MARCO TULIO CARVALHO DE ANDRADE

DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA EM SOFTWARE PARA SISTEMAS PARALELOS BASEADOS EM TRANSPUTERS

Dissertação Apresentada à Escola Politécnica da Universidade de São Paulo Para Obtenção do Título de Mestre em Engenharia de Eletricidade

SÃO PAULO

1990

MARCO TÚLIO CARVALHO DE ANDRADE

Engenheiro de Eletricidade, modalidade eletrônica, pela
Escola Politécnica da USP, 1982

DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA EM SOFTWARE PARA SISTEMAS PARALELOS BASEADOS EM TRANSPUTERS

Disertação Apresentada à Escola
Politécnica da Universidade de São
Paulo Para Obtenção do Título de
Mestre em Engenharia de Eletricidade

Orientador: Prof. Dr. Wilhelmus A. M. Van Noije.

SXO PAULO

1990

UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA

TERMO DE JULGAMENTO
DE
DEFESA DE DISSERTAÇÃO DE MESTRADO

Aos 24 *dias do mês de* agosto *de* 1990 *, às* 16:00 *horas,* no Departamento de Engenharia de Eletricidade

da Escola Politécnica da Universidade de São Paulo, presente a Comissão Julgadora, integrada pelos Senhores Professores Drs.Wilhelmus Adrianus Maria Van Noije Orientador do candidato, Edith Ranzini e Clésio Luis Tozzi

iniciou-se a Defesa de Dissertação de Mestrado do Senhor Engº Eletricista **MARCO TÚLIO CARVALHO DE ANDRADE**

Título da Dissertação: "Desenvolvimento de uma Interface Gráfica em Software para Sistemas Paralelos Baseados em Transputers"

Concluída a argüição, procedeu-se ao julgamento na forma regulamentar, tendo a Comissão Julgadora atribuído ao candidato as seguintes notas:

Prof. Dr. Wilhelmus Adrianus Maria Van Noije (10.0) (DEZ)

Profa.Dra.Edith Ranzini (10.0) (DEZ)

Prof. Dr. Clésio Luis Tozzi (10,0) (DEZ)

Para constar, é lavrado o presente termo, que vai assinado pela Comissão Julgadora e pelo Secretário da Seção de Pós-Graduação

São Paulo, 24 *de* agosto *de* 1990.

Presidente Wilhelmus Adrianus Maria Van Noije

Edith Ranzini

Clésio Luis Tozzi

Secretário Mara Fátima de Jesus Luz Sanches

Observações:

Homologada pela C.P.G. em reunião realizada à 10/09/1990

à meus Pais, José e Eny,

por possibilitarem a existência

de minhas emoçoes, e à

Marciléia, por materializá-las.

AGRADECIMENTOS

Ao Prof. Dr. Antonio Marcos de Aguirra Massola, inicialmente orientador, que numa demonstração de desprendimento com relação a interesses pessoais e visando exclusivamente as melhores condições possíveis para o encaminhamento desta dissertação, consentiu na transmissão da orientação deste trabalho. No entanto, deixou registradas suas contribuições, fundamentais ao desenrolar dos acontecimentos.

Ao Prof. Dr. Wilhelmus Adrianus M. Van Noije, pela orientação deste trabalho, e também pelo exemplo de competência e profissionalismo no qual tento me espelhar.

Ao Prof. Dr. Francisco J. O. Dias e à Profa. Dra. Edith Ranzini por incansáveis acompanhamentos administrativos indispensáveis à defesa desta dissertação.

A Escola Politécnica da USP e ao Instituto de Automática Industrial (do Ministério de Educação e Ciência do Governo Espanhol), pelo apoio à realização desta.

AGRADECIMIENTOS

Este trabajo ha sido posible gracias a la beca que me fue otorgada por el Ministerio de Educación y Ciencia de España, dentro del programa de Estancias Temporales de Científicos y Tecnólogos Extranjeros en España, y a la infraestructura proporcionada por el Instituto de Automática Industrial del CSIC. Deseo agradecer al Prof. Dr. Dn. José Antonio Cordero, director del IAI, Dr. Dn. Salvador Ros Torrecillas, secretario del IAI y al Dr. Dn. Ricardo García Rosa, jefe del Laboratorio de Informática, por la ayuda prestada para la realización de este trabajo.

Deseo hacer mención especial por la ayuda y la amistad recibida de parte de mis compañeros del Laboratorio de Informática, agradeciendo su colaboración en este trabajo. Además deseo agradecer a todo el personal del IAI que en todo momento ha demostrado interés en hacer mi estancia en España lo más agradable posible. De entre ellos me gustaría mencionar a Carlos Gonzalez (el "Rey de los colores en gráficos"), Lidia Pulido Lopez (por las buenas bromas), Nuria Julián Rigau (por las clases de Castellano), Jorge Gasos (por las conjugaciones verbales y las Comunidades Autónomas), Amadeo Ingerto Lechon (por su terminal del VAX y apoyo en la lucha contra el humo), Francisco Serradilla García (Dn. Paco, mi Poeta Español predilecto), Esther Santiago Rey (por las clases de gallego y geografía), Angela Ribeiro Seijas (por su simpatía), Fernando Cano (por las clases de VMS y por todas las cosas que me ha enseñado), Jesus Salido (porque hacia preguntas), Alejandro (porque no hacia preguntas), Pedro Daniel Fernandez Zuliani y Juan Tabera (ellos me permitieron ganar en el fútbol), Victor (de Méjico, me hizo perder en el fútbol), Cham (por no fumar en el laboratorio), Leopoldo (por el restaurante de Trujillo), Alfredo (por los pimientos rellenos), Pepe y Roberto (por las tardes de

fútbol), Alfonso Baños (por la amabilidad), M. Antonia Jiménez (por la sonrisa agradable), Amparo (por los caramelos), Victor (el compañero del "cuarto" autobus), José Nieto (por hablar de espacio), José Sanchez (por no hablar de espacio), Juan Gabriel (por las "minis"), Gotzone (por apuntarme para la furgoneta), Adelaida (por prestarme la tijera), Jesus (por el bueno comedor). A los Brasileños Dante José de Araújo, Jorge Silva, Leonardo Cunha Rosa, Evandro Ottoni y Jadir Eduardo Souza Lucas por recordarme a Brasil. A D. David Jiménez por las fotocopias y montaje de este y a Elena Agüero Gutierrez por "escucharme".

Finalmente, quiero expresar mi más sincera gratitud a mi directora de trabajo en España, Dra. Maria Teresa de Pedro Lucio, por su constante ayuda y motivación, profesional y personal, y al Dr. José Luiz Pedraza Domínguez, por brindarme con sus conocimientos, experiencia, "buen humor" y paciencia para la realización de este trabajo.

Quiero hacer hincapié en mi agradecimiento a estas personas a quienes tengo el honor de conocer y la ingenua pretensión de llamar amigos.

RESUMO

Este trabalho consta do desenvolvimento de uma interface gráfica em software para sistemas paralelos baseados em Transputers. Tal interface dispõe de um núcleo, cuja função é interpretar e ativar os comandos disponíveis, por meio de um protocolo definido, e um conjunto de rotinas gráficas básicas. A interface está estruturada de modo a realizar uma divisão espacial das tarefas de processamento para explorar o desempenho de sistemas paralelos.

De modo a introduzir o tema, há a apresentação de fundamentos sobre a arquitetura da família dos Transputers, à linguagem OCCAM 2 dos Transputers e ao sistema de desenvolvimento de programas para Transputers (tds2). Os recursos de hardware e a estruturação do software são mostrados. Também é feita uma discussão de questões de técnicas de otimização de "performance", concepção do sistema gráfico paralelo e considerações sobre os algoritmos utilizados. Algumas rotinas gráficas básicas foram desenvolvidas: linha, retângulo, círculo, semicírculo, quarto de círculo, oitavo de círculo, circunferência, semicircunferência, quarto de circunferência, elipses e rotinas para escrever cadeias de caracteres. Alguns algoritmos convencionais foram adaptados para adequá-los a atingir a melhor "performance" do sistema gráfico.

ABSTRACT

This work focuses the development of a graphical software interface for Transputer based parallel systems. This interface contains a kernel, which function is to interprete and activate the available commands, by means of a defined protocol, and a set of basic graphic routines. The interface is structured in order to reallize a spatial division of the processing tasks to explore the performance of parallel systems.

In order to introduce the subject, there is a tutorial on the Transputer family architecture, the OCCAM 2 Transputer language and the Transputer development system (tds2). The hardware resources and the software structure are shown. Furthermore, some performance otimization technics, the parallel graphic system conception and some considerations about the utillized algoritms are discussed. Some basic graphic routines were developed: line, rectangle, circle, semicircle, quarter of circle, eighth of circle, circumference, semicircumference, quarter of circumference, dognut, elipses and routines to write strings of characters. Some conventional algoritms were adapted to adequate them to achieve the best performance of the graphic system.

CONTEÚDO

LISTA DE SíMBOLOS E PALAVRAS RESERVADAS

ALT - construção alternativa da linguagem OCCAM

AND - comando de realização de um "E" lógico entre duas variáveis lógicas da linguagem OCCAM

assign - comando de associação de um valor a uma variável da linguagem OCCAM

A.U. - diz-se das rotinas que podem ser acessadas pelo usuário da interface gráfica

built-in - construído internamente ao sistema em referência

C - linguagem de programação

CHAN - usado declaração de canal em linguagem OCCAM

character - caractere

COMMENT - usado para definir uma linha ou um "fold" de comentário na linguagem OCCAM

Control Pantalla - usado para designar o subsistema de controle da tela. Conjunto de rotinas da interface que realizam o controle da tela.

Cp.Mp - canal definido entre o subsistema de "control pantalla" e o subsistema "memoria pantalla"

Cp.Org - canal definido entre o subsistema "control pantalla" e o programa organizador

default - definição assumida por um sistema ao ser inicializado, na ausência de quaisquer outras especificações de usuário

DOS - sistema operacional dos microcomputadores pessoais padrão IBM

DMA - "direct memory access" ou acesso direto à memória

event - canal de sincronização de eventos do gerador de temporização de vídeo programável

EXE - "fold" que contem programa executável

FALSE - atribuição lógica falsa a uma variável booleana da linguagem OCCAM

fold - traduz-se ao pé da letra por "dobra". Estrutura de hierarquização de programas em linguagem OCCAM

FOR - palavra reservada da linguagem OCCAM que designa quantas vezes uma repetição de comandos vai ocorrer

FUNCTION - palavra reservada da linguagem OCCAM que define uma função

G - prefixo que significa 10 elevado a nona potência (Giga)

HELP - palavra reservada para pedido de auxílio

IF - palavra reservada para construção condicional em linguagem OCCAM

IMS - prefixo de produtos da família INMOS

IMS B004 - placa do sistema de desenvolvimento de OCCAM

IMS B408 - módulo da placa gráfica que contêm as rotinas gráficas que atualizam a memória de tela

IMS B409 - módulo da placa gráfica que contem o temporizador de video programável e o gerador de "palette" de cores

INMOS - fabricante inglês da família de processadores denominados Transputers

INTn - definição de variável inteira de "n" bits em linguagem OCCAM

IS - palavra reservada para definição abreviada de variáveis em linguagem OCCAM

K - prefixo que significa mil vezes (Kilo)

Kbytes - mil bytes

Keyboard - Teclado

link - Elo físico de interligação para comunicação entre os Transputers

loop - Laço

M - prefixo que significa um milhão de vezes (Mega)

Mbytes - um milhão de bytes

Mbytes/s - um milhão de bytes por segundo

Memoria Pantalla - usado para designar o subsistema da memória de tela. Conjunto de rotinas da interface que realizam a escrita na tela.

Mflops - milhões de instruções de ponto flutuante por segundo

Mp.Cp - canal definido entre o subsistema de "memoria pantalla" e o subsistema "control pantalla"

Mp.Org - canal definido entre o subsistema de "memoria pantalla" e o subsistema "control pantalla"

N.A.U. - diz-se das rotinas que não podem ser acessadas pelo usuário da interface gráfica

NOT - palavra reservada que significa uma negação de uma variável lógica em OCCAM

OCCAM - linguagem de programação paralela desenvolvida para a INMOS para os Transputers

OF - palavra reservada en OCCAM para associar a definição de um canal com seu protocolo de utilização

on-chip - diz-se de algo que está contido dentro de um circuito integrado

OR - comando de realização de um "OU" lógico entre duas variáveis lógicas da linguagem OCCAM

Organizador - programa gerenciador da interface gráfica

Org.Cp - canal definido entre o programa "organizador" e o subsistema "control pantalla"

Org.Mp - canal definido entre o programa "organizador" e o subsistema de "memoria pantalla"

palette - palheta de cores adotada para o sistema gráfico

PAR - construção da linguagem OCCAM para execução dos subsequentes comandos em paralelo

PC - computador pessoal padrão IBM

PLACED - palavra reservada para realização de alocação de variáveis em posições de memória definidas na linguagem OCCAM

PROC - palavra reservada para definição de procedimentos em OCCAM

PROCESSOR - palavra reservada para definição dos processadores a serem utilizados para executar os programas OCCAM

PROGRAM - "fold" que contêm programas carregáveis em vários processadores

protocol - protocolo

RAM - "random access memory", ou memória de acesso aleatório

REALn - definição de um número real de "n" bits na linguagem OCCAM

REM - operação que extrai o resto de uma divisão

RESULT - define a variável que conterá o resultado de uma função OCCAM

ROUND - função que faz o arredondamento de valores quando se altera o tipo de uma variável

SC - designação do "fold" que contêm uma unidade de compilação separada

screen - tela

scroll - varredura de tela

SEQ - palavra reservada da construção OCCAM para sequencialização de comandos

slot - fenda, ou um dos conctores de placas para módulo de expansão do PC

string - cadeia de caracteres

timer - temporizador

tds2 - "Transputer development system 2", ou sistema de desenvolvimento de OCCAM II

TRUE - valor verdade de variáveis lógicas em OCCAM

TRUNC - palavra reservada para truncar o número de bits que representa uma variável em OCCAM

T212 - tipo de processador da família de Transputers da INMOS

T414 - tipo de processador da família de Transputers da INMOS

T800 - tipo de processador da família de Transputers da INMOS

UTIL - "fold" que contêm programas utilitários em OCCAM

VAL - palavra reservada que associa valores a variáveis em OCCAM

VALOF - palavra reservada que precede à definição da sequência de processos a ser executada por uma função em OCCAM

WHILE - principia uma repetição condicional de comandos em OCCAM

- símbolo que, precedendo uma sequência de números, significa que estes estão representados em hexadecimal

- - símbolo de operação de subtração

+ - símbolo de operação de adição

:= - símbolos de associação de valores em OCCAM

! - símbolo de envio de uma variável por um canal OCCAM

? - símbolo de leitura de uma variável por um canal OCCAM

<> - símbolo que significa "não igual a"

> - símbolo que significa "maior que"

< - símbolo que significa "menor que"

>= - símbolo que significa "maior ou igual que"

<= - símbolo que significa "menor ou igual que"

* - símbolo da operação de multiplicação

/ - símbolo da operação de divisão

~ - símbolo que implica na realização de uma negação de todos os bits da variável inteira que o precede

/\ - símbolo que implica na realização de um "E" lógico, bit a bit, dos valores de duas variáveis

\/ - símbolo implica na realização de um "OU" lógico, bit a bit, dos valores de duas variáveis

>< - símbolo implica na realização de um "OU EXCLUSIVO" lógico, bit a bit, dos valores de duas variáveis

<< - símbolo implica na realização de um deslocamento a esquerda, de um bit, na representação em bits de um inteiro

>> - símbolo implica na realização de um deslocamento a direita, de um bit, na representação em bits de um inteiro

LISTA DE FIGURAS

Figura B.4 - Diagrama em blocos do módulo IMS B409

Figura B.5 - Significado e estruturação dos campos do registrador de parâmetros (FIFO de parâmetros) do gerador de temporização programável do módulo IMS B409

LISTA DE TABELAS

1. INTRODUÇXO

1.1. Preliminares

A crescente penetração das arquiteturas paralelas no cenário da computaçào não se deve exclusivamente a tendências de poucos grupos de pesquisa, "lobbies" de grandes empresas ou modismos científicos de época mas fundamentalmente às características das necessidades atuais de capacidade de processamento. Ultrapassada a época em que o "software" dos sistemas era menos custoso que seu "hardware" a busca por soluções que barateassem seu desenvolvimento a custa de duplicação dos recursos de "hardware" também influiu para que se pensasse na estratégia do paralelismo como uma alternativa viável.

As aplicações que exigem alta capacidade computacional como por exemplo análises meteorológicas, modelamento por elementos finitos, aplicações em tempo real ou ainda problemas que exijam grande número de operações aritméticas em ponto flutuante, de alta precisão, compõe algumas das razões para o uso efetivo dos sistemas paralelos [VALERO-85]. Dentre as famílias de máquinas paralelas emergentes destaca-se a estabelecida pela linha de processadores INMOS, denominados Transputers [HOMEWOOD-87] [INMOS-87-3] [INMOS-87-4] [STEIN-88]. Um transputer não é em si um sistema paralelo mas sim um "tijolo" para a construção de sistemas paralelos. Devido à facilidade de constituição de arquiteturas paralelas nas mais variadas configurações possíveis e à disponibilidade de ambientes de programação em linguagem paralela (OCCAM) desenvolvidos pela própria INMOS para suporte aos usuários destes sistemas, a opção dos transputers passou a ser muito considerada pela comunidade científica. Com o estabelecimento de uma nova família de processadores para implementar sistemas

paralelos vêm a tona mesmas necessidades pelas quais passaram todas as arquiteturas desenvolvidas até então em termos de interface gráfica com o usuário [TEK-87]. Neste caso específico a própria INMOS apresentou uma solução a nível de "hardware" com uma placa gráfica implementada com transputers (vide apêndice B), que apresenta compatibilidade com sistemas construidos com base nestes processadores.

1.2. Objetivos

O que se propõe, com esta dissertação, é apresentar uma solução a nível de "software", utilizando a placa gráfica citada, desenvolvendo uma interface gráfica em "software" compatível com a modularidade dos processos que se executam em transputers [ANDRADE-90]. Tal interface é constituida por um programa organizador, que se encarrega de processar as ordens recebidas dos vários processadores de um sistema paralelo genérico com transputers e de uma biblioteca de rotinas gráficas básicas, tais como, desenho de linhas, retangulos, círculos, semicírculos, quarto de círculos, coroa circular, caracteres ASCII, etc. A interface foi idealizada de modo a garantir a total compatibilidade com a característica de modularidade dos sistemas a transputers.

Não se pretendeu canalizar a atenção deste trabalho em apenas dissertar sobre o tema de gráficos para Transputers do ponto de vista de sistema, mas na medida do desenrolar deste, apresentar alguns algorítmos ou técnicas de implementação de gráficos aplicáveis aos Transputers, usando como instrumento a linguagem OCCAM. Apresenta-se uma proposta de uma interface gráfica aderente às necessidades básicas de sistemas atuais, fornecendo as informações necessárias ao entendimento do contexto em que esta interface se insere e gerando a documentação dos programas e rotinas desenvolvidos. Alguns aspectos dos algorítmos

implementados são citados nos pontos onde sejam de interesse em termos de contribuição do trabalho e também para a compreensão da operação das rotinas, embora não seja necessário conhecer este nível de detalhe para seu entendimento e utilização.

A comunicação entre processos no sistema obedece a um protocolo próprio, desenvolvido neste trabalho, que foi feito para adequá-lo às características dos sistemas que vão utilizá-lo. Tal protocolo, idealizado com uma sequência de comandos a mais completa e simples possível, torna factível a inserção desta interface em sistemas paralelos genéricos baseados em Transputers e se encontra descrito no ítem 3.1.2.2. deste texto.

I.3 Constituição do Trabalho

Esta dissertação se compõe de quatro capítulos (incluindo a Introdução), uma lista de figuras, lista de tabelas, lista de símbolos e palavras reservadas utilizados, uma relação bibliográfica e três apêndices.

A apresentação dos transputers faz-se no capítulo 2, com suas características gerais em termos de arquitetura e apresentação de predicados específicos dos principais processadores da família dos Transputers. Para a linguagem de programação de alto nível utilizada (OCCAM) destinou-se a segunda parte do capítulo 2. No capítulo 3, primeira parte, tratamos sobre a infraestrutura de "hardware" disponível para a implementação e utilização desta interface e suas rotinas gráficas, e da estruturação do "software" desenvolvido para constituir o sistema. Trata-se ainda do tema de otimização de "performance" para sistmas baseados em Transputers e sobre a concepção do sistema gráfico paralelo, com os algorítmos

utilizados na sua implementação. Realiza-se uma descrição das rotinas do programa organizador, o qual executa as chamadas das rotinas gráficas, na segunda parte do capítulo 3. Mais adiante, ainda neste capítulo, temos a descrição das rotinas que fazem a carga do "palette" de cores e dos parâmetros necessários para o funcionamento do processador T222 que, junto com um "chip" gerador do "palette" de cores, realiza o controle da temporização e da conversão de dados de "pixels" digitais a analógicos. Em seguida faz-se a descrição das rotinas gráficas e do programa principal de chamada às mesmas (programa este que interpreta comandos enviados pelo programa organizador, segundo um protocolo específico). Além da clasificação de todas as rotinas desenvolvidas neste trabalho, feita nos capítulo 3 com base nas funções que realizam, desde o ponto de vista do usuário podemos dividir as rotinas de memória de tela em dois grupos: "rotinas acessíveis pelo usuário" (A.U.), às que se acessa segundo o protocolo do ítem 3.1.2.2; "rotinas não acessíveis pelo usuario" (N.A.U.), que são rotinas auxiliares de uso interno do sistema ou de acesso indireto desde outras rotinas. Sómente proceder-se-á a esta clasificação com as rotinas da memória de tela porque são elas que constituem a biblioteca gráfica propriamente dita, ou seja, são aquelas que importam desde o ponto de vista do usuário (não transparentes ao usuário) e as demais constituem o ambiente necessário para sua utilização e desenvolvimento. Caso haja o interesse de simplesmente utilizar as rotinas gráficas da interface que são acessíveis, pode-se dirigir diretamente ao ítem 3.1.2.2. e ao 3.2.3. onde se encontra toda a informação necessária para seu uso. Na apresentação das rotinas no capítulo 3 descreve-se a função que estas realizam e, em algumas, incluem-se informações de interesse como: algorítmos desenvolvidos, variáveis globais atualizadas, rotinas auxiliares empregadas.

A aplicação da interface na solução de um problema prático é apresentada no capítulo 3. Finalmente, as perspectivas futuras de desenvolvimento do sistema e as conclusões sobre o trabalho, serão apresentadas no capítulo 4. Acompanha esta dissertação uma bibliografia dos artigos e livros consultados.

Desenvolve-se uma introdução à placa de suporte ao sistema de desenvolvimento de OCCAM (tds2) no apêndice A. Uma descrição da placa gráfica com Transputers, que foi utilizada como base de "hardware" pode ser vista no apêndice B. O apêndice C consiste na listagem de todos os programas e rotinas desenvolvidos.

2. ARQUITETURA DOS TRANSPUTERS E LINGUAGEM OCCAM

Neste capítulo será apresentado na primeira parte uma introdução aos Transputers, do ponto de vista de hardware. Na segunda parte uma breve descrição de algumas características da linguagem OCCAM (nome derivado de um filósofo/monge Inglês do século XIV [POUNTAIN-90]), que serão úteis no desenrolar do texto, e fundamentos sobre o sistema de desenvolvimento de programas em OCCAM.

2.1. Arquitetura dos Transputers

2.1.1. Visão Geral

Um Transputer é um microcomputador com memória local própria e com enlaces físicos específicos para a interconexão com outros transputers (Figura 2.1).

Deste modo a arquitetura dos Transputers define uma família de componentes VLSI programáveis. A definição da arquitetura recai sobre dois aspectos de diferentes abordagens, quais sejam o aspecto lógico e o aspecto físico. O aspecto lógico consiste em como um sistema de Transputers é projetado e programado. O aspecto físico diz respeito a como os Transputers, sendo componentes VLSI, são interconectados e controlados.

Um membro típico da família de Transputers é um "chip" individual provido de processador, memória e enlaces de comunicação (" links ") os quais fornecem conexão ponto a ponto entre Transputers. Além disto cada Transputer contêm circuitaria especial e interfaces para adaptação a um uso específico. Por exemplo, um Transputer de controle de periféricos, tal como um

controlador gráfico, tem interfaces feitas sob medida para os requisitos de um dado dispositivo.

Os Transputers podem ser utilizados como processadores únicos ou para compor redes de processadores constituindo sistemas concorrentes de alta "performance". Uma rede de Transputers é facilmente construída usando comunicação ponto a ponto como indica a Figura 2.2. Cada Transputer pode ser conectado a quatro outros Transputers, como máximo, e os "links" utilizados para tais conexões podem ser quaisquer. Pode ocorrer que haja sistemas projetados para serem interconectados por meios de "links" especificados, mas isto deve normalmente vir especificado pelo projetista.

Transputers podem ser programados em linguagens de alto nível tais como C e são projetados para assegurar que os programas compilados terão execução eficiente.

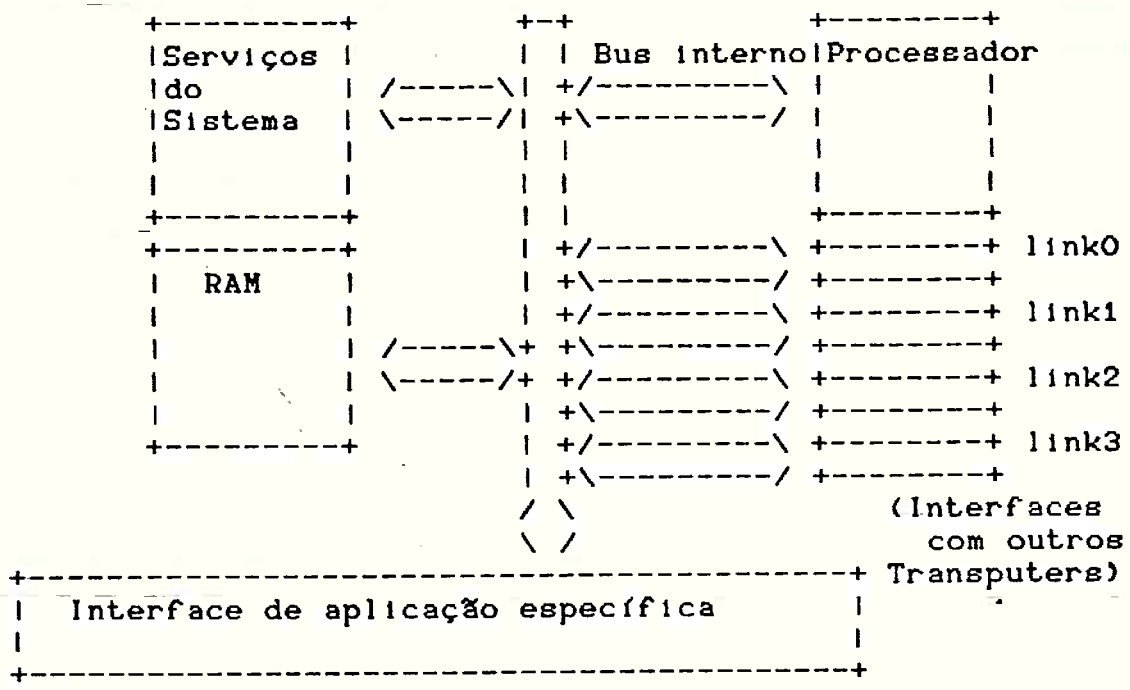


Figura 2.1 - Arquitetura genérica da família dos Transputers

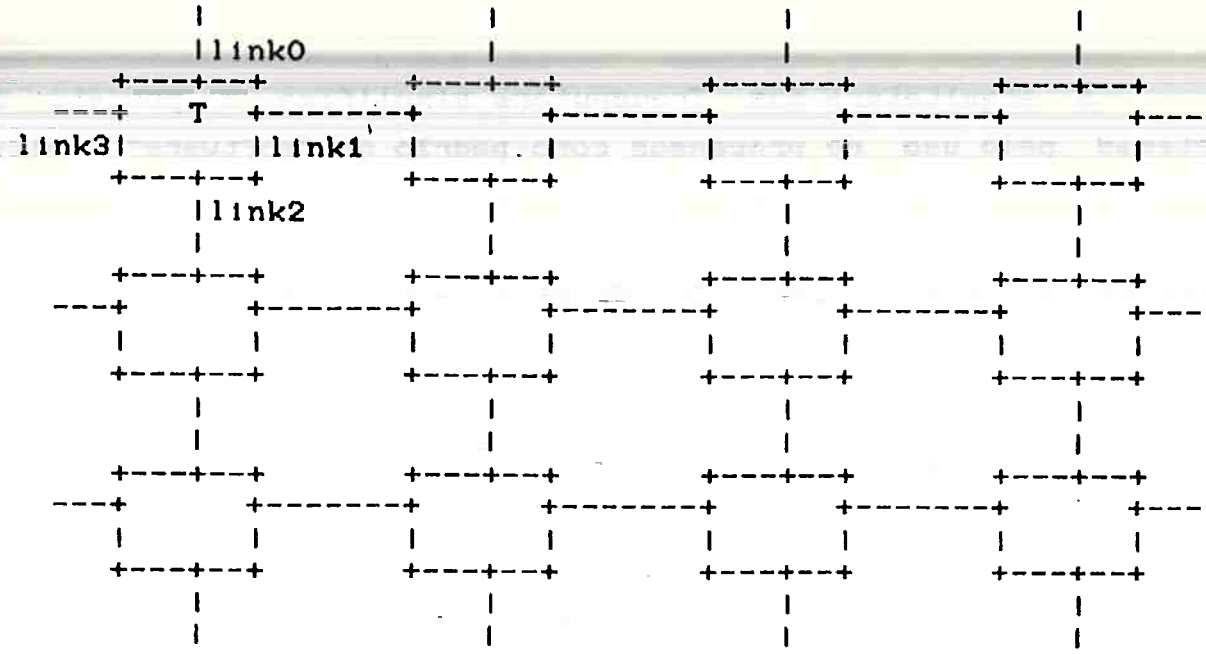


Figura 2.2 - Exemplo de configuração de Transputers para constituir sistema de processamento paralelo

Onde for necessário explorar os recursos de computação concorrente, mas ainda fazer uso de linguagens padrão, a linguagem paralela OCCAM (vide parte 2, capítulo 2) pode ser usada como um ambiente para ligar módulos objeto escritos nas linguagens escolhidas. Programando o sistema como um todo em OCCAM mantemos as vantagens de uma linguagem de alto nível com uma eficiência máxima de execução de programas. Além disto há a possibilidade de exploração das características especiais dos Transputers [INMOS-86]. A linguagem OCCAM constitue uma ferramenta de projeto de sistemas concorrentes com um papel semelhante ao da álgebra Booleana para o projeto de sistemas eletrônicos a partir de portas lógicas [INMOS-86-2].

2.2. Fundamentos

A arquitetura dos Transputers simplifica o projeto dos sistemas pelo uso de processos como padrão de "software" e blocos construtivos de "hardware". Um sistema inteiro pode ser concebido a partir de sua constituição em processos, desde de sua configuração até as operações de entrada e saída de baixo nível e as interrupções em tempo real.

O bloco construtivo de "software" é o processo, ou seja, um sistema é projetado em termos da interconexão de um conjunto de processos, onde cada processo é visto como uma unidade independente. Estas "unidades independentes" ou processos se comunicam entre si através de canais ponto a ponto construídos internamente ao Transputer e transparentes ao usuário. A especificação destes canais faz-se pela sua definição ao nível de "software" e pelas mensagens que envia e recebe. A sincronização entre a execução de todos os processos é automática eliminando a necessidade de fazer uso de qualquer mecanismo de sincronização adicional.

Um programa sendo executado em um Transputer é formalmente equivalente a um processo OCCAM, e uma rede de Transputers pode ser descrita diretamente como um programa OCCAM. Pode-se por extensão afirmar que um Transputer executando um processo descrito em OCCAM constitue um processo de hardware [INMOS-86], assim processos podem ser implementados em "hardware". Os processos podem ser projetados e compilados independentemente, materializando "caixas pretas" onde não se sabe o que ocorre em seu interior. O único que se sabe é o que aparece pelas comunicações entre seus enlaces físicos ("links"), os quais são os suportes físicos para os canais especificados em linguagem OCCAM.

A configuração típica da interligação entre quatro Transputers seria a apresentada na Figura 2.3, que pode ser

denominada de um nó de quatro Transputers. A possibilidade de se especificar funções como processos em "hardware" permite que um sistema de Transputers forneça infraestrutura para aplicações específicas, como por exemplo aplicações gráficas [McCONNEL-88]. Tal função, que se tomaria como exemplo, poderia ser especificada por um processo OCCAM e implementada em "hardware", com acesso por meio dos canais OCCAM. Em outras palavras, executam-se rotinas gráficas em um determinado Transputer, os outros Transputers que fazem os cálculos para a geração das saídas gráficas necessárias "enxergam" as rotinas que fazem os desenhos na tela como se fossem canais, cujo protocolo de troca de informação forçará a geração da atividade escolhida.

2.1.3. Arquitetura dos Sistemas

A seguir será apresentada uma introdução a arquitetura dos sistemas constituidos com os "chips" da família dos Transputers.

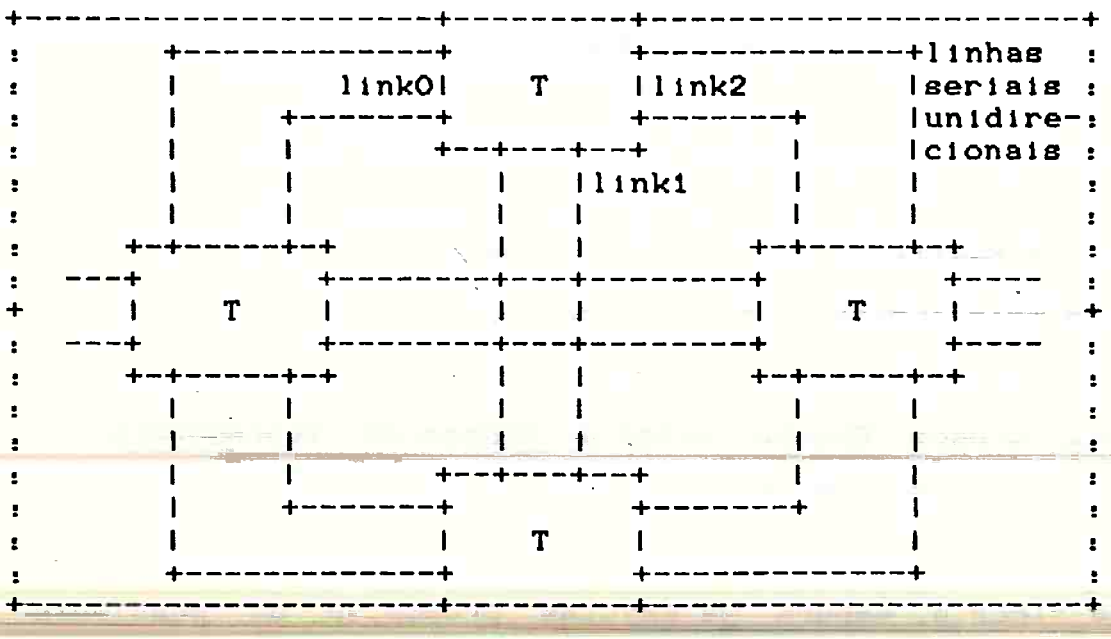


Figura 2.3 - Nó típico de quatro Transputers

Estamos diante de um sistema facilmente caracterizável por blocos básicos, um "chip" de características definidas onde se correm processos de "software", constituindo um processo de "hardware". A interligação entre vários processos de "hardware" pode gerar um novo e único processo, estabelecendo uma relação de hierarquia, e com isto tal sistema pode passar a ser agora um componente de um sistema maior.

A implementação de sistemas fica então simplificada pela geração e interconexão de primitivas básicas nos vários níveis de hierarquia definidos. Tal interconexão é possível por meio dos enlaces físicos ("links") que estabelecem uma comunicação ponto a ponto. Cada um dos membros da família Transputer possue um ou mais "links" de comunicação que podem ser interligados a um "link" de algum outro componente desta família. Isto permite a construção de redes de Transputers de configuração e tamanhos arbitrários, ou seja, várias topologias.

Um outro tipo de enfoque em termos de troca de informação interna em sistemas paralelos é o de compartilhamento de vias por vários processadores ("multi-processor buses"). Existem algumas vantagens da comunicação ponto a ponto em relação à comunicação por meio de vias compartilhadas [INMOS-86-2]:

- não há conflitos no sistema de comunicação a despeito do número de Transputers do sistema.

- não há penalidades quanto a cargas capacitivas a medida que se inserem mais Transputers ao sistema.

- a largura de banda de comunicação não satura na medida em que o sistema cresce. Quanto maior o número de Transputers no sistema, maior a largura de banda de comunicação total do sistema.

Entretanto, em um sistema muito grande as comunicações devem ser curtas e locais, embora já existam propostas de possíveis soluções para estes casos [POUNTAIN-90-2].

Cada Transputer possui sua própria memória local,sendo a largura de banda de memória proporcional ao número de Transputers do sistema. Por outro lado se os sistemas fossem constituidos de uma memória global, para acesso compartido de todos os processadores, um acréscimo no número de processadores configuraria um decréscimo na largura de banda de memória. Além disto as interfaces de memória são individuais, não compartilhadas, e são separadas das interfaces de comunicação. Estas interfaces de comunicação podem por sua vez serem otimizadas para cada aplicação (possibilidade sustentada pela implementação de vários tipos de componentes da família INMOS), resultando num máximo de eficiência com um mínimo de componentes externos.

2.1.4. Comunicação Entre Transputers

Cada enlace físico de comunicação fornece um meio de comunicação direta entre processos sendo executados em Transputers interligados. Para que ocorra uma comunicação sincronizada cada variável trocada entre processos, representada por uma mensagem, deve resultar no envio de um "acknowledge", assegurando o término e a corretude do processo. Como consequência deste protocolo de comunicação, um enlace físico, necessita como mínimo um caminho físico em cada sentido (vide Figura 2.4).

Pela simples conexão de uma interface de "link" de um Transputer com uma interface de "link" de outro Transputer se estabelece um "link" de comunicação entre estes dois Transputers. Este seria constituido de duas linhas de sinal unidirecionais por meio das quais os dados e o "acknowledge" seriam transmitidos serialmente. As duas linhas de sinal do "link" funcionam do ponto de vista lógico como dois canais OCCAM, um em cada sentido, que necessitam de um protocolo simples para trabalhar (vide parte 2, capítulo 2) e onde cada linha de sinal carrega informação de dados

e de controle. O protocolo dos "links" se encarrega da sincronização das comunicações e seu uso permite a transmissão de sequências de bytes de tamanhos arbitrários. Esta flexibilidade pode ser utilizada pelo usuário para conectar Transputers de distintos tamanhos de palavra.

Cada mensagem é constituida do envio de uma sequência de bytes (um por vez), necessitando apenas armazenar um único byte no Transputer receptor em cada etapa de modo a não haver perda de informação. O tipo de transmissão tem a característica "START/STOP", iniciando com dois "start bits" 1, sendo seguido pelo byte de informação e terminado por um "stop bit" 0 (vide Figura 2.4), para o caso de byte de dados. Para o caso de "acknowledge" o segundo "start bit" é um 0, isto faz com que o Transputer possa diferenciar se está recebendo um dado ou um "acknowledge".

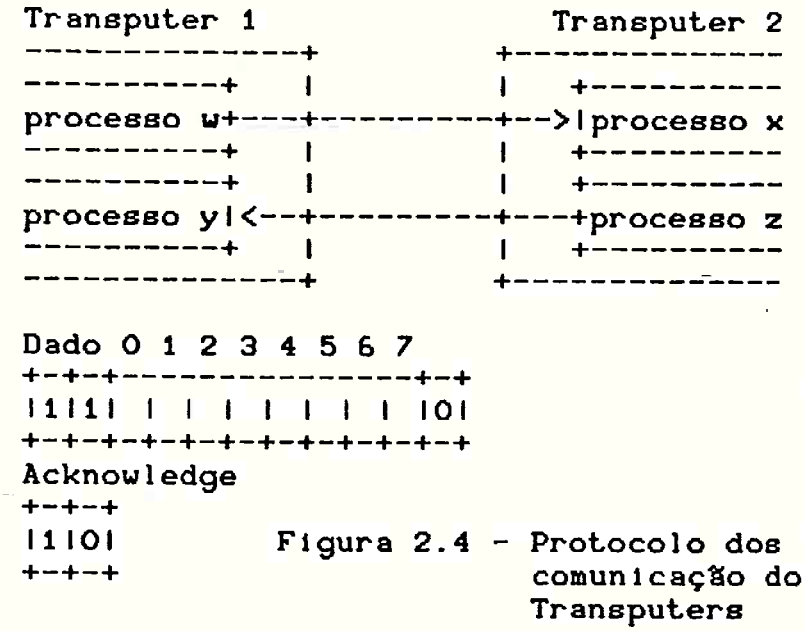


Figura 2.4 - Protocolo dos "links" de comunicação dos Transputers

Depois do envio de um byte de dados por uma das linhas, o transmissor espera até que receba um "acknowledge" que virá pela outra linha para poder enviar o próximo byte. Este "acknowledge" tardará o tempo de 2 bits para ser gerado porque este é o tempo necessário para que se verifique o valor do segundo "start bit" que determina se a sequência é de dados ou de "acknowledge". O recebimento do "acknowledge" tem o duplo significado de que o receptor foi capaz de receber o byte enviado e que está apto a receber o próximo. O "link" do transmissor pode reiniciar os processos de transmissão apenas quando o "acknowledge" do último byte foi recebido. O processador " sabe " qual é o último byte porque o programa executado neste definiu o tamanho em bytes da variável que foi transmitida e o programa executado no processador receptor espera receber uma variável deste mesmo tamanho de palavra. Caso contrário os dois processos estarão bloqueados sendo necessário para evitar tal problema uma correta definição dos tamanhos de variáveis que serão trocados entre estes.

Os dados a serem transmitidos por um determinado Transputer e os "acknowledges" gerados por este devido ao recebimento de dados, são multiplexados na mesma linha de sinal. Assim uma das linhas unidirecionais de um Transputer hora envia dados ao outro Transputer, hora envia "acknowledges", pelo mesmo caminho físico. Por esta razão a despeito de que a nível de programa se faça a definição de um canal lógico que apenas transmite dados do Transputer 1 ao Transputer 2 e nunca em sentido contrário, os dois "links" físicos tem que estar presentes, porque o Transputer 1 enviará os dados por um "link" e receberá os "acknowledges" pelo outro "link". Caso o Transputer 1 também recebesse dados isto seria pelo mesmo "link" físico pelo qual recebe os "acknowledges" e neste sentido é que ocorre uma multiplexação entre dados e "acknowledge" em cada "link" físico. A taxa de transmissão entre

"links" é da ordem de 10 Mbit/s, a despeito da "performance" individual de cada Transputer, o que permite que Transputers de diferentes "performances" sejam conectados.

2.1.5. Alguns Membros da Família de Transputers

A seguir apresentam-se algumas características individuais de alguns processadores e periféricos da família dos Transputers.

2.1.5.1. O Processador T414

O IMS T414 é o primeiro membro da família de Transputers (o primeiro a ser anunciado na literatura) que tem todos os seus constituintes compatíveis com a arquitetura apresentada anteriormente (Figura 2.1 e ítem 2.1.3) [INMOS-87-3]. A arquitetura específica pode ser vista na Figura 2.5..

O IMS T414 constitue-se de um processador de 32 bits, com quatro "links" padrão de comunicação, 2K bytes de memória "on-chip", com 0,1 MFLOPs [HOMEWOOD-87], uma interface de memória e é construido utilizando-se um processo CMOS de 1,5 micron. O projeto deste processador fornece suporte direto ao modelo concorrente da linguagem OCCAM.

O T414 usa transferência de blocos de memória via DMA ("direct memory access") para transferir mensagens entre memória e outros Transputers através dos "links" de comunicação. Os "links" e o processador podem operar concorrentemente permitindo a continuidade do processamento enquanto há transferência de dados [MAY-87-2]. Os 2K bytes de memória interna permitem uma taxa de transmissão de dados de 80 MBytes/s para acesso via processador. Esta memória interna também pode ser acessada via "links" de comunicação (a uma taxa menor).

O T414 pode diretamente acessar um espaço de memória de 4 G bytes. A interface de memória de palavra de 32bits usa a técnica e multiplexar linhas de dados e endereços e tem uma taxa de transferência de cerca de 27 MBytes/s (para acesso do processador à memória externa, via interface específica). Um controlador de memória configurável realiza todo o sincronismo de sinais de temporização, controle de sinais de refrescamento para memórias dinâmicas ou para sistemas mistos de memória.

O controlador de memória suporta dispositivos periféricos mapeados em memória, os quais usam DMA ("direct memory access"). Os "links" podem ser interfaceados com periféricos via dispositivos especiais de adaptação projetados pela INMOS [INMOS-86-2]. Um periférico pode requisitar a atenção de algum serviço via o pino de "event" [INMOS-86]. O processador dispõe de "timers" internos para temporização de processos.

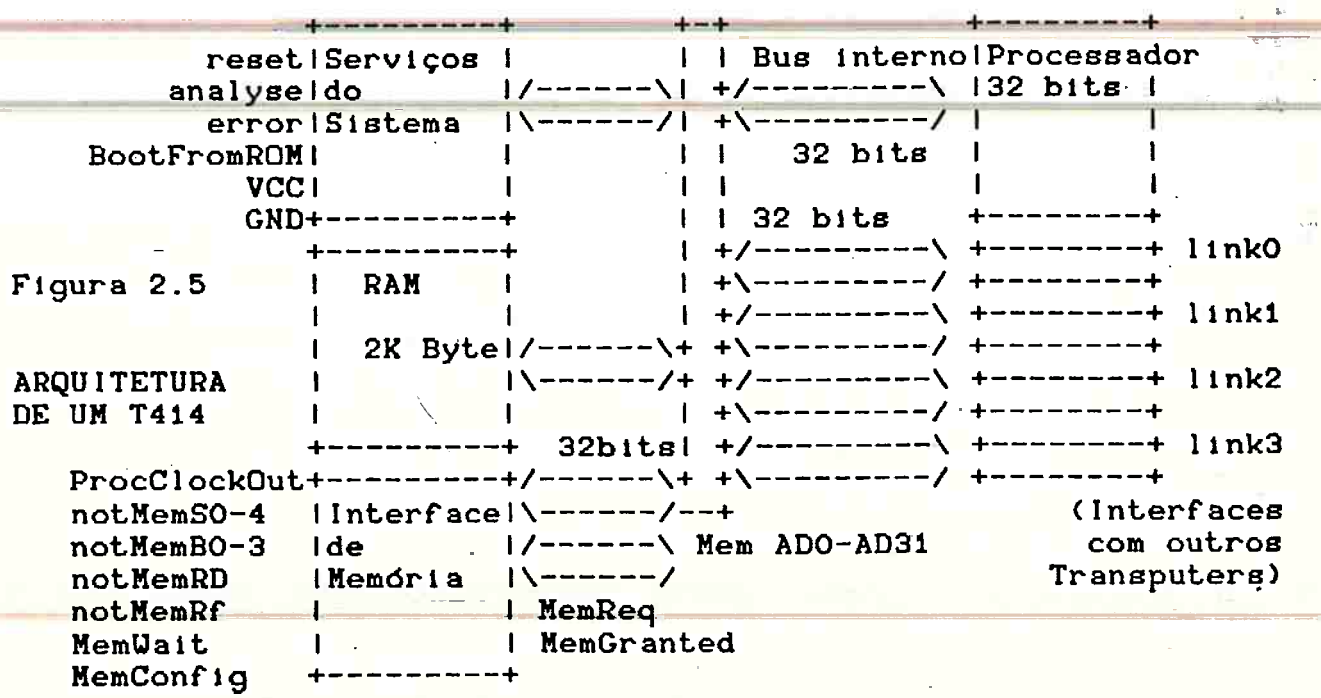


Figura 2.5 ARQUITETURA DE UM T414

O método ótimo de programação de um T414 é a utilização da linguagem OCCAM a despeito de que se possa fazê-lo por meio de outras linguagens de alto nível como C, Pascal ou Fortran.

O espaço de endereçamento interno do T414 se encontra na faixa de #80000000 a #800007FF (endereços em hexadecimal, indicados pelo símbolo "#").

2.1.5.2. O Processador T212

O IMS T212 é outro membro da família de Transputers que tem todos os seus constituintes compatíveis com a arquitetura apresentada anteriormente (Figura 2.1 e ítem 2.1.3), assim como explicado para o IMS T414. A arquitetura específica pode ser vista na Figura 2.6. [INMOS-86] [INMOS-86-2].

O IMS T212 constitue-se de um processador de 16 bits, com quatro "links" padrão de comunicação, 2K bytes de memória "on-chip", uma interface de memória e é construido utilizando-se um processo CMOS de 1,5 micron.

O projeto deste processador fornece suporte direto ao modelo concorrente da linguagem OCCAM.

O T212 usa transferência de blocos de memória via DMA ("direct memory access") para transferir mensagens entre memória e outros Transputers através dos "links" de comunicação. Os "links" e o processador podem operar concorrentemente permitindo a continuidade do processamento enquanto há transferência de dados. Os 2K bytes de memória interna permitem uma taxa de transmissão de dados de 40 MBytes/s com acesso interno via processador. Estes 2K

bytes de memória também podem ser acessados via "links" de comunicação.

O T212 pode diretamente acessar um espaço de memória de 64 KBytes. A interface de memória de palavra de 16 bits não usa a técnica de multiplexar linhas de dados e endereços como no IMS T414 e tem uma taxa de transferência de cerca de 20 MBytes/s. Aqui temos uma via de endereços de 16 bits e uma via de dados bidirecional de 16 bits.

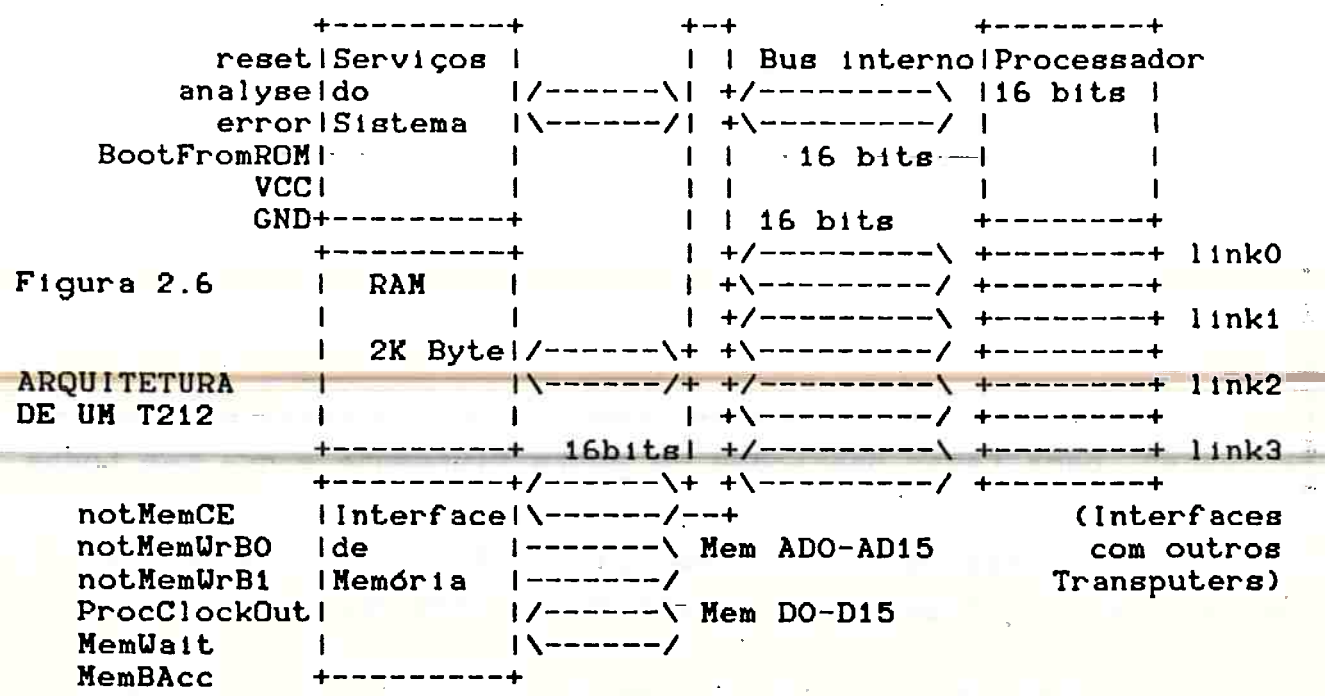


Figura 2.6

ARQUITETURA DE UM T212

Um controlador de memória configurável realiza todo o sincronismo de sinais de temporização, controle de sinais de refrescamento para memórias dinâmicas ou para sistemas mistos de memória.

O controlador de memória suporta dispositivos periféricos mapeados em memória, os quais usam DMA ("direct memory access"). Os "links" podem ser interfaceados com periféricos via dispositivos especiais de adaptação projetados pela INMOS. Um periférico pode requisitar a atenção de algum serviço via o pino de "event" e o processador também dispõe de "timers" internos para temporização de processos, como no IMS T414 [INMOS-86].

O método ótimo de programação de um T212 é a utilização da linguagem OCCAM a despeito de que se possa fazê-lo por meio de outras linguagens de alto nível como C, Pascal ou Fortran.

O espaço de endereçamento interno do T212 está definido entre #8000 e #87FF (endereços hexadecimais).

2.1.5.3. O Processador T800

O IMS T800 é um membro da família de Transputers dotado de "hardware" para fazer operações de ponto flutuante e que tem todos os seus blocos internos compatíveis com a arquitetura apresentada anteriormente (Figura 2.1 e ítem 2.1.3). A arquitetura específica pode ser vista na Figura 2.7. [STEIN-88] [INMOS-86].

O IMS T800 constitue-se de um processador de 32 bits, uma unidade de operações em ponto flutuante de 64 bits, quatro "links" padrão de comunicação, 4K bytes de memória "on-chip", uma interface de memória e é construido utilizando-se o mesmo processo CMOS de 1,5 micron dos processadores supracitados. O IMS T800-20 (20 MHZ) é capaz de trabalhar a uma taxa de até 1,5 MFLOPs e o IMS T800-30 (30 MHz, disponível a partir de 1988) pode chegar até 2.5 MFLOPs [HOMEWOOD-87]. O projeto deste processador assim como dos dois processadores anteriores, fornece suporte direto ao modelo concorrente da linguagem OCCAM.

A unidade de ponto flutuante de 64 bits fornece potencialidades de se fazer operações em tamanho de palavra simples ou tamanho de palavra duplo de acordo com a norma ANSI-IEEE 754-1985 que é o padrão para operações aritméticas em ponto flutuante. Além disto é capaz de realizar operações aritméticas em ponto flutuante concorrentemente com a atividade do processador, com uma "performance" estimada de 1,5 MFlops, conforme foi mostrado anteriormente.

A unidade de ponto flutuante consiste em uma máquina microcodificada a qual, além de operar concorrentemente o faz sob o comando do processador central. Contém uma pilha para operações de números em ponto flutuante, representados no formato IEEE. Toda comunicação entre memória e a unidade de ponto flutuante é feita sob a coordenação do processador central.

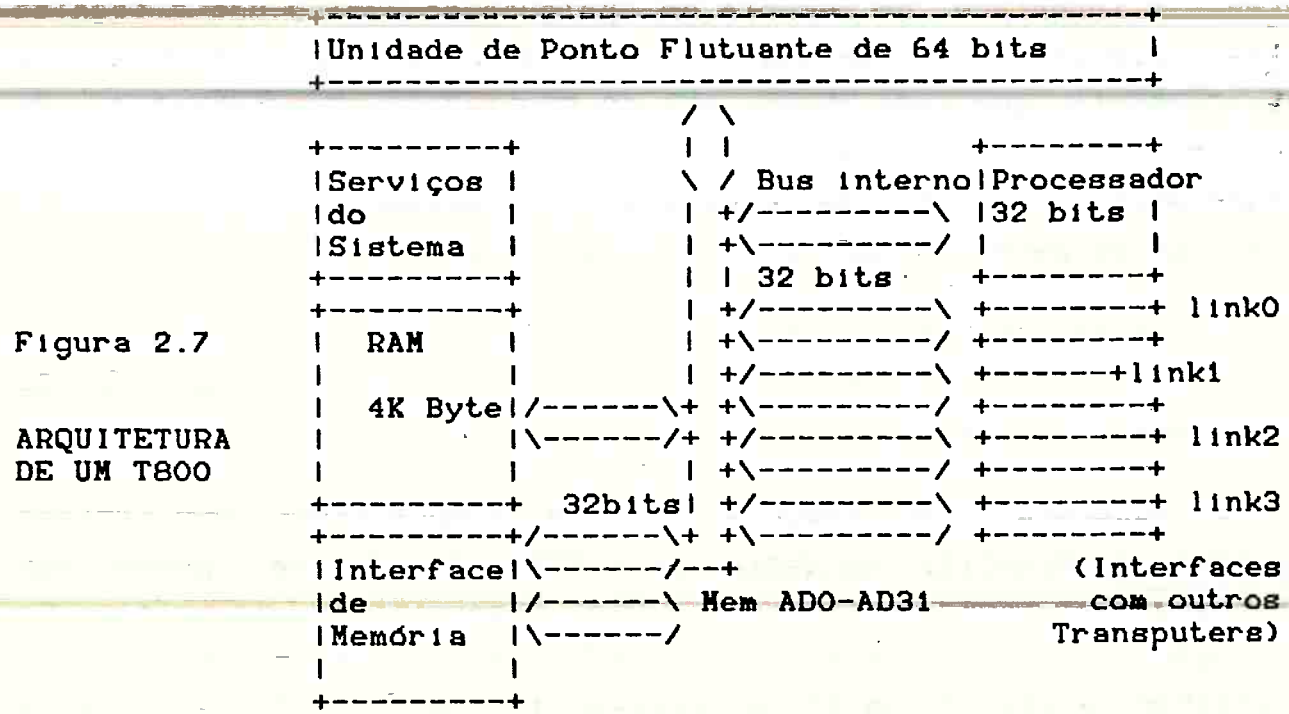


Figura 2.7

ARQUITETURA DE UM T800

O T800 usa transferência de blocos de memória via DMA para transferir mensagens entre memória e outros Transputers através dos "links" de comunicação. Os "links" e o processador podem operar concorrentemente permitindo a continuidade do processamento enquanto há transferência de dados. Caso haja transferência interna entre os "links" e o processdor, estes podem dar uma taxa de transmissão unidirecional de 1,7 MBytes/s e bidirecional (combinada entre mensagens e "acknowledge") de 2,3 MBytes/s. As velocidades de operação dos "links", para transmissões para o mundo exterior, podem ser de 10 Mbits/s ou 5 Mbits/s tornando os IMS T800 compatíveis com os processadores descritos anteriormente. Os 4K bytes de memória interna permitem uma taxa de transmissão de dados de 80 (ou até 120) MBytes/seg com acesso via processador ou via "links" de comunicação.

O T800 pode diretamente acessar um espaço de memória de 4 G bytes. A interface de memória de palavra de 32bits usa a técnica e multiplexar linhas de dados e endereços e tem uma taxa de transferência que pode chegar até 40 MBytes/seg. Um controlador de memória configurável realiza todo o sincronismo de sinais de temporização, controle de sinais de refrescamento para memórias dinâmicas ou para sistemas mistos de memória.

As duas memórias, interna e externa, são interpretadas do ponto de vista lógico como um único espaço de memória de endereçamento contíguo.

O controlador de memória suporta dispositivos periféricos mapeados em memória, os quais usam DMA. Os "links" podem ser interfaceados com periféricos via dispositivos especiais de adaptação projetados pela INMOS [INMOS-86-2]. Um periférico pode requisitar a atenção de algum serviço via o pino de "event" e este processador também dispõe de "timers" internos para temporização de processos [INMOS-86].

O método ótimo de programação de um T800 é a utilização da linguagem OCCAM a despeito de que se possa fazê-lo por meio de outras linguagens de alto nível como C, Pascal ou Fortran.

O espaço de endereçamento do T800 se encontra na faixa de #80000000 a #80001000 para o espaço interno.

O IMS T800 tem implementado em microcódigo facilidades para operações gráficas. Três novas instruções foram somadas ao grupo de instruções da família Transputer para aumentar a velocidade de operação quando há necessidade de se fazer transferência de estruturas de dados bidimensionais (matrizes de bytes). Movimentos de blocos de dados podem agora ser realizados no limite de velocidade de leitura da memória, passando a ser o "hardware" e não o "software" o fator de contenção da "performance" destas operações. As instruções de movimentos de blocos de dados bidimensionais podem ser utilizadas para fornecer operações gráficas tais como "scroll" de tela, centralização de figuras, janelamento e atualização de tela [INMOS-87].

Também existe agora uma instrução de contagem de bits 1 em um byte apropriado a aplicações de tratamento digital de imagens e reconhecimento de padrões [INMOS-87].

Este processador dispõe de uma interface de memória configurável projetada para permitir fácil conexão com uma variedade de tipos de memória externa com o mínimo de componentes externos. A interface permite a conexão com RAMs estáticas e dinâmicas, ROMs e dispositivos periféricos mapeados em memória. Possui uma via de dados e endereços de 32 bits multiplexada com um espaço de memória endereçável diretamente de 4 G bytes.

Existem até o momento as versões IMS T800-20 e IMS T800-30 que trabalham em uma frequência de 20 ou 30 MHz respectivamente.

Dispõe-se também de dois pinos para permitir que a frequência de operação interna do dispositivo seja definida pelo usuário. São eles os pinos "ProcSpeedSelect0" e "ProcSpeedSelect1" que quando estão ambos em nível baixo fazem com que o IMS T800 seja

compatível em velocidade de operação com o IMS T414 a 20 MHz [INMOS-87].

Concluindo, as diferenças básicas entre os três processadores apresentados são quanto a número de bits de palavra, quantidade de RAM interna e medida de "performance" em MFLOPs. O IMS T212 trabalha com 16 bits, enquanto o IMS T414 e o IMS T800, com 32 bits. O T414 e o T212 tem 2K bytes de RAM interna e o T800 dispõe de 4K bytes. A medida de "performance" em MFLOPs vai de 0,1 MFLOPs do T414 até 2,5 MFLOPs do T800, não se dispondo de dados concretos para o T212 [HOMEWOOD-87].

2.2. A Linguagem OCCAM

2.2.1. Fundamentos de OCCAM

OCCAM é uma linguagem de programação paralela que permite que um sistema seja descrito como um conjunto de processos concorrentes, os quais se comunicam um com o outro através de canais [POUNTAIN-87]. Fica estabelecido com esta linguagem uma estreita relação entre os processadores Transputer e os sistemas de processamento paralelo. A arquitetura interna dos Transputers está feita de modo que estes possam executar programas em OCCAM de forma quase direta, sendo OCCAM uma espécie de linguagem "assembler" para estes. É prudente advertir que esta "linguagem assembler" traz consigo todas as características básicas de uma linguagem de alto nível, como por exemplo C ou Pascal.

Se uma das funções de um computador é modelar e manipular processos que ocorrem no mundo real, deve-se ter em conta que os processos reais trazem características marcantemente concorrentes. Considerando nosso Universo de três dimensões espaciais e uma temporal podemos afirmar que existem eventos que ocorrem após outros (eventos sequenciais). Por outro lado existem também os eventos que ocorrem em lugares distintos mas ao mesmo tempo (eventos concorrentes).

Estas reflexões nos induzem a pensar que uma linguagem que se proponha a dar suporte à manipulação de um modelo de processos reais, deve permitir que se mesclem processos concorrentes e sequenciais. OCCAM é a primeira linguagem de programação baseada em conceitos de paralelismo em adição à execução sequencial e que além disto provoca sincronização automática da comunicação entre

processos concorrentes (construção decorrente da implementação interna da linguagem [POUNTAIN-87]). A comunicação entre processos é feita com a troca de informações por meio de canais definidos, diferentemente do conceito de passagem de informações por meio de variáveis, como na execução de procedimentos ou subrotinas de uma linguagem convencional (ainda que este mesmo processo está disponível também em OCCAM) [MATTOS-87].

Um canal pode passar valores entre dois processos sendo executados no mesmo processador ou entre processos sendo executados em processadores distintos. No primeiro caso o canal seria de fato uma posição de memória e no segundo caso um "link" de "hardware", um "Transputer link" ou uma linha serial de comunicação. Um canal OCCAM descreve uma comunicação abstrata independente da comunicação física. Um programa aplicativo que envolva troca de informações por canais pode ser desenvolvido em um único processador, testado, e depois de aprovado,pode-se distribuir os vários processos entre vários processadores para materializar a concorrência [POUNTAIN-89-2] [POUNTAIN-90]. Para isto basta introduzir algumas declarações no conteúdo do programa. Se um processo de entrada "descobre" que o dado a ser lido ainda não está disponível ele "espera" pelo fornecimento do dado sem nenhuma necessidade de haver alguma instrução explícita a este respeito.

Igualmente um canal de saída não enviará nada até que a entrada que a ele está conectada não esteja pronta para receber.

Os programas OCCAM são construídos a partir de três processos primitivos: atribuição, entrada por um canal e saída por um canal. Tais processos primitivos são combinados, para constituir

processos de mais alto nível, por meio de três construções: sequencial, paralela e condicional (ou alternativa). Será feita uma descrição dos processos primitivos, das construções básicas disponíveis e de aspectos relevantes ao entendimento da linguagem OCCAM.

2.2.1.1. Processos Primitivos

Todos os programas em OCCAM são constituídos pela combinação de três processos primitivos, quais sejam:

- **Processo de Atribuição** (" assign ") - um processo de atribuição altera o valor de uma variável exatamente como ocorre em linguagens convencionais. O símbolo utilizado para atribuição é ":=" (o símbolo "=" é usado para testes de igualdade) e assim:

contador := 4

faz com que o valor da variável "contador" assuma o valor "4".

- **Processo de Entrada** - este processo associa um parâmetro recebido por um canal a uma variável. O símbolo utilizado para entrada é "?" e um exemplo seria:

canal.A ? contador

que toma o valor recebido pelo canal "canal.A" e atribue este valor à variável "contador".

- **Processo de Saída** - este processo transmite o valor de que transmite o valor associado à variável "contador" pelo "canal.A".

Seria conveniente citar que a denominação de variáveis e canais em OCCAM tem um formato livre, devendo-se apenas evitar começar um nome por alguns caracteres especiais e também não definir nomes iguais a algumas palavras reservadas [POUNTAIN-87][INMOS-88-2].

Os nomes podem ser tão longos quanto se queirá começando por uma letra e depois sendo compostos por letras, algarismos ou o caractere "." . Letras maiúsculas e minúsculas são distinguidas por OCCAM de modo que "canal.A" e "canal.a" são canais distintos.

2.2.1.2. Construções Primitivas

Muitos processos primitivos de OCCAM podem ser combinados para constituir processos maiores por meio de três construções primitivas: sequencial, paralela e condicional (ou alternativa).

- **Construção Sequencial** - a construção sequencial é a mais simples das construções literalmente quer dizer "execute os seguintes processos em sequência um após o outro". A palavra reservada para esta construção é "SEQ":

```
SEQ
  canal.A ? contador
  contador := contador + 1
  canal.A ! mes
```

É conveniente salientar que os processos que vão ser executados sobre um SEQ são identificados por dois caracteres à direita. E assim também deve ser feito com todas as outras construções [POUNTAIN-87][INMOS-88-2].

- **Construção Paralela** - a construção paralela significa que todos os processos subsequentes a ela devem ser executados ao mesmo tempo, isto é em paralelo. O símbolo utilizado para isto é "PAR". A combinação entre construções também é fundamental para a constituição de processos mais complexos.

```
PAR
  SEQ
    canal.A ? contador
```

```
                contador := contador - 1

            SEQ

                canal.B ! mês

                mes := mês + 1
```

- **Construção Condicional (ou alternativa)** - todas as linguagens de programação fornecem um meio de escolher entre executar diferentes processos de acordo com determinadas condições, isto é, de acordo com os resultados de algum teste sobre o valor de variáveis.

Para isto OCCAM dispõe da construção "IF".

```
        IF

            contador = 0

                SEQ

                    contador := contador + mês


                    canal.A ! contador

            contador = mês

                SEQ

                    contador := contador + 1

                    canal.A ? mês

            TRUE

                SKIP
```

Em OCCAM a escolha sobre executar diferentes processos possui uma dimensão extra com respeito às linguagens de programação convencionais. Está apresentado acima como fazer escolhas de acordo com valores de variáveis de um programa

utilizando-se a construção "IF". Mas OCCAM permite que sejam feitas escolhas baseadas em estados de canais, ou condições em que se encontram os canais no momento de optar por uma "alternativa". Para tal existe a construção "ALTERNATIVE" e seu símbolo representativo é "ALT". É conveniente observar que "..." significa um "fold".

```
CHAN OF INT canal.A, canal.B, canal.C, :
INT x :
ALT
  canal.A ? x
    ... processo A
  canal.B ? x
    ... processo B
  canal.C ? x
    ... processo C
```

Ao passar por esta construção "ALT", se ocorre que "canal.B" é o primeiro a produzir uma entrada, então apenas "processo B" será executado. Aqui a escolha se enquanto não há uma entrada o sistema permanece aguardando. A construção "IF" e a construção "ALT" diferem apenas no objeto utilizado para a realização da escolha do processo a ser executado.

2.2.1.3. Outras Caracterlsticas

As palavras reservadas da linguagem são sempre maiúsculas como por exemplo SEQ, PAR, CHAN.Canais são todos do tipo:

```
CHAN OF <protocol>
```

É necessário especificar o tipo de dado e a estrutura dos valores que o canal transmitirá e a isto se denomina protocolo do canal [POUNTAIN-87][INMOS-88-2]. Um canal que transmitirá um único valor inteiro por vez deve ser definido como:

```
CHAN OF INT canal.A :
```

OCCAM não dispõe até o momento de qualquer tipo de dado "character" ou "string" para representar caracteres alfabéticos ou palavras. Em vez disto caracteres são representados como números do tipo "BYTE" e cadeias de caracteres ("strings") como sequências de números do tipo "BYTE". Os tipos de dados são:

INT	--	inteiro
INTn	--	inteiro de "n" bits, quando não se especifica o compilador assume o "default" de tamanho de palavra do Transputer utilizado (ex: INT64, INT32 ou INT16).
REALn	--	real (vale observação anterior para "n" exceto que não há REAL16)
BOOL	--	variável Booleana (variável que pode assumir somente os valores "TRUE" ou "FALSE")
BYTE	--	byte (inteiro de 0 a 255)

Para produzir "valores verdade" como resultado de testes de comparação entre variáveis estão disponíveis os seguintes testes:

=	--	igual a
<>	--	não igual a
>	--	maior que
<	--	menor que
>=	--	maior que ou igual a

<= -- menor que ou igual a

Os "valores verdade" disponíveis são "TRUE" e "FALSE".

Podem ser associados valores constantes à variáveis por meio da seguinte construção:

```
VAL INT mês IS 30:

VAL INT semana IS 7:
```

Em OCCAM variáveis e canais são locais aos processos que seguem imediatamente a suas especificações [POUNTAIN-87].

Todas as linguagens de programação fornecem meios para se realizar laços ("loops") de trechos de sequências de instruções, ou seja, perfazer uma determinada ação repetidamente. Em geral é conveniente dividir em dois tipos de repetição repetir um número especificado de vezes ou repetir enquanto se respeite uma determinada condição. OCCAM dispõe de recursos para a realização dos dois tipos de repetição. O primeiro é implementado pela construção "WHILE", a qual inclue teste sobre os valores de uma ou mais variáveis. O processo resultante é executado enquanto seja verdadeiro o teste de continuidade. Por exemplo:

```
INT a :

SEQ

  a := 1

  WHILE a <= 10

    SEQ

      canal.A ! a

      a := a + 1
```

Cada vez que o processo contido na construção "SEQ" terminar ocorrerá em seguida sua repetição enquanto a condição estabelecida seja verdadeira.

O outro tipo de repetição é conseguido por meio da utilização da construção "SEQ" estabelecendo um número de repetições de acordo com o valor de uma variável ou a declaração explícita do valor. Segue um exemplo para cada um dos casos:

```
INT n, a :

SEQ i=0 FOR n

  a := a + i
```

ou,

```
INT a :

SEQ i=0 FOR 10

  a := a + i
```

Note-se que não há necessidade de se declarar a variável "i" pois o fato de aparecer na construção de repetição já equivale a uma declaração para o compilador.

Estas construções podem realizar testes não apenas sobre o valor de variáveis mas também sobre o resultado de operações aritméticas. As operações aritméticas básicas disponíveis em OCCAM são:

```
x + y     --  somar "y" a "x"

x - y     --  subtrair "y" de "x"

x * y     --  multiplicar "x" por "y"

x / y     --  quociente da divisão de "x" por "y"

x REM y   --  resto da divisão de "x" por "y"
```

Estas operações podem ser realizadas entre variáveis com o mesmo tipo de dado sejam elas "INT" ou "REAL" mas não se pode

realizar entre tipos distintos de dado pois o compilador não permitirá. Quando for estritamente necessário para o andamento do programa deve-se proceder a utilização de variáveis intermediárias onde se faz uma alteração do tipo de dado. Estas converções de tipo de dados são permitidas e deve-se especificar se vai haver um arredondamento do valor ou se será truncado o valor para um tamanho de palavra menor que o anterior. Assim podemos considerar os seguintes exemplos, onde "n" e "m" são inteiros do tipo "INT64", "n" tem um valor "255" e "m" um valor "3":

```
BYTE n                    --  produz um byte de valor 255

REAL32 ROUND n            --  produz um REAL32 de valor 255,0

REAL64 TRUNC n            --  produz um REAL64 de valor 255,0

REAL64 ROUND (n * m)  --  produz um REAL64 de valor 765,0

(REAL64 ROUND n)*(REAL64 ROUND)-- REAL64 de valor 765,0
```

Para o caso de que seja necessário proceder a operações Booleanas ao utilizar variáveis Booleanas nos testes de construções de repetição também estão disponíveis três destas operações. Assim tem-se:

```
NOT ativo       --  assume o valor da negação do contedo
                    Booleano da variável "ativo"

AND             --  realiza o "e" lógico entre duas variáveis
                    Booleanas

OR              --  realiza o "ou" lógico entre duas
                    variáveis Booleanas
```

Para permitir operações de mais baixo nível OCCAM dispõe de operadores para alterar os bits que representam um determinado inteiro, ou seja:

```
~ n          -- realiza uma negação em todos os bits da
                variável inteira "n"

/\           -- realiza um "e" lógico, bit a bit,entre
                os valores de duas variáveis inteiras

\/           -- realiza um "ou" lógico, bit a bit,entre
                os valores de duas variáveis inteiras

><           -- realiza um "ou exclusivo" lógico, bit a
                bit, entre os valores de duas variáveis
                inteiras

<<           -- realiza um deslocamento a esquerda de um
                bit na representação em bits de um
                inteiro

>>           -- realiza um deslocamento a direita de um
                bit na representação em bits de um
                inteiro
```

Um procedimento em OCCAM é um processo com um nome e este nome é usado para a chamada deste procedimento dentro de outros processos. Para definir um procedimento utiliza-se a palavra reservada "PROC" seguida do nome debatismo do procedimento, dos parâmetros utilizados pelo procedimento entre parêntesis e do processo que implementa o chamado "corpo do procedimento". Uma definição de um procedimento seria a seguinte:

```
PROC Atraso (VAL INT atraso)
  INT n :
  SEQ
    n := atraso
    WHILE n < 0
      n := n - 1
:
```

Uma função dá nome a um tipo especial de processo oqual retorna um resultado, chamado valor do processo. As definições de funções tem a seguinte forma geral:

```
<tipo> FUNCTION <nome> (<parâmetros formais>)

   <declaração>

VALOF

   <processo>

RESULT <expressão>
```

Onde:

<tipo> - diz respeito ao tipo de dado que a função irá retornar

(ex: "INT" ou "REAL32").

<nome> - nome de chamada dentro do programa principal

<parâmetros formais - parâmetros que serão passados à função

<declaração> - declaração de variáveis internas utilizadas.

<processo> - corpo de ações da função.

<expressão> - expressão que fornece o resultado de saída da função

Um exemplo de uma função que retorna o maior valor dentre duas variáveis inteiras poderia ser:

```
INT FUNCTION maximo (VAL INT n, m)

   INT resposta :

   VALOF

     SEQ
```

```
        IF
          n > m
            resposta := n
          m > n
            resposta := m
          n = m
            resposta := n
        RESULT resposta
    :
```

Uma outra característica interessante de OCCAM é a possibilidade de se realizar uma declaração de variáveis chamada "abreviada" que pode trazer a um escopo local uma variável sem o ônus de ocupar mais espaço de memória para sua declaração. A declaração abreviada não duplica o espaço necessário, mas possibilita a utilização do espaço declarado em um contexto mais externo. Segue-se um exemplo:

```
[1000] INT vetor :
REAL32 q, r, t :
PROC ColocaZero ()
  [1000] INT vetor.interno IS vetor :
  SEQ i=0 FOR 1000
    vetor.interno[i] := i
:
SEQ
  ColocaZero ()
  q := REAL32 ROUND(1)
```

```
  r := 3.4
  SEQ i=0 FOR 1000
    vetor[i] := i
```

No exemplo acima o vetor "vetor" ocupa 4000 bytes (cada inteiro é representado por 4 bytes) e sua utilização é possível no escopo do procedimento "ColocaZero ()" através da definição abreviada de "vetor.interno" sem ter ocupar mais 4000 bytes.

Uma flexibilidade a mais quanto, a redefinição de dados, está disponível em OCCAM utilizando-se o comando "RETYPES", que possibilita uma mudança de tipos de dados desde que a variável redefinida seja representada pelo mesmo número de bytes internamente. Como exemplo:

```
[768][1024] BYTE matriz.tela :
INT n :
PROC LimpaTela ()
  [768*1024] vetor.tela IS matriz
   SEQ i=0 FOR [768*1024]
     vetor.tela[i] := BYTE(0)
:
SEQ
  LimpaTela ()
  n := 1
  SEQ i=512 FOR 100
    SEQ y=384 FOR 100
      matriz.tela[i][j] := BYTE(1)
```

No exemplo acima a "matriz.tela" do tipo matriz foi substituido por "vetor.tela" do tipo vetor.

2.2.2. O Sistema de Desenvolvimento de Programas em OCCAM (tds2)

Será apresentado em sequência o sistema de desenvolvimento utilizado para a implementação de programas em linguagem OCCAM.

2.2.2.1. Preliminares

O sistema de desenvolvimento tds2 ("Transputer Development System 2") é um sistema de INMOS para desenvolver programas em linguagem OCCAM. O tds2 consiste em uma placa (ver Apêndice A) que pode ser inserida em um dos "slots" de um PC e um "software" que é executado nesta placa. A combinação destes dois itens compõe um ambiente de desenvolvimento onde programas podem ser desenvolvidos, compilados e executados.

Programas a serem executados em uma rede de Transputers também podem ser desenvolvidos com este sistema.

A principal interface do sistema é um editor, tão logo o sistema inicia o usuário é inserido em um ambiente de edição onde a edição, compilação e execução de programas é possível ainda dentro deste ambiente.

2.2.2.2. Diretórios e Arquivos

Os programas componentes do tds2 são fornecidos como arquivos comprimidos e como parte da instalação um diretório \ARCD700D é criado, e é onde vão estar estes arquivos comprimidos. Os diretórios necessários são criados e os arquivos correspondentes são descomprimidos e colocados nestes, também como parte da instalação. O diretório principal criado é chamado de \TDS2 e os arquivos armazenados em subdiretórios, quais sejam:

\TDS2\SYSTEM - arquivos de sistema e utilitários

\TDS2\COMPLIBS - bibliotecas

\TDS2\TOOLS - ferramentas fornecidas em conjunto como sistema (por ex: "debugger")

\TDS2\MATHLIBS - funçoes matemáticas

\TDS2\TUTOR - "tutorial" para o usuário

\TDS2\EXAMPLE - programas de exemplo

\TDS2\SERVER - servidor de arquivos

Uma das características mais importantes do tds2 é sua estruturação de edição de arquivos em "folds" (ou em "dobras"). Estabelecendo uma analogia com uma longa folha de papel com muitas informações escritas podemos "esconder" parte da informação dobrando esta folha e deixando uma indicação de qual informação está "escondida" por meio de um título no início da dobra.

Assim, por meio de dobras, pode-se deixar parte da informação escrita (ou toda) fora do alcance visual direto. Para colocar esta informação a vista basta que se "abra" esta dobra. Extendendo este conceito pode-se ter dobras dentro de dobras extabelecendo-se uma estrutura hierárquica confortável para o desenvolvimento de programas. Num primeiro nível teríamos a vista uma série de títulos que conduziriam às dobras que poderiam conter por exemplo bibliotecas utilizadas, definição de variáveis, definição de constantes, definição de procedimentos, programa principal, etc. Continuando com este exemplo, ao abrir a dobra de definição de variáveis poderíamos ter outros títulos que conduziriam a outras dobras como variáveis inteiras, variáveis reais, variáveis lógicas, etc. Abrindo a dobra de variáveis inteiras teríamos então declarações em linguagem OCCAM que o compilador traduziria como definição de variáveis inteiras. Com

isto sempre podemos ter no nível de visualização do usuário apenas informação relevante ao entendimento deste nível. Desta forma os programas desenvolvidos guardam uma estrutura hierárquica intrínseca à sua edição, facilitando sua implementação, entendimento e depuração.

2.2.2.3. O Ambiente de Edição

O ambiente de edição fornece a infraestrutura adequada à criação de programas por meio de "folds" (ou dobras). Há uma redefinição de algumas teclas do PC para possibilitar o advento das novas funções que existem neste ambiente [INMOS - 88]. Sempre se pode sanar dúvidas dentro do editor por meio da tecla "HELP" que é uma das teclas de função definidas expecialmente para o uso do tds2. Informações mais detalhadas sempre podem ser buscadas na referência [INMOS - 88].

2.2.2.4. Compilação e "link" de Programas OCCAM

Serão apresentados em seguida os informações necessárias à compilaçào de programas utilizando-se o sistema de desenvolvimento de OCCAM.

2.2.2.4.1. Fundamentos

Há basicamente três modos de operação dos programas compilados obtidos, existindo para cada um deles uma sequência de procedimentos adequada:

- programas a serem executados no tds2
- programas a serem executados em uma rede de Transputers

- programas "standalone"

Os programas "standalone" são feitos para serem executados desde o DOS do PC.

Para realizar a compilação dos tipos de programas especificados acima deve-se conhecer uma série de ferramentas e procedimentos de preparação que serão descritos nos ítens seguintes.

2.2.2.4.2. Os Utilitários do Compilador

O tds2 não é apenas um editor mas um ambiente de desenvolvimento completo onde programas podem ser compilados, ligados e executados sem sair deste. Para fazer isto um conjunto de utilitários de compilação deve ser carregado no sistema. O conjunto de utilitários de compilação consiste em:

CHECK	- realiza uma checagem da sintaxe de um programa OCCAM
COMPILE	- compila um programa OCCAM
EXTRACT	- extrai a configuração dos programas e liga código ("link")
LOAD NETWORk	- carrega um programa compilado em uma rede de Transputers
RECOMPILE	- recompila um programa com parâmetros antigos
COMPILATION INFO	- mostra na tela do PC informação sobre um programa compilado
MAKE FOLDSET	- faz com que um "fold" seja adequado à compilação
SEARCH	- busca um conjunto de caracteres

REPLACE - substitue um conjunto de caracteres por outro

MAKE COMMENT - faz um "fold" de comentário

2.2.2.4.3. Preparação de um Programa Para Compilação

A simples edição do arquivo do programa a ser desenvolvido é condição necessária mas não suficiente para poder ser compilado, e assim que algumas medidas devem ser tomadas.

2.2.2.4.3.1. Criação de um "Fold" de Compilação

Antes de compilar um programa em OCCAM duas condições tem que ser encontradas, isto é, o "fold" que contêm o programa fonte deve estar em um arquivo e além disto deve estar envolto por um "compilation fold", ao qual o compilador será aplicado. O tipo de "compilation fold" indica qual tipo de unidade de compilação o "fold" contêm. Há cinco tipos destes "folds":

EXE - um programa executável desenvolvido para ser executado no tds2

UTIL - um programa para ser executado como um conjunto de utilitários do tds2

PROGRAM - um programa feito para ser executado em uma rede de Transputers

SC - uma unidade de compilação separada que não é um programa completo ("Separate Compilation Unit")

LIB - uma unidade de compilação de biblioteca

2.2.2.4.3.1. Criação de um "Fold" de Comentário

Quando se desenvolve um programa normalmente é necessário que se disponha do recurso de fazer comentários a parte do programa de modo que facilite a documentação e depuração deste, e sejam estes ignorados pelo compilador. Por vezes também é interessante que se possa transformar partes do código em comentários para que sejam utilizados mais tarde ou sejam mantidos como um exemplo anterior utilizado. Isto pode ser feito no tds2 colocando o texto do programa em um "fold" e aplicando sobre este o utilitário "MAKE COMMENT". Isto produz um novo "fold" que engloba o original tendo como cabeçalho o do original precedido por "COMMENT". O conteúdo deste "fold" será ignorado pelos outros utilitários de compilação exceto pelos de busca e substituição de conjuntos de caracteres.

2.2.2.5. Executando Programas OCCAM com tds2

Como já foi mostrado um programa compilado deve estar contido em um "compilation fold" do tipo "EXE". Uma vez que este programa tenha sido compilado e ligado o próprio compilador inclue neste "compilation fold" um outro "fold" denominado "CODE EXE", que é onde estará o código binário disponível para ser carregado e executado pelo sistema. Este "fold" pode ser removido daí e usado diretamente para ser executado.

Para carregar em memória o programa a ser executado deve-se colocar o cursor sobre o "EXE fold" (que já deve ter sido compilado e ligado) ou sobre o "CODE EXE" (que pode ter sido removido de um "compilation fold") e pressionar a tecla correspondente a ação de carregar programas, que é "GET CODE" [INMOS - 88]. Uma vez que o código tenha sido carregado tal

programa pode ser executado pressionando a tecla "RUN EXE" [INMOS --88].

Este programa permanece em memória até que se limpe a memória por meio dos recursos correspondentes disponíveis no tds2 [INMOS - 88]. Ao terminar a execução do programa o controle retorna automaticamente para o tds2.

2.2.2.6. Configuração e Carga de Programas em Redes de Transputers

Para fazer um uso efetivo das redes de Transputers uma aplicação deve ser transformada em um determinado número de processos paralelos. Uma vez feito isto, para que se possa atingir a "performance" esperada é necessário que se faça uma adaptação da topologia da rede que se dispõe aos processos que devem ser executados. Para que o tds2 realize isto de forma automática o programador deve adicionar aos seus programas informações que descrevam a topologia dos "links" e quais os fragmentos de código associados aos Transputers individuais. Este procedimento é denominado de configuração.

A carga de um programa previamente configurado é feita por meio do utilitário "LOAD NETWORK" acionado pela simples pressão de uma tecla no PC [INMOS - 88].

A alocação de código a processadores em uma rede de Transputers é alcançada usando-se duas extensões da linguagem OCCAM:

PLACED PAR

PROCESSOR <número do proces.> <tipo do Transputer>

Tal construção mapeia a distribuição de processos entre os "links" dos Transputers, permite ao configurador identificar o código destinado a um determinado Transputer e faz uma checagem se

a rede descrita pode ser carregada (cheque de consistência). O <número do processador> é um identificador lógico do processador e pode ter qualquer valor entre todos os inteiros positivos. A única restrição é que o processador imediatamente ligado à placa de desenvolvimento tenha o número "0" porque é condição imposta pelo funcionamento do sistema.

O tipo do Transputer define, como o nome diz, qual é o Transputer que será o processador deste nó. Esta informação é usado pelo configurador para verificar se o código que se quer carregar neste processador foi compilado para o "processador alvo" correto. Tipos válidos de processadores são: "T8" (IMS T800), "T4" (IMS T414) ou "T2" (IMS T212).

Concluindo, neste capítulo foram apresentados alguns conceitos sobre a arquitetura e modo de operação dos Transputers. Foram apresentados três constituintes da família destes processadores INMOS. Também discorreu-se sobre algumas características da linguagem de programação OCCAM. Foram apresentados os processos primitivos e as construções primitivas desta linguagem. Os principais comandos da linguagem foram mostrados, seguindo-se alguns exeplos de pequenos programas, para ilustração de seu uso. Foi feita uma introdução ao sistema de desenvolvimento de programas em linguagem OCCAM (tds2). Este sistema, possui características próprias que foram abordadas de uma forma introdutória. Maiores detalhes foram deixados para consultas posteriores à documentação apropriada, com referências na bibliografia.

Tais conhecimentos servirão de base para o desenvolvimento da interface gráfica, objeto deste trabalho.

3. DESENVOLVIMENTO E DESCRIÇÃO DA INTERFACE GRÁFICA

Neste capítulo, na primeira parte, faz-se uma descrição do suporte de "hardware" e a estruturação do "software", utilizados para o desenvolvimento da interface gráfica. Ainda na primeira parte segue-se uma discussão sobre as técnicas de otimização de "performance" implementadas e sobre os algorítmos abordados. Na segunda parte do capítulo são apresentadas todas as rotinas desenvolvidas para a composição da interface. Na terceira parte apresenta-se um caso prático de aplicação da interface, que é sua utilização em um roteador de placas de circuito impresso.

3.1. Infraestrutura, Técnicas e Algorítmos Utilizados

A seguir encontram-se descritos o suporte de "hardware" e a estruturação do "software", utilizados para o desenvolvimento da interface gráfica.

3.1.1. Suporte de "Hardware" do Sistema

Os processadores utilizados para implementar os sistemas de prova e desenvolvimento são Transputers de INMOS, com um tipo específico de processador para cada aplicação, ou seja, um processador T414 para a placa de desenvolvimento de programas OCCAM, um processador T800 para executar as rotinas gráficas e um processador T222 para carregar o "palette" e fazer o controle da temporização (além de fazer a transformação de dados de "pixels" digitais a analógicos) [INMOS-89]. Para melhor compreensão proceder-se-á a uma divisão deste ítem em partes que tratarão por separado do sistema global, do sistema de desenvolvimento de programas em OCCAM (de sua influência específica no sistema pois foi abordado do ponto de vista genérico com mais detalhes no ítem

foi abordado do ponto de vista genérico com mais detalhes no ítem 2.2.2.) e da placa gráfica (desta parte se tratará com mais detalhe no apêndice B).

3.1.1.1. Sistema Global

O sistema global está composto de um computador pessoal PC-AT compatível, em configuração padrão (512K RAM, 20 M bytes de disco rígido, unidade de disco flexível de 1.2 M bytes), uma placa de desenvolvimento IMS B004 de INMOS ("IMS B004 evaluation board") [GHEE-] [INMOS-85] [INMOS-88-3], uma placa gráfica de INMOS (onde há dois módulos Transputers, um IMS B408 e outro IMS B409), um monitor gráfico (policromático, marca Flexscan, modelo 9070S, cuja resolução pode variar entre 1024 "pixels" por 768 "pixels" de altura).

O computador PC tem a única função de oferecer as facilidades de entrada/saída por teclado/monitor e sua via interna para comunicação e envio/recepção de dados/endereços. Além disto oferece a fonte de alimentação como se fosse um sistema hospedeiro para as placas de INMOS. Neste sentido, daqui por diante quando falemos de placa de desenvolvimento, se entenderá como o conjunto placa IMS B004 mais teclado, monitor e via interna do PC. Quando falemos do sistema de desenvolvimento, se entenderá como o conjunto formado pela placa de desenvolvimento mais o "software" de desenvolvimento de programas em OCCAM (que é executado na placa de desenvolvimento). Para mais informação consultar o ítem 2.2.2 e o apêndice A. Quando falemos de "memoria pantalla" nos referiremos ao módulo IMS B408 (mais o programa e as rotinas gráficas que são executadas nele) e no caso de "control pantalla" ao módulo IMS B409 (mais o programa e as rotinas que são executadas nele).

A partir daqui se fará referência ao "software" de desenvolvimento de programas em OCCAM como tds2 ("Transputer Development System 2"), conforme já estabelecido no ítem 2.2.2

deste texto. Normalmente um "software" de desenvolvimento de programas está feito de modo que no mesmo processador onde éexecutado este "software", o farão posteriormente os programas de aplicação. Assim é como o realiza o tds2, mediante o qual se podem editar/compilar e ligar programas em OCCAM, já que é executado no processador T414 e se pode realizar a execução destes programas sobre o mesmo processador. Além disto o tds2 permite carregar módulos executáveis em outros processadores, por meio dos enlaces físicos dos Transputers. Então se pode fazer a compilação de módulos executáveis no tds2, escolhendo o tipo de processador como parâmetro da compilação (por exemplo, se temos um processador T414, ou T222, ou T800), e depois carregá-lo onde se queira.

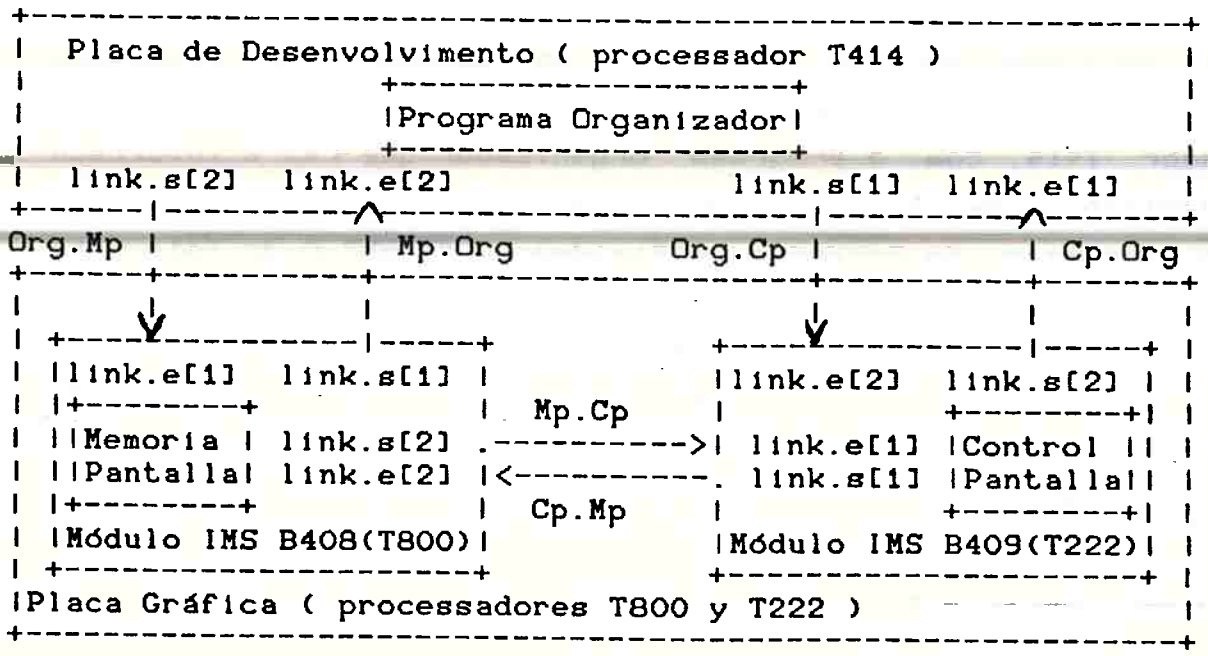


Figura 3.1 - Sistema global da interface

Com isto poderemos desenvolver as rotinas gráficas para carregá-las no módulo IMS B408 (processador T800, na placa gráfica), desenvolver as rotinas de carga de "palette" e controle de temporização do monitor para carregá-las no módulo IMS B409 (processador T222, na placa gráfica) e por último desenvolver o

sistema organizador para executar no processador T414 da placa de desenvolvimento.

Em resumo, para tentar aclarar as idéias, haverá duas placas: uma é a do sistema de desenvolvimento e a outra é a placa gráfica. Na placa gráfica há dois módulos processadores: o IMS B408 (processador T800) e o IMS B409 (processador T222). Todos os processadores têm canais de comunicação definidos entre eles (como se pode ver na Figura 3.1).

3.1.1.2. Placa de Desenvolvimento OCCAM

A placa de desenvolvimento de OCCAM [INMOS-85] está descrita no apêndice B e o sistema de desenvolvimento de OCCAM foi exposto com mais detalhes no ítem 2.2.2. . Um aspecto que interessa enfatizar neste ponto é que na placa de desenvolvimento ter-se-á o processador T414, com o programa organizador que faz a interface com o usuário, possibilitando que ele possa acessar às rotinas gráficas por meio de comandos definidos. O "link" 0 do T414 está conectado à via interna do PC para atividades como reiniciar sistema ("reset"). O "link" 2 está conectado ao módulo IMS B408, constituindo dois canais de comunicação definidos como Org.Mp (envia dados/ordens desde o organizador até a "memória pantalla") e Mp.Org (recebe dados da "memoria pantalla"). O "link" 1 está conectado ao módulo IMS B409, constituindo outros dois canais de comunicação definidos como Org.Cp (envia dados/comandos ao "control pantalla") e Cp.Org (recebe dados de "control pantalla").

3.1.1.3. Placa Gráfica

A placa gráfica (ver apêndice B) tem internamente definidos dois canais de comunicação que são o Mp.Cp (que envia dados/comandos desde a "memoria pantalla" até o "control pantalla"), e o canal Cp.Mp (que envia dados desde o "control

pantalla" até a "memoria pantalla"). Lembrar que o módulo "memoria pantalla" é o que contêm as rotinas que modificam a memória de tela, e no módulo "control pantalla" estão as rotinas de controle da programação dos parâmetros de temporização de vídeo e da carga do "palette". Além disto, existem canais de comunicação entre os módulos e o organizador, já descritos no ítem 3.1.1.2.

3.1.2. Estruturação do Software da Interface Gráfica

A estrutura do "software" do sistema de prova/desenvolvimento está feita visando a facilitar a obtenção de uma interface gráfica de aplicação genérica.

Assim que, além de apresentar esta estrutura neste capítulo, no capítulo 4. teremos um resumo de um caso prático de aplicação.

3.1.2.1. Estruturação dos Programas

A estruturação dos programas desenvolvidos está feita aproveitando a hierarquia que impõe o sistema de "folds" para armazenar arquivos no tds2. Esta estrutura permite fazer modificações independentes em todos os módulos, seja do programa organizador, "memoria pantalla" ou "control pantalla", e ainda, especificar o tipo de "fold" que se está utilizando. Como os processos "memoria pantalla" e "control pantalla" fazem parte de uma rede de transputers na placa gráfica eles estarão cada um em um "SC fold" e os dois em um "PROGRAM fold" de nível mais alto (ver ítem 2.2.2). O programa organizador que vai ser executado na placa de desenvolvimento estará contido em um "SC fold" que por sua vez estará em um "EXE fold" de mais alto nível. Os "folds" "PROGRAM" e "EXE" estarão no mesmo nível na hierarquia de edição do tds2. Assim que se terá a seguinte estruturação:

```
...PROGRAM        ...SC          ...Declaração de Bibliotecas
                  ("memoria pantalla",
                  IMS B408, processo0,  ...Declaração de Constantes
                  Transputer T800)
                                        ...Declaração de Variáveis
                                        ...Endereços de Variáveis
                                        ...Procedimentos
                                        ...Programa Principal
                        ...SC           ...Declaração de Variáveis
                  ("control pantalla",  ...Endereços de Variáveis
                  IMS B409, processo1,  ...Procedimentos
                  Transputer T222)      ...Programa Principal

...EXE            ...SC          ...Declaração de Bibliotecas
                  (organizador, TDS2,
                  Transputer T414)      ...Declaração de Constantes
                                        ...Declaração de Variáveis
                                        ...Endereços de Variáveis
                                        ...Procedimentos
                                        ...Programa Principal
```

3.1.2.2. Protocolo Para Acesso às Rotinas

Para acessar às rotinas deve-se obedecer a um protocolo específico que envia/recebe quatro variáveis pelos canais de comunicação. Estas variáveis são:

"origen", "destino", "tipo" e "longitud". As variáveis "origen" e "destino" identificam quem originou a chamada e onde vão estar as rotinas chamadas, respectivamente. A variável "tipo" especifica qual será a rotina chamada. Algumas rotinas não necessitam de parâmetros para sua execução enquanto que outras podem necessitar. Para que saibamos se há que enviar/receber parâmetros ou não temos a variável "longitud". Se o valor da variável "longitud" é zero não há que enviar parâmetros, mas se é distinto de zero seu valor especifica o número de parâmetros que se deve enviar. Assim que, desde o ponto de vista do programa organizador, se envia o cabeçalho que contem estas variáveis, os parâmetros (se há algum) e se espera receber uma variável de verificação de execução:

```
Org.Mp ! origen; destino; tipo; longitud

Org.Mp ! [ parametros FROM 0 FOR longitud ]

VerificacionEjecucion ()
```

Desde o ponto de vista do módulo "memoria pantalla" se recebe o cabeçalho, os parâmetros (se há, e para isto se faz uma verificação do valor de "longitud") e se envia uma variável de verificação de execução:

```
Org.Mp ? origen; destino; tipo; longitud

      IF

        longitud > 0
```

```
        Org.Mp ? [ parametros FROM 0 FOR longitud ]
    TRUE
      SKIP


VerificacionEjecucion ()
```

OBS: Ainda que as duas rotinas VerificaEjecucion() apresentadas tenham o mesmo nome elas são distintas (ver 3.2.1.4. e 3.2.3.5., respectivamente).

3.1.3. Técnicas de Otimização de "Performance"

As técnicas de otimização de "performance" poderiam ser divididas para análise em técnicas para um Transputer único e técnicas para uma topologia multiTransputer. Deste modo procede-se às discussões sobre ambos os tipos.

3.1.3.1 Otimização de "Performance" Para um único Transputer

Para extrair a máxima "performance" de um Transputer é vital o melhor uso possível de sua memória "on-chip" onde se sabe que os ciclos de memória são da ordem de 66 nseg. Para a memória externa estima-se um valor da ordem de 330 nseg [INMOS-86] [INMOS-86-2]. Esta diferença de tempos pode ser decisiva se um programa concentra-se em acessos à memória externa degradando a "performance" do sistema.

A utilização da memória "on-chip" é mais proveitosa para área de dados do que para área de instruções. O T414 por exemplo, realiza busca de instruções por meio de palavras de 32 bits, e cada ciclo de busca de código irá trazer 4 instruções. Assim, como o acesso a código ocorre com menos frequência que o acesso a dados, é conveniente reservar para a memória de dados aquela mais rápida. O compilador e o programa carregador do sistema de

desenvolvimento de OCCAM tiram vantagem deste desbalanceamento tentando colocar o maior número possível de variáveis de dados na memória "on-chip" [INMOS-88]. Uma região de endereçamento menos significativo da memória é deixada como espaço de sistema (para utilização pelos "links" de comunicação), a parte imediatamente seguinte é ocupada pelos dados e o restante pelo programa. Isto é possível pelo sistema de alocação estática dos dados em OCCAM já que depois da compilação o carregador do sistema de desenvolvimento sabe exatamente qual é o espaço de dados necessário ao programa. Diferentemente de linguagens como C, a linguagem OCCAM não dispõe de alocação dinâmica de memória e de recursividade. Caso algum programa necessite de espaço de dados superior a 2K bytes (tamanho da memória interna do T800, por exemplo) então alguns dados serão colocados na memória externa. Será importante então a intervenção do programador no sentido de encontrar a melhor posição dentro do programa para a definição das variáveis de dados. No desenvolvimento desta interface sempre houve a preocupação em que estes critérios fossem levados em conta. A aplicação destes critérios gerava por outro lado um conflito entre a melhor posição para as variáveis e um desenvolvimento de programas seguindo normas que conduzam a programas corretos e de fácil leitura ou entendimento por terceiros. Estas práticas, também conhecidas como estilos de programação e disponíveis na maioria dos livros de engenharia de programação, aconselham a que se tenha o menor número possível de variáveis globais no sistema para que o programador não perca o controle sobre atualização destas em momentos indesejáveis. Com isto pode-se evitar erros ocasionados por compartilhamento destas variáveis globais. Estaria também facilitado o entendimento das rotinas do sistema, que conteriam todas as definições necessárias para sua assimilação e depuração. Tendo em vista a "performance", o bom senso nos levaria a definir todas as variáveis no nível mais

externo possível do contexto do programa. Deve haver então uma relação de compromisso entre o desenvolvimento de acordo com métodos formais consolidados para sistemas convencionais ou detalhes de implementação de software específicos para aumento da "performance" em arquiteturas a Transputers.

Nos Transputers as variáveis são acessadas relativamente a um registrador apontador de espaço de trabalho [INMOS-86], onde cada processo OCCAM tem seu próprio espaço de trabalho. Para maximizar "performance" é importante que variáveis que pertençam a um espaço de trabalho utilizado com mais frequência estejam na memória interna. O compilador OCCAM reserva o começo da memória de dados para a última variável declarada no programa, ou seja, para o espaço de trabalho do último processo declarado. Caso haja a chamada de um procedimento dentro do espaço de trabalho de um processo, o procedimento terá suas variáveis alocadas em espaços de menor endereço que o programa chamador, ou seja, haverá maior probabilidade de ter sua alocação na memória interna. Novamente vem a tona a discussão do parágrafo anterior pois chamar um procedimento com necessidade de um grande espaço de trabalho resulta que o processo chamador possa vir a ter suas variáveis inteiramente na memória externa.

Uma solução atenuante para o problema seria a utilização do recurso de declarar variáveis que ocupam muito espaço por meio de abreviações, "trazendo-as" de um contexto mais externo para o contexto local [INMOS-88-2]. Abaixo segue um exemplo:

```
BOOL ativo :

[2000] INT vetor.inteiros :
```

```
PROC EscreveSequencialnteiros ()
  [2000] INT vetor.interno IS vetor.inteiros :
  SEQ i=0 FOR 2000
    vetor.inteiros[i] := i
:
SEQ
  EscreveSequencialnteiros ()
  ativo := TRUE
  SEQ i=0 FOR 2000
    vetor.inteiros := 0
```

Com a adoção desta notação abreviada a definição do vetor "vetor.inteiros" foi aproveitada dentro da rotina "EscreveSequencialnteiros()" sem necessidade de alocar mais duas mil posições de memória. Tal técnica foi empregada na fase de desenvolvimento da interface sempre que fosse preciso trabalhar com matrizes ou vetores de modo a que não fosse necessário redefinir estruturas cada vez que uma rotina executasse uma atualização sobre esta. Com o crescimento em tamanho da interface, e a consequente utilização forçada da memória externa, foi inevitável uma inversão nesta tendência, passando-se a dar mais importância a aplicar um estilo de programação mais adequado em detrimento da presença dos dados na memória externa.

Sob certas circunstâncias a declaração de variáveis na forma abreviada pode acelerar consideràvelmente a manipulação de bytes. Se existe a necessidade de frequentemente analizar uma variável inteira do ponto de vista de seus bytes constituintes então pode ser mais rápido redefinir esta palavra extraindo os campos de bytes que a compõe. Assim teríamos como exemplo:

```
INT palavra :
[4] BYTE palavra.em.bytes RETYPES palavra :
BYTE byte0 IS palavra.em.bytes[0] :
BYTE byte1 IS palavra.em.bytes[1] :
BYTE byte2 IS palavra.em.bytes[2] :
BYTE byte3 IS palavra.em.bytes[3] :
SEQ
  ... uso de byte0, byte1, byte2, byte3
```

Em neste exemplo o acesso às abreviações é mais rápido que deslocar ou mascarar os bytes constituintes da variável inteira "palavra".

No caso onde se define a construção "PAR", para executar processos paralelos em um único Transputer, a alocação de dados é feita com o menor endereço do espaço de dados, para a última variável do último processo declarado no programa. A minimização do espaço de trabalho por declaração global de variáveis também é importante quando mais de um processo está sendo executado em paralelo. Por exemplo:

```
PAR
  [2000] BYTE lista :
  CopiaLista (lista)
  [2000] BYTE lista :
  TrataLista (lista)
  [2000] BYTE lista :
  EscreveLista (lista)
```

```
[2000] BYTE lista :

ApagaLista (lista)
```

Suponhamos que os procedimentos "CopiaLista()", "TrataLista()", "EscreveLista()" e "ApagaLista()" possuem variáveis internas, de acesso constante, que ocupam quinhentas posições de memória para cada procedimento. Seguindo esta linha de raciocínio apenas as variáveis do procedimento "ApagaLista()" e uma parte do último vetor "lista" definido estariam na memória interna. Além disto os quatro vetores "lista" estão definidos dentro do contexto "PAR" e contribuirão para preencher os espaços de trabalho dos processos individuais. De modo a contornar a esta situação deveríamos reescrever o programa da seguinte forma:

```
[2000] BYTE lista.A, lista.B, lista.C, lista.D :

PAR

  CopiaLista (lista.A)

  TrataLista (lista.B)

  EscreveLista (lista.C)

  ApagaLista (lista.D)
```

Assim conseguimos com que os procedimentos "CopiaLista()", "TrataLista()", "EscreveLista()" e "ApagaLista()" tenham todo seu espaço de trabalho alocado na memória interna.

Há uma instrução de movimento de blocos de bytes nos Transputers, que é diretamente suportada pela linguagem OCCAM [MAY-87][MAY-87-3]. Com ele pode-se realizar a transferência rápida de vetores de bytes cuidando-se apenas para que o vetor fonte e o vetor destino sejam disjuntos, evitando problemas de conflito. Assim:

```
[1024] BYTE linha1, linha2 :

VAL largura IS 1024 :

[linha1 FROM 0 FOR largura] := [linha2 FROM 0 FOR largura]
```

Com este recurso é possível construir um inicializador de vetores de byte muito rápido, conforme será visto no ítem 3.1.4, relativo a algorítmos.

3.1.3.2 Otimização de Performance Para um Sistema MultiTransputer

O tema de obter uma maximização de "performance" para um sistema multiTransputer oferece hoje em dia poucas técnicas abrangentes a sistemas genéricos, dado que ainda é uma área de pesquisa ativa atualmente, não existindo soluções consolidadas. A maioria das soluções que aparecem tendem a ser específicas para os vários problemas existentes, no entanto, pode-se citar algumas idéias aplicáveis a sistemas multiprocessadores a Transputers.

Os "links" dos Transputers são dispositivos capazes de transferir dados bidirecionalmente por meio de uso de DMA a uma taxa de 10M bits/seg [RYGOL-87][INMOS-87][INMOS-86]. É possível atingir estas taxas de transferência sem comprometer seriamente a "performance" do processador e, para obter o máximo aproveitamento possível, os "links" e o processador devem estar ocupados o maior tempo possível. Para evitar que os "links" esperem o processador ou vice-versa, as comunicações entre os "links" devem ser desacopladas da atividade computacional dos processadores. Por exemplo suponhamos um procedimento que aplica transformações a um conjunto de dados:

```
PROC Transforma (CHAN entrada, saida)
  INT dado :
  SEQ
    entrada ? dado
    AplicaTransformacao(dado)
    saida ! dado
:
```

Se os canais "entrada" e "saida" são "links" de um Transputer, então a "performance" de execução do procedimento estará seriamente afetada. A construção "SEQ" força o Transputer a executar os processos descritos em sequência, um após o outro. Ele, configurado deste modo, primeiro aceita uma entrada, depois aplica uma transformação ao dado e depois fornece a saída. Este problema pode ser contornado fazendo-se uma reestruturação no programa e passando-se as variáveis por referência, como abaixo:

```
INT dado1, dado2, dado3 :
... proc_entrada (dado)
... aplica.transformacao(dado)
... proc saida (dado)
SEQ
  Entrada(dado1)
  PAR
    Entrada(dado2)
    AplicaTransformacao(dado1)
  WHILE TRUE
    SEQ
```

```
PAR
  Entrada(dado3)
  AplicaTransformacao(dado2)
  Saida(dado1)
PAR
  Entrada(dado1)
  AplicaTransformacao(dado3)
  Saida(dado2)
PAR
  Entrada(dado2)
  AplicaTransformacao(dado1)
  Saida(dado3)
```

Assim ao invés de termos operações de transferência entre os processos pelos canais "entrada" e "saida", os processos transferem dados entre eles mesmos de um modo cíclico, sem haver espera e superposição de valores.

3.1.4. Concepção do Sistema Gráfico Paralelo

A computação gráfica tem sempre necessitado de grandes quantidades de potência computacional, quer seja com demanda dos usuários por uma maior resolução, ou mais cores e rapidez de resposta gráfica, ou ainda maior velocidade de processamento e aumento na largura de banda para operações de entrada e saída. Aplicações gráficas tipicamente consomem grandes quantidades de operações matemáticas em ponto flutuante para realizar transformações, interpolação de curvas e avaliação de valor de cores em sistemas complexos de cálculo de sombreamento. Imagens reais podem conter milhares de primitivas a serem manipuladas e mostradas na tela dos monitores de alta resolução.

Uma tarefa gráfica em tempo real demanda uma largura de banda muito alta para o dispositivo de saída de tela. Para realizar uma imagem de apenas 16 quadros por segundo em um monitor de 512x512 pixels de 8 bits de representação de cor, é necessário transferir 4M bytes de dados por segundo para a tela. Esta taxa pode ser atingida facilmente por meio de um "link" de um Transputer. Como as taxas de passagem de quadros e a resolução das telas crescem continuamente a "performance" necessária também deve crescer. Existem soluções de hardware baseadas no uso de múltiplos Transputers para atingir "performances" superiores para sistemas gráficos que as exigem [McCONNEL-88].

3.1.4.1. Características de um Processador Gráfico

Algumas das principais características de um processador gráfico ideal seriam: alta velocidade em operações de ponto flutuante, alta velocidade de manipulação de grandes quantidades de bytes e alta taxa de transferência de dados de/para o processador. Os processadores da família dos Transputers preenchem os requisitos citados e adicionam alto poder computacional decorrente de sua arquitetura RISC, um grande espaço linear de endereçamento de memória (permite quantidade ilimitada de RAM, em termos práticos, evitando acesso lento a periféricos de memória de massa) e suporte para uma linguagem que explora características de paralelismo (OCCAM). Faz-se necessário então escolher, ou desenvolver, uma série de algorítmos para implementar as primitivas gráficas necessárias, que melhor explorem os recursos de hardware disponíveis. Tais algorítmos não necessitam ser necessariamente específicos para sistemas paralelos desde que operem de modo a extrair o máximo de "performance" que o hardware disponível tem para oferecer, ainda que de forma sequencial.

3.1.4.2. Características de um Sistema Gráfico Paralelo

Para poder atingir a melhor "performance" possível, em um sistema paralelo, é necessário que a tarefa de processamento seja dividida em um determinado número de subtarefas. Normalmente tal número é igual ao número de processadores definido para atingir a "performance" necessária. O problema passa a ser então de distribuição de tarefas entre processadores. Existem alguns métodos para distribuir tarefas gráficas entre processadores [McCONNEL-88]:

- **Espacial** - a tela gráfica é quebrada em um número de janelas. A administração de cada janela é deixada para um processador ou um grupo de processadores.

- **Cronológica** - este método distribue a tela toda, a todos os processadores do sistema. Em outras palavras, todo processador gerencia a tela inteira, mas cada um o faz em um instante distinto no tempo. Isto significa que há uma sucessão de quadros componentes da imagem, que podem ser gerados em paralelo, mas que tem sua apresentação em instantes de tempo subsequentes. Cada processador, ou grupo de processadores, é responsável por um quadro da imagem. Portanto uma aplicação mais direta seria em animação gráfica.

- **Objetiva** - Tal método realiza uma distribuição, aos processadores, dos diferentes objetos a serem desenhados. Este é um método intrinsecamente difícil, se considerarmos os problemas que podem advir de objetos escondidos, sombreados ou na própria intersecção entre estes. Boa para tratamentos gráficos com número de processadores da ordem do número de tipos de objetos, e ainda, número de objetos de cada tipo da mesma ordem de grandeza.

- **Característica** - Este método distribue as características de uma imagem, como por exemplo as cores, aos diferentes processadores. Menos utilizada.

3.1.4.3 Algorítmos Implementados

No caso da interface desenvolvida uma análise simples conduz a escolha do método de distribuição de tarefas que se vislumbra para esta. O método da divisão cronológica encontra uma melhor aplicação em sistemas de animação gráfica, onde as imagens se sucedem com a intensão de criar a ilusão do movimento. Os métodos de características e objetos poderiam comprometer a generalidade de aplicações da interface. Assim o método escolhido para explorar o paralelismo, onde fosse necessário, foi o de divisão espacial. Para cada uma das partes, o conjunto de primitivas desenvolvidas e o núcleo de interpretação de comandos seria o mesmo, sendo necessário apenas alterar endereços de janelas de memória de tela acessíveis a cada módulo. Outros parâmetros tais como variáveis que representam os limites máximos de desenho, ou escala dos desenhos também seriam alterados de um módulo para outro, mas todas as alterações seriam com respeito a valores de variáveis e não a questões estruturais ou de operação interna da interface. Além disto haveria uma total independência entre a operação dos vários módulos. Para estabelecimento de um primeiro protótipo o que houve foi o desenvolvimento de um destes - módulos, autosuficiente para todas as operações básicas necessárias, sem perder de vista o contexto de paralelismo espacial. Isto significa que vários módulos podem ser postos a trabalhar em conjunto, sem ônus de desenvolvimento adicional.

A primeira preocupação foi encontrar um algorítmo para transferência rápida de grandes quantidades de bytes que utilizasse a potencialidade dos Transputers de transferência de blocos [INMOS-88]. A apresentação deste fica clara por meio de sua descrição em um trecho de programa OCCAM:

```
vetor[origem] := cor.inicial

destino := origem + 1
```

```
WHILE quanto >= (2*quanto.interno)
  SEQ
    [vetor FROM destino FOR quanto.interno] :=
                [vetor FROM origem FOR quanto.interno]
    destino := destino + quanto.interno
    quanto.interno := quanto.interno + quanto.interno
```

Com tal algorítmo o tamanho dos trechos de vetor que vão sendo atualizados cresce com uma progressão geométrica de razão 2. Isto gerava a atualização de 1, depois 2, depois 4, 8, 16 bytes e assim sucessivamente. Ocorre que, obviamente, nem todos os vetores que se quer atualizar são de tamanho que pertença a esta sequência geométrica, de modo que ao algorítmo foi adicionado um fator de correção que o torna genérico para qualquer tamanho. Para tal basta inserir ao final (após o "loop" mostrado anteriormente) as seguintes linhas de comando:

```
quanto.interno := quanto - quanto.interno
[vetor FROM destino FOR quanto.interno] :=
                [vetor FROM origem FOR quanto.interno]
```

Além disto, aproveitando-se desta característica, e uma vez que havia necessidade de redefinições abreviadas, a bem da alocação de memória, uma redefinição da matriz de memória de tela foi adotada, em termos de um vetor de memória de tela. Em outras palavras, a matriz de bytes da memória de tela foi redefinida localmente, nas rotinas que a atualizavam, como um vetor de bytes. Foi feito isto para melhor explorar os recursos de otimização de "performance" com base nas características dos algorítmos escolhidos. Com esta medida foi possível transformar todas as necessidades de desenhos de figuras quaisquer, em desenhos sucessivos de vetores de bytes, que eram obtidos de maneira muito rápida utilizando a técnica exposta acima. A própria memória de tela estava agora descrita como um grande vetor de bytes onde tais operações eram imediatas.

O problema de busca de algorítmos específicos para cada figura geométrica, que se quer dispor como primitiva gráfica, é substituido agora por obter a melhor maneira de se representar a decomposição de tais figuras em linhas (ou semiretas), utilizando-se algorítmos convencionais para extrair os pontos de seus contornos.

- Traçado de Retas

O desenho de linhas retas quaisquer deve ser realizado ponto a ponto, seguindo a equação de uma reta dada pela geometria analítica:

$$y = m * x + q$$

onde "q" é o valor de "y" para "x" igual a zero e "m" é o coeficiente angular da reta. Faz-se uma varredura horizontal dos valores de "x" associados a "pixels", sequencialmente e de um em um, calculando-se os valores de "y" correspondentes. Para uma melhor precisão, os cálculos para obtenção dos valores de "y" são feitos utilizando valores reais e depois convertidos (ou arredondados) para inteiros de modo a encontrar seu "pixel" associado. No caso de linhas horizontais, ou de inclinação até cerca de 60 graus com relação a horizontal, obtêm-se um conjunto de pontos inteiros contíguos (associados a "pixels"). Com inclinações mais próximas de 90 graus o arredondamento dos valores reais para inteiros dá como resultado pontos espaçados (Figura 3.2.a). Este efeito, na tela, corresponderia ao desenho de uma série de pontos alinhados mas espaçados. Uma simples correção foi introduzida por meio da variável "ayu.rep" (ver Apêndice C, procedimento "LinesCualquiersEntero ()"), que realiza um teste para comprovar se os pontos gerados não estão espaçados por mais de um pixel na direção vertical. Em caso positivo, o procedimento introduz pontos de correção, ilustrado na Figura 3.2.b, onde os pontos originados por arredondamento estão designados por pequenas circunferências (pontos "vazados") e os de correção por pequenos círculos (pontos "cheios").

No caso particular de que a reta é horizontal, pode-se desenhá-la muito rapidamente por meio do algorítmo de inicializar um vetor de "bytes", já mostrado anteriormente.

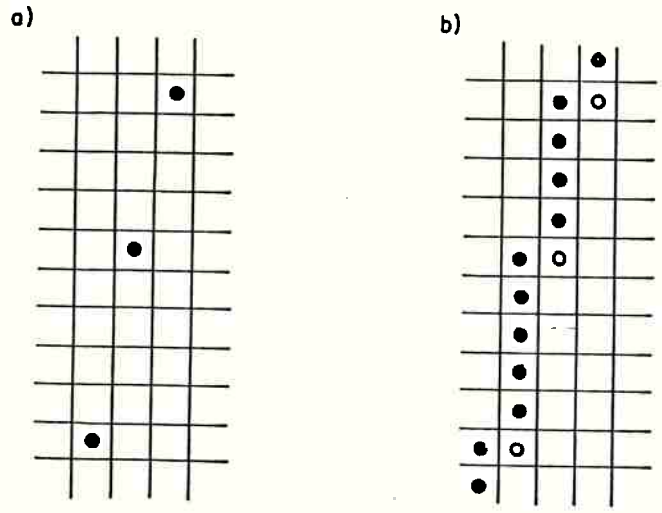


Figura 3.2 - Opção adotada para o desenho de retas de alta inclinação

- Traçado de Retângulos

Com respeito à geração de figuras retangulares, a transformação, para decomposição de retangulos em semiretas, é imediata. Basta fazer uma varredura do contorno vertical mais a esquerda do retângulo e com a informação da largura deste, utilizar os recursos de transferência de vetor de "bytes" para desenhar cada semireta que o compõe.

- Traçado de Figuras Circulares

As figuras circulares foram obtidas após o estudo de alguns algorítmos [DENERT-73] [PITEWAY-67] [PITEWAY-74]. As primitivas escolhidas foram: círculo, circunferência, semicírculo, semicircunferência, quarto de círculo, quarto de circunferência,

oitavo de círculo, coroa circular e algumas aplicações com elipses. Chegou-se a conclusão que, a melhor maneira de obter a decomposição das figuras circulares, era extrair os pontos de seu contorno (interno e externo no caso das coroas circulares), de onde estariam definidas as semiretas a serem desenhadas. Tais contornos são obtidos fazendo uma varredura dos pontos limite da equação da circunferência:

x.quadrado + y.quadrado := raio.quadrado

onde "x.quadrado" é a coordenada "x" (obtida de um sistema de coordenadas situado no centro da circunferência) elevada ao quadrado, "y.quadrado" é a coordenada "y" (mesmo sistema de coordenadas de "x") elevada ao quadrado e "raio.quadrado" é o raio da circunferência elevado ao quadrado. Um exemplo do resultado de tal varredura está mostrado na figura 3.3., que contêm os pontos de contorno de um quadrante de duas circunferências concêntricas. Igualmente ao que ocorreu com as retas inclinadas, surgem pontos espaçados originados por arredondamento, e novamente introduz-se uma variável de correção, tal que ao desenhar uma circunferência, esta não apareça interrompida. Para a determinação de quais semiretas irão compor um círculo, escolhe-se dentre os pontos de seu contorno, os que estão mais externos com relação ao eixo "x" e que mantenham uma continuidade com relação ao eixo "y". Estes estão designados na circunferência de maior raio da Figura 3.3., por pequenas circunferências (pontos "vazados"), sendo que os pontos de correção estão indicados por pequenos círculos (pontos "cheios").

A simetria das primitivas circulares permite que não sejam extraídos todos os pontos de seu contorno, mas apenas os pontos de um de seus quadrantes como está indicado na Figura 3.3., sendo os demais limites das semiretas obtidos por translações matemáticas simples.

Ainda fazendo uso da Figura 3.3, pode-se ilustrar a obtenção das coroas circulares considerando-se os dois contornos de circunferência mostrados, como sendo contorno interno e contorno externo. Os pontos eleitos para serem os limites das semiretas que se quer desenhar estão definidos nos dois contornos por pequenas circunferências (pontos "vazados").

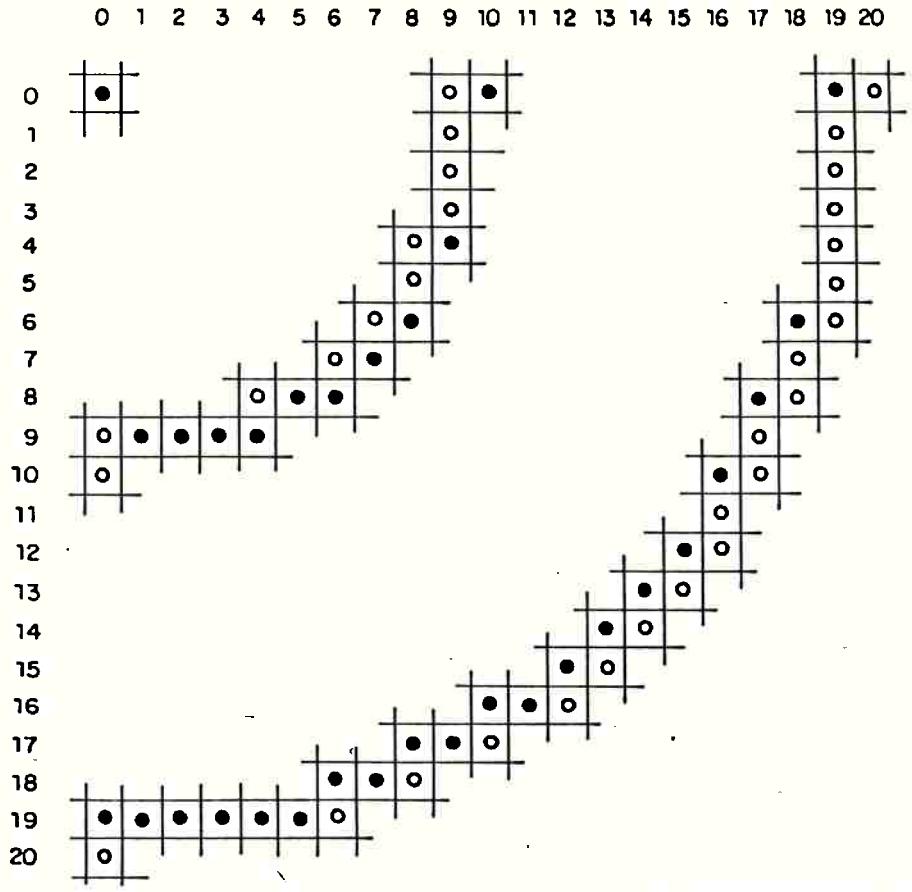


Figura 3.3 - Opção adotada para o desenho de coroas circulares

Realiza-se uma varredura nos pontos de contorno da circunferência externa, desenhando como se fosse um círculo normal, até que se chega em uma ordenada igual a ordenada inferior da circunferência interna. A partir daí as semiretas estarão definidas pelos pontos dados na Figura 3.3 por pequenas circunferências (pontos "vazados") e procede-se ao desenho do que resta.

Os semicírculos e os quartos de círculo decorrem imediatamente dos círculos, bastando para isto não desenhar as semiretas simétricas, no caso dos primeiros, e além disto, fixar o limite à esquerda no caso dos segundos. Para os oitavos de círculo, armazena-se em dois vetores unidimensionais os pontos "x" e "y" constituintes da semireta com origem no centro da circunferência, e que divide um quadrante em seus dois octantes constituintes. Após isto, e escolhendo o octante a ser desenhado por passagem de parâmetros, desenham-se as semiretas que o constituem definidas pelos pontos de contorno circular e pelos pontos de contorno da semireta que o gerou a partir da divisão do quadrante.

- Problema de Violação dos Limites da Tela

O desenvolvimento inicial das primitivas gráficas não levou em conta problemas decorrentes da violação dos limites disponíveis para desenho na tela gráfica. Isto significava que quando o desenho de uma geometria ultrapassava, por exemplo, o limite máximo imposto para a coordenada "x", nenhuma medida corretiva ou preventiva era adotada.

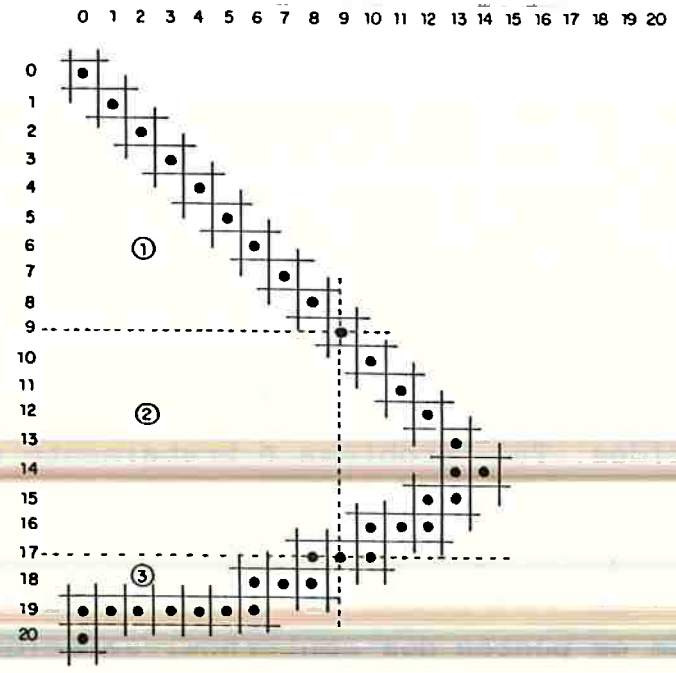


Figura 3.4 - Corte em oitavo de círculos

Associando isto ao fato de foi feito uma alteração de tipo na definição da memória de tela, que agora era um vetor contínuo, efeitos indesejáveis ocorriam se algum desses limites era ultrapassado.

Um exemplo típico de efeitos indesejáveis era no desenho de um círculo que violava o limite superior de "x". A memória de tela, por estar transformada em um vetor contínuo, faz com que o "pixel" mais a direita de uma linha da tela, ocupe uma posição de memória contígua ao "pixel" mais a esquerda da linha imediatamente abaixo. Com isto, quando se pretendia desenhar uma semireta que ultrapassava o limite superior de "x", ao desenhar o ponto imediatamente posterior ao limite superior de "x", o sistema o fazia no ponto mais a esquerda da linha imediatamenta abaixo. O efeito global era que, o limite excedente do desenho do círculo, ou a região que deveria ser cortada e não aparecer, por ultrapassar os limites de janela de tela, aparecia a esquerda da tela. Por outro lado, alterar estas rotinas, para que realizassem testes nos limites máximos, implicaria em se dispor de soluções mais lentas também para desenhos que não infringissem estes limites. Optou-se por manter todas estas rotinas que não faziam a verificação, e portanto eram mais rápidas, e implementar outras rotinas com as mesmas funções que desenhavam apenas nos limites da tela, ainda que mais lentas.

Um exemplo mais abrangente de verificação de limites encontra-se ilustrado na Figura 3.4., que indica o resultado de corte no desenho de um oitavo de círculo que ultrapassou o limite superior do eixo "x". No caso geral a atitude a tomar seria a verificação se cada semireta a ser desenhada estaria dentro dos limites estabelecidos. Tal problema é brutalmente simplificado se analizarmos os contornos gerados e estabelecermos quais pontos dos contornos da figura geram semiretas fora dos limites estabelecidos. Assim realizando-se uma varredura prévia nos vetores que contêm os pontos dos contornos, seleciona-se os que se

encontram dentro dos limites estabelecidos e que, por sua vez definem semiretas que também estão dentro dos limites. Como resultado, o desenho do oitavo de círculo da Figura 3.4., cortado pela linha vertical tracejada, por exemplo provocada pela borda da tela (efeito de "clipping"), ficaria reduzido ao desenho de três novas áreas: triangular (1), retangular (2) e circular (3).

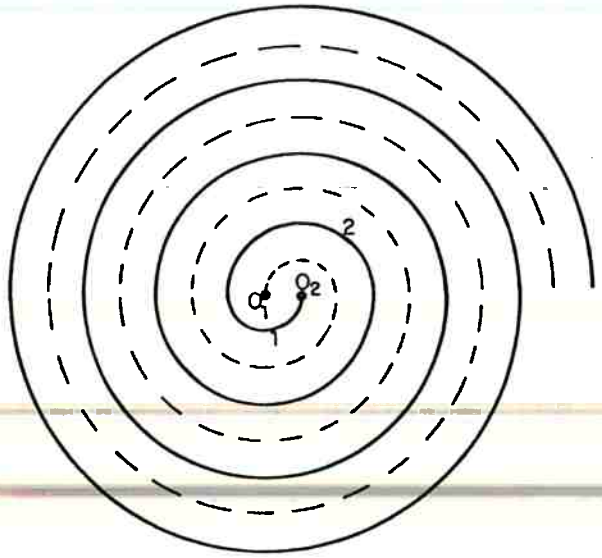


Figura 3.5 - Opção utilizada para desenhar elipses

Entre as rotinas desenvolvidas para demonstração da interface existem algumas que desenham elipses da forma mostrada na Figura 3.5., com superposição de desenhos de semicírculos, quarto de círculos ou oitavo de círculos. Parte-se de dois pontos iniciais "O1" e "O2" separados por metade da "largura" interna da espiral. Acompanhando a idéia pela Figura 3.5., parte-se do centro em "O1" e desenha-se uma semicircunferência (1) de raio "O1O2", centro em "O2" e desenha-se uma semicircunferência com raio do dobro de "O1O2", centro em "O1" e outra vez com raio do triplo de "O1O2", e assim sucessivamente.

A linha tracejada mostra a elipse obtida partindo-se inicialmente do centro "O2". Obviamente seguindo esta linha de desenho com semicírculos, os semicírculos posteriores encobririam os primeiros. Neste caso basta começar os desenhos pelos semicírculos mais externos.

- Impressão de Caracteres

Existe na interface a facilidade de se escrever caracteres ASCII na tela gráfica. Para tanto, quando o sistema é inicializado, carrega-se um arquivo desde o "DOS" do "PC", que contêm o "font" de caracteres a ser adotado. Este arquivo contêm uma sequência de caracteres que descreve, cada elemento do conjunto de caracteres imprimíveis ASCII, como uma matriz de pontos que o constituem. Optou-se por carregar o "font" de caracteres desta forma para que não fosse necessário recompilar toda a interface cada vez que se quisesse utilizar outro "font".

O fato de existirem dois processadores na placa gráfica (ver apêndice B) [INMOS-89] permitiu o aparecimento de um efeito interessante nas rotinas de demonstração desenvolvidas. O processador IMS T212 é responsável pelo controle e programação do gerador de temporização de vídeo programável (NEC D7220) e pelo controle do "chip" que gera e carrega o "palette" de cores [INMOS-87-2]. O processdor IMS T800 é responsável pela execução das rotinas gráficas propriamente ditas. Foi introduzida uma facilidade que permite que, ao mesmo tempo que o IMS T800 executa rotinas que desenham na tela, o IMS T212 executa uma outra rotina que desloca os "bytes" da tabela de cores do "palette" e vai recarregando-os. Isto faz com que, após 256 iterações da rotina executada no IMS T212, a um mesmo "pixel" da tela tenham sido associadas, em instantes sucessivos, todas as 256 cores disponíveis no "palette" gerado. Tal variação dinâmica nas cores de cada "pixel" produz um efeito visual bastante interessante.

3.2. ROTINAS DA INTERFACE

A seguir, nesta segunda parte do capítulo 3, são apresentadas as rotinas desenvolvidas para constituir a interface. Foram desenvolvidas cerca de seis mil linhas de programa em linguagem OCCAM, entre procedimentos, rotinas gráficas, programa principal e rotinas de demonstração.

3.2.1. Rotinas do Programa Organizador

No programa organizador, que se utiliza como interface com o usuário, são necessárias rotinas de entrada por teclado e de saída por tela, rotinas de geração de tabelas e menus, rotinas de tratamento de arquivos, "fonts" de caracteres e mensagens, e além disto rotinas para verificar a execução das rotinas gráficas. Nos processos que realizam a chamada às rotinas gráficas que se encontram no módulo "memória pantalla", por vezes faz-se referência à existência de parâmetros que não se encontram declarados explícitamente nestas. Para estes casos, a chamada para que sejam introduzidos os dados é feita em tempo de execução pelas próprias rotinas.

3.2.1. Entrada/Saída por Teclado/Tela

Descrição das rotinas utilizadas para entrada e saída por teclado ou tela, respectivamente:

1. SalidaCaracter (VAL INT caracter)

Imprime um caractere, na tela do PC, correspondente ao caractere ASCII de valor decimal (inteiro) dado pela variável "caracter". Este caractere será apresentado na tela com o tamanho, em número de algarismos, que seja necessário para representar a variável "caracter". Atualiza a variável global "caracter".

2. SalidaEntero (VAL INT entero)

Imprime um inteiro, na tela do PC, correspondente ao valor inteiro dado pela variável "entero". Este inteiro será apresentado na tela com o tamanho, em número de algarismos, que seja necessário para representar a variável "entero".

3. SalidaCadena (VAL []BYTE mensaje)

Imprime uma cadeia de caracteres armazenados como bytes na variável "mensaje" com um máximo de 80 caracteres.

4. VolverTds2 ()

Coloca uma mensagem na tela do PC e espera o ato de teclar qualquer caractere para retornar ao ambiente do TDS2.

Se utiliza para sair de programas OCCAM onde se queira que a saída não apague a tela do PC.

5. EntradaCaracter ()

Realiza a entrada de caracteres pelo teclado.

Atualiza a variável global "caracter" com o valor introduzido por teclado.

6. TeclearMandato ()

Aceita um comando por teclado guiando o usuário com uma mensagem de identificação.

7. TeclearParada (VAL []BYTE cadena)

Coloca uma mensagem de pedido de interrupção de um processo (de nome dado por "cadena") na tela, recebendo o caractere de parada do teclado e enviando à "memoria pantalla".

3.2.1.2. Gerar Tabelas/Telas

Nesta seção são apresentadas as rotinas de manipulação de telas e tabelas:

1. PantallaCogerNumero (VAL []BYTE cadena)

Prepara a tela para aceitar números que terão seu nome dado pelo vetor de bytes "cadena".

2. CabeceraInicial ()

Apresenta na tela o cabeçalho inicial do programa organizador.

3. MandatosDisponibles ()

Apresenta na tela uma relação de todos os comandos disponíveis e também de todos os códigos que devem ser introduzidos por teclado para a sua execução.

4. PantallaMandatos (VAL []BYTE cadena)

Prepara a tela do PC para aceitar o comando dado por "cadena".

5. PantallaNoImplementadoTodavia (VAL []BYTE cadena)

Coloca na tela uma mensagem de advertencia que indica que o comando dado por "cadena" não está ainda implementado.

Utilizado para guiar o usuário com relação às rotinas que ainda encontram-se em fase de desenvolvimento.

6. PantallaElegirLongitud (VAL []BYTE cadena1,
VAL []BYTE cadena2)

Prepara a tela para que seja introduzido pelo teclado a extensão (número de parâmetros) que terá a opção do comando "cadena1". A variável "cadena2" indica o número de pontos de um polígono que se vai desenhar, posto que, foi para este caso de aplicação em desenho de polígono de "n" lados, que a rotina foi inicialmente delineada.

Haverá dois parâmetros (coordenadas) para cada ponto e assim que o número de parâmetros estará determinado quando o usuário introduzir por teclado o número de pontos.

7. DistanciaEntreTablas (VAL INT n)

Coloca "n" linhas em branco de separação entre tabelas.

Aplicação imediata na estruturação de telas com tabelas.

3.2.1.3. Arquivos, Mensagens e "Font" de Caracteres

Nesta seção são apresentadas as rotinas de manipulação de arquivos, tratamento de mensagens e envio de arquivos que contêm os "fonts" de caracteres:

1. LeeBloque ()

Realiza a leitura de um bloco de 512 caracteres de arquivos que foram previamente abertos por outra operação.

Utilizada nas rotinas que fazem abertura e leitura de conteúdo de arquivos.

2. LeeFicheroFuenteNombreFijo ([]BYTE nombre.fichero,

VAL INT tamano,

VAL BOOL otra.vez,

CHAN OF ANY pantalla)

Realiza a leitura de um "font" de caracteres que está em um arquivo cujo nome é fixo e dado pela cadeia de caracteres "nombre.fichero".

Esta rotina é utilizada pela rotina que lê os parametros iniciais de um "fold" (3.2.1.3.-3) de onde se lê o nome do arquivo de "font" de caracteres que se vai utilizar para escrever mensagens na tela gráfica. A rotina que lê os parametros iniciais de um "fold" trás a definição dos canais da tela do PC e do teclado do PC como "pantalla" e "teclado". É feita a sua chamada desde o programa principal declarando estes canais como "screen" e "keyboard" posto que o sistema "sabe" que isto significa tela e teclado do PC, já que estes estão declarados nas bibliotecas definidas pela linguagem OCCAM. As palavras "screen" e "keyboard" são reservadas e utilizadas pelas bibliotecas de entrada/saída de OCCAM como sendo a tela e o teclado do PC. Neste conjunto de rotinas tais canais estão denominados como "pantalla" e "teclado", respectivamente. A rotina LeeFicheroFuenteCaracterFijo() tem suas subrotinas básicas de entrada e saída definidas com os nomes "screen" e "keyboard" e assim que, ao utilizá-las para a leitura de "folds" ocorriam problemas de compilação.

Para evitar os problemas de compilação foi feita esta versão que somente tem uma redefinição dos nomes dos canais de entrada/saída.

3. CargaParamInicDePliegue (CHAN OF INT teclado,

CHAN OF ANY pantalla,

CHAN OF ANY de.uf, a.uf)

Realiza a leitura do "fold" que está apontado pelo cursor no momento de sua chamada.

Se pode escolher o "palette" inicial que se vai carregar e o nome do arquivo de "font" de caracteres, simplesmente escrevendo qual é o "palette" que se deseja e qual é o nome do arquivo de "font" de caracteres em um "fold" exterior ao programa. Isto permite que alterações nestas condições iniciais não impliquem em ter que recompilar todos os programas.

4.LeeFicheroDOSPliegueFijo ([]BYTE

nombre.fichero.pliegue)

Em desenvolvimento. Realizará a leitura de um "fold" que foi transformado em arquivo DOS.

5. LeeFicheroFuenteCaracter ()

Faz a leitura de um "font" de caracteres que está em um arquivo cujo nome terá que ser introduzido por teclado.

Ver ítem 3.2.1.3-2 para referência.

6. LeeFicheroFuenteCaracterFijo ([]BYTE nombre.fichero,

VAL INT tamano,

VAL BOOL otra.vez)

Faz a leitura de um arquivo binário com nome fixo dado por "nombre.fichero", de tamanho dado pela variável "tamano".

Ver ítem 3.2.1.3.-2 para referência.

7. EscribirMensajePantallaGrafica ()

Escreve a mensagem que se introduz por teclado na posição (x,y), com a cor que também se introduz por teclado.

A rotina se encarrega de informar ao usuário que os parâmetros (e quais parâmetros) devem ser introduzidos por teclado.

8. EscribirMensajeCaracterDoble ()

Faz o mesmo da anterior mas com caracteres de tamanho duplo.

A rotina se encarrega de informar ao usuário que os parâmetros (e quais parâmetros) devem ser introduzidos por teclado.

9. EscribirMensajeCaracterVariable ()

Faz o mesmo das duas anteriores, mas com caracteres que tenham sua largura e altura variáveis (que se pode escolher por meio de parâmetros).

A rotina se encarrega de informar ao usuário que os parâmetros (e quais parâmetros) devem ser introduzidos por teclado.

3.2.1.4. Verificação

Rotina de verificação de execução:

1. VerificacionEjecucion ()

Recebe um caractere de advertência que indica o estado da execução das rotinas gráficas.

Atualiza a variável global "a.ver" com o valor deste caractere de verificação. Esta rotina, no caso de um desenvolvimento futuro, será a base para fazer o tratamento de erros que seria decorrente do tratamento da informação contida no caractere de advertência.

3.2.2. ROTINAS DE CONTROLE DA TELA ("CONTROL PANTALLA")

Nesta parte descreve-se as rotinas implementadas para a realização do controle da tela gráfica.

O módulo "control pantalla" faz a programação dos parâmetros que controlam a tela gráfica e carrega o "palette" cores iniciais. O usuário não tem acesso direto a estas rotinas.

As rotinas para configurar o gerador de temporização de vídeo programável (NEC D7220), as rotinas para o gerador da tabela de "palette" ("color look-up table" IMS G170) e o programa que inicializa o sistema e continuamente aguarda um pedido de alteração do "palette", estão presentes neste módulo.

As rotinas desenvolvidas são:

1. EscribeMandato (VAL INT mandato)

Espera que a fila de comandos e a de parâmetros ("command FIFO" e "parameter FIFO") do IMS B409 fique vazia para poder escrever o comando dado pela variável "mandato" na variável "registrador.mandato", que é a própria fila de comandos.

A informação sobre se a fila está vazia ou cheia é dada por testes no conteúdo da variável "registrador.estado".

2. EscribeParametro (VAL INT dato)

Espera que a fila de comandos e a de parâmetros ("command FIFO" e "parameter FIFO") do IMS B409 fique vazia para poder escrever o parâmetro dado pela variável "dato" na variável "parametro.fifo", que é a própria fila de parâmetros.

A informação sobre se a fila está vazia ou cheia é dada por testes no conteúdo da variável "registrador.estado".

3. Activar ()

Realiza a ativação do sistema de "hardware" gráfico por meio da rotina "EscribeMandato" com #0D como comando.

Consiste na aplicação do comando "BCTRL" que realiza uma restauração do conteúdo anterior da tela que estava definida previamente à sua inicialização (ver Apêndice B).

4. PonerModo (VAL INT modo)

Faz a variável "selec.modo.canal" igual a variável "modo".

Seleciona entre o modo do canal de "pixel" com 8 bits ou com 18 bits. Na placa gráfica há três canais de "pixel" que podem operar em dois modos, relativos ao número de bits que define cada "pixel" (ver apêndice B).

5. Reiniciar ()

Reinicializa o valor dos registradores de comando e parâmetros da placa gráfica.

O modo de obtenção dos parâmetros encontra-se mais detalhado no Apêndice B.

6. LeeEstado (INT32 estado)

Escreve o valor do registrador de estado na variável "estado".

Permite fazer testes para precisar o momento em que se podem tomar ações tais como enviar comandos à fila de comandos.

7. LoadTableA ([768]BYTE colours)

Realiza a carga de uma tabela de 768 cores no "palette".

8. CargaClut8 ()

Determina a tabela de cores que se vai carregar.

Carrega-se um "palette" padrão. Esta rotina leva o número 8 em seu nome para significar que está trabalhando com um canal de "pixel" representado por 8 bits.

9. CargaClut8Suave ()

Determina a tabela de cores com variação suave de tons, que se vai carregar.

Este "palette" apresenta uma variação suave de cores a medida que se aumenta o número correspondente à sua representação em binário. Esta rotina leva o número 8 em seu nome para significar que está trabalhando com um canal de "pixel" representado por 8 bits.

10. HaceClut8Suave ([768]BYTE tabla.colores)

Determina os valores dos bytes do "palette" de cores pondo estes bytes no vetor "tabla.colores".

Este "palette" apresenta uma variação suave de cores a medida que se aumenta o número correspondente à sua representação em binário. O que esta rotina faz é criar sua tabela de números binários. Esta rotina leva o número 8 em seu nome para significar que está trabalhando com um canal de "pixel" representado por 8 bits.

11. CambiaCargaClut8SuaveA ([768]BYTE tabla.colores)

Faz a mudança suave dos valores de cores do "palette" que está armazenado no vetor "tabla.colores", carregando este "palette" no canal A da placa gráfica. Isto é, além da variação suave de cores deste "palette", também realiza um deslocamento do "palette" ao longo do vetor onde está carregado produzindo o efeito de uma mudança dinâmica de cores independente das tarefas de desenho. Esta rotina leva o número 8 em seu nome para significar que está trabalhando com um canal de "pixel" representado por 8 bits.

3.2.3. ROTINAS DE MEMÓRIA DE TELA ("MEMORIA PANTALLA")

Já foi citado que existe um programa principal na memória associada ao processador T800 que recebe os comandos do programa organizador para a chamada das respectivas rotinas gráficas, descrito nos ítens 3.1.1.1 e 3.1.2.1 . Também neste processador (T800) são executadas as rotinas que constituem a saída gráfica para o monitor que se dispoe. É conveniente recordar que este conjunto de rotinas mais programa principal se denomina "memoria pantalla ". Tal conjunto realiza a intervenção na memória de tela.

Para uma melhor apresentação fez-se uma classificação das rotinas desenvolvidas dividindo-as de acordo com suas funções. Além da classificação por suas funções temos uma classificação adicional para o caso das rotinas que o usuário pode ou não acessar. As rotinas "não acessíveis ao usuário" estarão designadas por N.A.U. e as rotinas "acessíveis ao usuário" estarão designadas por A.U. na parte relativa a "Outras informações".

A seguir são apresentadas as rotinas desenvolvidas com as respectivas informações necessárias a sua utilização.

3.2.3.1 Iniciais

A seguir é feita uma descrição das rotinas utilizadas na fase de inicialização de parâmetros do sistema:

1. Escala (VAL REAL32 x, VAL REAL32 y)

Calcula a escala para a transformação de coordenadas de usuário em coordenadas internas. Se utiliza para obter as coordenadas internas do "pixel" a selecionar.

Utilizada no procedimento "IniciaPantalla". Atualiza as seguintes variáveis globais: escala, x.maximo.ent, y.maximo.ent, x.minimo.ent, y.minimo.ent. (N.A.U.)

2. IniciaVectorBytes (VAL INT px, VAL INT py,

VAL INT cuanto,

VAL BYTE color.inic)

Inicia o valor de um vetor de bytes com o valor do byte "color.inic".

Inicialmente temos a tela gráfica, ou a memória de tela, definida como uma matriz de bytes ([alto][ancho] BYTE pantalla). Para operações mais rápidas sobre ela a linguagem OCCAM oferece a facilidade de fazer executar uma redefinição do tipo de dado "pantalla" para mudar de uma matriz para um vetor de bytes, segundo se indica:

[(ancho*alto)] BYTE vec.pant RETYPES pantalla

E assim que isto se faz dentro desta rotina e as modificações feitas no vetor "vec.pant" são efetivamente feitas na memória de tela, como se fizesse as modificações na matriz "pantalla", pois agora "vec.pant" substitue esta matriz. Com isto podemos fazer modificações mais rápidas pois existem soluções algorítmicas mais velozes para escrever um mesmo byte em um vetor de bytes fazendo duplicações de partes deste vetor. Com os parâmetros "px" e "py" se pode obter a posição do vetor a partir do qual se vai escrever o valor "color.inic" por um número "cuanto" de vezes. Esta rotina atualiza a variável global "pantalla". (N.A.U.)

3. Color (VAL BYTE color.elegido)

Faz a associação da cor escolhida com o valor da variável "color.elegido", determinando a cor dos pixels desenhados a partir

de então.

Atualiza a variável global "color". (A.U.)

4. BorraPantalla ()

Apaga a tela gráfica. (A.U.)

5. IniciaPantalla (VAL REAL32 x, VAL REAL32 y)

Define o valor máximo de "x" e "y" das coordenadas da tela (define a janela máxima de trabalho na tela). (A.U.)

Inicia o valor dos registradores de controle associados ao "pixel port" e gerador de posições de memória do módulo IMS B408. Atualiza as seguintes variáveis globais: habilita.salida, modo.evento, pronto, color, habilita.entrelazado, color.elegida.ent, inicio.pantalla. A variável "inicio.pantalla" ("display start") contem o endereço do pixel mais a esquerda e mais acima da tela e o valor associado a esta funciona como um "offset" a ser adicionado ao endereço inicial da RAM "dual port". A variável "habilita.entrelazado" seleciona entre modo entrelaçado e não entrelaçado. Escrever 1 em "modo.evento" significa forçar a ocorrência da interrupção "event" com o ocorrência simultânea dos sinais "FieldSync" e "SysReady". A variável "habilita.salida" habilita os registradores de saída do "pixel port".

3.2.3.2. Rotinas de Desenho

Segue-se com a descrição das rotinas que realizam desenhos na tela gráfica.

3.2.3.2.1. Desenhar Pontos

Rotina para "pintar" um pixel:

1. DibujaPuntos (VAL INT px, VAL INT py)

Seleciona o "pixel" de coordenadas inteiras "px", "py" e associa à sua posição na matriz de tela o valor do byte "color". (N.A.U.)

Atualiza a matriz "pantalla".

3.2.3.2.2. Figuras Retilíneas

A seguir encontra-se a descrição das rotinas que desenham figuras retilíneas:

1. ObtenerYdeRecta (VAL INT px, VAL INT py,

VAL REAL32 pendiente,

VAL INT px.nuevo, INT py.nuevo)

Rotina definida, mas em fase de depuração. Em uma semi reta definida pelo ponto "px", "py" e pelo coeficiente angular "pendiente", determina para uma das coordenadas dada "px.nuevo", sua homóloga "py.nuevo" correspondente. Em outras palavras, para uma reta definida por um de seus pontos e o coeficiente angular, dado um ponto de abcissa "px.nuevo" encontrar sua ordenada "py.nuevo" correspondente, de tal forma que o par ("px.nuevo","py.nuevo"), pertença a reta. (N.A.U.)

2. ObtenerXdeRecta (VAL INT px, VAL INT py,

VAL REAL32 pendiente,

VAL INT py.nuevo, INT px.nuevo)

Rotina definida, mas em fase de depuração. Em uma semi reta definida pelo ponto "px", "py" e pelo coeficiente angular, determina para uma das coordenadas dada "py.nuevo", sua homóloga "px.nuevo" correspondente. Em outras palavras, para uma reta definida por um de seus pontos e o coeficiente angular, dado um ponto de ordenada "py.nuevo" encontrar sua abcissa "px.nuevo" correspondente, de tal forma que o par ("px.nuevo","py.nuevo"), pertença a reta. (N.A.U.)

3. VerificaRectaPertenecePantalla (INT pxi, INT pyi,

INT pxf, INT pyf,

BOOL pertenece)

Rotina definida, mas em fase de depuração. Em uma semireta definida com os pontos "pxi", "pyi" (iniciais) e "pxf", "pyf" (finais), faz-se a verificação se está contida na janela de tela definida. (N.A.U.)

Atualiza a variável Booleana "pertenece" com "TRUE" se pertence ou com "FALSE" se não pertence à janela de tela.

4. LimiteSuperiorRecta (INT pxi, INT pyi,

INT pxf, INT pyf)

Rotina definida, mas em fase de depuração. Atualiza os valores de "pxi", "pyi", "pxf", "pyf" para que fiquem dentro dos limites mínimos e máximos da janela de tela definida inicialmente. (N.A.U.)

5. AlmacenaPuntosSegmentoParaleloEje(VAL INT delta,

VAL INT coord.i, VAL INT coord.f,

VAL INT coord.fija, VAL BOOL paralela.x,

INT tamano, []INT contorno.x,

[]INT contorno.y)

Calcula e armazena em "contorno.x", e "contorno.y" todos os pontos que pertencem a um segmento paralelo a um dos eixos de coordenadas, que começa no ponto "coord.i", "coord.fija" se "paralela.x=TRUE", ou "coord.fija", "coord.i" se "paralela.x=FALSE". Quando da explanação sobre os algorítmos utilizados para desenho, no ítem 3.1.4., foi falado sobre a necessidade de se transformar as figuras a serem desenhadas em uma série de segmentos de reta. Foi falado também que uma maneira de realizar isto seria extrair o contorno das figuras, armazenando tais pontos de contorno em vetores, e, posteriormente, utilizando estes pontos armazenados para definir os segmentos de reta a serem desenhados. Esta rotina é uma das que realiza este trabalho. (N.A.U.)

Atualiza a variável tamanho com o número de pontos que constituem o segmento.

6. AlmacenaPuntosSegmentoCualquiera (VAL INT pxi,

VAL INT pyi, VAL INT pxf,

VAL INT pyf, VAL REAL32 m,

INT tamano, []INT contorno.x,

[]INT contorno.y)

Calcula e armazena em "contorno.x", e "contorno.y" todos os pontos de um segmento qualquer que começa no ponto "pxi", "pyi" e termina no ponto "pxf", "pyf" e com coeficiente angular = m. Quando da explanação sobre os algorítmos utilizados para desenho, no ítem 3.1.4., foi falado sobre a necessidade de se transformar as figuras a serem desenhadas em uma série de segmentos de reta. Foi falado também que uma maneira de realizar isto seria extrair o contorno das figuras, armazenando tais pontos de contorno em vetores, e, posteriormente, utilizando estes pontos armazenados para definir os segmentos de reta a serem desenhados. Esta rotina é outra das que realiza este trabalho. (N.A.U.)

Atualiza a variável tamanho com o número de pontos do contorno.

7. AlmacenaPuntosSegmentoNA (VAL INT pxi, VAL INT pyi,
VAL INT pxf, VAL INT pyf,
INT tamano, []INT contorno.x,
[]INT contorno.y)

Calcula e armazena em "contorno.x", e "contorno.y" todos os pontos que constituem o segmento que começa no ponto "pxi", "pyi" e termina no ponto "pxf", "pyf". Quando da explanação sobre os algorítmos utilizados para desenho, no ítem 3.1.4., foi falado sobre a necessidade de se transformar as figuras a serem desenhadas em uma série de segmentos de reta. Foi falado também que uma maneira de realizar isto seria extrair o contorno das figuras, armazenando tais pontos de contorno em vetores, e, posteriormente, utilizando estes pontos armazenados para definir os segmentos de reta a serem desenhados. Esta rotina é outra das que realiza este trabalho. (N.A.U.)

Não se pode ali associar novos valores aos parâmetros de coordenadas pelo modo como estão definidos (VAL INT). Atualiza a variável "tamano" com o número de pontos do segmento.

8. AlmacenaPuntosSegmentoA (INT pxi, INT pyi,

INT pxf, INT pyf,

INT tamano,

[]INT contorno.x,

[]INT contorno.y)

Calcula e armazena em "contorno.x", e "contorno.y" todos os pontos que constituem o segmento que começa no ponto "pxi", "pyi" e termina no ponto "pxf", "pyf". Quando da explanação sobre os algorítmos utilizados para desenho, no ítem 3.1.4., foi falado sobre a necessidade de se transformar as figuras a serem desenhadas em uma série de segmentos de reta. Foi falado também que uma maneira de realizar isto seria extrair o contorno das figuras, armazenando tais pontos de contorno em vetores, e, posteriormente, utilizando estes pontos armazenados para definir os segmentos de reta a serem desenhados. Esta rotina é outra das que realiza este trabalho. (N.A.U.)

Pode-se ali associar novos valores aos parâmetros de coordenadas pelo modo como estão definidos (INT). Atualiza a variável "tamano" com o número de pontos do segmento.

9. LineaParalelaEje (VAL INT delta, VAL INT coord.i,

VAL INT coord.f, VAL INT coord.fija,

VAL BOOL paralela.x)

Desenha uma linha, paralela a um dos eixos de cooordenadas,

que começa no ponto "coord.i", "coord.fija" se paralela.x = TRUE, ou "coord.fija", "coord.i" se paralela.x=FALSE. (N.A.U.)

10. LineaCualquieraEntero (VAL INT pxi, VAL INT pyi,
VAL INT pxf, VAL INT pyf,
VAL REAL32 m)

Desenha uma linha qualquer que começa no ponto "pxi", "pyi" e termina no ponto "pxf", "pyf". (N.A.U.)

11. TiraLineaEntero (VAL INT pxi, VAL INT pyi,
VAL INT pxf, VAL INT pyf)

Desenha uma linha que começa no ponto "pxi", "pyi" e termina no ponto "pxf", "pyf", sendo as coordenadas números inteiros. No início do desenvolvimento da interface, o protocolo de comunicação aceitava a transmissão de números reais como parâmetros. Com a definição final do protocolo, decidiu-se enviar apenas parâmetros inteiros, e para isto agregou-se ao nome da rotina que desenhava linhas a palavra "entero", não trazendo outro significado senão o de diferenciá-la da rotina desenvolvida anteriormente. (A.U.)

12. TiraLineaAlmacenaPuntosA (INT pxi, INT pyi,
INT pxf, INT pyf)

Desenha una linea que começa no ponto "pxi", "pyi" e termina no ponto "pxf", "pyf", sendo as coordenadas números inteiros. Esta rotina além de desenhar uma linha, armazena os pontos desenhados, para utilização em extração de contornos. (N.A.U.)

As coordenadas são parâmetros aos quais se podem associar valores e se utilizam aqui as rotinas de armazenar pontos que constituem segmentos.

13. TiraLineaAlmacenaPuntosNA (VAL INT pxi,VAL INT pyi,

VAL INT pxf, VAL INT pyf)

Desenha uma linha que começa no ponto "xi", "yi" e termina no ponto "xf", "yf", sendo as coordenadas números inteiros. Esta rotina, do mesmo modo que a anterior, além de desenhar uma linha, armazena os pontos desenhados, para utilização em extração de contornos. (N.A.U.)

As coordenadas são parâmetros aos quais não se pode associar valores e se utilizam aqui as rotinas de armazenar pontos que constituem segmentos.

14. Rectangulo (VAL REAL32 xi, VAL REAL32 yi,

VAL REAL32 xf, VAL REAL32 yf)

Desenha um retangulo definido pelas coordenadas de usuário "xi", "yi" e "xf", "yf". (A.U.)

15. Rectangulo1 (VAL REAL32 xi, VAL REAL32 yi,

VAL REAL32 xf, VAL REAL32 yf)

Desenha um retangulo definido pelos pontos "xi", "yi" e "xf", "yf", sendo os parâmetros variáveis de tipo de dado real. (A.U.)

O faz mais lentamente que a anterior porque realiza o desenho ponto a ponto e não utiliza as técnicas de decompor o retângulo em uma sequência de semiretas.

16. ParalelogramoParaleloX (VAL INT px1, VAL INT px2,

VAL INT py1.2, VAL INT pxo,

VAL INT pyo)

Rotina definida, mas em fase de depuração. Desenha um paralelogramo paralelo ao eixo "x" e definido por tres pontos que se percorrem em sentido antihorário. (N.A.U.)

17. ParalelogramoParaleloY (VAL INT py1, VAL INT py2,

VAL INT px1.2, VAL INT pxo,

VAL INT pyo)

Rotina definida, mas em fase de depuração. Desenha um paralelogramo paralelo ao eixo "y" e definido por tres pontos que se percorrem em sentido antihorário. (N.A.U.)

18. Paralelogramo (VAL INT px1, VAL INT py1,

VAL INT pxf, VAL INT pyf,

VAL INT pxo, VAL INT pyo)

Rotina definida, mas em fase de depuração. Desenha um paralelogramo qualquer definido pelos pontos "px1", "py1", e "pxf", "pyf" e finalmente "pxo", "pyo", sendo os parâmetros definidos como inteiros aos quais não se pode associar valores. (A.U.)

3.2.3.2.3. Figuras Curvilíneas

Descrição das rotinas que fazem desenhos de figuras curvilíneas:

1. Circulo (VAL REAL32 xo, VAL REAL32 yo, VAL REAL32 r)

Desenha um círculo de centro no ponto (xo,yo) e raio "r". (A.U.)

Utiliza vetores de bytes (mais rápido) sem verificação dos limites da janela de tela definida, ou seja, não faz uma checagem se o círculo se encontra todo dentro dos limites estabelecidos pela rotina "IniciaPantalla". Isto pode ocasionar o aparecimento de trechos do círculo em locais não esperados.

2. Circulo1 (VAL REAL32 xo, VAL REAL32 yo, VAL REAL32 r)

Desenha um círculo de centro no ponto (xo,yo) e raio "r". (A.U.)

Não utiliza a rotina de iniciar valor de vetor de bytes e portanto é mais lenta que a anterior. Realiza o desenho ponto a ponto e sem verificação dos limites da janela de tela definida.

3. Circulo2 (VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r)

Desenha um círculo de centro no ponto (xo,yo) e raio "r". (A.U.)

Utiliza a rotina de iniciar valor de vetor de bytes e portanto é mais rápida que a anterior (mas não tanto quanto a primeira) e realiza verificação dos limites da janela de tela definida.

4. SemiCirculo (VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r, VAL INT sup)

Desenha um semicírculo de centro no ponto (xo,yo) e raio "r". O semicírculo será superior se ao parâmetro "sup" se associa o valor zero e inferior, se a este se associa valor um. (A.U.)

Não realiza verificação dos limites da janela de tela definida.

5. SemiCirculo1 (VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r, VAL INT sup)

Desenha um semicírculo de centro no ponto (xo,yo) e raio "r". O semicírculo será superior se ao parâmetro "sup" se associa o valor zero e inferior, se a este se associa valor um. (A.U.)

Realiza verificação dos limites da janela de tela definida.

6. CuartoDeCirculo (VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r, VAL INT cuadrante)

Desenha um quarto de círculo (ou um quadrante) de centro no ponto (xo,yo) e raio "r". O quadrante será dado pelo parâmetro "quadrante". (A.U.)

Não realiza verificação dos limites da janela de tela definida.

7. CuartoDeCirculo1(VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r, VAL INT cuadrante)

Desenha um quarto de círculo (ou um quadrante) de centro no ponto (xo,yo) e raio "r". O quadrante será dado pelo parâmetro "cuadrante". (A.U.)

Realiza verificação dos limites da janela de tela definida.

8. OitavoDeCirculo (VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r, VAL INT oitavo)

Desenha um oitavo de círculo (ou um octante) de centro no ponto (xo,yo) e raio "r". O octante será dado pelo parâmetro "oitavo". (A.U.)

Não realiza verificação dos limites da janela de tela definida.

9. OitavoDeCirculo1(VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r, VAL INT oitavo)

Desenha um oitavo de círculo (ou um octante) de centro no ponto (xo,yo) e raio "r". O octante será dado pelo parâmetro "oitavo". (A.U.)

Realiza verificação dos limites da janela de tela definida.

10. Circunferencia (VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r)

Desenha uma circunferência de centro no ponto (xo,yo) e raio "r". (A.U.)

Não realiza verificação dos limites da janela de tela definida.

11. SemiCircunferencia (VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r, VAL INT sup)

Desenha um semicírculo de centro no ponto (xo,yo) e raio "r". O semicírculo será superior se ao parâmetro "sup" se associa o valor zero e inferior, se associamos valor um. (A.U.)

Não realiza verificação dos limites da janela de tela definida.

12. CuartoDeCircunferencia (VAL REAL32 xo,VAL REAL32 yo,

VAL REAL32 r, VAL INT cuadrante)

Desenha um quarto de circunferência (ou um quadrante de uma circunferência) de centro no ponto (xo,yo) e radio "r". O quadrante será dado pelo parâmetro "cuadrante". (A.U.)

Não realiza verificação dos limites da janela de tela definida.

13. EspiralSemiDosColores (VAL REAL32 xo,VAL REAL32 yo,

VAL REAL32 r.ext, VAL REAL32 ancho.int.r,

VAL BYTE color1, VAL BYTE color2)

Desenha uma espiral de centro no ponto (xo,yo), e raio externo "r.ext", feita com semicírculos cujos raios se vão diminuindo por um valor dado por "ancho.int.r". A cada conjunto de dois semicírculos consecutivos é uma cor dada por "color1" ou "color2" alternativamente. Consultar Figura 3.5., para esclarecimentos. (A.U.)

Não realiza verificação dos limites da janela de tela definida.

14. EspiralCuarto (VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r.ext, VAL REAL32 ancho.int.r)

Desenha uma espiral de centro no ponto (xo,yo), e raio externo "r.ext", feita com quartos de círculo cujos raios se vão diminuindo por um valor dado por "ancho.int.r". (A.U.)

Não realiza verificação dos limites da janela de tela definida.

15. EspiralCuartoContorno (VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r.ext, VAL REAL32 ancho.int.r)

Desenha uma espiral de centro no ponto (xo,yo), e raio externo "r.ext", feita com quartos de círculo cujos raios se vão diminuindo por um valor dado por "ancho.int.r". Além disto desenha um contorno na espiral. (A.U.)

Não realiza verificação dos limites da janela de tela definida. A variação de cores de um quarto de círculo para outro é lenta, ou seja, a atribuição de cores a cada quarto é feita somando 2 ao byte de cor anterior.

16. EspiralCuartoContorno1 (VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r.ext, VAL REAL32 ancho.int.r)

Desenha uma espiral de centro no ponto (xo,yo), e raio externo "r.ext", feita com quartos de círculo cujos raios se vão diminuindo por um valor dado por "ancho.int.r". Além disto desenha um contorno na espiral. (A.U.)

Não realiza verificação dos limites da janela de tela definida. A variação de cores de um quarto de círculo para outro é mais rápida que a anterior, ou seja, a atribuição de cores a cada quarto é feita somando 4 ao byte de cor anterior.

17. EspiralOitavoContorno (VAL REAL32 xo, VAL REAL32 yo,
VAL REAL32 r.ext, VAL REAL32 ancho.int.r)

Desenha uma espiral de centro no ponto (xo,yo), e raio externo "r.ext", feita com oitavos de círculo cujos raios se vão diminuindo por um valor dado por "ancho.int.r". Além disto desenha um contorno na espiral. (A.U.)

Não realiza verificação dos limites da janela de tela definida. A variação de cores de um oitavo de círculo para outro é rápida, ou seja, a atribuição de cores a cada quarto é feita somando 4 ao byte de cor anterior.

18. EspiralOitavoDosCores1 (VAL REAL32 xo, VAL REAL32 yo,
VAL REAL32 r.ext, VAL REAL32 ancho.int.r,
VAL BYTE color1, VAL BYTE color2)

Desenha uma espiral de centro no ponto (xo,yo), e raio externo "r.ext", feita com oitavos de círculo cujos raios se vão diminuindo por um valor dado por "ancho.int.r". Além disto desenha um contorno na espiral. (A.U.)

Não realiza verificação dos limites da janela de tela definida. A variação de cores de um oitavo de círculo para outro rápida, ou seja, a atribuição de cores a cada quarto é feita somando 4 ao byte de cor anterior. A cada conjunto de quatro oitavos consecutivos é associado uma cor dada por "color1" ou "color2" alternativamente.

19. Corona (VAL REAL32 xo, VAL REAL32 yo,

VAL REAL32 r.int, VAL REAL32 r.ext)

Desenha una coroa de centro no ponto (xo,yo), raio interno "r.int" e raio externo "r.ext". Ver Figura 3.3., onde apresenta-se a opçào adotada para o desenho de coroas circulares. (A.U.)

3.2.3.3. Rotinas Para Caracteres e Mensagens

Apresentam-se a seguir rotinas para manusear caracteres e imprimir mensagens na tela gráfica:

1. EscribirMensajePantallaGrafica (VAL INT color.ent,

VAL INT x.mens, VAL INT y.mens,

VAL []INT cadena.ent)

Escreve na tela gráfica a mensagem contida em "cadena.ent", na posição "x.mens", "y.mens", com a cor "color.ent". (A.U.)

2. EscribirMensajeCaracterDoble (VAL INT color.ent,

VAL INT x.mens, VAL INT y.mens,

VAL []INT cadena.ent)

Escreve na tela gráfica a mensagem contida em "cadena.ent", utilizando caracteres de tamanho duplo, na posição "x.mens", "y.mens", com a cor "color.ent". (A.U.)

3. EscribirMensajeCaracterVariable (VAL INT color.ent,

VAL INT x.mens, VAL INT y.mens,

VAL INT ancho.esc, VAL INT alto.esc,

VAL []INT cadena.ent)

Escreve na tela gráfica a mensagem contida em "cadena.ent", utilizando caracteres de tamanho proporcional em largura e altura a "ancho.esc" e "alto.esc" respectivamente, na posição "x.mens", "y.mens", com a cor "color.ent". (A.U.)

4. EscribirFicherosASCII ()

Rotina definida, mas em fase de depuração. Copia o conteudo de um arquivo ASCII na tela gráfica utilizando o "font" de caracteres escolhido inicialmente. (N.A.U.)

5. CargaFuenteCaracteres (VAL INT num.carac,

VAL INT font.x, VAL INT font.y,

VAL INT longitud.font)

Carrega um "font" de padrão de caracteres, que define os tipos de caracteres que vão ser impressos na tela gráfica, ao se escrever alguma mensagem nesta. (A.U.)

Carrega o "font" de caracteres escolhido, recebido desde o organizador, em uma matriz de bytes :

[num.carac][font.y]font.elegido.by

3.2.3.4. Rotinas de Demonstração

Foram desenvolvidas algumas rotinas para servirem de exemplos de utilização das rotinas gráficas implementadas com a interface:

1. DibujaEjemploO (VAL BYTE color.inicial)

Desenha janelas de tamanho fixo partindo de uma cor inicial dada. (A.U.)

2. DibujaEjemplo1 (VAL BYTE color.inicial)

Desenha quadrados e círculos de tamanho fixo partindo de uma cor inicial dada. (A.U.)

3. DibujaEjemplo2Patron (VAL INT32 semilla)

Desenha círculos de tamanho variável, cujos centros, raios e cores vão mudando dinâmicamente, partindo das condições iniciais de uma "semilla" (semente para geração de um número pseudo aleatório) dada. Aqui é utilizado o recurso de geração de números pseudo-aleatórios, dado pela linguagem OCCAM, através de sua função "RAN". (N.A.U.)

A mudança de cores se realiza segundo um "palette" padrão de cores.

4. DibujaEjemplo2Suave (VAL INT32 semilla)

Desenha círculos de tamanho variável, cujos centros, raios e cores vão mudando dinâmicamente, partindo das condições iniciais de uma "semilla" (semente para a geração de um número pseudoaleatório) dada. (N.A.U.)

A mudança de cores se realiza segundo outro "palette" que faz uma alteração suave de cores. Além disto desenha um contorno nos círculos.

5. DibujaEjemplo2Suave1 (VAL INT32 semilla)

Desenha círculos de tamanho variável, cujos centros, raios e cores se vão alterando dinâmicamente, partindo das condições iniciais de uma "semilla" (semente) dada. (N.A.U.)

A mudança de cores se realiza segundo outro "palette" que faz uma alteração suave de cores. Não desenha um contorno nos círculos.

6. DibujaEjemplo2 (VAL INT32 semilla)

Desenha círculos de tamanho variável, cujos centros, raios e cores se vão alterando dinâmicamente, partindo das condições iniciais de uma "semilla" dada. (A.U.)

A mudança de cores se realiza segundo o "palette" que está carregado no momento da chamada da rotina (que pode ser a de padrão de cores, ou a de alteração suave de cores).

7. Mandelbroot ()

Calcula o conjunto de Mandelbroot por meio de até 255 iterações em cada ponto, associando uma cor aos pontos do conjunto que é igual ao número de iterações necessárias para verificar se ele faz parte do conjunto. (A.U.)

8. Julia ()

Calcula o conjunto de Julia por meio de até 255 iterações, associando uma cor aos pontos do conjunto que é igual ao número de iterações necessárias para verificar se ele faz parte do conjunto. (A.U.) Em desenvolvimento.

9. MostraPaleta ()

Desenha 256 retangulos na tela mostrando em cada um uma cor do "palette" atual. (A.U.) Rotina definida, mas em fase de depuração.

10. DibujaEjemplo9 (VAL BYTE color.inicial)

Faz o mesmo que a rotina DibujaEjemplo1, mas com algoritmos que tornam sua execução mais lenta. (A.U.)

Estes algorítmos fazem o desenho ponto a ponto e não utilizam a rotina de iniciar vetor de bytes.

3.2.3.5. Verificação

Descrição da rotina de verificação:

1. VerificacionEjecucion ()

Envia um caractere para "memoria pantalla" para a verificação da execução das rotinas gráficas. (A.U.)

Atualiza a variável global "a.ver" com o valor deste caractere de verificação.

3.3. Um Caso Prático de Aplicação

A seguir é apresentado um caso prático de aplicação desta interface.

D.O.C.I.L. (Sistema Roteador de Pistas Para Placas de Cicuito Impresso)

Está em desenvolvimento, no Instituto de Automática Industrial do Ministério de Educação e Ciência do Governo Espanhol, um sistema roteador para placas de circuito impresso em linguagem OCCAM [PEDRAZA-87] [DE PEDRO-88] [PEDRAZA-88]. Tal sistema recebeu a denominação de D.O.C.I.L. ("Diseño por Ordenador de Circuitos Impresos" acrescentando "L" para compor o nome). Este sistema é executado em uma arquitetura paralela com Transputers, com a distribuição dos processos de busca de caminhos e escolha das melhores rotas entre os vários processadores.

Utiliza-se aqui todos os recursos disponíveis na linguagem OCCAM e a facilidade da realização de arquiteturas específicas com Transputers. Este problema pode ser enunciado como a busca de caminhos que unam entre si pontos terminais, agrupados em diferentes subconjuntos disjuntos m(i), de um conjunto N de pontos [DE PEDRO-88]. Estes caminhos devem cumprir certas restrições, como por exemplo que um ponto nao pode pertencer a caminhos de diferentes subconjuntos. Para atacar o problema foi feita uma decomposição das tarefas a serem executadas em blocos modulares. Assim temos a seguinte divisão:

- **Bloco buscador** - procede à seleção do caminho mais promissor

- **Bloco traçador** - realiza a geração dos caminhos sucessores

- **Bloco comprovador** - faz a comprovação se o sistema alcançou o objetivo

O processo global que decorre desta divisão atua com o bloco buscador selecionando o melhor nó para ser expandido e enviando este pedido de expansão deste nó ao bloco traçador, o qual estará encarregado de gerar os caminhos sucessores do nó eleito pelo buscador. Os novos nós gerados pelo traçador serão enviados tanto ao buscador como ao comprovador, de forma que enquanto o comprovador verifica se alguns dos novos caminhos atingiram seu objetivo, o buscador elege o novo nó a ser expandido, emitindo um novo pedido de geração ao traçador. Isto faz com que haja uma sobreposição temporal entre as etapas de buscar e comprovar, com o consequente aumento que isto incorpora à execução dos processos. O processo global de busca, traçado e comprovação será iterativo até que se atinja o objetivo.

Este roteador utilizava para interface gráfica com o usuário, um conjunto de rotinas básicas para desenhar na tela gráfica, disponíveis em um sistema Tektronix. Para alterar a configuração para um sistema independiente do Tektronix, utilizou-se a interface gráfica objeto desta dissertação cujas rotinas gráficas implementadas em OCCAM abrangem ao conjunto de rotinas básicas que estavam disponíveis no Tektronix, e que agora se poderiam executar em uma placa gráfica feita com Transputers. Além de executar todas as rotinas básicas necessárias ao sistema e disponíveis no Tektronix se disporia agora de rotinas para carregar "fonts" de caracteres, escrever mensagens na tela gráfica e rotinas para carregar parâmetros diretamente de "folds" (isto é, rotinas que fazem leitura de "folds").

Ganhou-se em velocidade de apresentação das telas gráficas com relação a anterior utilização com o Tektronix como interface porque agora existem processadores específicos para os cálculos matemáticos das rotinas, para o controle de sinais de temporização do monitor gráfico e geração dinâmica de novos "palettes".

A localização de falhas no desenvolvimento dos processos do programa de roteamento que enviam os comandos gráficos para a interface tornou-se facilitada porque os módulos que executam as tarefas gráficas estao agora integrados ao sistemas com compatibilidade de "hardware" e "software".

4. CONSIDERAÇÕES FINAIS E CONCLUSÕES

A seguir apresentam-se alguns comentários sobre uma possível continuidade do trabalho e conclusões finais.

4.1. Continuidade do Trabalho

Numa tentativa de realizar um texto o mais independente possível de outras referências extensas, ou na medida do possível, um texto autocontido, optou-se por discorrer sobre a arquitetura dos Transputers, alguns fundamentos da linguagem OCCAM e sobre o sistema de desenvolvimento de programas em OCCAM (tds2). Estes temas foram desenvolvidos no capítulo 2, de modo a fornecer a base de alguns dos conceitos e ferramentas utilizados no desenrolar da realização prática desta interface. Para situar o desenvolvimento quanto aos aspectos de infraestrutura de hardware e software, estruturação do software da interface, técnicas de otimização de "performance" e algorítmos implementados, destinou-se a primeira parte do capítulo 3. Com relação a infraestrutura de hardware, em primeiro lugar comenta-se sobre o sistema global em que se insere esta interface, passando-se pela placa de desenvolvimento de OCCAM, e, finalmente, sobre a placa gráfica, onde serão executadas as rotinas gráficas efetivamente. Segue-se um relato sobre a estruturação utilizada para os programas desenvolvidos e a apresentação da definição do protocolo para acesso às rotinas gráficas. Uma parte mais teórica vem, com a apresentação e discussão, de técnicas de otimização de "performance" para sistemas a Transputers, abordando em casos separados, as características específicas para um único Transputer e para mais de um Transputer. Tais características nos permitem, estudar a aplicabilidade das técnicas para o caso real desta interface, e além disto, obter elementos para julgar sua efetividade na solução

deste problema. Não poderiam deixar de ser abordados neste trabalho os critérios empregados na concepção do sistema gráfico paralelo, tanto do ponto de vista do que se espera de um sistema paralelo, como do ponto de vista de quais são as características desejáveis em um único processador gráfico. Considerações sobre os algorítmos implementados foram feitas à luz das idéias emergentes da abordagem e discussão destes critérios. Na segunda parte do capítulo 3 é feita uma descrição de todas as rotinas desenvolvidas para compor a interface, onde são dadas informações necessárias para seu entendimento e utilização. A descrição está dividida pelos módulos onde cada rotina se insere. Na terceira parte deste mesmo capítulo apresenta-se a aplicação desta interface em um caso prático real, que embora ainda não explore suas características de paralelismo gráfico, demonstrou sua compatibilidade imediata com sistemas genéricos a Transputers.

O trabalho desenvolvido constitue uma base sólida para desenvolvimentos posteriores pois, houve a preocupação em se manter a compatibilidade com sistemas genéricos a Transputers, e ainda, realizar um conjunto mínimo de rotinas gráficas, que cobrissem a maior parte das aplicações mais usuais.

Podemos falar em desenvolvimentos posteriores deste sistema com relação a duas linhas que seriam o desenvolvimento de novas rotinas gráficas e o aprimoramento das funções de mais alto nível. Abaixo procura-se relatar aspectos destas duas linhas.

Foi deixado para uma etapa posterior o desenvolvimento de rotinas mais específicas tais como por exemplo "zoom" de tela, muito importante para algumas aplicações mas dispensável por hora em uma interface que se propunha de âmbito geral.

Outras aplicações poderiam ter interesse em figuras geométricas tais como elipses ou paralelogramos, mas pode-se dizer

que já foi dado o passo principal na direção de obter eficientemente estas e outros tipos de figuras. Dando a estas um tratamento semelhante ao do desenho de círculos onde primeiro se calculam os pontos do contorno de um quarto (ou quadrante) do círculo por meio da equação da circunferência que o circunda, extraída das lições de geometria analítica. Uma vez obtidos estes pontos pode-se transformar o preenchimento do círculo no preenchimento de dois semi círculos simétricos, que, por sua vez, seriam o desenho de uma série de vetores, os quais se pode realizar rapidamente pela rotina de iniciar o valor de um vetor de bytes. Para utilizar o mesmo método para elipses bastaria agora substituir a equação do cálculo de pontos para a de elipses e os processos de simetria e de transformação para vetores seriam os mesmos. Em analogia para paralelogramos podem-se usar as rotinas já desenvolvidas para extração (ou armazenagem) dos pontos de um segmento de reta dado, transformando os preenchimentos em desenhos de vetores de bytes.

Com relação ao sistema, um aperfeiçoamento interessante seria o estabelecimento de um gerenciador para uma base de dados gráfica onde todas as geometrias desenhadas estivessem representadas logicamente junto ao programa. Isto forneceria a infraestrutura necessária para operações do tipo apagar geometrias ou fazer "zoom" de janelas da tela. Um gerenciador de versões de desenhos também seria útil para aplicações em que o sistema gráfico fosse suportar sistemas de CAD para projeto em geral, tendo que constituir uma ferramenta mais poderosa.

Também fica como sugestão a implementação de uma fila de comandos, ou fila de pedidos de execução de rotinas gráficas. Para sistemas a Transputers com a utilização de muitos processadores, ou de características marcantemente concorrentes, pode ocorrer que muitos processos façam pedidos de execução de rotinas deixando

estes processos em um estado de espera. Tal estado de espera pode ser muito grande em face dos tempos do sistema, comprometendo seu desempenho. Existem rotinas que tardam mais em ser executadas e podem elas mesmo ser a causa dos atrasos. Assim se houvesse uma fila de pedidos de execução das rotinas isto poderia liberar os vários processadores mais rapidamente para que cuidassem da execução de seus processos.

4.2. Conclusões

O estado atual de desenvolvimento da interface e de suas rotinas é suficiente para as exigências do que se vislumbra como sendo as possíveis aplicações atuais, desde o ponto de vista dos desenvolvimentos realizados no IAI.

Um dos aspectos que podemos destacar é a velocidade de execução das rotinas que fazem desenhos e apagam a tela, onde se utilizam de recursos de DMA em sua implementação em linguagem OCCAM.

O problema da padronização gráfica, com relação ao fato de seguir padrões de núcleos gráficos já existentes, como por exemplo GKS ("graphical kernel system"), não se considerou preponderante para esta primeira fase por não se dispor de critérios concretos para sua adoção em sistemas paralelos. O desenvolvimento e utilização desta interface gráfica para sistemas paralelos poderia trazer a tona parâmetros que passassem despercebidos em sistemas gráficos puramente sequenciais. Estes parâmetros poderiam constituir os subsídios necessários para o estabelecimento de padrões próprios para os sistemas paralelos. Embora nao se tenha deparado com problemas que tornassem inviáveis a adoção dos padrões de núcleos gráficos convencionais, a sua não adoção contribuiu para diminuir o número de restrições iniciais de projeto.

Outro aspecto relevante é a questão da modularidade da interface, comprovada com o sucesso da utilização prática descrita no ítem 3.5 e com uma análise de sua estruturação interna. Tal modularidade permite a inserção de características adicionais com um mínimo de esforço de programação.

Uma maior auto suficiência com esta interface poderia ser obtida implementando, dentro de algum de seus módulos, ou ainda, em um módulo de nível superior, serviços de mais alto nível, como por exemplo um gerenciador de uma base de dados gráfica. Nesta primeira etapa optou-se por deixar de fora quaisquer características que embora mais poderosas pudessem de alguma forma comprometer a adaptabilidade a sistemas de aplicação genérica com Transputers. Um estudo mais detalhado desta portabilidade poderia gerar as informações necessárias para os desenvolvimentos posteriores.

BIBLIOGRAFIA

[ANDRADE-90]

M.T.C. ANDRADE, J.L. PEDRAZA, M.T. DE PEDRO, " Realização de Uma Interface Gráfica em Software Para Sistemas Paralelos Baseados em Transputers", Relatório Técnico Interno, Instituto de Automática Industrial, Consejo Superior de Investigaciones Científicas, Madrid, Espanha, julho de 1990.

[DENERT-73]

E. DENERT, " A Method for Computing Points of a Circles Using Only Integers ", Computer Graphics and Image Processing (1973) 2, pp 83-91, 1973.

[GHEE-]

S. GHEE, " IMS B004 IBM PC Adding Board ", Technical Note 11, INMOS.

[HOMEWOOD-87]

M. HOMEWOOD & D. MAY, " The IMS T800 Transputer ", IEEE Micro, pp 10-26, October 1987.

[INMOS-85]

" IMS B004 Evaluation Board ", INMOS, november, 1985.

[INMOS-86]

" Transputer Reference Manual ", INMOS, october, 1986.

[INMOS-86-2]

" The Transputer Family ", Product Information, INMOS, june, 1986.

[INMOS-87]

" IMS T800 Transputer ", Product Overview, INMOS, 1987.

[INMOS-87-2]

" IMS G170 High Performance CMOS Color Look-up Table ", Product Data Sheet, INMOS, march, 1987.

[INMOS-87-3]

" IMS T414 Transputer ", Preliminary Data, INMOS, February, 1987.

[INMOS-87-4]

" IMS T800 Architecture ", INMOS, Technical Note 6, Central Applications Group, Bristol, March, 1987.

[INMOS-88]

" Transputer Development System ", INMOS, Prentice Hall, 1988.

[INMOS-88-2]

" OCCAM2 Reference Manual ", INMOS, Prentice Hall, 1988.

[INMOS-88-3]

" IMS B008 User Guide and Reference Manual ", INMOS, Prentice Hall, 1988.

[INMOS-89]

" IMS B408 And IMS B409 User Manual ", INMOS, 1989.

[MATTOS-87]

P. MATTOS, " Program Design for Concurrent Systems ", Technical Note 5, INMOS, February, 1987.

[MAY-87]

D. MAY, R. SHEPHERD, " The Transputer Implementation of OCCAM ", Technical Note 21, INMOS, February, 1987.

[MAY-87-2]

D. MAY, " Comunicating Processes and OCCAM ", Technical Note 20, INMOS, February 1987.

[MAY-87-3]

D. MAY, C. KEANE, " Compiling OCCAM Into Silicon", Technical Note 23, INMOS, February 1987.

[McCONNEL-88]

R. McCONNEL, "A Transputer Based Distributed Graphics Display", Technical Note 46, INMOS, July 1988.

[PEDRAZA-87]

J. L. PEDRAZA D., " Arquitecturas Paralelas de Computador Especializadas en Procesos de Busqueda en Espacio de Estados ", Tesis Doctoral, Universidade de Salamanca, Facultad de Ciencias, 1987.

[PEDRAZA-88]

J. L. PEDRAZA, M. T. DE PEDRO, A. RIBEIRO, " A Parallel Architecture For PCB Routing ", 4th International Conference on Systems Research Informatics and Cybernetics, Baden-Baden, West Germany, August, 1988.

[DE PEDRO-88]

M. T. DE PEDRO, J. L. PEDRAZA, A. RIBEIRO, " A Parallel Architecture For Cooperative Heuristic Searches: Design Methodology And Application to the PCB Routing ", Mini and Microcomputers and Their Applications, San Feliu de Guixols, España, 1988.

[POUNTAIN-90-2]

D. POUNTAIN, " Virtual Channels : The Next Generation of Transputers", BYTE, PP E&W3-E&W10, April 1990.

[RYGOL-87]

M. RYGOL, T. WATSON, "Connecting INMOS Links", Technical Note 18, INMOS, April 1987.

[STEIN-88]

R. M. STEIN, " T800 And Counting ", BYTE, pp 287-296, November, 1988.

[TEK-87]

" An Introduction to Computer Color Graphics ", TEK Users Handbook, Information Display Group, Tektronix, 1987.

[VALERO-85]

M. VALERO, " Arquitectura Para los Computadores de Alta Velocidad ", VII Escuela de Verano de Informática, La Rábida, España, 15 a 26 de Julio, 1985.

[PITEWAY-67]

M. L. V. PITEWAY, " Algorithm for Drawing Elipses or Hiperbolae With a Digital Plotter ", Computer Journal, 10, 1967, pp 282-289, 1967.

[PITEWAY-74]

M. L. V. PITEWAY, " Integers Circles, Etc. - Some Further Thoughts ", Computer Graphics and Image Processing (1974) 3, pp 262-265, 1974.

[POUNTAIN-87]

D. POUNTAIN & D. MAY, " A Tutorial Introduction to OCCAM Programming ", Blackwell Scientific, 1987.

[POUNTAIN-89]

D. POUNTAIN, " OCCAM II ", BYTE, pp 279-284, October, 1989.

[POUNTAIN-89-2]

D. POUNTAIN, " Configuring Parallel Programs - Part 1", BYTE, pp 349-352, December 1989.

[POUNTAIN-90]

D. POUNTAIN, " Configuring Parallel Programs - Part 2", BYTE, pp 327-334, January 1990.

APÊNDICE A

PLACA DO SISTEMA DE DESENVOLVIMENTO DE PROGRAMAS OCCAM (IMS B004)

A.1. Introdução

Realiza-se aqui uma breve introdução às principais características da placa de desenvolvimento IMS B004.

A placa IMS B004 está logicamente dividida em três partes distintas :

1. O Transputer com seus links para comunicação e 1 ou 2 Mbytes de RAM.
2. A lógica de subsistema para o PC, a qual permite a um programa que é executado no PC reinicializar e controlar sistemas.
3. O IMS C002 (adaptador de links), que faz a "interface" com uma via de dados/endereços paralela, como o que há nos "slots" de um computador PC. O adaptador de "link" é acessado por um programa que é executado no PC, para transferir dados de/para o Transputer.

Há um detalhamento de seus blocos na Figura A.1 .

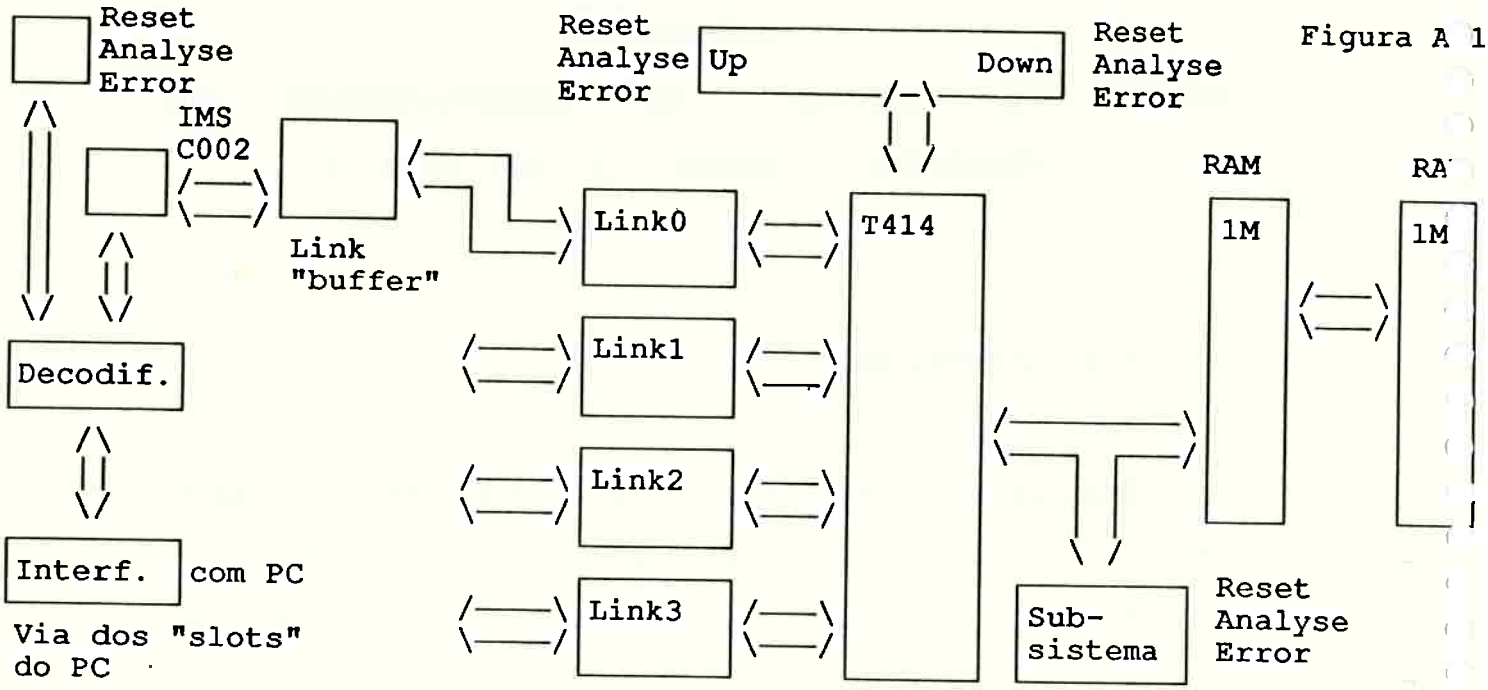


Figura A 1

As três partes da placa citadas acima são postas para trabalhar em conjunto por meio de "jumpers" de conexão fornecidos pelo fabricante. O "reset jumper" permite ao subsistema que faz a interface com o PC responder a endereçamentos provenientes do PC e conectar os sinais "reset", "analyse" e "error" aos sinais controlados pelo PC. O "jumper link" conecta o adaptador de link a um dos links do Transputer e permite ao adaptador de links responder aos endereçamemtos provenientes do PC.

A placa IMS B004 pode ser instalada em qualquer "slot" de expansão disponível no PC. Isto é feito do mesmo modo que se procede para instalar placas convencionais no PC.

A.2. Configuração por "Jumpers"

Antes que qualquer programa seja carregado na IMS B004, a partir do PC, é necessário que se coloquem dois "jumpers" que permitem que mais de uma placa IMS B004 esteja presente no PC, mas que apenas uma responda aos comandos do sistema de desenvolvimento de OCCAM (tds2). A placa que tem os "jumpers" colocados é denominada de "Master" (mestre) e qualquer número de placas adicionais pode ser interligado a esta por meio dos links físicos.

O "link jumper" é utilizado para interconectar o link do adaptador de links (o qual gerencia toda a comunicação com a via paralela do PC) com o link 0 do Transputer. O efeito final é que este "jumper" liga o adaptador de link ao "bus" paralelo do PC. Quando a placa é inicialmente configurada por meio deste "jumper", a comunicação para o PC é possível utilizando comunicação por canal OCCAM padrão no link 0 do Transputer.

O "reset jumper" é utilizado para conectar o PC aos sinais de serviços do sistema disponíveis no conector "Up" (Figura A.1). Quando a placa está configurada com este "jumper" é possível para o PC fornecer os sinais "reset" e "analyse" e receber o sinal "error" do conector "Up". Este "jumper" deve estar disponível na placa mestre e assim o tds2 será capaz de reinicializar

todo o sistema antes de carregar algum programa. Se o "jumper" não for utilizado a placa mestre ignorará qualquer comando de sistema vindo do PC.

Uma placa (a mestre), ou o computador hospedeiro (no caso o PC) podem controlar um número arbitrário de placas que por sua vez podem constituir subsistemas de placas. Um diagrama seria o da Figura A.2 para ilustrar esta situação.

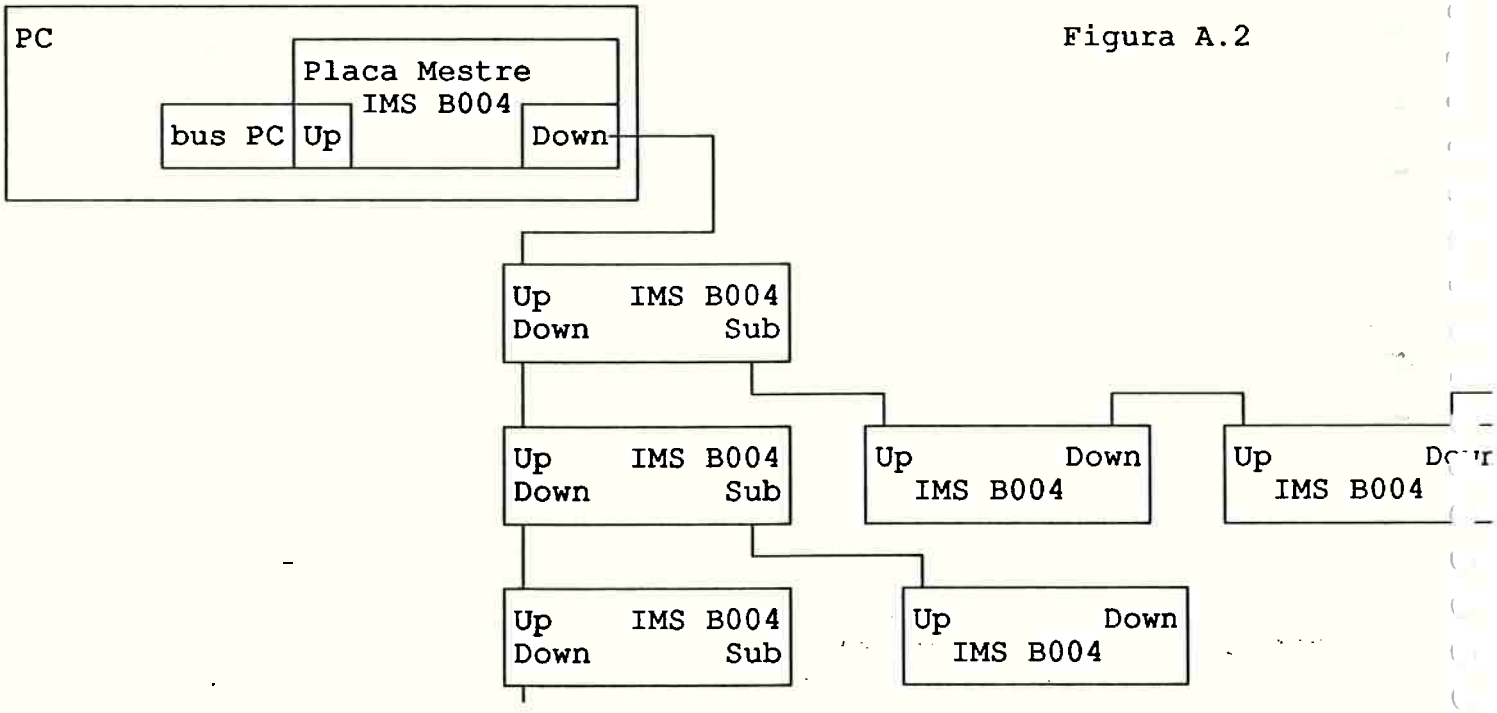


Figura A.2

Tais configurações são possíveis pela disponibilidade de três conectores denominados "Up", "Down" e "Subsystem". Placas adicionais são interligadas por meio de "jumpers" entre estes conectores (ver Figura A.2). Cada uma placas das placas em um subsistema pode por si mesma controlar uma série de outras placas configurando um outro subsistema. Uma

placa, no caso a placa mestre, pode ser controlada pelo PC a partir da execução do tds2 nesta máquina. As ações de controle são do tipo reinicializar ("reset") ou análise de erros e estes sinais de controle são "vistos" como posições de memória pelo PC.

A placa IMS B004 é controlada somente pelo PC e há alguns programas de teste que se pode executar desde este.

A.3. Espaço de Endereçamento e Interface com PC

O espaço de endereçamento é apenas parcialmente decodificado, usando A31 e A20. Se a placa possui 1 Mbyte de memória RAM os endereços são:

80000000 a 800007FF -- 2 Kbytes de RAM interna
80000800 a 800FFFFF -- 1 Mbyte de RAM externa

Para 2 Mbyte de memória teríamos:

80000000 a 800007FF -- 2 Kbytes de RAM interna
80000800 a 802007FF -- 2 Mbytes de RAM externa

A interface com o PC requer uma via paralela para compatibilizar as comunicações com a via interna ("slot bus") deste. Tal interface possibilitaria a comunicação e poderia ser implementada por dois métodos:

- mapear o "hardware" da via de comunicação na memória externa do Transputer.

- comunicar com o Transputer por meio de um dos links seriais.

O segundo método foi adotado pela INMOS, já que esta solução estaria de acordo com o conceito de comunicação via canais de OCCAM. Neste caso o PC terá o comportamento de um processo que está conectado ao Transputer por meio de um canal, e isto determina que o Transputer escolhido para a comunicação com o PC terá um de seus links utilizados exclusivamente para esta tarefa. Para materializar esta solução INMOS produziu alguns dispositivos periféricos entre eles o IMS C002 que converte dados paralelos em seriais e vice-versa. Além disto a utilização de uma PAL para a decodificação de endereço permite que o endereço da placa seja alterado pela simples troca da PAL. A placa é usualmente alocada no endereço hexadecimal "150" no espaço de endereçamento de I/O do PC.

APÊNDICE B

PLACA GRÁFICA PARA TRANSPUTERS (IMS B408/B409)

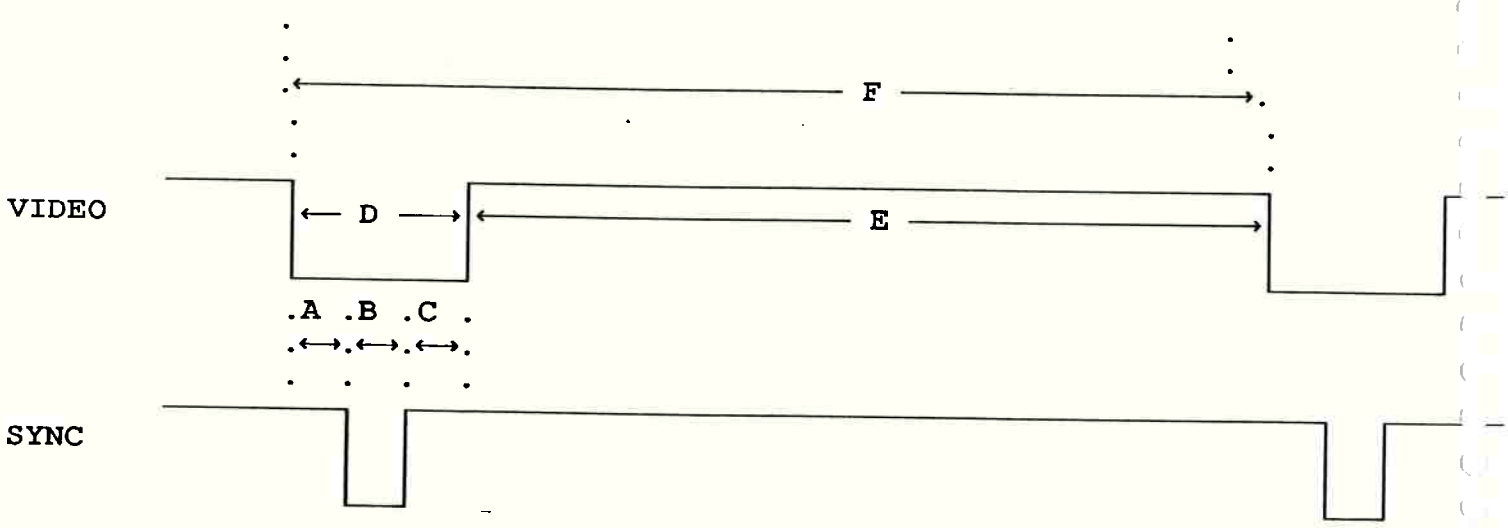
B.1. Introdução

A função desta placa gráfica é a de executar as rotinas gráficas gerando os sinais de vermelho, verde, azul e sincronismo composto ("R", "G", "B" e "SYNC") enviados ao monitor de alta resolução colorido. Para o monitor é necessário que isto ocorra mediante a obediência de uma carta de tempos fornecida por seu fabricante. Com respeito à temporização relativa ao sincronismo horizontal e ao sincronismo vertical pode-se fazer uso da mesma ilustração, dada pela Figura B.1 já que estes sinais teriam o mesmo aspecto com ordens de grandeza distintas.

Fazendo então uso da Figura B.1 teríamos uma série de definições de carta de tempos para os sincronismos horizontal e vertical. Para o sincronismo horizontal temos o tempo designado por "A" que é o tempo de pórtico frontal ("horizontal front-porch" ou HFP) ou o tempo decorrido entre o término do sinal de sincronismo horizontal e o início de uma linha ativa. O tempo designado por "B" é o tempo de duração do sinal de sincronismo horizontal ("horizontal sync-width" ou HS). O tempo designado por "C" seria o tempo de pórtico traseiro ("horizontal back-porch" ou HBP) ou o tempo

decorrido entre o término de uma linha ativa e o início do próximo sinal de sincronismo horizontal. O tempo designado por "D" é o tempo decorrido entre entre duas linhas horizontais ativas consecutivas. O tempo designado por "E" é o tempo de duração de uma linha horizontal ativa ("horizontal active line time" ou AW). O tempo designado por "F" é o tempo de duração total de uma linha horizontal ("horizontal line time").

Figura B.1



Ainda fazendo uso da Figura B.1 teríamos uma série de definições de carta de tempos também para o sincronismo vertical. Temos assim o tempo designado por "A" que é o tempo de pórtico frontal ("vertical front-porch" ou VFP) ou o tempo decorrido entre o término do sinal de sincronismo vertical e o início das linhas verticais ativas (ou um campo de imagem). Sabe-se da teoria de televisões que a imagem em um monitor de televisão é reproduzida por campos subsequentes cuja

superposição é integrada pelo olho humano constituindo uma imagem contínua. Cada campo por sua vez é constituido por um grupo de linhas ativas durante a varredura vertical. O tempo designado por "B" é o tempo de duração do sinal de sincronismo vertical ("vertical sync-width" ou VS). O tempo designado por "C" seria o tempo de pórtico traseiro ("vertical back-porch" ou VBP) ou o tempo decorrido entre o término das linhas ativas (ou o término de um campo) e o início do próximo sinal de sincronismo vertical. O tempo designado por "D" é o tempo decorrido entre entre dois grupos de linhas ativas verticais consecutivas (ou o tempo decorrido entre a apresentação de dois campos subsequentes). O tempo designado por "E" é o tempo de duração de um grupo de linhas ativas verticais ("vertical active lines time" ou AL). O tempo designado por "F" é o tempo de duração total de um grupo de linhas ativas verticais ou um campo ("vertical line time" ou "field time").

As IMS B408 e IMS B409 são dois módulos de Transputers construidos para serem utilizados em conjunto e constituir um sistema gráfico (ver Figura B.2) para computadores distribuidos, ou seja, sistemas onde a tarefa de desenhar é distribuida entre vários processadores. Um sistema completo consiste em um módulo IMS B409 e um ou mais módulos IMS B408.

Neste texto será utilizado o termo em ingles "pixel" para designar a menor unidade de desenho (ou cada ponto que se pode desenhar).

O IMS B408 e o IMS B409 são interligados por meio de uma via de pixels ("pixel bus"). O IMS B409 gera a carta de tempos do sistema e converte pixels digitais em sinais RGB analógicos disponíveis para suportar um monitar de video colorido.

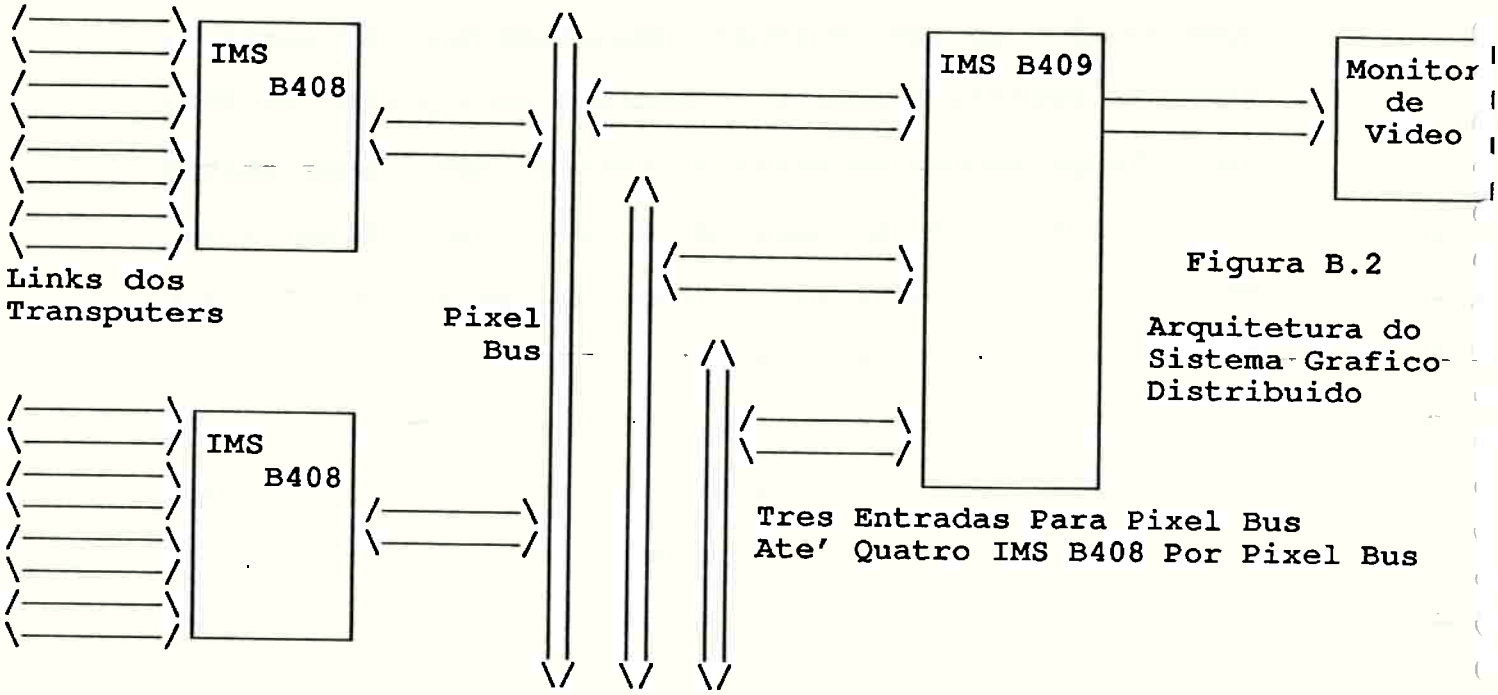


Figura B.2

Arquitetura do Sistema Grafico Distribuido

Cada IMS B408 tem um Transputer com facilidades de operação em ponto flutuante (IMS T800) e um bloco de memória "dual port". O IMS T800 desenha na memória via um "port" e o outro "port" é conectado no "pixel bus". O "pixel bus" distribue os sinais de temporização do IMS B409 para o IMS B408 e combina a saída de dados do IMS B408.

O sistema gráfico é projetado em torno do "pixel bus" que é uma via de 32 bits capaz de movimentar dados a uma taxa de 100 MBytes/seg e também distribue os sinais de temporização do sistema. Um sistema gráfico

genérico contem um módulo de controle de tela (IMS B409) que possue três canais de pixels os quais recebem uma sequência de pixels de um "pixel bus" e geram os sinais RGB de saída para suportar monitores de video coloridos. Ele gera os sinais de sincronismo necessários ao funcionamento do monitor e também aqueles que são distribuidos pelo "pixel bus". O IMS B409 pode trabalhar em taxas de até 64 MHz. Tal sistema genérico pode dispor tambem de um ou mais módulos IMS B408. Os módulos IMS B408 contem, cada um, um Transputer IMS T800, processador de 32 bits com unidade de ponto flutuante interna, e 1,25 M Bytes de RAM "dual port" conectada ao "pixel bus". Isto permite ao Transputer escrever na RAM (que equivale a desenhar na tela) ao mesmo tempo que dados fluem da "dual port" para o "pixel bus".

O "pixel bus" é uma via síncrona o qual distribue temporização de sistema e transporta dados de 32 bits (dados relativos à constituição de um pixel) em velocidades de até 25 MHz.

B.2. Módulo IMS B408

O módulo IMS B408 implementa o armazenamento dos desenhos realizados e a memória de execução das rotinas gráficas utilizadas. Incorpora para a realização desta tarefa um processador de 32 bits (IMS T800) que tem unidade de ponto flutuante interna, 1 M Byte de RAM de trabalho e 1,25 M Byte de RAM de tela acessável pelo

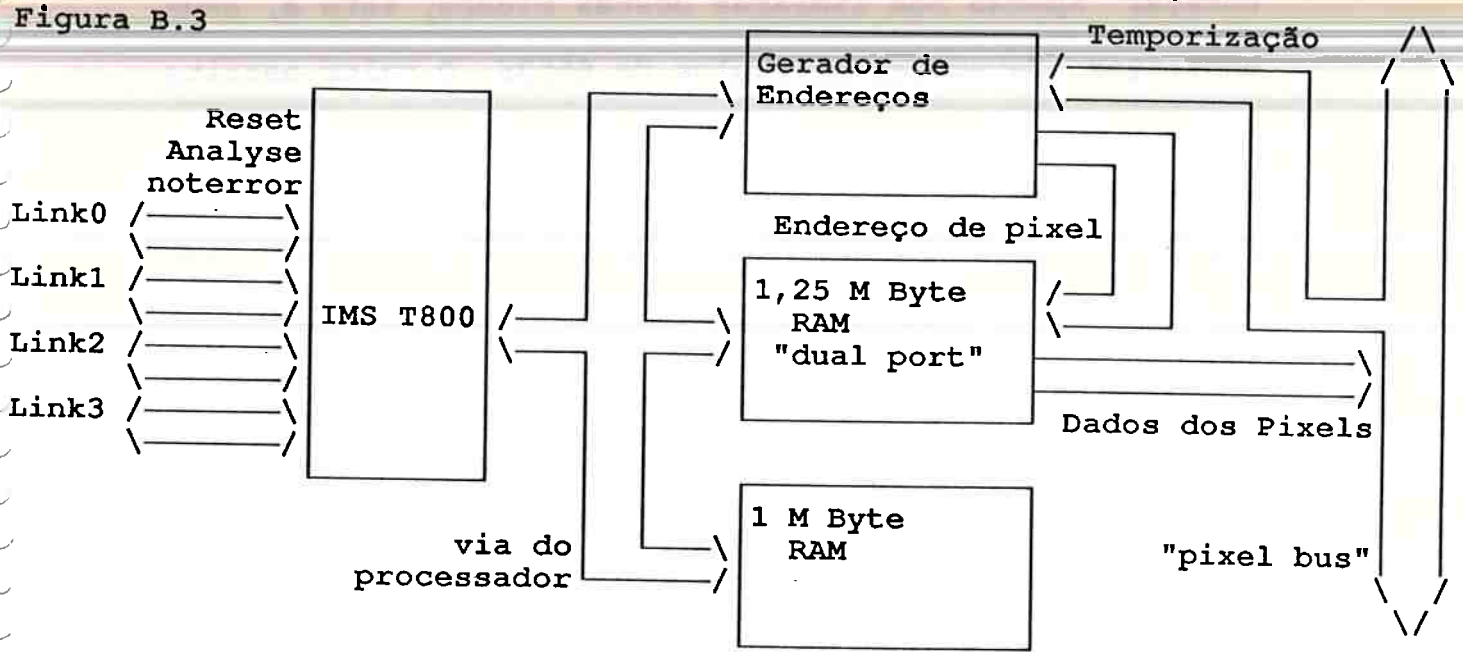
processador e com um dos "ports" (recordar que ela é "dual port") ligado ao "pixel bus". Este "port" conectado ao "pixel bus" é capaz de realizar transmissão de dados a uma velocidade de até 100 M Bytes/seg independentemente do processador sob o controle de um gerador de endereços autônomo. O IMS B408 suporta os modos de vídeo entrelaçado e não entrelaçado de resoluções arbitrárias de até 1024 x 768 pixels. A resoluções mais baixas pode-se ter múltiplos "buffers" de tela, por exemplo 4 "buffers" de tela de 640 x 480 pixels. Este módulo se encontra esquematizado na Figura B.2 em diagrama de blocos.

O IMS B408 é projetado para ser usado com o IMS B409, e quando são usados em conjunto constituem um sistema completo de desenho. Um aumento da performance do sistema é obtido adicionando mais módulos IMS B408. O módulo IMS B408 realiza a função de desenhar propriamente dita e a imagem é armazenada na memória de tela de capacidade de 1,25 M Byte ("dual port"). Uma memória adicional de 1 M Byte para o armazenamento de programas e dados também está disponível. Os dois "ports" da memória de tela são: o "port" que está conectado à via do processador e o "port" que está conectado ao "pixel bus" (denominado de "pixel port"). Os dados de imagem fluem através do "pixel port" sob o controle do gerador de endereços (ver Figura B.3).

O "pixel port" do IMS B408 possibilita sua conexão ao "pixel bus" que por sua vez estará conectado a um IMS

B409 e possivelmente a outros IMS B408.

Figura B.3



Existem alguns registradores de controle associados com o "pixel port" e o gerador de endereços, mencionados abaixo:

REGISTRADOR	LEITURA/ESCRITA	ENDEREçO HEXA
Display Start	escrita	#00000000
Ready	escrita/leitura	#00040000
SysReady	leitura	#00080000
Interlace Enable	escrita/leitura	#000C0000
Event Mode	escrita/leitura	#00100000
Output Enable	escrita/leitura	#00140000

- Display Start - este registrador contem o endereço do pixel mais acima a esquerda da tela gráfica. A memória "dual port" pode ser pensada como sendo dividida por 20

blocos de 64 KBytes. Armazenadores de desenhos podem começar apenas nos limiares destes blocos, isto é, seus endereços devem ser múltiplos de 65536. O valor escrito neste registrador é um "offset" para ser usado em conjunto com uma base e gerar o endereço absoluto, isto é :

(endereço - #80100000)/4

- Interlace Enable - seleciona uma varredura de tela em modo entrelaçado (escrever 1) ou modo não entrelaçado (escrever 0). Os 31 bits mais significativos deste registrador são ignorados.

- Event Mode - seleciona a fonte da interrupção "EVENT" como sendo o surgimento do sinal "FieldSync" (Event Mode = 1) ou o surgimento dos sinais "FieldSync" e "SysReady" ao mesmo tempo (Event Mode = 0). Os 31 bits mais significativos deste registrador são ignorados.

- Output Enable - Habilita (escrever 1) e desabilita (escrever 0) a saída do "pixel port". Os 31 bits mais significativos deste registrador são ignorados.

- Ready - escrever 0 força o sinal "SysReady" para "low" e escrever 1 permite que "SysReady" vá para "high". Quando lido, os 31 bits mais significativos deste registrador são ignorados.

- SysReady - é um endereço apenas de leitura que reflete as condições deste sinal que está presente no "pixel port". Quando lido, os 31 bits mais significativos deste registrador são ignorados.

Estão previstos 2304 KBytes de memória. Isto consiste de 4 KBytes de memória interna do Transputer e 2300 KBytes de memória RAM dinâmica externa. Os 1280 KBytes superiores compõe a RAM "dual port" ligada ao "pixel port". Os restantes 1024 KBytes inferiores são usados para armazenamento de programas e dados. O mapeamento se encontra abaixo:

RAM interna do T800	#80000000 - #80000FFF
RAM externa do T800	#80001000 - #800FFFFF
RAM "dual port"	#80100000 - #8023FFFF

Os dados de pixel são mapeados em memória de tal modo que o pixel imediatamente a direita do pixel de endereço "m" terá endereço "m+1" e o pixel imediatamente abaixo deste terá endereço "m+w" onde w é o número total de pixels em uma linha horizontal da tela. Assim definindo um "array" unidimensional e alocando em memória por meio de aumentos em seu indexador pode-se desenhar na tela da esquerda para a direita e de cima

para baixo. Com um "array" bidimensional resulta do modo como OCCAM está feito que incrementos no primeiro índice resultarão em movimentos de cima para baixo e no segundo índice da esquerda para a direita, ou seja, o "array" deve ser definido e dimensionado como abaixo:

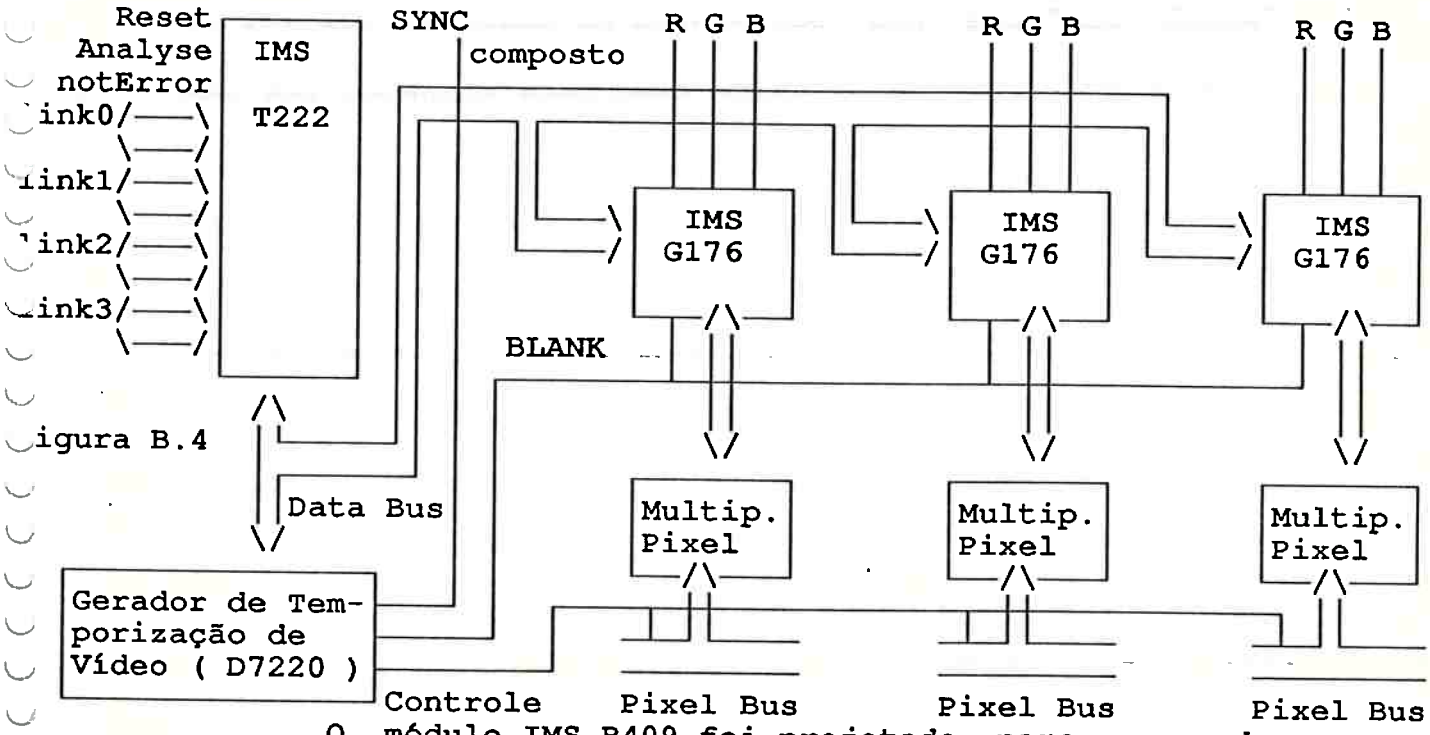
```
tela [altura] [largura]
```

Cada pixel ocupa uma palavra de 32 bits, onde o byte menos significativo é o vermelho, o seguinte é o verde e depois o azul. O byte mais significativo não é utilizado e pode ter qualquer conteúdo. As palavras são armazenadas em memória tendo os bytes menos significativos endereços menores. O endereço de uma palavra é o endereço do byte menos significativo e é sempre um múltiplo de 4.

B.3. Módulo IMS B409

O IMS B409 (ver Figura B.4) implementa a geração de temporização e a definição do "palette" de cores acompanhado de "drivers" para os monitores. Ele consiste de três "pixel bus", chamados de canais de pixels, cada um com um multiplexador e um periférico para definição do "palette" e geração dos sinais RGB (IMS G170 "colour look-up table"), e um periférico programável para geração de temporização de video (NEC D7220 "Graphics Display Controller"). A entrada para cada canal de

pixel é feita por um "pixel bus" separado e cada canal gera um conjunto de saídas RGB. O módulo IMS B409 suporta ambos os modos entrelaçado e não entrelaçado com resolução arbitrário e uma taxa de pixel de até 64 MHz.



igura B.4

O módulo IMS B409 foi projetado para ser usado em conjunto com um ou mais módulos IMS B408 que compõe juntos um sistemas gráfico completo. A performance do sistema pode crescer com a adição de mais módulos IMS B408. A conexão entre eles é permitida pela existencia no IMS B409 de três canais de pixel compatíveis com a relação de sinais de um "pixel bus" do IMS B408. Os três canais são denominados de A, B e C e consistem de um multiplexador 4-1 bytes e um IMS G176 ("colour look-up table" - CLUT). A entrada para cada canal é feita através do conector para o "pixel bus" e a saída é o conjunto dos sinais RGB e sincronismo composto.

Há dois modos de operação para o número de bits por pixel. Cada um dos canais aceita 32 bits de dados vindos do "pixel bus" e em um modo são multiplexados para constituir uma palavra de 8 bits a qual é fornecida ao IMS G176 para gerar um dos sinais de cor. Desta forma o "pixel bus" tem uma velocidade de operação de 1/4 da taxa máxima. Cada canal neste caso pode fornecer até 256 cores por imagem e não é necessário usar os três canais e desde que eles são sincronizados pode-se usar um para cada sinal de cor (um canal para R, outro para G e o terceiro para B). No outro modo o IMS B409 fornece uma imagem de 262.144 cores e há a passagem de uma palavra de pixel de 32 bits diretamente para o canal A. Neste caso as entradas dos canais B e C não podem ser usadas, mas eles recebem a carga diretamente do canal A. Assim o byte menos significativo da cadeia de 32 bits é dirigido para o canal A, o segundo menos significativo é roteado diretamente para o canal B e o terceiro menos significativo é roteado diretamente para o canal C. Cada IMS G176 está provido com um conversor digital-analógico de 6 bits, deste modo apenas os seis primeiros bits de cada byte de informação de cor são utilizados desprezando-se os outros dois bits finais. Com isto a saída de vídeo do canal A é controlada pelo byte 0 da palavra de pixel, a saída de vídeo do canal B é controlada pelo byte 1 da palavra de pixel, a saída de vídeo do canal C é controlada pelo byte 2 da palavra de pixel, sendo uma única palavra de entrada a

especificação completa das cores R, G e B.

Há um registrador para escolher o modo do canal de pixel ("pixel channel mode select register") onde se escreve 1 para palavra multiplexada (8 bits) e 0 para palavra não-multiplexada (32 bits), este registrador é de escrita apenas e sua posição de memória é dada pela Tabela B.1 onde também estão outros endereços de interesse.

Tabela B.1

REGISTRADOR	ENDEREçO
"Channel Mode Selet"	#B000
"Channel A pixel address" (modo escrita)	#0000
"Channel A colour value"	#0400
"Channel A pixel mask"	#0800
"Channel A pixel address" (modo leitura)	#0C00
"Channel B pixel address" (modo escrita)	#1000
"Channel B colour value"	#1400
"Channel B pixel mask"	#1800
"Channel B pixel address" (modo leitura)	#1C00
"Channel C pixel address" (modo escrita)	#2000
"Channel C colour value"	#2400
"Channel C pixel mask"	#2800
"Channel C pixel address" (modo leitura)	#2C00

Há três IMS G176 ("color look-up table", CLUT) no módulo IMS B409, cada um interligado a um canal de pixel. Cada um destes dispositivos dispõe de uma RAM interna de 256 palavras de 18 bits e três conversores digital-analógico de 6 bits. Estes 6 bits do dado em operação são aplicados aos conversores gerando os sinais R, G e B podendo formar um conjunto de até 256 cores selecionáveis de um conjunto de 262.144 cores. Acessos isolados aos CLUTs podem ser feitos pelos endereços da Tabela B.1 mas na Tabela B.2 encontram-se os endereços para acesso por meio de movimentação direta de blocos.

Tabela B.2

REGISTRADOR	ENDEREÇO
"Channel A colour value"	#4400 a #47FF
"Channel A colour value"	#5400 a #57FF
"Channel A colour value"	#6400 a #67FF

O conteúdo do "pixel mask register" é utilizado para fazer uma operação "AND" com o dado de pixel em uso e é normalmente configurado para #FF. Ele pode ser usado para criar objetos que piscam ou animação simples.

Cada CLUT tem um único "pixel address register" mas este é acessado desde duas posições de memória e com

dois modos distintos. Escrever no "pixel address register" mas usando o modo escrita (ver Tabela B.2) significa que uma cor vai ser associada a esta posição na tabela de cores. Escrever no "pixel address register" mas usando o modo leitura (ver Tabela B.2) significa que uma cor vai lida lida desta posição. Uma leitura desde qualquer um dos modos retorna o mesmo resultado, ou seja, o conteúdo do "pixel address register".

A geração de temporização de video é feita por um NEC D7220 que está mapeado no espaço de memória do IMS T222, ou seja, seus registradores de comando, de parâmetros ("FIFOs" de comando e parâmetros) e estado estão mapeados no espaço de memória do Transputer e na Tabela B.3 se apresenta tal endereçamento:

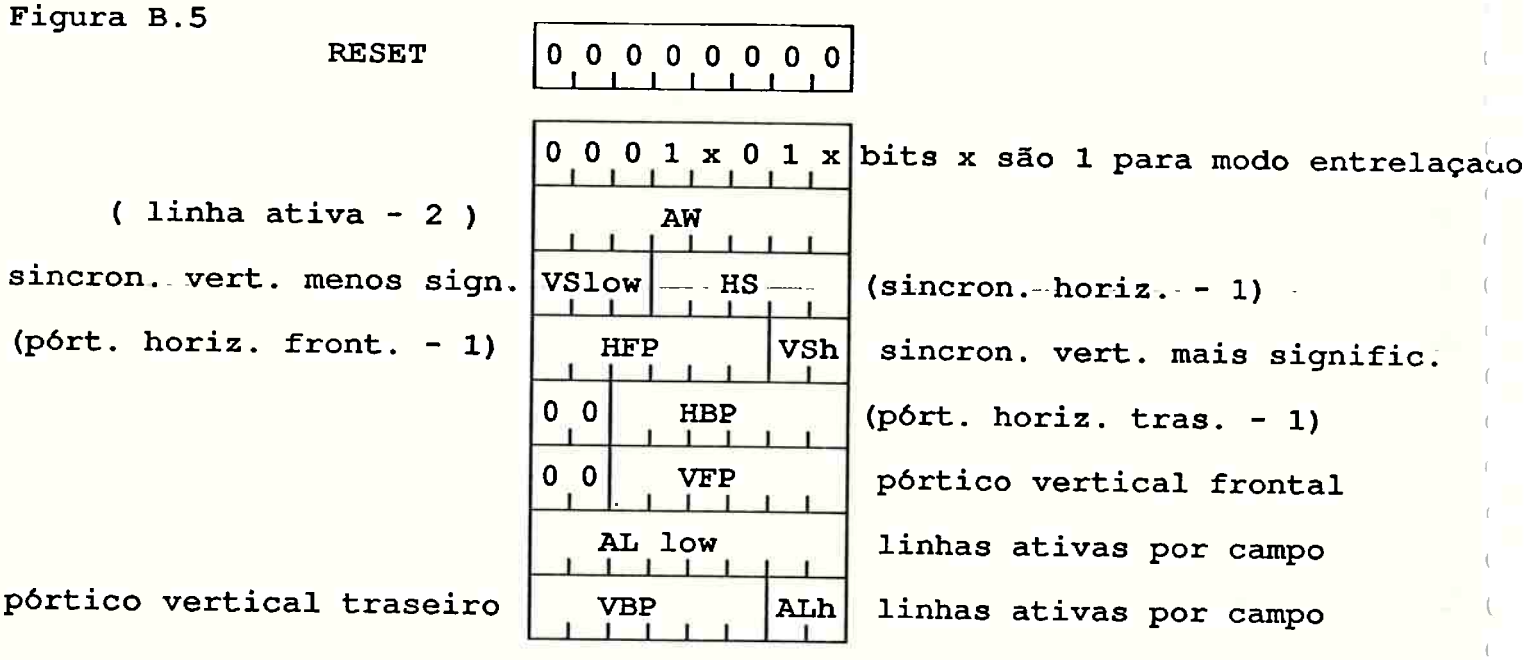
Tabela B.3

REGISTRADOR	ESCRITA/LEITURA	ENDEREçO
"parameter FIFO"	apenas escrita	#A000
"status register"	apenas leitura	#A000
"command FIFO"	apenas escrita	#A002
"FIFO read"	leitura apenas	#A002

Ele é utilizado apenas como um gerador de temporização programável e não tem funções de desenho. Frequência de linha, de campo e resolução podem ser programadas. A programação é feita por meio de um

comando de "reset" (ver Figura B.5). Escreve-se um comando de "reset" no endereço #A002 e isto prepara a "FIFO" de parâmetros para receber 8 parâmetros em seguida que são os estipulados pela Figura B.5 e constituem a programação de temporização.

Figura B.5



Existe um comando ("BCTRL") que permite o reaparecimento na tela da imagem armazenada na memória de tela antes de um comando "reset". Após o comando "reset" a tela pode ser restaurada por meio deste comando que é ativado escrevendo #0D na "FIFO" de comando.

APÊNDICE C

LISTAGEM DOS PROGRAMAS

```
  --{{{  De n Hasta z
  VAL  t.n              IS   INT('n')     :
  VAL  t.otras          IS   INT('o')     :
  VAL  t.p              IS   INT('p')     :
  VAL  t.q              IS   INT('q')     :
  VAL  t.rectang        IS   INT('r')     :
  VAL  t.s              IS   INT('s')     :
  VAL  t.t              IS   INT('t')     :
  VAL  t.circulares     IS   INT('u')     :
  VAL  t.ventanas       IS   INT('v')     :
  VAL  t.w              IS   INT('w')     :
  VAL  t.x              IS   INT('x')     :
  VAL  t.y              IS   INT('y')     :
  VAL  t.z              IS   INT('z')     :
  --}}}
  --{{{  De 0 Hasta 9
  VAL  t.cero           IS   INT('0')     :
  VAL  t.uno            IS   INT('1')     :
  VAL  t.dos            IS   INT('2')     :
  VAL  t.tres           IS   INT('3')     :
  VAL  t.cuatro         IS   INT('4')     :
  VAL  t.cinco          IS   INT('5')     :
  VAL  t.seis           IS   INT('6')     :
  VAL  t.siete          IS   INT('7')     :
  VAL  t.ocho           IS   INT('8')     :
  VAL  t.nueve          IS   INT('9')     :
  --}}}
  --{{{  De A Hasta M
  VAL  t.A              IS   INT('A')     :
  VAL  t.B              IS   INT('B')     :
  --}}}
  --}}}
  --}}}
  --
  --{{{  Declaracion de Variables y sus Direciones
  --
  --{{{  Variables
  --
  --{{{  Variables Enteras
  --{{{  Para Caracteres
  INT longitud.fuente, num.carac, fuente.x, fuente.y :
  INT x.mensaje, y.mensaje :
  INT ancho.esc, alto.esc :
  --}}}
  INT16 ayu16 :
  INT32 ayu32, semilla.ayu :
  INT64 ayu64 :
  INT a.ver, paleta, paleta.ant, sup, cuadrante :
  INT numero.ent, tamano, ayu.t :
  INT caracter, color.elegido.ent, cont :
  INT delta.x, delta.y :
  INT habilita.entrelazado, habilita.salida :
  INT inicio.pantalla, modo.evento :
  INT n, numero.bytes.dibujo :
  INT opcion :
  INT origen, destino, longitud :
  INT pronto, sistema.pronto :
  INT tipo :
  INT xf.ent, xi.ent :
  INT yf.ent, yi.ent :
  INT x.maximo.ent, y.maximo.ent :
  INT x.maximo.ant.ent, y.maximo.ant.ent :
  INT x.minimo.ent, y.minimo.ent :
  --}}}
  --
  --{{{  Variables Logicas
```

```
**Listed On  28-5-90 17:5  **
**List of Fold**       "V 3.6 Aplicacion Grafica"
**List all lines  with Fold Headers
**Excluding : NO LIST folds
--{{{  PROGRAM Grafic00 - Operacion Sobre las Tarjetas B408 y B409
--:::A  3 11
--{{{F Configuracion
--:::F Config27.tsr
--{{{  Definiciones de la configuracion
--
--{{{  Definicion de canales fisicos
VAL link.e IS [4, 5, 6, 7]:
VAL link.s IS [0, 1, 2, 3]:
--}}}
--
--{{{  Definicion de canales (Organizador, MemoriaPantalla y ControlePantalla)
--
CHAN OF ANY Org.Mp, Mp.Org, Mp.Cp, Cp.Mp , Cp.Org, Org.Cp :
--
--}}}
--
--}}}
--{{{  Procedimientos
--{{{  SC Operaciones Graficas - Memoria de la Pantalla en el B408
--:::A  3 10
--{{{F MemoriaPantalla
--:::F Memori27.tsr
PROC MemoriaPantalla ( CHAN OF ANY Org.Mp, Mp.Org, Mp.Cp, Cp.Mp )
  --{{{  Procesos
  --
  -- -------- Implementacion de Rutinas Graficas --------
  --
  --{{{  Declaracion de Librerias
  #USE snglmath
  #USE mathhdr
  --}}}
  --
  --{{{  Declaracion de Constantes
  --{{{  Parametros
  VAL ancho             IS                              1024    :
  VAL alto              IS                               768    :
  VAL numero.bytes      IS                     ancho * alto     :
  VAL ancho.r           IS                REAL32 ROUND(ancho)   :
  VAL alto.r            IS                 REAL32 ROUND(alto)   :
  VAL esc.comp          IS                   ancho.r / alto.r   :
  VAL ancho.pixel       IS                             ancho - 1 :
  VAL alto.pixel        IS                              alto - 1 :
  VAL ancho.pixel.r     IS          REAL32 ROUND(ancho.pixel)   :
  VAL alto.pixel.r      IS           REAL32 ROUND(alto.pixel)   :
  --}}}
  --{{{  Para llamada de rutinas graficas
  --{{{  De a Hasta m
  VAL  t.a              IS   INT('a')     :
  VAL  t.borrar         IS   INT('b')     :
  VAL  t.color          IS   INT('c')     :
  VAL  t.d              IS   INT('d')     :
  VAL  t.e              IS   INT('e')     :
  VAL  t.fin            IS   INT('f')     :
  VAL  t.g              IS   INT('g')     :
  VAL  t.h              IS   INT('h')     :
  VAL  t.iniciar        IS   INT('i')     :
  VAL  t.ejemplos       IS   INT('j')     :
  VAL  t.k              IS   INT('k')     :
  VAL  t.linea          IS   INT('l')     :
  VAL  t.m              IS   INT('m')     :
  --}}}
```

```
    IF
      ( x.max.int.ent = 0 ) OR ( y.max.int.ent = 0 )
        SEQ
          x.max.int := REAL32 ROUND(1023)
          y.max.int := REAL32 ROUND(767)
      TRUE
        SKIP
    --}}}
    IF
      ( x.max.int/y.max.int ) >= esc.comp
        SEQ
          --{{{  Procesos
          escala := ancho.pixel.r / x.max.int
          x.maximo.ent := ancho.pixel
          y.maximo.ent := INT ROUND( y.max.int *.escala )
          --}}}
      TRUE
        SEQ
          --{{{  Procesos
          escala := alto.pixel.r / y.max.int
          y.maximo.ent := alto.pixel
          x.maximo.ent := INT ROUND( x.max.int*escala )
          --}}}
    numero.bytes.dibujo := x.maximo.ent * y.maximo.ent
    --{{{  Limite Minimo de x/y
    x.minimo.ent  := 0
    y.minimo.ent  := 0
    --}}}
  --}}}
:
--}}}
--
--{{{  Trabajar con Vector de Bytes
PROC IniciaVectorBytes ( VAL INT px, VAL INT py,
                         VAL INT cuanto, VAL BYTE color.inic )
  --{{{  Procesos
  [(ancho*alto)]BYTE vec.pant RETYPES pantalla :
  INT cuanto.int, origen, destino, y.arregla, y.arregla1 :
  INT px.ref, py.ref :
  SEQ
    --{{{  Asignaciones
    px.ref := px
    py.ref := py
    cuanto.int := 1
    --{{{  y.arregla
    y.arregla  := alto.pixel - py.ref
    y.arregla1 := y.arregla * ancho
    --}}}
    origen :=  y.arregla1  + px
    destino := origen + 1
    vec.pant[origen] := color.inic
    --}}}
    WHILE   cuanto  >= ( 2 * cuanto.int )
      --{{{  Procesos
      SEQ
        [ vec.pant FROM destino FOR cuanto.int ] :=
                        [ vec.pant FROM origen FOR cuanto.int ]
        destino := destino + cuanto.int
        cuanto.int := cuanto.int + cuanto.int
      --}}}
    --{{{  Inicia los que Quedan
    cuanto.int :=  cuanto - cuanto.int
    [ vec.pant FROM destino FOR cuanto.int ] :=
                    [ vec.pant FROM origen FOR cuanto.int ]
    --}}}
  --}}}
```

```
  BOOL negativo :
  BOOL activo :
  BOOL opcion.activa :
  BOOL opcion.de.opcion.activa :
  BOOL paralelo.x :
  --}}}
  --
  --{{{  Variables Reales
  REAL32 n.r, numero.real, angulo :
  REAL32 ancho.int.r :
  REAL32 escala, ayu.real, pend.ret :
  REAL32 r, r.int, r.ext :
  REAL32 xf, xi, x.maximo, xn, xo :
  REAL32 yf, yi, y.maximo, yn, yo :
  --}}}
  --
  --{{{  Variables Tipo Byte
  BYTE color :
  BYTE color1, color2 :
  BYTE color.elegido :
  --}}}
  --
  --{{{  Variables Tipo Vector o Array de Enteros
  [20]INT trace :
  [80]INT cadena.mensaje.ent :
  [100]INT parametros :
  [256][12]INT fuente.elegido.ent :
  --}}}
  --
  --{{{  Variables Tipo Vector o Array de Bytes
  [alto][ancho]BYTE pantalla :
  [80]BYTE cadena.mensaje.by :
  [256][12]BYTE fuente.elegido.by :
  [12]BYTE fuente.8x12 :
  --}}}
  --
--}}}
--
--{{{  Direciones
PLACE   habilita.salida        AT (#00140000/4) + #20000000:
PLACE   modo.evento            AT (#00100000/4) + #20000000:
PLACE   habilita.entrelazado   AT (#000C0000/4) + #20000000:
PLACE   sistema.pronto         AT (#00080000/4) + #20000000:
PLACE   pronto                 AT (#00040000/4) + #20000000:
PLACE   inicio.pantalla        AT (#00000000/4) + #20000000:
PLACE   pantalla               AT (#80100000/4) + #20000000:
--}}}
--
--}}}
--
--{{{  Declaracion de Procedimientos Graficos
--
--{{{F Iniciales
--:::F inicgr03.tar
--
--{{{  Calcular Escala
PROC Escala ( VAL REAL32 x, VAL REAL32 y )
  --{{{  Procesos
  REAL32 x.max.int, y.max.int :
  INT x.max.int.ent, y.max.int.ent :
  SEQ
    x.max.int     := x
    y.max.int     := y
    x.max.int.ent := INT ROUND(x.max.int)
    y.max.int.ent := INT ROUND(y.max.int)
    --{{{  Limite Maximo x/y
```

```
            pyi := ayu
            IF
              pyi < y.minimo.ent
                SEQ
                  ObtenerXdeRecta ( pxi, pyi, pendiente, y.minimo.ent, ayu )
                  pxi := ayu
                  pyi := y.minimo.ent

      --}}}

      --}}}
    :
    --}}}
    --
    --{{{  Encontra Limite Superior de Recta
    PROC LimiteSuperiorRecta ( INT pxi, INT pyi , INT pxf, INT pyf )
      --{{{  Declaraciones
      REAL32 delta.x.real, delta.y.real, pendiente :
      INT delta.x.ent, delta.y.ent, ayu :
      BOOL fora :
      --}}}
      SEQ
        --{{{  Proc
        --{{{  Arreglar xi,yi,xf,yf
        IF
          ( pxi > pxf )
            SEQ
              --{{{  Proc
              ayu  :=  pxi
              pxi  :=  pxf
              pxf  :=  ayu
              ayu  :=  pyi
              pyi  :=  pyf
              pyf  :=  ayu
              --}}}
          TRUE
            SKIP
        --}}}
        --{{{  Asignaciones
        delta.x.ent     :=  pxf - pxi
        delta.y.ent     :=  pyf - pyi
        delta.x.real    :=  REAL32 ROUND(delta.x.ent)
        delta.y.real    :=  REAL32 ROUND(delta.y.ent)
        pendiente       :=  delta.y.real / delta.x.real
        fora            :=  FALSE
        --}}}
        --{{{  Verifica x.min
        IF
          pxi < x.minimo.ent
            SEQ
              ObtenerYdeRecta ( pxi, pyi, pendiente, x.minimo.ent, ayu )
              pxi := x.minimo.ent
              pyi := ayu
              IF
                pyi < y.minimo.ent
                  SEQ
                    ObtenerXdeRecta ( pxi, pyi, pendiente, y.minimo.ent, ayu )
                    pxi := ayu
                    pyi := y.minimo.ent

        --}}}

        --}}}
    :
    --}}}
    --
```

```
    --}}}
    --
    --{{{  Almacenar Puntos de Segmentos
    --
    --{{{  Almacena Puntos Recta Paralela a Eje
    PROC AlmacenaPuntosSegmentoParaleloEje ( VAL INT delta, VAL INT coord.i,
                                             VAL INT coord.f, VAL INT coord.fija,
                                             VAL BOOL paralela.x, INT tamano,
                                             []INT contorno.x, []INT contorno.y )
      --{{{  Declaraciones
      INT coord.f.int, coord.i.int, cuanto :
      --}}}
      SEQ
        --{{{  Proc
        --{{{  Ordena Coordenadas
        IF
          delta < 0
            SEQ
              coord.f.int := coord.i
              coord.i.int := coord.f
          TRUE
            SEQ
              coord.f.int := coord.f
              coord.i.int := coord.i
        --}}}
        --{{{  Eje x o y
        tamano  := 0
        IF
          paralela.x = FALSE
            --{{{  Procesos
            SEQ
              IF
                ( coord.fija >= x.minimo.ent ) AND ( coord.fija <= x.maximo.ent )
                  --{{{  Procesos
                  SEQ
                    IF
                      coord.i.int < y.minimo.ent
                        coord.i.int := y.minimo.ent
                      TRUE
                        SKIP
                    IF
                      coord.f.int > y.maximo.ent
                        coord.f.int := y.maximo.ent
                      TRUE
                        SKIP
                    WHILE coord.i.int <= coord.f.int
                      --{{{  Procesos
                      SEQ
                        contorno.y[tamano] := coord.i.int
                        contorno.x[tamano] := coord.fija
                        coord.i.int        := coord.i.int + 1
                        tamano             := tamano + 1
                      --}}}
                  --}}}
                TRUE
                  SKIP
            --}}}
          paralela.x = TRUE
            --{{{  Procesos
            SEQ
              IF
                ( coord.fija  >= y.minimo.ent ) AND ( coord.fija <= y.maximo.ent )
                  --{{{  Procesos
                  SEQ
                    IF
                      coord.i.int < x.minimo.ent
```

```
:
--}}}
--
--{{{  Asignar Color
PROC Color ( VAL BYTE color.elegido )
  --{{{  Procesos
  color := color.elegido
  --}}}
:
--}}}
--
--{{{  Borrar Pantalla
PROC BorraPantalla ()
  --{{{  Procesos
  INT color.ayu :
  BYTE color.temp :
  SEQ
    color.temp := color
    color.ayu := 0
    color :=  BYTE(color.ayu)
    IniciaVectorBytes ( 0, alto.pixel, numero.bytes, color )
    color := color.temp
  --}}}
:
--}}}
--
--{{{  Iniciar Pantalla
PROC IniciaPantalla ( VAL REAL32 x, VAL REAL32 y )
  --{{{  Procesos
  SEQ
    habilita.salida := 1
    modo.evento := 0
    habilita.entrelazado := 0
    pronto := 0
    inicio.pantalla := 0
    Escala  ( x, y )
    color.elegido.ent := 255
    color := BYTE(color.elegido.ent)
  --}}}
:
--}}}
--
--}}}F
--
--{{{  Dibujantes
--
--{{{  Dibujar Puntos
PROC DibujaPuntos ( VAL INT px, VAL INT py )
  --{{{  Procesos
  INT y.arregla :
  SEQ
    y.arregla :=  alto.pixel - py
    pantalla[ y.arregla ][ px ] := color
  --}}}
:
--}}}
--
--{{{F Rectil - Dibujar Figuras Rectilineas
--:::F Rectil02.tsr
--
--{{{  Auxiliares
--
--{{{  Dado x, y, pendiente y nuevo x, obtener nuevo y
PROC ObtenerYdeRecta ( VAL INT px, VAL INT py, VAL REAL32 pendiente,
                       VAL INT px.nuevo, INT py.nuevo )
  --{{{  Declaraciones
  INT delta.y.ent :
  REAL32 delta.x.r :
  --}}}
  SEQ
    --{{{  Proc
    delta.x.r     :=  REAL32 ROUND(px.nuevo - px)
    delta.y.ent   :=  INT ROUND(pendiente * delta.x.r)
    py.nuevo      :=  py + delta.y.ent
    --}}}
:
--}}}
--
--{{{  Dado x, y, pendiente y nuevo y, obtener nuevo x
PROC ObtenerXdeRecta ( VAL INT px, VAL INT py, VAL REAL32 pendiente,
                       VAL INT py.nuevo, INT px.nuevo )
  --{{{  Declaraciones
  INT delta.x.ent, delta.y.ent :
  REAL32 delta.y.r :
  --}}}
  SEQ
    --{{{  Proc
    delta.y.r     :=  REAL32 ROUND(py.nuevo - py)
    delta.y.ent   :=  INT ROUND(pendiente * delta.y.r)
    px.nuevo      :=  px + delta.x.ent
    --}}}
:
--}}}
--
--{{{  Verifica se Recta Pertenece a Pantalla
PROC VerificaRectaPertenecePantalla ( INT pxi,  INT pyi ,  INT pxf,
                                      INT pyf, BOOL pertenece )
  --{{{  Declaraciones
  REAL32 delta.x.real, delta.y.real, pendiente :
  INT delta.x.ent, delta.y.ent, ayu :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Arreglar xi,yi,xf,yf
    IF
      ( pxi > pxf )
        SEQ
          --{{{  Proc
          ayu  :=  pxi
          pxi  :=  pxf
          pxf  :=  ayu
          ayu  :=  pyi
          pyi  :=  pyf
          pyf  :=  ayu
          --}}}
      TRUE
        SKIP
    --}}}
    --{{{  Asignaciones
    delta.x.ent    :=  pxf - pxi
    delta.y.ent    :=  pyf - pyi
    delta.x.real   :=  REAL32 ROUND(delta.x.ent)
    delta.y.real   :=  REAL32 ROUND(delta.y.ent)
    pendiente      :=  delta.y.real / delta.x.real
    pertenece      :=  FALSE
    --}}}
    --{{{  Verifica x.min
    ObtenerYdeRecta ( pxi, pyi, pendiente, x.minimo.ent, ayu )
    IF
      pxi < x.minimo.ent
        SEQ
          pxi := x.minimo.ent
```

```
          -- nuevo y
          ayu.y.ant       :=  ayu.y
          ayu.y           :=  pyi + delta.y.ayu.ent
          ayu.rep         :=  ayu.y - ayu.y.ant
          --}}}
          --{{{  Coge  Puntos "y" Extra
          IF
            ayu.rep < (-1)
              --{{{  Proc
              SEQ
                ayu.rep  :=  -ayu.rep
                SEQ i=0 FOR ( ayu.rep - 1 )
                  SEQ
                    ayu.limite.y := ( ayu.y.ant - i ) - 1
                    IF
                      ( ayu.limite.y >= y.minimo.ent )
                        SEQ
                          contorno.x[tamano] := ayu.x.ant
                          contorno.y[tamano] := ayu.limite.y
                          tamano             := tamano + 1
                      TRUE
                        SKIP
              --}}}
            ayu.rep > 1
              --{{{  Proc
              SEQ i=0 FOR ( ayu.rep - 1 )
                SEQ
                  ayu.limite.y := ( ayu.y.ant + i ) + 1
                  IF
                    ( ayu.limite.y <= y.maximo.ent )
                      SEQ
                        contorno.x[tamano] := ayu.x.ant
                        contorno.y[tamano] := ayu.limite.y
                        tamano             := tamano + 1
                    TRUE
                      SKIP
              --}}}
            TRUE
              SKIP
          --}}}
          --{{{  Coge Puntos x y y
          IF
            ( ayu.y <= y.maximo.ent ) AND ( ayu.y >= y.minimo.ent )
              SEQ
                contorno.x[tamano] := ayu.x
                contorno.y[tamano] := ayu.y
                tamano             := tamano + 1
            TRUE
              SKIP
          --}}}
          --}}}
      --}}}
:
--}}}
--
--{{{  Almacena Puntos Recta ( Parametros no Asignables )
PROC AlmacenaPuntosSegmentoNA ( VAL INT pxi, VAL INT pyi,  VAL INT pxf,
                                VAL INT pyf, INT tamano, []INT contorno.x,
                                []INT contorno.y )
  --{{{  Declaraciones
  REAL32 m :
  REAL32 delta.x.r, delta.y.r :
  INT delta.x.ent, delta.y.ent, ayu :
  INT cuanto.x, cuanto.y :
  INT x.arregla, y.arregla :
  INT pxi.int, pyi.int, pxf.int, pyf.int :
```

```
  INT coord.f.int, coord.i.int, cuanto :
  SEQ
    --{{{  Ordena Coordenadas
    IF
      delta < 0
        SEQ
          coord.f.int := coord.i
          coord.i.int := coord.f
      TRUE
        SEQ
          coord.f.int := coord.f
          coord.i.int := coord.i
    --}}}
    --{{{  Eje x o y
    IF
      paralela.x = FALSE
        --{{{  Procesos
        SEQ
          IF
            ( coord.fija >= x.minimo.ent ) AND ( coord.fija <= x.maximo.ent )
              --{{{  Procesos
              SEQ
                IF
                  coord.i.int < y.minimo.ent
                    coord.i.int := y.minimo.ent
                  TRUE
                    SKIP
                IF
                  coord.f.int > y.maximo.ent
                    coord.f.int := y.maximo.ent
                  TRUE
                    SKIP
                WHILE coord.i.int <= coord.f.int
                  --{{{  Procesos
                  SEQ
                    DibujaPuntos ( coord.fija, coord.i.int )
                    coord.i.int := coord.i.int + 1
                  --}}}
              --}}}
            TRUE
              SKIP
        --}}}
      paralela.x = TRUE
        --{{{  Procesos
        SEQ
          IF
            ( coord.fija  >= y.minimo.ent ) AND ( coord.fija <= y.maximo.ent )
              --{{{  Procesos
              SEQ
                IF
                  coord.i.int < x.minimo.ent
                    coord.i.int := x.minimo.ent
                  TRUE
                    SKIP
                IF
                  coord.f.int > x.maximo.ent
                    coord.f.int := x.maximo.ent
                  TRUE
                    SKIP
                cuanto := ( coord.f.int - coord.i.int ) + 1
                IniciaVectorBytes ( coord.i.int, coord.fija, cuanto, color )
              --}}}
            TRUE
              SKIP
      --}}}
  --}}}
```

```
                coord.i.int := x.minimo.ent
              TRUE
                SKIP
            IF
              coord.f.int > x.maximo.ent
                coord.f.int := x.maximo.ent
              TRUE
                SKIP
            WHILE coord.i.int <= coord.f.int
              --{{{  Procesos
              SEQ
                contorno.x[tamano] := coord.i.int
                contorno.y[tamano] := coord.fija
                coord.i.int        := coord.i.int + 1
                tamano             := tamano + 1
              --}}}
          --}}}
        TRUE
          SKIP
      --}}}
  --}}}
  --}}}
:
--}}}
--
--{{{  Almacena Puntos Recta Cualquiera
PROC AlmacenaPuntosSegmentoCualquiera ( VAL INT pxi, VAL INT pyi,
                                        VAL INT pxf, VAL INT pyf,
                                        VAL REAL32 m, INT tamano,
                                        []INT contorno.x, []INT contorno.y )

  --{{{  Declaraciones
  INT ayu.x, ayu.y, ayu.x.ant, ayu.y.ant :
  INT ayu.rep, ayu.limite.y :
  INT delta.x.ayu.ent, delta.y.ayu.ent :
  REAL32 delta.x.ayu.real, delta.y.ayu.real :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Asignaciones
    ayu.x    := pxi
    ayu.y    := pyi
    tamano   := 0
    --}}}
    IF
      ( ayu.x >= x.minimo.ent )  AND ( ayu.x <= x.maximo.ent ) AND
      ( ayu.y >= y.minimo.ent )  AND ( ayu.y <= y.maximo.ent )
        SEQ
          contorno.x[tamano] := ayu.x
          contorno.y[tamano] := ayu.y
          tamano             := tamano + 1
      TRUE
        SKIP
    WHILE ( ayu.x < pxf ) AND ( ayu.x >= x.minimo.ent )  AND
          ( ayu.x <= x.maximo.ent )
      SEQ
        --{{{  Procesos
        --{{{  Calcula el Proximo Punto
        ayu.x.ant        := ayu.x
        -- avanzar x
        ayu.x            :=  ayu.x + 1
        -- calcular delta x real
        delta.x.ayu.ent  := ayu.x - pxi
        delta.x.ayu.real := REAL32 ROUND( delta.x.ayu.ent )
        -- calcular delta y entero
        delta.y.ayu.real := m * delta.x.ayu.real
        delta.y.ayu.ent  := INT ROUND(delta.y.ayu.real)
```

```
        --}}}
    --}}}
    SEQ
      --{{{  Procesos
      SEQ
        --{{{  Asignaciones
        --{{{  Asignaciones Iniciales
        pxi.int := pxi
        pyi.int := pyi
        pxf.int := pxf
        pyf.int := pyf
        --}}}
        --{{{  Arregla xi y xf
        IF
          pxi.int > pxf.int
            SEQ
              ayu      := pxi.int
              pxi.int  := pxf.int
              pxf.int  := ayu
              ayu      := pyi.int
              pyi.int  := pyf.int
              pyf.int  := ayu
          TRUE
            SKIP
        --}}}
        delta.x.ent   :=  pxf.int - pxi.int
        delta.y.ent   :=  pyf.int - pyi.int
        cuanto.x      :=  delta.x.ent + 1
        cuanto.y      :=  delta.y.ent + 1
        delta.x.r     :=  REAL32 ROUND(delta.x.ent)
        delta.y.r     :=  REAL32 ROUND(delta.y.ent)
        tamano        :=  0
        --}}}
        IF
          ( delta.x.ent = 0 ) AND ( delta.y.ent = 0 )
            --{{{  Proc
            SEQ
              contorno.x[0]  := pxi.int
              contorno.x[0]  := pyi.int
              tamano         := cuanto.x
            --}}}
          ( delta.x.ent = 0 ) AND ( delta.y.ent <> 0 )
            AlmacenaPuntosSegmentoParaleloEje ( delta.y.ent, pyi.int, pyf.int, p
                                                FALSE, tamano, contorno.x,
                                                contorno.y )
          ( delta.x.ent <> 0 ) AND ( delta.y.ent = 0 )
            AlmacenaPuntosSegmentoParaleloEje ( delta.x.ent, pxi.int, pxf.int, p
                                                TRUE, tamano, contorno.x,
                                                contorno.y  )
          ( delta.x.ent <> 0 ) AND ( delta.y.ent <> 0 )
            SEQ
              m := delta.y.r / delta.x.r
              AlmacenaPuntosSegmentoCualquiera ( pxi.int, pyi.int, pxf.int,
                                                 pyf.int, m, tamano,
                                                 contorno.x, contorno.y )
          TRUE
            SKIP
    --}}}
:
--}}}
--
--{{{  Almacena Puntos Recta ( Parametros  Asignables )
PROC AlmacenaPuntosSegmentoA ( INT pxi, INT pyi,  INT pxf, INT pyf,
                               INT tamano, []INT contorno.x,
                               []INT contorno.y )

  --{{{  Declaraciones
  REAL32 m :
```

```
  --}}}
:
--}}}
--
--{{{  Dibujar Linea Cualquiera Con Parametros Enteros
PROC LineaCualquieraEntero ( VAL INT pxi, VAL INT pyi, VAL INT pxf,
                             VAL INT pyf, VAL REAL32 m )
  --{{{  Declaraciones
  INT ayu.x, ayu.y, ayu.x.ant, ayu.y.ant :
  INT ayu.rep, ayu.limite.y :
  INT delta.x.ayu.ent, delta.y.ayu.ent :
  REAL32 delta.x.ayu.real, delta.y.ayu.real :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Asignaciones
    ayu.x    := pxi
    ayu.y    := pyi
    --}}}
    IF
      ( ayu.x >= x.minimo.ent )  AND ( ayu.x <= x.maximo.ent ) AND
      ( ayu.y >= y.minimo.ent )  AND ( ayu.y <= y.maximo.ent )
        DibujaPuntos ( ayu.x, ayu.y )
      TRUE
        SKIP
    WHILE ( ayu.x < pxf ) AND ( ayu.x >= x.minimo.ent )  AND
          ( ayu.x <= x.maximo.ent )
      SEQ
        --{{{  Procesos
        --{{{  Calcula el Proximo Punto
        ayu.x.ant        := ayu.x
        -- avanzar x
        ayu.x            :=  ayu.x + 1
        -- calcular delta x real
        delta.x.ayu.ent  := ayu.x - pxi
        delta.x.ayu.real := REAL32 ROUND( delta.x.ayu.ent )
        -- calcular delta y entero
        delta.y.ayu.real := m * delta.x.ayu.real
        delta.y.ayu.ent  := INT ROUND(delta.y.ayu.real)
        -- nuevo y
        ayu.y.ant        :=  ayu.y
        ayu.y            :=  pyi + delta.y.ayu.ent
        ayu.rep          :=  ayu.y - ayu.y.ant
        --}}}
        --{{{  Dibujo de Puntos y Extra
        IF
          ayu.rep < (-1)
            --{{{  Proc
            SEQ
              ayu.rep  :=  -ayu.rep
              SEQ i=0 FOR ( ayu.rep - 1 )
                SEQ
                  ayu.limite.y := ( ayu.y.ant - i ) - 1
                  IF
                    ( ayu.limite.y >= y.minimo.ent )
                      DibujaPuntos( ayu.x.ant, ayu.limite.y )
                    TRUE
                      SKIP
            --}}}
          ayu.rep > 1
            --{{{  Proc
            SEQ i=0 FOR ( ayu.rep - 1 )
              SEQ
                ayu.limite.y := ( ayu.y.ant + i ) + 1
                IF
                  ( ayu.limite.y <= y.maximo.ent )
```

```
    [768]INT contorno.y :
    INT xi.ent, xf.ent, yi.ent, yf.ent :
    INT tamano :
    --}}}
    SEQ
      --{{{  Proc
      --{{{  Asignaciones
      xi.ent  := pxi
      yi.ent  := pyi
      xf.ent  := pxf
      yf.ent  := pyf
      --}}}
      AlmacenaPuntosSegmentoA ( xi.ent, yi.ent, xf.ent, yf.ent, tamano,
                                contorno.x, contorno.y )
      SEQ i=0 FOR tamano
        DibujaPuntos ( contorno.x[i], contorno.y[i] )
      --}}}
:
--}}}
--
--{{{  Dibujar Tira Linea con Coge Contorno no Asignable
PROC TiraLineaAlmacenaPuntosNA ( VAL INT pxi, VAL INT pyi,
                                 VAL INT pxf, VAL INT pyf )
  --{{{  Declaraciones
  [1024]INT contorno.x :
  [768]INT contorno.y :
  INT xi.ent, xf.ent, yi.ent, yf.ent :
  INT tamano :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Asignaciones
    xi.ent  := pxi
    yi.ent  := pyi
    xf.ent  := pxf
    yf.ent  := pyf
    --}}}
    AlmacenaPuntosSegmentoNA ( xi.ent, yi.ent, xf.ent, yf.ent,
                               tamano, contorno.x, contorno.y )
    SEQ i=0 FOR tamano
      DibujaPuntos ( contorno.x[i], contorno.y[i] )
    --}}}
:
--}}}
--
--}}}
--
--{{{  Dibujar Rectangulos
--
--{{{  Dibujar Rectangulo
PROC Rectangulo ( VAL REAL32 xi, VAL REAL32 yi,
                  VAL REAL32 xf, VAL REAL32 yf )
  --{{{  Procesos
  --{{{  Declaraciones
  [(ancho*alto)]BYTE vec.pant RETYPES pantalla :
  INT elegir.ayu :
  INT y.arregla, y.arregla1 :
  INT vec.ayu1, vec.ayu2 :
  INT origen.x, cuanto.x, cuanto.y, cuanto.y.ayu :
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Asignaciones
    xi.ent :=  INT ROUND( xi*escala )
    yi.ent :=  INT ROUND( yi*escala )
    xf.ent :=  INT ROUND( xf*escala )
```

```
REAL32 delta.x.r, delta.y.r :
INT delta.x.ent, delta.y.ent, ayu :
INT cuanto.x, cuanto.y :
INT x.arregla, y.arregla :
--}}}
SEQ
  --{{{  Procesos
  SEQ
    --{{{  Asignaciones
    --{{{  Arregla xi y xf
    IF
      pxi > pxf
        SEQ
          ayu  := pxi
          pxi  := pxf
          pxf  := ayu
          ayu  := pyi
          pyi  := pyf
          pyf  := ayu
      TRUE
        SKIP
    --}}}
    delta.x.ent   :=  pxf - pxi
    delta.y.ent   :=  pyf - pyi
    cuanto.x      :=  delta.x.ent + 1
    cuanto.y      :=  delta.y.ent + 1
    delta.x.r     :=  REAL32 ROUND(delta.x.ent)
    delta.y.r     :=  REAL32 ROUND(delta.y.ent)
    tamano        :=  0
    --}}}
    IF
      ( delta.x.ent = 0 ) AND ( delta.y.ent = 0 )
        --{{{  Proc
        SEQ
          contorno.x[0]  := pxi
          contorno.x[0]  := pyi
          tamano         := cuanto.x
        --}}}
      ( delta.x.ent = 0 ) AND ( delta.y.ent <> 0 )
        AlmacenaPuntosSegmentoParaleloEje ( delta.y.ent, pyi, pyf, pxi,
                                            FALSE, tamano, contorno.x,
                                            contorno.y )
      ( delta.x.ent <> 0 ) AND ( delta.y.ent = 0 )
        AlmacenaPuntosSegmentoParaleloEje ( delta.x.ent, pxi, pxf, pyi,
                                            TRUE, tamano, contorno.x,
                                            contorno.y )
      ( delta.x.ent <> 0 ) AND ( delta.y.ent <> 0 )
        SEQ
          m := delta.y.r / delta.x.r
          AlmacenaPuntosSegmentoCualquiera ( pxi, pyi, pxf, pyf, m,
                                             tamano, contorno.x,
                                             contorno.y )

      TRUE
        SKIP
  --}}}
:
--}}}
--
--}}}
--
--{{{  Dibujar Tira Linea
--
--{{{  Dibujar Linea Paralela a un Eje
PROC LineaParalelaEje ( VAL INT delta, VAL INT coord.i, VAL INT coord.f,
                        VAL INT coord.fija,          VAL BOOL paralela.x )
  --{{{  Procesos
    --{{{  Asignaciones
    --{{{  Arregla xi y xf
    IF
      coord.i > coord.f
        SEQ
          ayu  := coord.i
          coord.i  := coord.f
          coord.f  := ayu
      TRUE
        SKIP
    --}}}
    IF
      paralela.x
        DibujaPuntos( coord.i, coord.fija )
      TRUE
        SKIP
    --}}}
  TRUE
    SKIP
--}}}
--{{{  Dibujo de Puntos
IF
  ( ayu.y <= y.maximo.ent ) AND ( ayu.y >= y.minimo.ent )
    DibujaPuntos ( ayu.x, ayu.y )
  TRUE
    SKIP
--}}}
--}}}
--}}}
:
--}}}
--
--{{{  TiraLinea Con Parametros Enteros
PROC TiraLineaEntero ( INT pxi, INT pyi,
                       INT pxf, INT pyf )
  --{{{  Declaraciones
  REAL32 delta.x.real, delta.y.real, m :
  REAL32 xi.r, yi.r, xf.r, yf.r :
  INT delta.x.ent, delta.y.ent, ayu :
  INT pxi1, pyi1, pxf1, pyf1 :
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Escala
    xi.r  := REAL32 ROUND(pxi)
    yi.r  := REAL32 ROUND(pyi)
    xf.r  := REAL32 ROUND(pxf)
    yf.r  := REAL32 ROUND(pyf)
    pxi1  := INT ROUND(escala*xi.r)
    pyi1  := INT ROUND(escala*yi.r)
    pxf1  := INT ROUND(escala*xf.r)
    pyf1  := INT ROUND(escala*yf.r)
    --}}}
    --{{{  Arregla xi y xf
    IF
      pxi1 > pxf1
        SEQ
          ayu   := pxi1
          pxi1  := pxf1
          pxf1  := ayu
          ayu   := pyi1
          pyi1  := pyf1
          pyf1  := ayu
      TRUE
        SKIP.
    --}}}
    --{{{  Asignaciones
    delta.x.ent   :=  pxf1 - pxi1
    delta.y.ent   :=  pyf1 - pyi1
    delta.x.real  :=  REAL32 ROUND(delta.x.ent)
    delta.y.real  :=  REAL32 ROUND(delta.y.ent)
    --}}}
    IF
      ( delta.x.ent = 0 ) AND ( delta.y.ent = 0 )
        DibujaPuntos( pxi1, pyi1 )
      ( delta.x.ent = 0 ) AND ( delta.y.ent <> 0 )
        LineaParalelaEje ( delta.y.ent, pyi1, pyf1,
                                        pxi1, FALSE )
      ( delta.x.ent <> 0 ) AND ( delta.y.ent = 0 )
```

```
yf.ent :=  INT ROUND( yf*escala )
--}}}
--{{{  Elegir ( xi < xf ) y ( yi < yf )
--{{{  xi < xf
IF
  xf.ent < xi.ent
    SEQ
      elegir.ayu := xi.ent
      xi.ent     := xf.ent
      xf.ent     := elegir.ayu
  TRUE
    SKIP
--}}}
--{{{  yi < yf
IF
  yf.ent < yi.ent
    SEQ
      elegir.ayu := yi.ent
      yi.ent     := yf.ent
      yf.ent     := elegir.ayu
  TRUE
    SKIP
--}}}
--}}}
--{{{  Limites de x e y
IF
  xi.ent < 0
    xi.ent := 0
  TRUE
    SKIP
IF
  yi.ent < 0
    yi.ent := 0
  TRUE
    SKIP
IF
  xf.ent > x.maximo.ent
    xf.ent := x.maximo.ent
  TRUE
    SKIP
IF
  yf.ent > y.maximo.ent
    yf.ent := y.maximo.ent
  TRUE
    SKIP
--}}}
--{{{  Asignaciones
delta.x     := xf.ent - xi.ent
delta.y     := yf.ent - yi.ent
cuanto.x    := delta.x + 1
cuanto.y    := delta.y + 1
origen.x    := xi.ent
y.arregla   := alto.pixel - yf.ent
y.arregla1  := y.arregla * ancho
--}}}
--}}}
IniciaVectorBytes ( origen.x, yf.ent, cuanto.x, color )
vec.ayu1 := y.arregla1 + origen.x
vec.ayu2 := vec.ayu1
cuanto.y.ayu := 1
WHILE   cuanto.y >= ( 2 * cuanto.y.ayu )
  SEQ
    SEQ j = 1 FOR cuanto.y.ayu
      SEQ
        vec.ayu2 := vec.ayu2 + ancho
        [ vec.pant FROM vec.ayu2 FOR cuanto.x ] :=
```

```
          SKIP
      IF
        yi.ent < 0
          yi.ent := 0
        TRUE
          SKIP
      delta.y := yf.ent - yi.ent
      cuanto.y := delta.y + 1
      --}}}
      --}}}
      SEQ j = xi.ent FOR cuanto.x
        SEQ k = yi.ent FOR cuanto.y
          DibujaPuntos ( j, k )
  --}}}
:
--}}}
--
--}}}
--
--{{{  Dibujar Paralelogramos
--
--{{{  Paralelogramo de Base Paralela a x
PROC ParalelogramoParaleloX ( VAL INT px1, VAL INT px2, VAL INT py1.2,
                              VAL INT pxo, VAL INT pyo )
  --{{{  Declaraciones
  INT delta.x.1.2 :
  INT delta.x.1.o :
  INT delta.x.2.o :
  INT cuanto.x.1.2 :
  INT tamano :
  [1024]INT contorno.x :
  [768]INT contorno.y :
  REAL32 xi.r, yi.r, xo.r, yo.r :
  --}}}
  SEQ
    --{{{  Asignaciones
    delta.x.1.2  := px2 - px1
    delta.x.1.o  := pxo - px1
    delta.x.2.o  := pxo - px2
    cuanto.x.1.2 := delta.x.1.2 + 1
    --}}}
    IF
      delta.x.2.o = 0
        SEQ
          --{{{  Proc
          xi.r    := REAL32 ROUND(px1)
          yi.r    := REAL32 ROUND(py1.2)
          xo.r    := REAL32 ROUND(pxo)
          yo.r    := REAL32 ROUND(pyo)
          Rectangulo ( xi.r, yi.r, xo.r, yo.r )
          --}}}
      delta.x.1.o = 0
        SEQ
          --{{{  Proc
          xi.r    := REAL32 ROUND(px2)
          yi.r    := REAL32 ROUND(py1.2)
          xo.r    := REAL32 ROUND(pxo)
          yo.r    := REAL32 ROUND(pyo)
          Rectangulo ( xi.r, yi.r, xo.r, yo.r )
          --}}}
      TRUE
        SEQ
          --{{{  Proc
          AlmacenaPuntosSegmentoNA ( pxo, pyo, px1, py1.2, tamano,
                                     contorno.x, contorno.y )
          SEQ i=0 FOR tamano
```

```
        LineaParalelaEje ( delta.x.ent, pxi1, pxf1,
                                    pyi1,  TRUE )
      ( delta.x.ent <> 0 ) AND ( delta.y.ent <> 0 )
        SEQ
          m  :=  delta.y.real / delta.x.real
          LineaCualquieraEntero ( pxi1, pyi1, pxf1, pyf1, m )
      TRUE
        SKIP
    --}}}
:
--}}}
--{{{  COMMENT TiraLinea Con Parametros Enteros
--:::A COMMENT FOLD
--{{{  TiraLinea Con Parametros Enteros
PROC TiraLineaEntero ( INT pxi, INT pyi,
                       INT pxf, INT pyf )
  --{{{  Declaraciones
  REAL32 delta.x.real, delta.y.real, m :
  INT delta.x.ent, delta.y.ent, ayu :
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Arregla xi y xf
    IF
      pxi > pxf
        SEQ
          ayu  := pxi
          pxi  := pxf
          pxf  := ayu
          ayu  := pyi
          pyi  := pyf
          pyf  := ayu
      TRUE
        SKIP
    --}}}
    --{{{  Asignaciones
    delta.x.ent   :=  pxf - pxi
    delta.y.ent   :=  pyf - pyi
    delta.x.real  :=  REAL32 ROUND(delta.x.ent)
    delta.y.real  :=  REAL32 ROUND(delta.y.ent)
    --}}}
    IF
      ( delta.x.ent = 0 ) AND ( delta.y.ent = 0 )
        DibujaPuntos( pxi, pyi )
      ( delta.x.ent = 0 ) AND ( delta.y.ent <> 0 )
        LineaParalelaEje ( delta.y.ent, pyi, pyf,
                                    pxi, FALSE )
      ( delta.x.ent <> 0 ) AND ( delta.y.ent = 0 )
        LineaParalelaEje ( delta.x.ent, pxi, pxf,
                                    pyi,  TRUE )
      ( delta.x.ent <> 0 ) AND ( delta.y.ent <> 0 )
        SEQ
          m  :=  delta.y.real / delta.x.real
          LineaCualquieraEntero ( pxi, pyi, pxf, pyf, m )
      TRUE
        SKIP
    --}}}
:
--}}}
--}}}
--
--{{{  Dibujar Tira Linea con Coge Contorno Asignable
PROC TiraLineaAlmacenaPuntosA  ( INT pxi, INT pyi,
                                 INT pxf, INT pyf )
  --{{{  Declaraciones
  [1024]INT contorno.x :
```

```
                                 [ vec.pant FROM vec.ayu1 FOR cuanto.x ]
      cuanto.y.ayu := cuanto.y.ayu + cuanto.y.ayu
    --{{{  Dibuja los que Quedan
    cuanto.y.ayu := cuanto.y - cuanto.y.ayu
    SEQ j = 1 FOR cuanto.y.ayu
      SEQ
        vec.ayu2 := vec.ayu2 + ancho
        [ vec.pant FROM vec.ayu2 FOR cuanto.x ] :=
                [ vec.pant FROM vec.ayu1 FOR cuanto.x ]
    --}}}
  --}}}
:
--}}}
--
--{{{  Dibujar Rectangulo 1 ( Viejo, Lento )
PROC Rectangulo1 ( VAL REAL32 xi, VAL REAL32 yi,
                   VAL REAL32 xf, VAL REAL32 yf )
  --{{{  Procesos
  INT j, k :
  INT cuanto.x, cuanto.y :
  INT delta.x, delta.y :
  SEQ
    --{{{  Procesos
    xi.ent := ( INT ROUND( xi*escala ) )
    yi.ent := ( INT ROUND( yi*escala ) )
    xf.ent := ( INT ROUND( xf*escala ) )
    yf.ent := ( INT ROUND( yf*escala ) )
    --{{{  Busca x.max
    IF
      xf.ent >= xi.ent
        SKIP
      TRUE
        SEQ
          j := xi.ent
          xi.ent := xf.ent
          xf.ent := j
    --}}}
    --{{{  Busca y.max
    IF
      yf.ent >= yi.ent
        SKIP
      TRUE
        SEQ
          k := yi.ent
          yi.ent := yf.ent
          yf.ent := k
    --}}}
    --{{{  Arregla limites de x
    IF
      xf.ent > x.maximo.ent
        xf.ent := x.maximo.ent
      TRUE
        SKIP
    IF
      xi.ent < 0
        xi.ent := 0
      TRUE
        SKIP
    delta.x := xf.ent - xi.ent
    cuanto.x := delta.x + 1
    --}}}
    --{{{  Arregla limites de y
    IF
      yf.ent > y.maximo.ent
        yf.ent := y.maximo.ent
      TRUE
```

```
              cuanto.x := ( pxo - contorno.x[i] ) + 1
              IniciaVectorBytes ( contorno.x[i], contorno.y[i],
                                  cuanto.x, color )
              IniciaVectorBytes ( px1.2, ( contorno.y[i] + delta.y ),
                                  cuanto.x, color )
              delta.y  := delta.y - 2
            --}}}
        --}}}
      TRUE
        SKIP
      --}}}
:
--}}}
--
--{{{  Paralelogramos generales ( Tipos de )
--
--{{{  t1
PROC t1 ()
  --{{{  Declaraciones
  --}}}
  SEQ
    --{{{  Proc
    --}}}
:
--}}}
--
--{{{  t2
PROC t2 ()
  --{{{  Declaraciones
  --}}}
  SEQ
    --{{{  Proc
    --}}}
:
--}}}
--
--{{{  t3
PROC t3 ()
  --{{{  Declaraciones
  --}}}
  SEQ
    --{{{  Proc
    --}}}
:
--}}}
--
--{{{  t4
PROC t4 ()
  --{{{  Declaraciones
  --}}}
  SEQ
    --{{{  Proc
    --}}}
:
--}}}
--
--{{{  t5
PROC t5 ()
  --{{{  Declaraciones
  --}}}
  SEQ
    --{{{  Proc
    --}}}
:
--}}}
--
```

```
--{{{  t6
PROC t6 ()
  --{{{  Declaraciones
  --}}}
  SEQ
    --{{{  Proc
    --}}}
:
--}}}
--
--}}}
--
--{{{  Paralelogramo Cualquier
PROC Paralelogramo ( VAL INT pxi, VAL INT pyi, VAL INT pxf,
                     VAL INT pyf, VAL INT pxo, VAL INT pyo )
  --{{{  Declaraciones
  INT delta.x.i.f, delta.y.i.f :
  INT delta.x.i.f.v, delta.y.i.f.v :
  INT delta.x.f.o, delta.y.f.o :
  INT delta.x.f.o.v, delta.y.f.o.v :
  INT delta.x.o.i, delta.y.o.i :
  INT delta.x.o.i.v, delta.y.o.i.v :
  INT cuanto.i.f, cuanto.f.o :
  INT cuanto.repetir :
  INT tamano.i.f, tamano.f.o :
  [1024]INT contorno.x.i.f :
  [768]INT contorno.y.i.f :
  [1024]INT contorno.x.f.o :
  [768]INT contorno.y.f.o :
  INT xi.ent, yi.ent, xf.ent, yf.ent, xo.ent, yo.ent :
  INT xi.ent.v, yi.ent.v, xf.ent.v, yf.ent.v, xo.ent.v, yo.ent.v :
  REAL32 xi.r, yi.r, xf.r, yf.r, xo.r, yo.r :
  REAL32 xi.r.v, yi.r.v, xf.r.v, yf.r.v, xo.r.v, yo.r.v :
  --}}}
  SEQ
    --{{{  Proc
    SEQ
      --{{{  Asignaciones
      --{{{  Enteros
      xi.ent.v        := pxi
      yi.ent.v        := pyi
      xf.ent.v        := pxf
      yf.ent.v        := pyf
      xo.ent.v        := pxo
      yo.ent.v        := pyo
      delta.x.i.f.v   := xf.ent.v - xi.ent.v
      delta.y.i.f.v   := yf.ent.v - yi.ent.v
      delta.x.f.o.v   := xo.ent.v - xf.ent.v
      delta.y.f.o.v   := yo.ent.v - yf.ent.v
      delta.x.o.i.v   := xi.ent.v - xo.ent.v
      delta.y.o.i.v   := yi.ent.v - yo.ent.v
      --}}}
      --{{{  Reales
      xi.r.v          := REAL32 ROUND(pxi)
      yi.r.v          := REAL32 ROUND(pyi)
      xf.r.v          := REAL32 ROUND(pxf)
      yf.r.v          := REAL32 ROUND(pyf)
      xo.r.v          := REAL32 ROUND(pxo)
      yo.r.v          := REAL32 ROUND(pyo)
      --}}}
      --}}}
      IF
        ( xo.ent.v <= x.maximo.ent ) AND ( xf.ent.v <= x.maximo.ent ) AND
        ( xi.ent.v >= 0 ) AND ( yi.ent.v <= y.maximo.ent ) AND
        ( yo.ent.v <= y.maximo.ent ) AND ( yi.ent.v >= 0 ) AND
        ( yo.ent >= 0 ) AND ( yf.ent >= 0 )
```

```
          IniciaVectorBytes ( contorno.x[i], contorno.y[i],
                              cuanto.x.1.2, color )
        --}}}
:
--}}}
--
--{{{  Paralelogramo de Base Paralela a y
PROC ParalelogramoParaleloY ( VAL INT py1, VAL INT py2, VAL INT px1.2,
                              VAL INT pxo, VAL INT pyo )
  --{{{  Declaraciones
  INT delta.y.1.2 :
  INT delta.y.1.o :
  INT delta.y.2.o :
  INT delta.x, delta.y, cuanto.x :
  INT tamano, ayu.x, ayu.y :
  [1024]INT contorno.x :
  [768]INT contorno.y :
  REAL32 xi.r, yi.r, xo.r, yo.r, delta.x.r, delta.y.r, m :
  --}}}
  SEQ
    --{{{  Asignaciones
    delta.y.1.2  := py2 - py1
    delta.y.1.o  := pyo - py1
    delta.y.2.o  := pyo - py2
    --}}}
    IF
      --{{{  Ifs
      delta.y.2.o = 0
        SEQ
          --{{{  Proc
          xi.r    := REAL32 ROUND(px1.2)
          yi.r    := REAL32 ROUND(py1)
          xo.r    := REAL32 ROUND(pxo)
          yo.r    := REAL32 ROUND(pyo)
          Rectangulo ( xi.r, yi.r, xo.r, yo.r )
          --}}}
      ( delta.y.2.o > 0 ) AND ( ( pyo - delta.y.1.2 ) > py2 )
        SEQ
          --{{{  Proc
          ayu.y     := pyo - delta.y.1.2
          delta.y   := ayu.y - py1
          delta.y.r := REAL32 ROUND(delta.y)
          delta.x   := pxo - px1.2
          delta.x.r := REAL32 ROUND(delta.x)
          m         := delta.y.r / delta.x.r
          ObtenerXdeRecta ( px1.2, py1, m, py2, ayu.x )
          ParalelogramoParaleloX ( px1.2, ayu.x, py2, pxo, ayu.y )
          delta.y   := ( pyo - delta.y.1.2 ) - py2
          AlmacenaPuntosSegmentoNA ( px1.2, py1, ayu.x, py2, tamano,
                                     contorno.x, contorno.y )
          DibujaPuntos ( pxo, pyo )
          DibujaPuntos ( px1.2, py1 )
          ayu.x   := pxo - px1.2
          delta.y := pyo - py1
          SEQ i=0 FOR tamano
            --{{{  Proc
            SEQ
              cuanto.x := ( contorno.x[i] - px1.2 ) + 1
              IniciaVectorBytes ( px1.2, contorno.y[i],
                                  cuanto.x, color )
              IniciaVectorBytes ( ( contorno.x[i] + ayu.x ), ( contorno.y[i] +
                                  cuanto.x, color )
              ayu.x    := ayu.x - 2
              delta.y  := delta.y - 2
            --}}}
          --}}}
      ( delta.y.2.o > 0 ) AND ( ( pyo - delta.y.1.2 ) <= py2 )
        SEQ
          --{{{  Proc
          xi.r      := REAL32 ROUND(px1.2)
          yi.r      := REAL32 ROUND(py2)
          xo.r      := REAL32 ROUND(pxo)
          delta.y   := pyo - delta.y.1.2
          yo.r      := REAL32 ROUND( delta.y )
          Rectangulo ( xi.r, yi.r, xo.r, yo.r )
          AlmacenaPuntosSegmentoNA ( px1.2, py2, pxo, pyo, tamano,
                                     contorno.x, contorno.y )
          DibujaPuntos ( pxo, pyo )
          DibujaPuntos ( px1.2, py1 )
          delta.y := py2 - delta.y
          SEQ i=0 FOR tamano
            SEQ
              cuanto.x := ( pxo - contorno.x[i] ) + 1
              IniciaVectorBytes ( contorno.x[i], contorno.y[i],
                                  cuanto.x, color )
              IniciaVectorBytes ( px1.2, ( contorno.y[i] - delta.y ),
                                  cuanto.x, color )
              delta.y  := delta.y + 2
          --}}}
      ( delta.y.2.o < 0 ) AND ( delta.y.1.o >= 0 )
        SEQ
          --{{{  Proc
          xi.r      := REAL32 ROUND(px1.2)
          yi.r      := REAL32 ROUND(py1)
          xo.r      := REAL32 ROUND(pxo)
          yo.r      := REAL32 ROUND(pyo)
          Rectangulo ( xi.r, yi.r, xo.r, yo.r )
          AlmacenaPuntosSegmentoNA ( px1.2, py1, pxo, ( pyo - delta.y.1.2 ), t
                                     contorno.x, contorno.y )
          DibujaPuntos ( pxo, pyo )
          DibujaPuntos ( px1.2, py1 )
          delta.y   := pyo - py1
          SEQ i=0 FOR tamano
            SEQ
              cuanto.x := ( pxo - contorno.x[i] ) + 1
              IniciaVectorBytes ( contorno.x[i], contorno.y[i],
                                  cuanto.x, color )
              IniciaVectorBytes ( px1.2, ( contorno.y[i] + delta.y ),
                                  cuanto.x, color )
              delta.y  := delta.y + 2
          --}}}
      ( delta.y.2.o < 0 ) AND ( delta.y.1.o < 0 )
        SEQ
          --{{{  Proc
          ayu.y     := pyo - delta.y.1.2
          delta.y   := ayu.y - py1
          delta.y.r := REAL32 ROUND(delta.y)
          delta.x   := pxo - px1.2
          delta.x.r := REAL32 ROUND(delta.x)
          m         := delta.y.r / delta.x.r
          ObtenerXdeRecta ( px1.2, py1, m, pyo, ayu.x )
          ParalelogramoParaleloX ( ayu.x, pxo, pyo, px1.2, py1 )
          delta.y   := py1 - pyo
          AlmacenaPuntosSegmentoNA ( pxo, ayu.y, ayu.x, pyo, tamano,
                                     contorno.x, contorno.y )
          DibujaPuntos ( pxo, ayu.y )
          DibujaPuntos ( px1.2, py2 )
          ayu.x   := pxo - px1.2
          delta.y := py2 - ayu.y
          SEQ i=0 FOR tamano
            --{{{  Proc
            SEQ
```

```
                --{{{  Proc xf < xo < xi AND yi < yf < yo
                --}}}
              ( yf.ent.v < yi.ent.v ) AND ( yi.ent.v < yo.ent.v )
                SEQ
                  --{{{  Proc xf < xo < xi AND yf < yi < yo
                  --}}}
              ( yf.ent.v < yo.ent.v ) AND ( yo.ent.v < yi.ent.v )
                SEQ
                  --{{{  Proc xf < xo < xi AND yf < yo < yi
                  --}}}
              ( yi.ent.v < yo.ent.v ) AND ( yo.ent.v < yf.ent.v )
                SEQ
                  --{{{  Proc xf < xo < xi AND yi < yo < yf
                  --}}}
              ( yo.ent.v < yi.ent.v ) AND ( yi.ent.v < yf.ent.v )
                SEQ
                  --{{{  Proc xf < xo < xi AND yo < yi < yf
                  --}}}
              ( yo.ent.v < yf.ent.v ) AND ( yf.ent.v < yi.ent.v )
                SEQ
                  --{{{  Proc xf < xo < xi AND yo < yf < yi
                  --}}}
              TRUE
                SKIP

            --}}}
          ( xi.ent.v < xo.ent.v ) AND ( xo.ent.v < xf.ent.v )
            SEQ
              --{{{  Proc xi < xo < xf
              IF
                ( yi.ent.v < yf.ent.v ) AND ( yf.ent.v < yo.ent.v )
                  SEQ
                    --{{{  Proc xi < xo < xf AND yi < yf < yo
                    --}}}
                ( yf.ent.v < yi.ent.v ) AND ( yi.ent.v < yo.ent.v )
                  SEQ
                    --{{{  Proc xi < xo < xf AND yf < yi < yo
                    --}}}
                ( yf.ent.v < yo.ent.v ) AND ( yo.ent.v < yi.ent.v )
                  SEQ
                    --{{{  Proc xi < xo < xf AND yf < yo < yi
                    --}}}
                ( yi.ent.v < yo.ent.v ) AND ( yo.ent.v < yf.ent.v )
                  SEQ
                    --{{{  Proc xi < xo < xf AND yi < yo < yf
                    --}}}
                ( yo.ent.v < yi.ent.v ) AND ( yi.ent.v < yf.ent.v )
                  SEQ
                    --{{{  Proc xi < xo < xf AND yo < yi < yf
                    --}}}
                ( yo.ent.v < yf.ent.v ) AND ( yf.ent.v < yi.ent.v )
                  SEQ
                    --{{{  Proc xi < xo < xf AND yo < yf < yi
                    --}}}
                TRUE
                  SKIP

              --}}}
            ( xo.ent.v < xi.ent.v ) AND ( xi.ent.v < xf.ent.v )
              SEQ
                --{{{  Proc xo < xi < xf
                IF
                  ( yi.ent.v < yf.ent.v ) AND ( yf.ent.v < yo.ent.v )
                    SEQ
                      --{{{  Proc xo < xi < xf AND yi < yf < yo
                      --}}}
                  ( yf.ent.v < yi.ent.v ) AND ( yi.ent.v < yo.ent.v )
                    SEQ
                      --{{{  Proc xo < xi < xf AND yf < yi < yo
                      --}}}
                  ( yf.ent.v < yo.ent.v ) AND ( yo.ent.v < yi.ent.v )
                    SEQ
                      --{{{  Proc xo < xi < xf AND yf < yo < yi
                      --}}}
                  ( yi.ent.v < yo.ent.v ) AND ( yo.ent.v < yf.ent.v )
                    SEQ
                      --{{{  Proc xo < xi < xf AND yi < yo < yf
                      --}}}
                  ( yo.ent.v < yi.ent.v ) AND ( yi.ent.v < yf.ent.v )
                    SEQ
                      --{{{  Proc xo < xi < xf AND yo < yi < yf
                      --}}}
                  ( yo.ent.v < yf.ent.v ) AND ( yf.ent.v < yi.ent.v )
                    SEQ
                      --{{{  Proc xo < xi < xf AND yo < yf < yi
                      --}}}
                  TRUE
                    SKIP

                --}}}
              ( xo.ent.v < xf.ent.v ) AND ( xf.ent.v < xi.ent.v )
                SEQ
                  --{{{  Proc xo < xf < xi
                  IF
                    ( yi.ent.v < yf.ent.v ) AND ( yf.ent.v < yo.ent.v )
                      SEQ
                        --{{{  Proc xo < xf < xi AND yi < yf < yo
                        --}}}
                    ( yf.ent.v < yi.ent.v ) AND ( yi.ent.v < yo.ent.v )
                      SEQ
                        --{{{  Proc xo < xf < xi AND yf < yi < yo
                        --}}}
                    ( yf.ent.v < yo.ent.v ) AND ( yo.ent.v < yi.ent.v )
                      SEQ
                        --{{{  Proc xo < xf < xi AND yf < yo < yi
                        --}}}
                    ( yi.ent.v < yo.ent.v ) AND ( yo.ent.v < yf.ent.v )
                      SEQ
                        --{{{  Proc xo < xf < xi AND yi < yo < yf
                        --}}}
                    ( yo.ent.v < yi.ent.v ) AND ( yi.ent.v < yf.ent.v )
                      SEQ
                        --{{{  Proc xo < xf < xi AND yo < yi < yf
                        --}}}
                    ( yo.ent.v < yf.ent.v ) AND ( yf.ent.v < yi.ent.v )
                      SEQ
                        --{{{  Proc xo < xf < xi AND yo < yf < yi
                        --}}}
                    TRUE
                      SKIP

                  --}}}
                TRUE
                  SKIP
              --}}}
            TRUE
              SKIP
        --}}}
      TRUE
        SKIP
  --}}}
:
--}}}
--
--}}}
```

```
SEQ
  --{{{  Proc
  IF
    ( delta.y.i.f.v = 0 ) AND ( delta.x.f.o.v = 0 )
      Rectangulo ( xi.r, yi.r, xo.r, yo.r )
    ( xi.ent.v < xf.ent.v ) AND ( xf.ent.v < xo.ent.v )
      SEQ
        --{{{  Proc xi < xf < xo
        IF
          ( yi.ent.v < yf.ent.v ) AND ( yf.ent.v < yo.ent.v )
            SEQ
              --{{{  Proc xi < xf < xo AND yi < yf < yo
              --{{{  Asignaciones xi<xf<xo, yi<yf<yo
              --{{{  Enteros xi<xf<xo, yi<yf<yo
              xi.ent       := xi.ent.v
              yi.ent       := yi.ent.v
              xf.ent       := xf.ent.v
              yf.ent       := yf.ent.v
              xo.ent       := xo.ent.v
              yo.ent       := yo.ent.v
              --}}}
              --{{{  Reales xi<xf<xo, yi<yf<yo
              xi.r         := REAL32 ROUND(xi.ent)
              yi.r         := REAL32 ROUND(yi.ent)
              xf.r         := REAL32 ROUND(xf.ent)
              yf.r         := REAL32 ROUND(yf.ent)
              xo.r         := REAL32 ROUND(xo.ent)
              yo.r         := REAL32 ROUND(yo.ent)
              --}}}
              --}}}
              t1 ()
              --}}}
          ( yf.ent.v < yi.ent.v ) AND ( yi.ent.v < yo.ent.v )
            SEQ
              --{{{  Proc xi < xf < xo AND yf < yi < yo
              --{{{  Asignaciones xi<xf<xo, yf<yi<yo
              --{{{  Enteros xi<xf<xo, yf<yi<yo
              xi.ent       := xi.ent.v
              yi.ent       := yi.ent.v
              xf.ent       := xf.ent.v
              yf.ent       := yf.ent.v
              xo.ent       := xo.ent.v
              yo.ent       := yo.ent.v
              --}}}
              --{{{  Reales xi<xf<xo, yf<yi<yo
              xi.r         := REAL32 ROUND(xi.ent)
              yi.r         := REAL32 ROUND(yi.ent)
              xf.r         := REAL32 ROUND(xf.ent)
              yf.r         := REAL32 ROUND(yf.ent)
              xo.r         := REAL32 ROUND(xo.ent)
              yo.r         := REAL32 ROUND(yo.ent)
              --}}}
              --}}}
              t2 ()
              --}}}
          ( yf.ent.v < yo.ent.v ) AND ( yo.ent.v < yi.ent.v )
            SEQ
              --{{{  Proc xi < xf < xo AND yf < yo < yi
              --{{{  Asignaciones xi<xf<xo, yf<yo<yi
              --{{{  Enteros xi<xf<xo, yf<yo<yi
              xi.ent       := xi.ent.v
              yi.ent       := yi.ent.v
              xf.ent       := xf.ent.v
              yf.ent       := yf.ent.v
              xo.ent       := xo.ent.v
              yo.ent       := yo.ent.v
              --}}}
              --}}}
              --{{{  Reales xi<xf<xo, yf<yo<yi
              xi.r         := REAL32 ROUND(xi.ent)
              yi.r         := REAL32 ROUND(yi.ent)
              xf.r         := REAL32 ROUND(xf.ent)
              yf.r         := REAL32 ROUND(yf.ent)
              xo.r         := REAL32 ROUND(xo.ent)
              yo.r         := REAL32 ROUND(yo.ent)
              --}}}
              --}}}
              t3 ()
              --}}}
          ( yi.ent.v < yo.ent.v ) AND ( yo.ent.v < yf.ent.v )
            SEQ
              --{{{  Proc xi < xf < xo AND yi < yo < yf
              --}}}
          ( yo.ent.v < yi.ent.v ) AND ( yi.ent.v < yf.ent.v )
            SEQ
              --{{{  Proc xi < xf < xo AND yo < yi < yf
              --}}}
          ( yo.ent.v < yf.ent.v ) AND ( yf.ent.v < yi.ent.v )
            SEQ
              --{{{  Proc xi < xf < xo AND yo < yf < yi
              --}}}
          TRUE
            SKIP

        --}}}
    ( xf.ent.v < xi.ent.v ) AND ( xi.ent.v < xo.ent.v )
      SEQ
        --{{{  Proc xf < xi < xo
        IF
          ( yi.ent.v < yf.ent.v ) AND ( yf.ent.v < yo.ent.v )
            SEQ
              --{{{  Proc xf < xi < xo AND yi < yf < yo
              --}}}
          ( yf.ent.v < yi.ent.v ) AND ( yi.ent.v < yo.ent.v )
            SEQ
              --{{{  Proc xf < xi < xo AND yf < yi < yo
              --}}}
          ( yf.ent.v < yo.ent.v ) AND ( yo.ent.v < yi.ent.v )
            SEQ
              --{{{  Proc xf < xi < xo AND yf < yo < yi
              --}}}
          ( yi.ent.v < yo.ent.v ) AND ( yo.ent.v < yf.ent.v )
            SEQ
              --{{{  Proc xf < xi < xo AND yi < yo < yf
              --}}}
          ( yo.ent.v < yi.ent.v ) AND ( yi.ent.v < yf.ent.v )
            SEQ
              --{{{  Proc xf < xi < xo AND yo < yi < yf
              --}}}
          ( yo.ent.v < yf.ent.v ) AND ( yf.ent.v < yi.ent.v )
            SEQ
              --{{{  Proc xf < xi < xo AND yo < yf < yi
              --}}}
          TRUE
            SKIP

        --}}}
    ( xf.ent.v < xo.ent.v ) AND ( xo.ent.v < xi.ent.v )
      SEQ
        --{{{  Proc xf < xo < xi
        IF
          ( yi.ent.v < yf.ent.v ) AND ( yf.ent.v < yo.ent.v )
            SEQ
```

```
      TRUE
        SKIP
    --}}}
:
--}}}
--
--{{{  Circulos con Verificacion de limites
PROC Circulo2 ( VAL REAL32 xo, VAL REAL32 yo,
                VAL REAL32 r )
  --{{{  Declaraciones
  [(ancho*alto)]BYTE vec.pant RETYPES pantalla :
  [1024]INT contorno.x :
  [768]INT contorno.y :
  INT i, i.ayu, j, j.ayu, k, k.ayu :
  INT xo.ent, yo.ent, delta.l.x, delta.l.y :
  INT x.quad, y.quad, xq.mas.yq :
  INT r.ent, r.quad, cuanto.r :
  INT cuanto.x, cuanto.ayu, delta.ayu :
  INT x.arregla, y.arregla :
  INT condicion :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Asignaciones Iniciales
    xo.ent := INT ROUND( xo * escala )
    yo.ent := INT ROUND( yo * escala )
    r.ent  := INT ROUND(  r * escala )
    r.quad := r.ent * r.ent
    --}}}
    --{{{  Procesos
    SEQ
      --{{{  Asignaciones
      cuanto.r := r.ent + 1
      k        := 0
      k.ayu    := 0
      i        := xo.ent
      i.ayu    := 0
      x.quad   := 0
      --}}}
      SEQ
        --{{{  Procesos
        --{{{  Coge el Contorno
        SEQ j = ( yo.ent - r.ent ) FOR cuanto.r
          SEQ
            --{{{  Procesos
            j.ayu     := j - yo.ent
            y.quad    := j.ayu * j.ayu
            xq.mas.yq := x.quad + y.quad
            WHILE xq.mas.yq  <= r.quad
              SEQ
                --{{{  Procesos
                contorno.x[k] := xo.ent + i.ayu
                contorno.y[k] := yo.ent + j.ayu
                i             := i + 1
                k             := k + 1
                i.ayu         := i - xo.ent
                x.quad        := i.ayu * i.ayu
                xq.mas.yq     := x.quad + y.quad
                --}}}
            IF
              i > xo.ent
                SEQ
                  --{{{  Procesos
                  i      := i - 1
                  i.ayu  := i - xo.ent
                  x.quad := i.ayu * i.ayu
                  --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        TRUE
          SKIP
      --}}}
  --}}}
  --{{{  Dibuja el Circulo
  SEQ k.ayu = 0 FOR k
    SEQ
      --{{{  Procesos
      IF
        contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu]
          SEQ
            --{{{  Procesos
            delta.l.x   := contorno.x[k.ayu] - xo.ent
            delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
            cuanto.x    := ( 2 * delta.l.x ) + 1
            x.arregla   := xo.ent - delta.l.x
            y.arregla   := yo.ent + delta.l.y
            --{{{  Prueba se Esta en la Ventana Definida
            IF
              --{{{  Condicion 1
              ( x.arregla >= x.minimo.ent ) AND
              ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent ) AND
              ( contorno.y[k.ayu] >= y.minimo.ent ) AND
              ( y.arregla <= y.maximo.ent )
              --}}}
                SEQ
                  --{{{  Dibuja Linea
                  condicion  := 1
                  IniciaVectorBytes ( x.arregla, contorno.y[k.ayu],
                                      cuanto.x, color )
                  IniciaVectorBytes ( x.arregla, y.arregla, cuanto.x, co
                  --}}}
              --{{{  Condicion 2
              ( x.arregla < x.minimo.ent ) AND
              ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent ) AND
              ( contorno.y[k.ayu] >= y.minimo.ent ) AND
              ( y.arregla <= y.maximo.ent )
              --}}}
                SEQ
                  --{{{  Dibuja Linea
                  condicion  := 2
                  delta.ayu := x.minimo.ent - x.arregla
                  cuanto.x  := cuanto.x - delta.ayu
                  IniciaVectorBytes ( x.minimo.ent, contorno.y[k.ayu],
                                      cuanto.x, color )
                  IniciaVectorBytes ( x.minimo.ent, y.arregla, cuanto.x,
                  --}}}
              --{{{  Condicion 3
              ( x.arregla >= x.minimo.ent ) AND
              ( ((x.arregla+cuanto.x)-1) > x.maximo.ent ) AND
              ( contorno.y[k.ayu] >= y.minimo.ent ) AND
              ( y.arregla <= y.maximo.ent )
              --}}}
                SEQ
                  --{{{  Dibuja Linea
                  condicion  := 3
                  delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maxi
                  cuanto.x  := cuanto.x - delta.ayu
                  IniciaVectorBytes ( x.arregla, contorno.y[k.ayu],
                                      cuanto.x, color )
                  IniciaVectorBytes ( x.arregla, y.arregla, cuanto.x, co
                  --}}}
              --{{{  Condicion 4
              ( x.arregla < x.minimo.ent ) AND
              ( ((x.arregla+cuanto.x)-1) > x.maximo.ent ) AND
```

```
--
--}}}F
--
--{{{  Curvil - Dibujar Figuras Circulares
--
--{{{  Dibujar Circulos/Semicirculos/Cuarto de Circulos/Octavo de Circulos
--
--{{{F Dibujar Circulos
--:::F Dicirc01.tsr
--
--{{{  Circulos
PROC Circulo ( VAL REAL32 xo, VAL REAL32 yo,
               VAL REAL32 r )
  --{{{  Declaraciones
  [(ancho*alto)]BYTE vec.pant RETYPES pantalla :
  [1024]INT contorno.x :
  [768]INT contorno.y :
  INT i, i.ayu, j, j.ayu, k, k.ayu :
  INT xo.ent, yo.ent, delta.l.x, delta.l.y :
  INT lim.inf.x.ent, lim.inf.y.ent, lim.sup.x.ent, lim.sup.y.ent :
  INT x.quad, y.quad, xq.mas.yq :
  INT r.ent, r.quad, cuanto.r :
  INT cuanto.x :
  INT x.arregla, y.arregla :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Procesos
    xo.ent := INT ROUND( xo * escala )
    yo.ent := INT ROUND( yo * escala )
    r.ent  := INT ROUND(  r * escala )
    r.quad := r.ent * r.ent
    --}}}
    --{{{  Arregla Limites de x
    IF
      ( xo.ent-r.ent ) >= 0
        lim.inf.x.ent := xo.ent - r.ent
      TRUE
        lim.inf.x.ent := x.minimo.ent
    IF
      ( xo.ent+r.ent ) <= x.maximo.ent
        lim.sup.x.ent := xo.ent + r.ent
      TRUE
        lim.sup.x.ent := x.maximo.ent
    --}}}
    --{{{  Arregla Limites de y
    IF
      ( yo.ent-r.ent ) >= 0
        lim.inf.y.ent := yo.ent - r.ent
      TRUE
        lim.inf.y.ent := y.minimo.ent
    IF
      ( yo.ent+r.ent ) <= y.maximo.ent
        lim.sup.y.ent := yo.ent + r.ent
      TRUE
        lim.sup.y.ent := y.maximo.ent
    --}}}
    IF
      ( lim.sup.x.ent <= x.maximo.ent ) AND ( lim.inf.x.ent >= x.minimo.ent )
        ( lim.sup.y.ent <= y.maximo.ent ) AND ( lim.inf.y.ent >= y.minimo.ent
        --{{{  Procesos
        SEQ
          --{{{  Asignaciones
          cuanto.r := r.ent + 1
          k        := 0
          k.ayu    := 0
          i        := xo.ent
          i.ayu    := 0
          x.quad   := 0
          --}}}
          SEQ
            --{{{  Procesos
            --{{{  Coge el Contorno
            --SEQ j = lim.inf.y.ent FOR cuanto.r
            SEQ j = ( yo.ent - r.ent ) FOR cuanto.r
              SEQ
                --{{{  Procesos
                j.ayu      := j - yo.ent
                y.quad     := j.ayu * j.ayu
                xq.mas.yq  := x.quad + y.quad
                WHILE xq.mas.yq  <= r.quad
                  SEQ
                    --{{{  Procesos
                    contorno.x[k] := xo.ent + i.ayu
                    contorno.y[k] := yo.ent + j.ayu
                    i             := i + 1
                    k             := k + 1
                    i.ayu         := i - xo.ent
                    x.quad        := i.ayu * i.ayu
                    xq.mas.yq     := x.quad + y.quad
                    --}}}
                  IF
                    i > xo.ent
                      SEQ
                        --{{{  Procesos
                        i       := i - 1
                        i.ayu   := i - xo.ent
                        x.quad  := i.ayu * i.ayu
                        --}}}
                    TRUE
                      SKIP
                --}}}
            --}}}
            --{{{  Dibuja el Circulo
            SEQ k.ayu = 0 FOR k
              SEQ
                --{{{  Procesos
                IF
                  contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu]
                    SEQ
                      --{{{  Procesos
                      delta.l.x    := contorno.x[k.ayu] - xo.ent
                      delta.l.y    := - ( contorno.y[k.ayu] - yo.ent )
                      cuanto.x     := ( 2 * delta.l.x ) + 1
                      x.arregla    := xo.ent - delta.l.x
                      IniciaVectorBytes ( x.arregla, contorno.y[k.ayu],
                                          cuanto.x, color )
                      y.arregla    := yo.ent + delta.l.y
                      IniciaVectorBytes ( x.arregla, y.arregla, cuanto.x, colo
                      --}}}
                  TRUE
                    SKIP
                --}}}
            --}}}
            --{{{  Dibuja la Ultima Linea
            x.arregla := xo.ent - r.ent
            cuanto.x := ( 2 * r.ent ) + 1
            IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
            --}}}
            --}}}

--}}}
```

```
                    TRUE
                      SKIP
                --}}}
            --}}}
          --}}}
        TRUE
          SKIP
    --}}}
:

--}}}
--
--}}}F
--
--{{{F Dibujar Semicirculos
--:::F Diseci01.tsr
--
--{{{  Semicirculos
PROC SemiCirculo ( VAL REAL32 xo, VAL REAL32 yo,
                   VAL REAL32 r , VAL INT sup )
  --{{{  Declaraciones
  [(ancho*alto)]BYTE vec.pant RETYPES pantalla :
  [1024]INT contorno.x :
  [768]INT contorno.y :
  INT i, i.ayu, j, j.ayu, k, k.ayu :
  INT xo.ent, yo.ent, delta.l.x, delta.l.y :
  INT lim.inf.x.ent, lim.inf.y.ent, lim.sup.x.ent, lim.sup.y.ent :
  INT x.quad, y.quad, xq.mas.yq :
  INT r.ent, r.quad, cuanto.r :
  INT cuanto.x :
  INT x.arregla, y.arregla :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Procesos
    xo.ent := INT ROUND( xo * escala )
    yo.ent := INT ROUND( yo * escala )
    r.ent  := INT ROUND(  r * escala )
    r.quad := r.ent * r.ent
    --}}}
    --{{{  Arregla Limites de x
    IF
      ( xo.ent-r.ent ) >= 0
        lim.inf.x.ent := xo.ent - r.ent
      TRUE
        lim.inf.x.ent := x.minimo.ent
    IF
      ( xo.ent+r.ent ) <= x.maximo.ent
        lim.sup.x.ent := xo.ent + r.ent
      TRUE
        lim.sup.x.ent := x.maximo.ent
    --}}}
    --{{{  Arregla Limites de y
    IF
      ( yo.ent-r.ent ) >= 0
        lim.inf.y.ent := yo.ent - r.ent
      TRUE
        lim.inf.y.ent := y.minimo.ent
    IF
      ( yo.ent+r.ent ) <= y.maximo.ent
        lim.sup.y.ent := yo.ent + r.ent
      TRUE
        lim.sup.y.ent := y.maximo.ent
    --}}}
    IF
      ( lim.sup.x.ent <= x.maximo.ent ) AND ( lim.inf.x.ent >= x.minimo.ent )
      ( lim.sup.y.ent <= y.maximo.ent ) AND ( lim.inf.y.ent >= y.minimo.ent
    --{{{  Procesos
    SEQ
      --{{{  Asignaciones
      cuanto.r := r.ent + 1
      k        := 0
      k.ayu    := 0
      i        := xo.ent
      i.ayu    := 0
      x.quad   := 0
      --}}}
      SEQ
        --{{{  Procesos
        --{{{  Coge el Contorno
        --SEQ j = lim.inf.y.ent FOR cuanto.r
        SEQ j = ( yo.ent - r.ent ) FOR cuanto.r
          SEQ
            --{{{  Procesos
            j.ayu      := j - yo.ent
            y.quad     := j.ayu * j.ayu
            xq.mas.yq  := x.quad + y.quad
            WHILE xq.mas.yq  <= r.quad
              SEQ
                --{{{  Procesos
                contorno.x[k] := xo.ent + i.ayu
                contorno.y[k] := yo.ent + j.ayu
                i             := i + 1
                k             := k + 1
                i.ayu         := i - xo.ent
                x.quad        := i.ayu * i.ayu
                xq.mas.yq     := x.quad + y.quad
                --}}}
              IF
                i > xo.ent
                  SEQ
                    --{{{  Procesos
                    i       := i - 1
                    i.ayu   := i - xo.ent
                    x.quad  := i.ayu * i.ayu
                    --}}}
                TRUE
                  SKIP
              --}}}
        --}}}
        --{{{  Dibuja el Semicirculo
        IF
          sup = 1
            --{{{  Dibuja el Semicirculo Inferior
            SEQ k.ayu = 0 FOR k
              SEQ
                --{{{  Procesos
                IF
                  ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
                    ( contorno.y[(k.ayu + 1)] <= yo.ent )
                    SEQ
                      --{{{  Procesos
                      delta.l.x   := contorno.x[k.ayu] - xo.ent
                      delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
                      cuanto.x    := ( 2 * delta.l.x ) + 1
                      x.arregla   := xo.ent - delta.l.x
                      IniciaVectorBytes ( x.arregla, contorno.y[k.ayu],
                                          cuanto.x, color )
                      --}}}
                  TRUE
                    SKIP
                --}}}
```

```
                ( contorno.y[k.ayu] >= y.minimo.ent ) AND
                ( y.arregla <= y.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 4
                    delta.ayu := x.minimo.ent - x.arregla
                    cuanto.x  := cuanto.x - delta.ayu
                    delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maxi
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.minimo.ent, contorno.y[k.ayu],
                                        cuanto.x, color )
                    IniciaVectorBytes ( x.minimo.ent, y.arregla, cuanto.x,
                    --}}}
                TRUE
                  SKIP
              --}}}
              --}}}
          TRUE
            SKIP
        --}}}
    --}}}
    --{{{  Dibuja la Ultima Linea
    IF
      condicion = 1
        SEQ
          --{{{  Dibuja Linea
          x.arregla := xo.ent - r.ent
          cuanto.x := ( 2 * r.ent ) + 1
          IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
          --}}}
      condicion = 2
        SEQ
          --{{{  Dibuja Linea
          x.arregla := xo.ent - r.ent
          cuanto.x := ( 2 * r.ent ) + 1
          delta.ayu := x.minimo.ent - x.arregla
          cuanto.x  := cuanto.x - delta.ayu
          IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color )
          --}}}
      condicion = 3
        SEQ
          --{{{  Dibuja Linea
          x.arregla := xo.ent - r.ent
          cuanto.x := ( 2 * r.ent ) + 1
          delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.ent
          cuanto.x  := cuanto.x - delta.ayu
          IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
          --}}}
      condicion = 4
        SEQ
          --{{{  Dibuja Linea
          x.arregla := xo.ent - r.ent
          cuanto.x := ( 2 * r.ent ) + 1
          delta.ayu := x.minimo.ent - x.arregla
          cuanto.x  := cuanto.x - delta.ayu
          delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.ent
          cuanto.x  := cuanto.x - delta.ayu
          IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color )
          --}}}
      TRUE
        SKIP
    --}}}
    --}}}
--}}}
--}}}
```

```
:
--}}}
--
--{{{  Dibujar Circulos ( Lento )
PROC Circulo1 ( VAL REAL32 xo, VAL REAL32 yo,
                VAL REAL32 r )
  --{{{  Procesos
  INT j, j.ayu, k, k.ayu :
  INT xo.ent, yo.ent, delta.x.ent, delta.y.ent :
  INT lim.inf.x.ent, lim.inf.y.ent, lim.sup.x.ent, lim.sup.y.ent :
  INT xo.quad, yo.quad, xoq.mas.yoq :
  INT r.ent, r.quad, diam :
  INT cuanto.x, cuanto.y :
  SEQ
    --{{{  Procesos
    xo.ent := INT ROUND( xo * escala )
    yo.ent := INT ROUND( yo * escala )
    r.ent  := INT ROUND(  r * escala )
    r.quad := r.ent * r.ent
    --}}}
    --{{{  Arregla Limites de x
    IF
      ( xo.ent-r.ent ) >= 0
        lim.inf.x.ent := xo.ent - r.ent
      TRUE
        lim.inf.x.ent := 0
    IF
      ( xo.ent+r.ent ) <= x.maximo.ent
        lim.sup.x.ent := xo.ent + r.ent
      TRUE
        lim.sup.x.ent := x.maximo.ent
    delta.x.ent := lim.sup.x.ent - lim.inf.x.ent
    cuanto.x := delta.x.ent + 1
    --}}}
    --{{{  Arregla Limites de y
    IF
      ( yo.ent-r.ent ) >= 0
        lim.inf.y.ent := yo.ent - r.ent
      TRUE
        lim.inf.y.ent := 0
    IF
      ( yo.ent+r.ent ) <= y.maximo.ent
        lim.sup.y.ent := yo.ent + r.ent
      TRUE
        lim.sup.y.ent := y.maximo.ent
    delta.y.ent := lim.sup.y.ent - lim.inf.y.ent
    cuanto.y := delta.y.ent + 1
    --}}}
    IF
      ( lim.inf.x.ent <= x.maximo.ent ) AND ( lim.sup.x.ent >=0 ) AND
                    ( lim.inf.y.ent <= y.maximo.ent ) AND ( lim.sup.y.ent >=0
        --{{{  Procesos
        SEQ j = lim.inf.x.ent FOR  cuanto.x
          --{{{  Procesos
          SEQ
            j.ayu   := j - xo.ent
            xo.quad := j.ayu * j.ayu
            SEQ k = lim.inf.y.ent FOR cuanto.y
              --{{{  Procesos
              SEQ
                k.ayu       := k - yo.ent
                yo.quad     := k.ayu * k.ayu
                xoq.mas.yoq := xo.quad + yo.quad
                IF
                  xoq.mas.yoq <= r.quad
                    DibujaPuntos ( j, k )
```

```
                ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 2
                    delta.ayu := x.minimo.ent - x.arregla
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.minimo.ent, contorno.y[k.ayu
                                        cuanto.x, color )
                    --}}}
                --{{{  Condicion 3
                ( x.arregla >= x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 3
                    delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.arregla, contorno.y[k.ayu],
                                        cuanto.x, color )
                    --}}}
                --{{{  Condicion 4
                ( x.arregla < x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 4
                    delta.ayu := x.minimo.ent - x.arregla
                    cuanto.x  := cuanto.x - delta.ayu
                    delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.minimo.ent, contorno.y[k.ayu
                                        cuanto.x, color )
                    --}}}
                TRUE
                  SKIP
              --}}}
              --}}}
          TRUE
            SKIP
        --}}}
    --}}}
  sup := 0
    --{{{  Dibuja el Semicirculo Superior
    SEQ k.ayu = 0 FOR k
      SEQ
        --{{{  Procesos
        IF
          ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
          ( contorno.y[(k.ayu + 1)] <= yo.ent )
          SEQ
            --{{{  Procesos
            delta.l.x    := contorno.x[k.ayu] - xo.ent
            delta.l.y    := - ( contorno.y[k.ayu] - yo.ent )
            cuanto.x     := ( 2 * delta.l.x ) + 1
            x.arregla    := xo.ent - delta.l.x
            y.arregla    := yo.ent + delta.l.y
            --{{{  Prueba se Esta en la Ventana Definida
            IF
              --{{{  Condicion 1
              ( x.arregla >= x.minimo.ent ) AND
              ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
              --}}}
                SEQ
                  --{{{  Dibuja Linea
                  condicion  := 1
                  IniciaVectorBytes ( x.arregla, y.arregla, cuanto.x
                  --}}}
              --{{{  Condicion 2
              ( x.arregla < x.minimo.ent ) AND
              ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
              --}}}
                SEQ
                  --{{{  Dibuja Linea
                  condicion  := 2
                  delta.ayu := x.minimo.ent - x.arregla
                  cuanto.x  := cuanto.x - delta.ayu
                  IniciaVectorBytes ( x.minimo.ent, y.arregla, cuant
                  --}}}
              --{{{  Condicion 3
              ( x.arregla >= x.minimo.ent ) AND
              ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
              --}}}
                SEQ
                  --{{{  Dibuja Linea
                  condicion  := 3
                  delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.
                  cuanto.x  := cuanto.x - delta.ayu
                  IniciaVectorBytes ( x.arregla, y.arregla, cuanto.x
                  --}}}
              --{{{  Condicion 4
              ( x.arregla < x.minimo.ent ) AND
              ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
              --}}}
                SEQ
                  --{{{  Dibuja Linea
                  condicion  := 4
                  delta.ayu := x.minimo.ent - x.arregla
                  cuanto.x  := cuanto.x - delta.ayu
                  delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.
                  cuanto.x  := cuanto.x - delta.ayu
                  IniciaVectorBytes ( x.minimo.ent, y.arregla, cuant
                  --}}}
              TRUE
                SKIP
            --}}}
            --}}}
        TRUE
          SKIP
      --}}}
  --}}}
  --{{{  Dibuja la Ultima Linea
  IF
    condicion = 1
      SEQ
        --{{{  Dibuja Linea
        x.arregla := xo.ent - r.ent
        cuanto.x := ( 2 * r.ent ) + 1
        IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
        --}}}
    condicion = 2
      SEQ
        --{{{  Dibuja Linea
        x.arregla := xo.ent - r.ent
        cuanto.x := ( 2 * r.ent ) + 1
        delta.ayu := x.minimo.ent - x.arregla
        cuanto.x  := cuanto.x - delta.ayu
```

```
            --}}}
          sup = 0
            --{{{  Dibuja el Semicirculo Superior
            SEQ k.ayu = 0 FOR k
              SEQ
                --{{{  Procesos
                IF
                  ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
                    ( contorno.y[(k.ayu + 1)] <= yo.ent )
                    SEQ
                      --{{{  Procesos
                      delta.l.x   := contorno.x[k.ayu] - xo.ent
                      delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
                      cuanto.x    := ( 2 * delta.l.x ) + 1
                      x.arregla   := xo.ent - delta.l.x
                      y.arregla   := yo.ent + delta.l.y
                      IniciaVectorBytes ( x.arregla, y.arregla, cuanto.x,
                      --}}}
                  TRUE
                    SKIP
                --}}}
            --}}}
          TRUE
            SKIP
        --}}}
        --{{{  Dibuja la Ultima Linea
        x.arregla := xo.ent - r.ent
        cuanto.x := ( 2 * r.ent ) + 1
        IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
        --}}}
        --}}}
      --}}}
    TRUE
      SKIP
  --}}}
:
--}}}
--
--{{{  Semicirculos con Verificacion de Limites
PROC SemiCirculo1 ( VAL REAL32 xo, VAL REAL32 yo,
                    VAL REAL32 r , VAL INT sup )
  --{{{  Declaraciones
  [(ancho*alto)]BYTE vec.pant RETYPES pantalla :
  [1024]INT contorno.x :
  [768]INT contorno.y :
  INT i, i.ayu, j, j.ayu, k, k.ayu :
  INT xo.ent, yo.ent, delta.l.x, delta.l.y :
  INT x.quad, y.quad, xq.mas.yq :
  INT r.ent, r.quad, cuanto.r :
  INT cuanto.x, delta.ayu, cuanto.ayu :
  INT x.arregla, y.arregla :
  INT condicion :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Asignaciones Iniciales
    xo.ent := INT ROUND( xo * escala )
    yo.ent := INT ROUND( yo * escala )
    r.ent  := INT ROUND(  r * escala )
    r.quad := r.ent * r.ent
    --}}}
    --{{{  Procesos
    SEQ
      --{{{  Asignaciones
      cuanto.r := r.ent + 1
      k        := 0
      k.ayu    := 0
      i        := xo.ent
      i.ayu    := 0
      x.quad   := 0
      --}}}
      SEQ
        --{{{  Procesos
        --{{{  Coge el Contorno
        SEQ j = ( yo.ent - r.ent ) FOR cuanto.r
          SEQ
            --{{{  Procesos
            j.ayu      := j - yo.ent
            y.quad     := j.ayu * j.ayu
            xq.mas.yq  := x.quad + y.quad
            WHILE xq.mas.yq  <= r.quad
              SEQ
                --{{{  Procesos
                contorno.x[k] := xo.ent + i.ayu
                contorno.y[k] := yo.ent + j.ayu
                i             := i + 1
                k             := k + 1
                i.ayu         := i - xo.ent
                x.quad        := i.ayu * i.ayu
                xq.mas.yq     := x.quad + y.quad
                --}}}
            IF
              i > xo.ent
                SEQ
                  --{{{  Procesos
                  i       := i - 1
                  i.ayu   := i - xo.ent
                  x.quad  := i.ayu * i.ayu
                  --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        --{{{  Dibuja el Semicirculo
        IF
          sup = 1
            --{{{  Dibuja el Semicirculo Inferior
            SEQ k.ayu = 0 FOR k
              SEQ
                --{{{  Procesos
                IF
                  ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
                    ( contorno.y[(k.ayu + 1)] <= yo.ent )
                    SEQ
                      --{{{  Procesos
                      delta.l.x   := contorno.x[k.ayu] - xo.ent
                      cuanto.x    := ( 2 * delta.l.x ) + 1
                      x.arregla   := xo.ent - delta.l.x
                      --{{{  Prueba se Esta en la Ventana Definida
                      IF
                        --{{{  Condicion 1
                        ( x.arregla >= x.minimo.ent ) AND
                        ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                        --}}}
                          SEQ
                            --{{{  Dibuja Linea
                            condicion  := 1
                            IniciaVectorBytes ( x.arregla, contorno.y[k.ayu],
                                                cuanto.x, color )
                            --}}}
                        --{{{  Condicion 2
                        ( x.arregla < x.minimo.ent ) AND
```

```
      SEQ
        --{{{  Dibuja el Cuadrante 3
        SEQ k.ayu = 0 FOR k
          SEQ
            --{{{  Procesos
            IF
              ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
                ( contorno.y[(k.ayu + 1)] <= yo.ent )
                SEQ
                  --{{{  Procesos
                  delta.1.x    := contorno.x[k.ayu] - xo.ent
                  cuanto.x     := delta.1.x + 1
                  x.arregla    := xo.ent - delta.1.x
                  IniciaVectorBytes ( x.arregla, contorno.y[k.ayu],
                                      cuanto.x, color )
                  --}}}
              TRUE
                SKIP
            --}}}
        --{{{  Dibuja la Ultima Linea
        x.arregla := xo.ent - r.ent
        cuanto.x  := r.ent + 1
        IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
        --}}}
        --}}}
      cuadrante = 4
        SEQ
          --{{{  Dibuja el Cuadrante 4
          SEQ k.ayu = 0 FOR k
            SEQ
              --{{{  Procesos
              IF
                ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
                  ( contorno.y[(k.ayu + 1)] <= yo.ent )
                  SEQ
                    --{{{  Procesos
                    delta.1.x    := contorno.x[k.ayu] - xo.ent
                    cuanto.x     := delta.1.x + 1
                    IniciaVectorBytes ( xo.ent, contorno.y[k.ayu],
                                        cuanto.x, color )
                    --}}}
                TRUE
                  SKIP
              --}}}
          --{{{  Dibuja la Ultima Linea
          cuanto.x := r.ent + 1
          IniciaVectorBytes ( xo.ent , yo.ent, cuanto.x, color )
          --}}}
          --}}}
        cuadrante = 1
          SEQ
            --{{{  Dibuja el Cuadrante 1
            SEQ k.ayu = 0 FOR k
              SEQ
                --{{{  Procesos
                IF
                  ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
                    ( contorno.y[(k.ayu + 1)] <= yo.ent )
                    SEQ
                      --{{{  Procesos
                      delta.1.x    := contorno.x[k.ayu] - xo.ent
                      delta.1.y    := - ( contorno.y[k.ayu] - yo.ent )
                      cuanto.x     := delta.1.x + 1
                      y.arregla    := yo.ent + delta.1.y
                      IniciaVectorBytes ( xo.ent, y.arregla, cuanto.x, c
                      --}}}
                  TRUE
                    SKIP
                --}}}
            --{{{  Dibuja la Ultima Linea
            cuanto.x := r.ent + 1
            IniciaVectorBytes ( xo.ent , yo.ent, cuanto.x, color )
            --}}}
            --}}}
          cuadrante = 2
            SEQ
              --{{{  Dibuja el Cuadrante 2
              SEQ k.ayu = 0 FOR k
                SEQ
                  --{{{  Procesos
                  IF
                    ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
                      ( contorno.y[(k.ayu + 1)] <= yo.ent )
                      SEQ
                        --{{{  Procesos
                        delta.1.x    := contorno.x[k.ayu] - xo.ent
                        delta.1.y    := - ( contorno.y[k.ayu] - yo.ent )
                        cuanto.x     := delta.1.x + 1
                        x.arregla    := xo.ent - delta.1.x
                        y.arregla    := yo.ent + delta.1.y
                        IniciaVectorBytes ( x.arregla, y.arregla, cuanto.x
                        --}}}
                    TRUE
                      SKIP
                  --}}}
              --{{{  Dibuja la Ultima Linea
              x.arregla := xo.ent - r.ent
              cuanto.x  := r.ent + 1
              IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
              --}}}
              --}}}
            TRUE
              SKIP
          --}}}
          --}}}
      --}}}
    TRUE
      SKIP
  --}}}
:
--}}}
--
--{{{  Cuarto de circulos con Verificacion de Limites
PROC CuartoDeCirculo1 ( VAL REAL32 xo, VAL REAL32 yo,
                        VAL REAL32 r , VAL INT cuadrante )
  --{{{  Declaraciones
  [(ancho*alto)]BYTE vec.pant RETYPES pantalla :
  [1024]INT contorno.x :
  [768]INT contorno.y :
  INT i, i.ayu, j, j.ayu, k, k.ayu :
  INT xo.ent, yo.ent, delta.1.x, delta.1.y :
  INT lim.inf.x.ent, lim.inf.y.ent, lim.sup.x.ent, lim.sup.y.ent :
  INT x.quad, y.quad, xq.mas.yq :
  INT r.ent, r.quad, cuanto.r :
  INT cuanto.x, delta.ayu, cuanto.ayu :
  INT x.arregla, y.arregla :
  INT condicion :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Asignaciones Iniciales
    xo.ent := INT ROUND( xo * escala )
```

```
            IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color )
            --}}}
          condicion = 3
            SEQ
              --{{{  Dibuja Linea
              x.arregla := xo.ent - r.ent
              cuanto.x := ( 2 * r.ent ) + 1
              delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.ent
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
              --}}}
          condicion = 4
            SEQ
              --{{{  Dibuja Linea
              x.arregla := xo.ent - r.ent
              cuanto.x := ( 2 * r.ent ) + 1
              delta.ayu := x.minimo.ent - x.arregla
              cuanto.x  := cuanto.x - delta.ayu
              delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.ent
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color )
              --}}}
          TRUE
            SKIP
        --}}}
        --}}}
    --}}}
    --}}}
:
--}}}
--
--}}}F
--
--{{{F Dibujar Cuarto de Circulos
--:::F Dicuac01.tsr
--
--{{{  Cuarto de circulos
PROC CuartoDeCirculo ( VAL REAL32 xo, VAL REAL32 yo,
                       VAL REAL32 r , VAL INT cuadrante )
  --{{{  Declaraciones
  [(ancho*alto)]BYTE vec.pant RETYPES pantalla :
  [1024]INT contorno.x :
  [768]INT contorno.y :
  INT i, i.ayu, j, j.ayu, k, k.ayu :
  INT xo.ent, yo.ent, delta.l.x, delta.l.y :
  INT lim.inf.x.ent, lim.inf.y.ent, lim.sup.x.ent, lim.sup.y.ent :
  INT x.quad, y.quad, xq.mas.yq :
  INT r.ent, r.quad, cuanto.r :
  INT cuanto.x :
  INT x.arregla, y.arregla :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Procesos
    xo.ent      := INT ROUND( xo * escala )
    yo.ent      := INT ROUND( yo * escala )
    r.ent       := INT ROUND(  r * escala )
    r.quad := r.ent * r.ent
    --}}}
    --{{{  Arregla Limites de x
    IF
      ( xo.ent-r.ent ) >= 0
        lim.inf.x.ent := xo.ent - r.ent
      TRUE
        lim.inf.x.ent := x.minimo.ent
    IF
      ( xo.ent+r.ent ) <= x.maximo.ent
        lim.sup.x.ent := xo.ent + r.ent
      TRUE
        lim.sup.x.ent := x.maximo.ent
    --}}}
    --{{{  Arregla Limites de y
    IF
      ( yo.ent-r.ent ) >= 0
        lim.inf.y.ent := yo.ent - r.ent
      TRUE
        lim.inf.y.ent := y.minimo.ent
    IF
      ( yo.ent+r.ent ) <= y.maximo.ent
        lim.sup.y.ent := yo.ent + r.ent
      TRUE
        lim.sup.y.ent := y.maximo.ent
    --}}}
    IF
      ( lim.sup.x.ent <= x.maximo.ent ) AND ( lim.inf.x.ent >= x.minimo.ent )
        ( lim.sup.y.ent <= y.maximo.ent ) AND ( lim.inf.y.ent >= y.minimo.ent
        --{{{  Procesos
        SEQ
          --{{{  Asignaciones
          cuanto.r := r.ent + 1
          k        := 0
          k.ayu    := 0
          i        := xo.ent
          i.ayu    := 0
          x.quad   := 0
          --}}}
          SEQ
            --{{{  Procesos
            --{{{  Coge el Contorno
            --SEQ j = lim.inf.y.ent FOR cuanto.r
            SEQ j = ( yo.ent - r.ent ) FOR cuanto.r
              SEQ
                --{{{  Procesos
                j.ayu      := j - yo.ent
                y.quad     := j.ayu * j.ayu
                xq.mas.yq  := x.quad + y.quad
                WHILE xq.mas.yq  <= r.quad
                  SEQ
                    --{{{  Procesos
                    contorno.x[k] := xo.ent + i.ayu
                    contorno.y[k] := yo.ent + j.ayu
                    i             := i + 1
                    k             := k + 1
                    i.ayu         := i - xo.ent
                    x.quad        := i.ayu * i.ayu
                    xq.mas.yq     := x.quad + y.quad
                    --}}}
                  IF
                    i > xo.ent
                      SEQ
                        --{{{  Procesos
                        i       := i - 1
                        i.ayu   := i - xo.ent
                        x.quad  := i.ayu * i.ayu
                        --}}}
                    TRUE
                      SKIP
                --}}}
            --}}}
            --{{{  Dibuja el Cuarto de Circulo
            IF
              cuadrante = 3
```

```
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
                --}}}
            condicion = 3
              SEQ
                --{{{  Dibuja Linea
                delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
                --}}}
            condicion = 4
              SEQ
                --{{{  Dibuja Linea
                delta.ayu := x.minimo.ent - x.arregla
                cuanto.x  := cuanto.x - delta.ayu
                delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
                --}}}
            TRUE
              SKIP
          --}}}
          --}}}
      cuadrante = 4
        SEQ
          --{{{  Dibuja el Cuadrante 4
          SEQ k.ayu = 0 FOR k
            SEQ
              --{{{  Procesos
              IF
                ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
                  ( contorno.y[(k.ayu + 1)] <= yo.ent )
                  SEQ
                    --{{{  Procesos
                    delta.l.x   := contorno.x[k.ayu] - xo.ent
                    cuanto.x    := delta.l.x + 1
                    x.arregla   := xo.ent
                    --{{{  Prueba se Esta en la Ventana Definida
                    IF
                      --{{{  Condicion 1
                      ( x.arregla >= x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 1
                          IniciaVectorBytes ( x.arregla, contorno.y[k.ayu]
                                              cuanto.x, color )
                          --}}}
                      --{{{  Condicion 2
                      ( x.arregla < x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 2
                          delta.ayu := x.minimo.ent - x.arregla
                          cuanto.x  := cuanto.x - delta.ayu
                          IniciaVectorBytes ( x.minimo.ent, contorno.y[k.a
                                              cuanto.x, color )
                          --}}}
                      --{{{  Condicion 3
                      ( x.arregla >= x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 3
                          delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                          cuanto.x  := cuanto.x - delta.ayu
                          IniciaVectorBytes ( x.arregla, contorno.y[k.ayu]
                                              cuanto.x, color )
                          --}}}
                      --{{{  Condicion 4
                      ( x.arregla < x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 4
                          delta.ayu := x.minimo.ent - x.arregla
                          cuanto.x  := cuanto.x - delta.ayu
                          delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                          cuanto.x  := cuanto.x - delta.ayu
                          IniciaVectorBytes ( x.minimo.ent, contorno.y[k.a
                                              cuanto.x, color )
                          --}}}
                      TRUE
                        SKIP
                    --}}}
                    --}}}
                TRUE
                  SKIP
              --}}}
          --{{{  Dibuja la Ultima Linea
          x.arregla := xo.ent
          cuanto.x := r.ent + 1
          IF
            condicion = 1
              SEQ
                --{{{  Dibuja Linea
                IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
                --}}}
            condicion = 2
              SEQ
                --{{{  Dibuja Linea
                delta.ayu := x.minimo.ent - x.arregla
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
                --}}}
            condicion = 3
              SEQ
                --{{{  Dibuja Linea
                delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
                --}}}
            condicion = 4
              SEQ
                --{{{  Dibuja Linea
                delta.ayu := x.minimo.ent - x.arregla
                cuanto.x  := cuanto.x - delta.ayu
                delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
                --}}}
            TRUE
              SKIP
          --}}}
          --}}}
      cuadrante = 1
        SEQ
```

```
yo.ent := INT ROUND( yo * escala )
r.ent  := INT ROUND(  r * escala )
r.quad := r.ent * r.ent
--}}}
--{{{  Procesos
SEQ
  --{{{  Asignaciones
  cuanto.r := r.ent + 1
  k        := 0
  k.ayu    := 0
  i        := xo.ent
  i.ayu    := 0
  x.quad   := 0
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Coge el Contorno
    SEQ j = ( yo.ent - r.ent ) FOR cuanto.r
      SEQ
        --{{{  Procesos
        j.ayu      := j - yo.ent
        y.quad     := j.ayu * j.ayu
        xq.mas.yq  := x.quad + y.quad
        WHILE xq.mas.yq  <= r.quad
          SEQ
            --{{{  Procesos
            contorno.x[k] := xo.ent + i.ayu
            contorno.y[k] := yo.ent + j.ayu
            i             := i + 1
            k             := k + 1
            i.ayu         := i - xo.ent
            x.quad        := i.ayu * i.ayu
            xq.mas.yq     := x.quad + y.quad
            --}}}
        IF
          i > xo.ent
            SEQ
              --{{{  Procesos
              i       := i - 1
              i.ayu   := i - xo.ent
              x.quad  := i.ayu * i.ayu
              --}}}
          TRUE
            SKIP
        --}}}
    --}}}
    --{{{  Dibuja el Cuarto de Circulo
    IF
      cuadrante = 3
        SEQ
          --{{{  Dibuja el Cuadrante 3
          SEQ k.ayu = 0 FOR k
            SEQ
              --{{{  Procesos
              IF
                ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
                  ( contorno.y[(k.ayu + 1)] <= yo.ent )
                  SEQ
                    --{{{  Procesos
                    delta.l.x   := contorno.x[k.ayu] - xo.ent
                    cuanto.x    := delta.l.x + 1
                    x.arregla   := xo.ent - delta.l.x
                    --{{{  Prueba se Esta en la Ventana Definida
                    IF
                      --{{{  Condicion 1
                      ( x.arregla >= x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 1
                          IniciaVectorBytes ( x.arregla, contorno.y[k.ayu]
                                              cuanto.x, color )
                          --}}}
                      --{{{  Condicion 2
                      ( x.arregla < x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 2
                          delta.ayu := x.minimo.ent - x.arregla
                          cuanto.x  := cuanto.x - delta.ayu
                          IniciaVectorBytes ( x.minimo.ent, contorno.y[k.a
                                              cuanto.x, color )
                          --}}}
                      --{{{  Condicion 3
                      ( x.arregla >= x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 3
                          delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                          cuanto.x  := cuanto.x - delta.ayu
                          IniciaVectorBytes ( x.arregla, contorno.y[k.ayu]
                                              cuanto.x, color )
                          --}}}
                      --{{{  Condicion 4
                      ( x.arregla < x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 4
                          delta.ayu := x.minimo.ent - x.arregla
                          cuanto.x  := cuanto.x - delta.ayu
                          delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                          cuanto.x  := cuanto.x - delta.ayu
                          IniciaVectorBytes ( x.minimo.ent, contorno.y[k.a
                                              cuanto.x, color )
                          --}}}
                      TRUE
                        SKIP
                    --}}}
                    --}}}
              TRUE
                SKIP
            --}}}
    --{{{  Dibuja la Ultima Linea
    x.arregla := xo.ent - r.ent
    cuanto.x  := r.ent + 1
    IF
      condicion = 1
        SEQ
          --{{{  Dibuja Linea
          IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
          --}}}
      condicion = 2
        SEQ
          --{{{  Dibuja Linea
          delta.ayu := x.minimo.ent - x.arregla
```

```
--{{{  Dibuja el Cuadrante 1
SEQ k.ayu = 0 FOR k
  SEQ
    --{{{  Procesos
    IF
      ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
        ( contorno.y[(k.ayu + 1)] <= yo.ent )
        SEQ
          --{{{  Procesos
          delta.l.x   := contorno.x[k.ayu] - xo.ent
          delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
          cuanto.x    := delta.l.x + 1
          x.arregla   := xo.ent
          y.arregla   := yo.ent + delta.l.y
          --{{{  Prueba se Esta en la Ventana Definida
          IF
            --{{{  Condicion 1
            ( x.arregla >= x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 1
                IniciaVectorBytes ( x.arregla, y.arregla, cuanto
                --}}}
            --{{{  Condicion 2
            ( x.arregla < x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 2
                delta.ayu := x.minimo.ent - x.arregla
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, y.arregla, cua
                --}}}
            --{{{  Condicion 3
            ( x.arregla >= x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 3
                delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.arregla, y.arregla, cuanto
                --}}}
            --{{{  Condicion 4
            ( x.arregla < x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 4
                delta.ayu := x.minimo.ent - x.arregla
                cuanto.x  := cuanto.x - delta.ayu
                delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, y.arregla, cua
                --}}}
            TRUE
              SKIP
          --}}}
        --}}}
  TRUE
    SKIP
          --}}}
        --{{{  Dibuja la Ultima Linea
        delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
        x.arregla   := xo.ent
        y.arregla   := yo.ent + delta.l.y
        cuanto.x    := r.ent + 1
        IF
          condicion = 1
            SEQ
              --{{{  Dibuja Linea
              IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
              --}}}
          condicion = 2
            SEQ
              --{{{  Dibuja Linea
              delta.ayu := x.minimo.ent - x.arregla
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
              --}}}
          condicion = 3
            SEQ
              --{{{  Dibuja Linea
              delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
              --}}}
          condicion = 4
            SEQ
              --{{{  Dibuja Linea
              delta.ayu := x.minimo.ent - x.arregla
              cuanto.x  := cuanto.x - delta.ayu
              delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
              --}}}
          TRUE
            SKIP
      --}}}
    --}}}
cuadrante = 2
  SEQ
    --{{{  Dibuja el Cuadrante 2
    SEQ k.ayu = 0 FOR k
      SEQ
        --{{{  Procesos
        IF
          ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
            ( contorno.y[(k.ayu + 1)] <= yo.ent )
            SEQ
              --{{{  Procesos
              delta.l.x   := contorno.x[k.ayu] - xo.ent
              delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
              cuanto.x    := delta.l.x + 1
              x.arregla   := xo.ent - delta.l.x
              y.arregla   := yo.ent + delta.l.y
              --{{{  Prueba se Esta en la Ventana Definida
              IF
                --{{{  Condicion 1
                ( x.arregla >= x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 1
                    IniciaVectorBytes ( x.arregla, y.arregla, cuanto
```

```
            SEQ
              --{{{  Procesos
              delta.l.x    := contorno.x.inf[k.ayu] - xo.ent
              delta.l.y    := - ( contorno.y.inf[k.ayu] - yo.ent )
              y.arregla    := yo.ent + delta.l.y
              cuanto.x     := delta.l.x + 1
              IniciaVectorBytes ( xo.ent, y.arregla,
                                  cuanto.x, color )
              --}}}
            TRUE
              SKIP
          --}}}
      --}}}
      --{{{  Dibuja parte triangular
      SEQ l.ayu = 0 FOR tamano
        SEQ
          --{{{  Procesos
          IF
            ( contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu] ) A
              ( contorno.y.sup[(l.ayu + 1)] <= yo.ent )
              SEQ
                --{{{  Procesos
                delta.l.x    := contorno.x.sup[l.ayu] - xo.ent
                delta.l.y    := - ( contorno.y.sup[l.ayu] - yo.ent )
                y.arregla    := yo.ent + delta.l.y
                cuanto.x     := delta.l.x + 1
                IniciaVectorBytes ( xo.ent, y.arregla,
                                    cuanto.x, color )
                --}}}
            TRUE
              SKIP
          --}}}
      --}}}
      --}}}
  oitavo = 3
    SEQ
      --{{{  Dibuja el Oitavo 3
      AlmacenaPuntosSegmentoNA ( xo.ent, yo.ent,
                                 contorno.x.inf[(k-1)], contorno.y.inf
                                 tamano, contorno.x.sup,
                                 contorno.y.sup )
      --{{{  Dibuja parte circular
      SEQ k.ayu = 0 FOR k
        SEQ
          --{{{  Procesos
          IF
            contorno.y.inf[(k.ayu + 1)] <> contorno.y.inf[k.ayu]
              SEQ
                --{{{  Procesos
                delta.l.x    := contorno.x.inf[k.ayu] - xo.ent
                delta.l.y    := - ( contorno.y.inf[k.ayu] - yo.ent )
                x.arregla    := xo.ent - delta.l.x
                y.arregla    := yo.ent + delta.l.y
                cuanto.x     := delta.l.x + 1
                IniciaVectorBytes ( x.arregla, y.arregla,
                                    cuanto.x, color )
                --}}}
            TRUE
              SKIP
          --}}}
      --}}}
      --{{{  Dibuja parte triangular
      SEQ l.ayu = 0 FOR tamano
        SEQ
          --{{{  Procesos
          IF
```

```
            ( contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu] ) A
              ( contorno.y.sup[(l.ayu + 1)] <= yo.ent )
              SEQ
                --{{{  Procesos
                delta.l.x    := contorno.x.sup[l.ayu] - xo.ent
                delta.l.y    := - ( contorno.y.sup[l.ayu] - yo.ent )
                x.arregla    := xo.ent - delta.l.x
                y.arregla    := yo.ent + delta.l.y
                cuanto.x     := delta.l.x + 1
                IniciaVectorBytes ( x.arregla, y.arregla,
                                    cuanto.x, color )
                --}}}
            TRUE
              SKIP
          --}}}
      --}}}
      --}}}
  oitavo = 4
    SEQ
      --{{{  Dibuja el Oitavo 4
      --{{{  Almacena Puntos de Segmento
      AlmacenaPuntosSegmentoNA ( contorno.x.sup[0], contorno.y.sup[0],
                                 xo.ent, yo.ent,
                                 tamano, contorno.x.inf,
                                 contorno.y.inf )
      --}}}
      --{{{  Dibuja el Oitavo
      k.ayu := tamano - 1
      l.ayu := 0
      WHILE ( contorno.y.sup[l.ayu] < yo.ent ) AND ( k.ayu > 0 )
        SEQ
          --{{{  Procesos
          IF
            contorno.y.inf[k.ayu] <> contorno.y.inf[(k.ayu - 1 )]
              SEQ
                IF
                  contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu]
                    SEQ
                      --{{{  Procesos
                      delta.l.x    := contorno.x.sup[l.ayu] - contorno.
                      delta.l.x.1  := contorno.x.sup[l.ayu] - xo.ent
                      delta.l.y    := - ( contorno.y.sup[l.ayu] - yo.en
                      cuanto.x     := delta.l.x  + 1
                      x.arregla    := xo.ent - delta.l.x.1
                      y.arregla    := yo.ent + delta.l.y
                      IniciaVectorBytes ( x.arregla, y.arregla,
                                          cuanto.x, color )
                      --}}}
                      k.ayu := k.ayu - 1
                      l.ayu := l.ayu + 1
                  TRUE
                    l.ayu := l.ayu + 1
            TRUE
              k.ayu := k.ayu - 1
          --}}}
      --}}}
      --{{{  Dibuja la Ultima Linea
      cuanto.x  := r.ent + 1
      IniciaVectorBytes ( (xo.ent-r.ent), yo.ent, cuanto.x, color )
      --}}}
      --}}}
  --}}}
  TRUE
    SKIP
--{{{  COMMENT Dibuja el Cuarto de Circulo
--:::A COMMENT FOLD
```

```
                    IniciaVectorBytes ( x.arregla, contorno.y[k.ayu]
                                        cuanto.x, color )
                    --}}}
                --{{{  Condicion 2
                ( x.arregla < x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 2
                    delta.ayu := x.minimo.ent - x.arregla
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.minimo.ent, contorno.y[k.a
                                        cuanto.x, color )
                    --}}}
                --{{{  Condicion 3
                ( x.arregla >= x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 3
                    delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.arregla, contorno.y[k.ayu]
                                        cuanto.x, color )
                    --}}}
                --{{{  Condicion 4
                ( x.arregla < x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 4
                    delta.ayu := x.minimo.ent - x.arregla
                    cuanto.x  := cuanto.x - delta.ayu
                    delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.minimo.ent, contorno.y[k.a
                                        cuanto.x, color )
                    --}}}
                TRUE
                  SKIP
              --}}}
              --}}}
          TRUE
            SKIP
        --}}}
    --{{{  Dibuja la Ultima Linea
    x.arregla := xo.ent
    cuanto.x := r.ent + 1
    IF
      condicion = 1
        SEQ
          --{{{  Dibuja Linea
          IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
          --}}}
      condicion = 2
        SEQ
          --{{{  Dibuja Linea
          delta.ayu := x.minimo.ent - x.arregla
          cuanto.x  := cuanto.x - delta.ayu
          IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
          --}}}
      condicion = 3
        SEQ
          --{{{  Dibuja Linea
          delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
          cuanto.x  := cuanto.x - delta.ayu
          IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
          --}}}
      condicion = 4
        SEQ
          --{{{  Dibuja Linea
          delta.ayu := x.minimo.ent - x.arregla
          cuanto.x  := cuanto.x - delta.ayu
          delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
          cuanto.x  := cuanto.x - delta.ayu
          IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
          --}}}
      TRUE
        SKIP
    --}}}
    --}}}
cuadrante = 1
  SEQ
    --{{{  Dibuja el Cuadrante 1
    SEQ k.ayu = 0 FOR k
      SEQ
        --{{{  Procesos
        IF
          ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
            ( contorno.y[(k.ayu + 1)] <= yo.ent )
            SEQ
              --{{{  Procesos
              delta.l.x   := contorno.x[k.ayu] - xo.ent
              delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
              cuanto.x    := delta.l.x + 1
              x.arregla   := xo.ent
              y.arregla   := yo.ent + delta.l.y
              --{{{  Prueba se Esta en la Ventana Definida
              IF
                --{{{  Condicion 1
                ( x.arregla >= x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 1
                    IniciaVectorBytes ( x.arregla, y.arregla, cuanto
                    --}}}
                --{{{  Condicion 2
                ( x.arregla < x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 2
                    delta.ayu := x.minimo.ent - x.arregla
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.minimo.ent, y.arregla, cua
                    --}}}
                --{{{  Condicion 3
                ( x.arregla >= x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 3
                    delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.arregla, y.arregla, cuanto
```

```
--{{{  Dibuja el Cuarto de Circulo
IF
  cuadrante = 3
    SEQ
      --{{{  Dibuja el Cuadrante 3
      SEQ k.ayu = 0 FOR k
        SEQ
          --{{{  Procesos
          IF
            ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
              ( contorno.y[(k.ayu + 1)] <= yo.ent )
              SEQ
                --{{{  Procesos,
                delta.1.x   := contorno.x[k.ayu] - xo.ent
                cuanto.x    := delta.1.x + 1
                x.arregla   := xo.ent - delta.1.x
                --{{{  Prueba se Esta en la Ventana Definida
                IF
                  --{{{  Condicion 1
                  ( x.arregla >= x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 1
                      IniciaVectorBytes ( x.arregla, contorno.y[k.ayu]
                                          cuanto.x, color )
                      --}}}
                  --{{{  Condicion 2
                  ( x.arregla < x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 2
                      delta.ayu := x.minimo.ent - x.arregla
                      cuanto.x  := cuanto.x - delta.ayu
                      IniciaVectorBytes ( x.minimo.ent, contorno.y[k.a
                                          cuanto.x, color )
                      --}}}
                  --{{{  Condicion 3
                  ( x.arregla >= x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 3
                      delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                      cuanto.x  := cuanto.x - delta.ayu
                      IniciaVectorBytes ( x.arregla, contorno.y[k.ayu]
                                          cuanto.x, color )
                      --}}}
                  --{{{  Condicion 4
                  ( x.arregla < x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 4
                      delta.ayu := x.minimo.ent - x.arregla
                      cuanto.x  := cuanto.x - delta.ayu
                      delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                      cuanto.x  := cuanto.x - delta.ayu
                      IniciaVectorBytes ( x.minimo.ent, contorno.y[k.a
                                          cuanto.x, color )
                      --}}}
                  TRUE
                    SKIP
                --}}}
                --}}}
            TRUE
              SKIP
          --}}}
      --{{{  Dibuja la Ultima Linea
      x.arregla := xo.ent - r.ent
      cuanto.x  := r.ent + 1
      IF
        condicion = 1
          SEQ
            --{{{  Dibuja Linea
            IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
            --}}}
        condicion = 2
          SEQ
            --{{{  Dibuja Linea
            delta.ayu := x.minimo.ent - x.arregla
            cuanto.x  := cuanto.x - delta.ayu
            IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
            --}}}
        condicion = 3
          SEQ
            --{{{  Dibuja Linea
            delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
            cuanto.x  := cuanto.x - delta.ayu
            IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
            --}}}
        condicion = 4
          SEQ
            --{{{  Dibuja Linea
            delta.ayu := x.minimo.ent - x.arregla
            cuanto.x  := cuanto.x - delta.ayu
            delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
            cuanto.x  := cuanto.x - delta.ayu
            IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
            --}}}
        TRUE
          SKIP
      --}}}
      --}}}
  cuadrante = 4
    SEQ
      --{{{  Dibuja el Cuadrante 4
      SEQ k.ayu = 0 FOR k
        SEQ
          --{{{  Procesos
          IF
            ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
              ( contorno.y[(k.ayu + 1)] <= yo.ent )
              SEQ
                --{{{  Procesos
                delta.1.x   := contorno.x[k.ayu] - xo.ent
                cuanto.x    := delta.1.x + 1
                x.arregla   := xo.ent
                --{{{  Prueba se Esta en la Ventana Definida
                IF
                  --{{{  Condicion 1
                  ( x.arregla >= x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 1
```

```
                cuanto.x    := r.ent + 1
                IF
                  condicion = 1
                    SEQ
                      --{{{  Dibuja Linea
                      IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
                      --}}}
                  condicion = 2
                    SEQ
                      --{{{  Dibuja Linea
                      delta.ayu := x.minimo.ent - x.arregla
                      cuanto.x  := cuanto.x - delta.ayu
                      IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
                      --}}}
                  condicion = 3
                    SEQ
                      --{{{  Dibuja Linea
                      delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
                      cuanto.x  := cuanto.x - delta.ayu
                      IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
                      --}}}
                  condicion = 4
                    SEQ
                      --{{{  Dibuja Linea
                      delta.ayu := x.minimo.ent - x.arregla
                      cuanto.x  := cuanto.x - delta.ayu
                      delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
                      cuanto.x  := cuanto.x - delta.ayu
                      IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
                      --}}}
                  TRUE
                    SKIP
                --}}}
                --}}}
              TRUE
                SKIP
            --}}}
            --}}}
            --}}}
            --}}}
        --}}}
        --}}}
  :
  --}}}F
  --}}}
  --
  --}}}
  --
  --}}}
  --
  --{{{F Dibujar Circunferencias/Semicircunferencias/Cuarto de Circunferencias
  --:::F Dibcir01.tsr
  --
  --{{{  Dibujar Circunferencia
  PROC Circunferencia ( VAL REAL32 xo, VAL REAL32 yo,
                        VAL REAL32 r )
    --{{{  Procesos
    --{{{  Declaraciones
    [1024]INT contorno.x :
    [768]INT contorno.y :
    INT i, i.ayu, j, j.ayu, k, k.ayu :
    INT xo.ent, yo.ent, delta.l.x, delta.l.y :
    INT lim.inf.x.ent, lim.inf.y.ent, lim.sup.x.ent, lim.sup.y.ent :
    INT x.quad, y.quad, xq.mas.yq :
    INT r.ent, r.quad, cuanto.r :
    INT cuanto.x :
    INT x.arregla, y.arregla :
    INT px1, py1, px2, py2 :
    INT px3, py3, px4, py4 :
    --}}}
    SEQ
      --{{{  Procesos
      xo.ent := INT ROUND( xo * escala )
      yo.ent := INT ROUND( yo * escala )
      r.ent  := INT ROUND(  r * escala )
      r.quad := r.ent * r.ent
      --}}}
      --{{{  Arregla Limites de x
      IF
        ( xo.ent-r.ent ) >= x.minimo.ent
          lim.inf.x.ent := xo.ent - r.ent
        TRUE
          lim.inf.x.ent := x.minimo.ent
      IF
        ( xo.ent+r.ent ) <= x.maximo.ent
          lim.sup.x.ent := xo.ent + r.ent
        TRUE
          lim.sup.x.ent := x.maximo.ent
      --}}}
      --{{{  Arregla Limites de y
      IF
        ( yo.ent-r.ent ) >= y.minimo.ent
          lim.inf.y.ent := yo.ent - r.ent
        TRUE
          lim.inf.y.ent := y.minimo.ent
      IF
        ( yo.ent+r.ent ) <=.y.maximo.ent
          lim.sup.y.ent := yo.ent + r.ent
        TRUE
          lim.sup.y.ent := y.maximo.ent
      --}}}
      IF
        ( lim.sup.x.ent <= x.maximo.ent ) AND
          ( lim.inf.x.ent >= x.minimo.ent ) AND
          ( lim.sup.y.ent <= y.maximo.ent ) AND
          ( lim.inf.y.ent >= y.minimo.ent )
          --{{{  Procesos
          SEQ
            --{{{  Asignaciones
            cuanto.r := r.ent + 1
            k        := 0
            k.ayu    := 0
            i        := xo.ent
            i.ayu    := 0
            x.quad   := 0
            --}}}
            SEQ
              --{{{  Procesos
              --{{{  Coge el Contorno
              --SEQ j = lim.inf.y.ent FOR cuanto.r
              SEQ j = ( yo.ent - r.ent ) FOR cuanto.r
                SEQ
                  --{{{  Procesos
                  j.ayu      := j - yo.ent
                  y.quad     := j.ayu * j.ayu
                  xq.mas.yq  := x.quad + y.quad
                  WHILE xq.mas.yq  <= r.quad
                    SEQ
                      --{{{  Procesos
                      contorno.x[k] := xo.ent + i.ayu
                      contorno.y[k] := yo.ent + j.ayu
                      i             := i + 1
```

```
                --}}}
              --{{{  Condicion 4
              ( x.arregla < x.minimo.ent ) AND
              ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
              --}}}
                SEQ
                  --{{{  Dibuja Linea
                  condicion  := 4
                  delta.ayu := x.minimo.ent - x.arregla
                  cuanto.x  := cuanto.x - delta.ayu
                  delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                  cuanto.x  := cuanto.x - delta.ayu
                  IniciaVectorBytes ( x.minimo.ent, y.arregla, cua
                  --}}}
              TRUE
                SKIP
            --}}}
          --}}}
        TRUE
          SKIP
      --}}}
    --{{{  Dibuja la Ultima Linea
    delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
    x.arregla   := xo.ent
    y.arregla   := yo.ent + delta.l.y
    cuanto.x    := r.ent + 1
    IF
      condicion = 1
        SEQ
          --{{{  Dibuja Linea
          IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
          --}}}
      condicion = 2
        SEQ
          --{{{  Dibuja Linea
          delta.ayu := x.minimo.ent - x.arregla
          cuanto.x  := cuanto.x - delta.ayu
          IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
          --}}}
      condicion = 3
        SEQ
          --{{{  Dibuja Linea
          delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
          cuanto.x  := cuanto.x - delta.ayu
          IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
          --}}}
      condicion = 4
        SEQ
          --{{{  Dibuja Linea
          delta.ayu := x.minimo.ent - x.arregla
          cuanto.x  := cuanto.x - delta.ayu
          delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
          cuanto.x  := cuanto.x - delta.ayu
          IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
          --}}}
      TRUE
        SKIP
    --}}}
    --}}}
cuadrante = 2
  SEQ
    --{{{  Dibuja el Cuadrante 2
    SEQ k.ayu = 0 FOR k
      SEQ
        --{{{  Procesos
        IF
          --{{{  Condicion 1
          ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
          ( contorno.y[(k.ayu + 1)] <= yo.ent )
          --}}}
            SEQ
              --{{{  Procesos
              delta.l.x   := contorno.x[k.ayu] - xo.ent
              delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
              cuanto.x    := delta.l.x + 1
              x.arregla   := xo.ent - delta.l.x
              y.arregla   := yo.ent + delta.l.y
              --{{{  Prueba se Esta en la Ventana Definida
              IF
                --{{{  Condicion 1
                ( x.arregla >= x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 1
                    IniciaVectorBytes ( x.arregla, y.arregla, cuanto
                    --}}}
                --{{{  Condicion 2
                ( x.arregla < x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 2
                    delta.ayu := x.minimo.ent - x.arregla
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.minimo.ent, y.arregla, cua
                    --}}}
                --{{{  Condicion 3
                ( x.arregla >= x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 3
                    delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.arregla, y.arregla, cuanto
                    --}}}
                --{{{  Condicion 4
                ( x.arregla < x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 4
                    delta.ayu := x.minimo.ent - x.arregla
                    cuanto.x  := cuanto.x - delta.ayu
                    delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.minimo.ent, y.arregla, cua
                    --}}}
                TRUE
                  SKIP
              --}}}
              --}}}
          TRUE
            SKIP
        --}}}
    --{{{  Dibuja la Ultima Linea
    delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
    x.arregla   := xo.ent - r.ent
    y.arregla   := yo.ent + delta.l.y
```

```
      --SEQ j = lim.inf.y.ent FOR cuanto.r
      SEQ j = ( yo.ent - r.ent ) FOR cuanto.r
        SEQ
          --{{{  Procesos
          j.ayu       := j - yo.ent
          y.quad      := j.ayu * j.ayu
          xq.mas.yq   := x.quad + y.quad
          WHILE xq.mas.yq  <= r.quad
            SEQ
              --{{{  Procesos
              contorno.x[k] := xo.ent + i.ayu
              contorno.y[k] := yo.ent + j.ayu
              i             := i + 1
              k             := k + 1
              i.ayu         := i - xo.ent
              x.quad        := i.ayu * i.ayu
              xq.mas.yq     := x.quad + y.quad
              --}}}
          IF
            i > xo.ent
              SEQ
                --{{{  Procesos
                i       := i - 1
                i.ayu   := i - xo.ent
                x.quad  := i.ayu * i.ayu
                --}}}
            TRUE
              SKIP
          --}}}
      --}}}
      --{{{  Dibuja la Circunferencia
      SEQ k.ayu = 0 FOR k
        SEQ
          --{{{  Procesos
          delta.l.x   := contorno.x[k.ayu] - xo.ent
          delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
          cuanto.x    := 2 * delta.l.x
          y.arregla   := yo.ent + delta.l.y
          px4         := contorno.x[k.ayu]
          py4         := contorno.y[k.ayu]
          py3         := contorno.y[k.ayu]
          px3         := contorno.x[k.ayu] -  cuanto.x
          px1         := contorno.x[k.ayu]
          py1         := yo.ent + delta.l.y
          px2         := contorno.x[k.ayu] -  cuanto.x
          py2         := yo.ent + delta.l.y
          --{{{  Dibuja
          IF
            ( px1 < x.maximo.ent ) AND ( px1 > x.minimo.ent )
              IF
                sup.int = 0
                  DibujaPuntos ( px1, py1 )
                TRUE
                  DibujaPuntos ( px4, py4 )
            TRUE
              SKIP
          IF
            ( px2 < x.maximo.ent ) AND ( px2 > x.minimo.ent )
              IF
                sup.int = 0
                  DibujaPuntos ( px2, py2 )
                TRUE
                  DibujaPuntos ( px3, py3 )
            TRUE
              SKIP
          --}}}
          --}}}
        --}}}
        --{{{  Dibuja los Ultimos Puntos
        DibujaPuntos ( ( xo.ent + r.ent ), yo.ent )
        DibujaPuntos ( ( xo.ent - r.ent ), yo.ent )
        DibujaPuntos ( ( ( xo.ent - r.ent ) - 1 ), yo.ent )
        DibujaPuntos ( ( ( xo.ent + r.ent ) - 1 ), yo.ent )
        --}}}
        --}}}
    --}}}
  TRUE
    SKIP
--}}}
:
--}}}
--
--{{{  Dibujar Cuarto de Circunferencia
PROC CuartoDeCircunferencia ( VAL REAL32 xo, VAL REAL32 yo,
                              VAL REAL32 r, VAL INT cuadrante )
  --{{{  Procesos
  --{{{  Declaraciones
  [1024]INT contorno.x :
  [768]INT contorno.y :
  INT i, i.ayu, j, j.ayu, k, k.ayu :
  INT xo.ent, yo.ent, delta.l.x, delta.l.y :
  INT lim.inf.x.ent, lim.inf.y.ent, lim.sup.x.ent, lim.sup.y.ent :
  INT x.quad, y.quad, xq.mas.yq :
  INT r.ent, r.quad, cuanto.r :
  INT cuanto.x, cuadrante.int :
  INT x.arregla, y.arregla :
  INT px1, py1, px2, py2 :
  INT px3, py3, px4, py4 :
  --}}}
  SEQ
    --{{{  Procesos
    xo.ent := INT ROUND( xo * escala )
    yo.ent := INT ROUND( yo * escala )
    r.ent  := INT ROUND(  r * escala )
    r.quad := r.ent * r.ent
    --}}}
    --{{{  Arregla Limites de x
    IF
      ( xo.ent-r.ent ) >= x.minimo.ent
        lim.inf.x.ent := xo.ent - r.ent
      TRUE
        lim.inf.x.ent := x.minimo.ent
    IF
      ( xo.ent+r.ent ) <= x.maximo.ent
        lim.sup.x.ent := xo.ent + r.ent
      TRUE
        lim.sup.x.ent := x.maximo.ent
    --}}}
    --{{{  Arregla Limites de y
    IF
      ( yo.ent-r.ent ) >= y.minimo.ent
        lim.inf.y.ent := yo.ent - r.ent
      TRUE
        lim.inf.y.ent := y.minimo.ent
    IF
      ( yo.ent+r.ent ) <= y.maximo.ent
        lim.sup.y.ent := yo.ent + r.ent
      TRUE
        lim.sup.y.ent := y.maximo.ent
    --}}}
    IF
      ( lim.sup.x.ent <= x.maximo.ent ) AND
```

```
                k               := k + 1
                i.ayu           := i - xo.ent
                x.quad          := i.ayu * i.ayu
                xq.mas.yq       := x.quad + y.quad
                --}}}
            IF
              i > xo.ent
                SEQ
                  --{{{  Procesos
                  i       := i - 1
                  i.ayu   := i - xo.ent
                  x.quad  := i.ayu * i.ayu
                  --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        --{{{  Dibuja la Circunferencia
        SEQ k.ayu = 0 FOR k
          SEQ
            --{{{  Procesos
            delta.l.x   := contorno.x[k.ayu] - xo.ent
            delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
            cuanto.x    := 2 * delta.l.x
            y.arregla   := yo.ent + delta.l.y
            px4         := contorno.x[k.ayu]
            py4         := contorno.y[k.ayu]
            py3         := contorno.y[k.ayu]
            px3         := contorno.x[k.ayu] -  cuanto.x
            px1         := contorno.x[k.ayu]
            py1         := yo.ent + delta.l.y
            px2         := contorno.x[k.ayu] -  cuanto.x
            py2         := yo.ent + delta.l.y
            --{{{  Dibuja
            IF
              ( px1 < x.maximo.ent ) AND ( px1 > x.minimo.ent )
                SEQ
                  DibujaPuntos ( px1, py1 )
                  DibujaPuntos ( px4, py4 )
              TRUE
                SKIP
            IF
              ( px2 < x.maximo.ent ) AND ( px2 > x.minimo.ent )
                SEQ
                  DibujaPuntos ( px2, py2 )
                  DibujaPuntos ( px3, py3 )
              TRUE
                SKIP
            --}}}
            --}}}
        --}}}
        --{{{  Dibuja los Ultimos Puntos
        DibujaPuntos ( ( xo.ent + r.ent ), yo.ent )
        DibujaPuntos ( ( ( xo.ent + r.ent ) - 1 ), yo.ent )
        DibujaPuntos ( ( xo.ent - r.ent ), yo.ent )
        DibujaPuntos ( ( ( xo.ent - r.ent ) - 1 ), yo.ent )
        --}}}
        --}}}
    --}}}
  TRUE
    SKIP
--}}}
:
--}}}
--
--{{{  Dibujar Semicircunferencia
```

```
PROC SemiCircunferencia ( VAL REAL32 xo, VAL REAL32 yo,
                          VAL REAL32 r, VAL INT sup )
  --{{{  Procesos
  --{{{  Declaraciones
  [1024]INT contorno.x :
  [768]INT contorno.y :
  INT i, i.ayu, j, j.ayu, k, k.ayu :
  INT xo.ent, yo.ent, delta.l.x, delta.l.y :
  INT lim.inf.x.ent, lim.inf.y.ent, lim.sup.x.ent, lim.sup.y.ent :
  INT x.quad, y.quad, xq.mas.yq :
  INT r.ent, r.quad, cuanto.r :
  INT cuanto.x, sup.int :
  INT x.arregla, y.arregla :
  INT px1, py1, px2, py2 :
  INT px3, py3, px4, py4 :
  --}}}
  SEQ
    --{{{  Procesos
    xo.ent  := INT ROUND( xo * escala )
    yo.ent  := INT ROUND( yo * escala )
    r.ent   := INT ROUND(  r * escala )
    r.quad  := r.ent * r.ent
    sup.int := sup
    --}}}
    --{{{  Arregla Limites de x
    IF
      ( xo.ent-r.ent ) >= x.minimo.ent
        lim.inf.x.ent := xo.ent - r.ent
      TRUE
        lim.inf.x.ent := x.minimo.ent
    IF
      ( xo.ent+r.ent ) <= x.maximo.ent
        lim.sup.x.ent := xo.ent + r.ent
      TRUE
        lim.sup.x.ent := x.maximo.ent
    --}}}
    --{{{  Arregla Limites de y
    IF
      ( yo.ent-r.ent ) >= y.minimo.ent
        lim.inf.y.ent := yo.ent - r.ent
      TRUE
        lim.inf.y.ent := y.minimo.ent
    IF
      ( yo.ent+r.ent ) <= y.maximo.ent
        lim.sup.y.ent := yo.ent + r.ent
      TRUE
        lim.sup.y.ent := y.maximo.ent
    --}}}
    IF
      ( lim.sup.x.ent <= x.maximo.ent ) AND
        ( lim.inf.x.ent >= x.minimo.ent ) AND
        ( lim.sup.y.ent <= y.maximo.ent ) AND
        ( lim.inf.y.ent >= y.minimo.ent )
        --{{{  Procesos
        SEQ
          --{{{  Asignaciones
          cuanto.r := r.ent + 1
          k        := 0
          k.ayu    := 0
          i        := xo.ent
          i.ayu    := 0
          x.quad   := 0
          --}}}
          SEQ
            --{{{  Procesos
            --{{{  Coge el Contorno
```

```
      xos.int.r   := xo
      yo.int.r    := yo
      h.r         := ancho.int.r / dos.r
      xoi.int.r   := xos.int.r - h.r
      --}}}
      WHILE r.ext.int.r >= h.r
        SEQ
          --{{{  Proc
          --{{{  Prueba si es superior u inferior
          sup := cont \ 2
          IF
            sup=0
              xo.ayu.r  := xos.int.r
            TRUE
              xo.ayu.r  := xoi.int.r
          --}}}
          Color (color1)
          SemiCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, sup )
          Color (color2)
          SemiCirculo ( xo.ayu.r, yo.int.r, (r.ext.int.r-h.r), sup )
          --{{{  Actualiza radio y contador
          r.ext.int.r := r.ext.int.r - h.r
          cont := cont + 1
          --}}}
          --}}}
      --{{{  Prueba si es superior u inferior
      sup := cont \ 2
      IF
        sup=0
          xo.ayu.r  := xos.int.r
        TRUE
          xo.ayu.r  := xoi.int.r
      --}}}
      Color (color1)
      SemiCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, sup )
      --}}}
:
--}}}
--
--{{{  Espiral con Cuarto de Circulos
PROC EspiralCuarto ( VAL REAL32 xo, VAL REAL32 yo,
                     VAL REAL32 r.ext , VAL REAL32 ancho.int.r )
  --{{{  Declaraciones
  REAL32 xo.int.r, yo.int.r, xoi.int.r, h.r, r.ext.int.r :
  REAL32 xo.ayu.r :
  INT cont, cuadrante, sup, color.ayu :
  BYTE color.by :
  --}}}
  --{{{  Constantes
  VAL    dos.r   IS    REAL32 ROUND(2) :
  VAL   cero.r   IS    REAL32 ROUND(0) :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Asignaciones
    color.ayu   :=  INT(color)
    cont        :=  0
    sup         :=  0
    cuadrante   :=  0
    r.ext.int.r := r.ext
    xo.int.r    := xo
    yo.int.r    := yo
    h.r         := ancho.int.r / dos.r
    xoi.int.r   := xo.int.r - h.r
    --}}}
    WHILE r.ext.int.r > cero.r
      SEQ
        --{{{  Proc
        --{{{  Prueba si es superior u inferior
        sup := cont \ 2
        IF
          sup=0
            SEQ
              --{{{  Proc
              xo.ayu.r  := xo.int.r
              cuadrante := 1
              CuartoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, cuadrante )
              --{{{  Color
              color.ayu := color.ayu + 1
              IF
                ( color.ayu > 255 ) OR ( color.ayu = 0 )
                  color.ayu := 1
                TRUE
                  SKIP
              color.by := BYTE(color.ayu)
              Color (color.by)
              --}}}
              cuadrante := 2
              CuartoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, cuadrante )
              --}}}
          TRUE
            SEQ
              --{{{  Proc
              xo.ayu.r  := xoi.int.r
              cuadrante := 3
              CuartoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, cuadrante )
              --{{{  Color
              color.ayu := color.ayu + 1
              IF
                ( color.ayu > 255 ) OR ( color.ayu = 0 )
                  color.ayu := 1
                TRUE
                  SKIP
              color.by := BYTE(color.ayu)
              Color (color.by)
              --}}}
              cuadrante := 4
              CuartoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, cuadrante )
              --}}}
        --}}}
        --{{{  Color
        color.ayu := color.ayu + 1
        IF
          ( color.ayu > 255 ) OR ( color.ayu = 0 )
            color.ayu := 1
          TRUE
            SKIP
        color.by := BYTE(color.ayu)
        Color (color.by)
        --}}}
        --{{{  Actualiza radio y contador
        r.ext.int.r := r.ext.int.r - h.r
        cont := cont + 1
        --}}}
        --}}}
    --}}}
:
--}}}
--
--{{{  Espiral con Cuarto de Circulos y Contorno (color.varia.rapido)
PROC EspiralCuartoContorno1 ( VAL REAL32 xo, VAL REAL32 yo,
                              VAL REAL32 r.ext , VAL REAL32 ancho.int.r )
```

```
( lim.inf.x.ent >= x.minimo.ent ) AND
( lim.sup.y.ent <= y.maximo.ent ) AND
( lim.inf.y.ent >= y.minimo.ent )
--{{{  Procesos
SEQ
  --{{{  Asignaciones
  cuanto.r       := r.ent + 1
  k              := 0
  k.ayu          := 0
  i              := xo.ent
  i.ayu          := 0
  x.quad         := 0
  cuadrante.int  := cuadrante
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Coge el Contorno
    --SEQ j = lim.inf.y.ent FOR cuanto.r
    SEQ j = ( yo.ent - r.ent ) FOR cuanto.r
      SEQ
        --{{{  Procesos
        j.ayu       := j - yo.ent
        y.quad      := j.ayu * j.ayu
        xq.mas.yq   := x.quad + y.quad
        WHILE xq.mas.yq  <= r.quad
          SEQ
            --{{{  Procesos
            contorno.x[k] := xo.ent + i.ayu
            contorno.y[k] := yo.ent + j.ayu
            i             := i + 1
            k             := k + 1
            i.ayu         := i - xo.ent
            x.quad        := i.ayu * i.ayu
            xq.mas.yq     := x.quad + y.quad
            --}}}
        IF
          i > xo.ent
            SEQ
              --{{{  Procesos
              i       := i - 1
              i.ayu   := i - xo.ent
              x.quad  := i.ayu * i.ayu
              --}}}
          TRUE
            SKIP
        --}}}
    --}}}
    --{{{  Dibuja la Circunferencia
    SEQ k.ayu = 0 FOR k
      SEQ
        --{{{  Procesos
        delta.l.x    := contorno.x[k.ayu] - xo.ent
        delta.l.y    := - ( contorno.y[k.ayu] - yo.ent )
        cuanto.x     := 2 * delta.l.x
        y.arregla    := yo.ent + delta.l.y
        px4          := contorno.x[k.ayu]
        py4          := contorno.y[k.ayu]
        py3          := contorno.y[k.ayu]
        px3          := contorno.x[k.ayu] -  cuanto.x
        px1          := contorno.x[k.ayu]
        py1          := yo.ent + delta.l.y
        px2          := contorno.x[k.ayu] -  cuanto.x
        py2          := yo.ent + delta.l.y
        --{{{  Dibuja
        IF
          ( px1 < x.maximo.ent ) AND ( px1 > x.minimo.ent )
            IF
              cuadrante.int = 1
                DibujaPuntos ( px1, py1 )
              cuadrante.int = 4
                DibujaPuntos ( px4, py4 )
              TRUE
                SKIP
          TRUE
            SKIP
        IF
          ( px2 < x.maximo.ent ) AND ( px2 > x.minimo.ent )
            IF
              cuadrante.int = 2
                DibujaPuntos ( px2, py2 )
              cuadrante.int = 3
                DibujaPuntos ( px3, py3 )
              TRUE
                SKIP
          TRUE
            SKIP
        --}}}
        --}}}
    --}}}
    --{{{  Dibuja los Ultimos Puntos
    IF
      ( cuadrante.int = 1 ) OR ( cuadrante.int = 4 )
        SEQ
          DibujaPuntos ( ( xo.ent + r.ent ), yo.ent )
          DibujaPuntos ( ( ( xo.ent + r.ent ) - 1 ), yo.ent )
      TRUE
        SEQ
          DibujaPuntos ( ( xo.ent - r.ent ), yo.ent )
          DibujaPuntos ( ( ( xo.ent - r.ent ) - 1 ), yo.ent )
    --}}}
    --}}}
  --}}}
TRUE
  SKIP
--}}}
:
--}}}
--
--}}}F
--
--{{{F Dibujar Espirales
--:::F Dibespo1.tsr
--
--{{{  Espiral con Semicirculos y dos colores
PROC EspiralSemiDosColores ( VAL REAL32 xo, VAL REAL32 yo,
                             VAL REAL32 r.ext , VAL REAL32 ancho.int.r,
                             VAL BYTE color1, VAL BYTE color2 )
  --{{{  Declaraciones
  REAL32 xos.int.r, yo.int.r, xoi.int.r, h.r, r.ext.int.r :
  REAL32 xo.ayu.r :
  INT cont, sup :
  --}}}
  --{{{  Constantes
  VAL    dos.r   IS    REAL32 ROUND(2) :
  VAL   cero.r   IS    REAL32 ROUND(0) :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Asignaciones
    cont         :=  0
    sup          :=  0
    r.ext.int.r := r.ext
```

```
          xo.ayu.r  := xo.int.r
          oitavo    := 1
          OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
          --{{{  Color
          color.ayu := color.ayu + 5
          IF
            ( color.ayu > 255 ) OR ( color.ayu = 0 )
              color.ayu := 1
            TRUE
              SKIP
          color.by := BYTE(color.ayu)
          Color (color.by)
          --}}}
          --{{{  COMMENT Color
          --:::A COMMENT FOLD
          --{{{  Color
          color.ayu := color.ayu + 4
          IF
            ( color.ayu > 255 ) OR ( color.ayu = 0 )
              color.ayu := 1
            TRUE
              SKIP
          color.by := BYTE(color.ayu)
          Color (color.by)
          --}}}
          --}}}
          oitavo    := 2
          OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
          --{{{  Color
          color.ayu := color.ayu + 5
          IF
            ( color.ayu > 255 ) OR ( color.ayu = 0 )
              color.ayu := 1
            TRUE
              SKIP
          color.by := BYTE(color.ayu)
          Color (color.by)
          --}}}
          --{{{  COMMENT Color
          --:::A COMMENT FOLD
          --{{{  Color
          color.ayu := color.ayu + 4
          IF
            ( color.ayu > 255 ) OR ( color.ayu = 0 )
              color.ayu := 1
            TRUE
              SKIP
          color.by := BYTE(color.ayu)
          Color (color.by)
          --}}}
          --}}}
          oitavo    := 3
          OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
          --{{{  Color
          color.ayu := color.ayu + 5
          IF
            ( color.ayu > 255 ) OR ( color.ayu = 0 )
              color.ayu := 1
            TRUE
              SKIP
          color.by := BYTE(color.ayu)
          Color (color.by)
          --}}}
          --{{{  COMMENT Color
          --:::A COMMENT FOLD
          --{{{  Color
          color.ayu := color.ayu + 4.
          IF
            ( color.ayu > 255 ) OR ( color.ayu = 0 )
              color.ayu := 1
            TRUE
              SKIP
          color.by := BYTE(color.ayu)
          Color (color.by)
          --}}}
          --}}}
          oitavo    := 4
          OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
          --{{{  Contorno
          Color ( BYTE( 255 - color.ayu ) )
          SemiCircunferencia ( xo.ayu.r, yo.int.r, r.ext.int.r, sup )
          Color (color.by)
          --}}}
          --}}}
        TRUE
          SEQ
            --{{{  Proc/Dibuja
            xo.ayu.r  := xo1.int.r
            oitavo    := 5
            OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
            --{{{  Color
            color.ayu := color.ayu + 5
            IF
              ( color.ayu > 255 ) OR ( color.ayu = 0 )
                color.ayu := 1
              TRUE
                SKIP
            color.by := BYTE(color.ayu)
            Color (color.by)
            --}}}
            --{{{  COMMENT Color
            --:::A COMMENT FOLD
            --{{{  Color
            color.ayu := color.ayu + 4
            IF
              ( color.ayu > 255 ) OR ( color.ayu = 0 )
                color.ayu := 1
              TRUE
                SKIP
            color.by := BYTE(color.ayu)
            Color (color.by)
            --}}}
            --}}}
            oitavo    := 6
            OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
            --{{{  Color
            color.ayu := color.ayu + 5
            IF
              ( color.ayu > 255 ) OR ( color.ayu = 0 )
                color.ayu := 1
              TRUE
                SKIP
            color.by := BYTE(color.ayu)
            Color (color.by)
            --}}}
            --{{{  COMMENT Color
            --:::A COMMENT FOLD
            --{{{  Color
            color.ayu := color.ayu + 4
            IF
              ( color.ayu > 255 ) OR ( color.ayu = 0 )
                color.ayu := 1
```

```
    --{{{  Declaraciones
    REAL32 xo.int.r, yo.int.r, xo1.int.r, h.r, r.ext.int.r :
    REAL32 xo.ayu.r :
    INT cont, cuadrante, sup, color.ayu :
    BYTE color.by :
    --}}}
    --{{{  Constantes
    VAL    dos.r   IS    REAL32 ROUND(2) :
    VAL   cero.r   IS    REAL32 ROUND(0) :
    --}}}
    SEQ
      --{{{  Proc
      --{{{  Asignaciones
      color.ayu   :=  INT(color)
      cont        :=  0
      sup         :=  0
      cuadrante   :=  0
      r.ext.int.r := r.ext
      xo.int.r    := xo
      yo.int.r    := yo
      h.r         := ancho.int.r / dos.r
      xo1.int.r   := xo.int.r - h.r
      --}}}
      WHILE r.ext.int.r > cero.r
        SEQ
          --{{{  Proc
          --{{{  Prueba si es superior u inferior
          sup := cont \ 2
          IF
            sup=0
              SEQ
                --{{{  Proc/Dibuja
                xo.ayu.r  := xo.int.r
                cuadrante := 1
                CuartoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, cuadrante )
                --{{{  Color
                color.ayu := color.ayu + 4
                IF
                  ( color.ayu > 255 ) OR ( color.ayu = 0 )
                    color.ayu := 1
                  TRUE
                    SKIP
                color.by := BYTE(color.ayu)
                Color (color.by)
                --}}}
                cuadrante := 2
                CuartoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, cuadrante )
                --{{{  Contorno
                Color ( BYTE( 255 - color.ayu ) )
                SemiCircunferencia ( xo.ayu.r, yo.int.r, r.ext.int.r, sup )
                Color (color.by)
                --}}}
                --}}}
            TRUE
              SEQ
                --{{{  Proc/Dibuja
                xo.ayu.r  := xo1.int.r
                cuadrante := 3
                CuartoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, cuadrante )
                --{{{  Color
                color.ayu := color.ayu + 2
                IF
                  ( color.ayu > 255 ) OR ( color.ayu = 0 )
                    color.ayu := 1
                  TRUE
                    SKIP
                color.by := BYTE(color.ayu)
                Color (color.by)
                --}}}
                cuadrante := 4
                CuartoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, cuadrante )
                --{{{  Contorno
                Color ( BYTE( 255 - color.ayu ) )
                SemiCircunferencia ( xo.ayu.r, yo.int.r, r.ext.int.r, sup )
                Color (color.by)
                --}}}
                --}}}
          --}}}
          --{{{  Color
          color.ayu := color.ayu + 4
          IF
            ( color.ayu > 255 ) OR ( color.ayu = 0 )
              color.ayu := 1
            TRUE
              SKIP
          color.by := BYTE(color.ayu)
          Color (color.by)
          --}}}
          --{{{  Actualiza radio y contador
          r.ext.int.r := r.ext.int.r - h.r
          cont := cont + 1
          --}}}
          --}}}
      --}}}
    :
    --}}}
    --
    --{{{  Espiral con Oitavo de Circulos y Contorno (color.varia.rapido)
    PROC EspiralOitavoContorno  ( VAL REAL32 xo, VAL REAL32 yo,
                                  VAL REAL32 r.ext , VAL REAL32 ancho.int.r )
      --{{{  Declaraciones
      REAL32 xo.int.r, yo.int.r, xo1.int.r, h.r, r.ext.int.r :
      REAL32 xo.ayu.r :
      INT cont, oitavo, sup, color.ayu :
      BYTE color.by :
      --}}}
      --{{{  Constantes
      VAL    dos.r   IS    REAL32 ROUND(2) :
      VAL   cero.r   IS    REAL32 ROUND(0) :
      --}}}
      SEQ
        --{{{  Proc
        --{{{  Asignaciones
        color.ayu   :=  INT(color)
        cont        :=  0
        sup         :=  0
        oitavo      :=  0
        r.ext.int.r := r.ext
        xo.int.r    := xo
        yo.int.r    := yo
        h.r         := ancho.int.r / dos.r
        xo1.int.r   := xo.int.r - h.r
        --}}}
        WHILE r.ext.int.r > cero.r
          SEQ
            --{{{  Proc
            --{{{  Prueba si es superior u inferior
            sup := cont \ 2
            IF
              sup=0
                SEQ
                  --{{{  Proc/Dibuja
```

```
            --{{{  Proc
            IF
              ( oitavo REM 2 ) = 0
                Color (color2)
              TRUE
                Color (color1)
            --}}}
        --}}}
        OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
        oitavo    := 3
        --{{{  Color
        IF
          cambia = TRUE
            SEQ
              --{{{  Proc
              IF
                ( oitavo REM 2 ) = 0
                  Color (color1)
                TRUE
                  Color (color2)
              --}}}
          TRUE
            SEQ
              --{{{  Proc
              IF
                ( oitavo REM 2 ) = 0
                  Color (color2)
                TRUE
                  Color (color1)
              --}}}
        --}}}
        OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
        oitavo    := 4
        --{{{  Color
        IF
          cambia = TRUE
            SEQ
              --{{{  Proc
              IF
                ( oitavo REM 2 ) = 0
                  Color (color1)
                TRUE
                  Color (color2)
              --}}}
          TRUE
            SEQ
              --{{{  Proc
              IF
                ( oitavo REM 2 ) = 0
                  Color (color2)
                TRUE
                  Color (color1)
              --}}}
        --}}}
        OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
        --}}}
    TRUE
      SEQ
        --{{{  Proc/Dibuja
        xo.ayu.r  := xoi.int.r
        oitavo    := 5
        --{{{  Color
        IF
          cambia = TRUE
            SEQ
              --{{{  Proc
              IF
                ( oitavo REM 2 ) = 0
                  Color (color1)
                TRUE
                  Color (color2)
              --}}}
          TRUE
            SEQ
              --{{{  Proc
              IF
                ( oitavo REM 2 ) = 0
                  Color (color2)
                TRUE
                  Color (color1)
              --}}}
        --}}}
        OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
        oitavo    := 6
        --{{{  Color
        IF
          cambia = TRUE
            SEQ
              --{{{  Proc
              IF
                ( oitavo REM 2 ) = 0
                  Color (color1)
                TRUE
                  Color (color2)
              --}}}
          TRUE
            SEQ
              --{{{  Proc
              IF
                ( oitavo REM 2 ) = 0
                  Color (color2)
                TRUE
                  Color (color1)
              --}}}
        --}}}
        OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
        oitavo    := 7
        --{{{  Color
        IF
          cambia = TRUE
            SEQ
              --{{{  Proc
              IF
                ( oitavo REM 2 ) = 0
                  Color (color1)
                TRUE
                  Color (color2)
              --}}}
          TRUE
            SEQ
              --{{{  Proc
              IF
                ( oitavo REM 2 ) = 0
                  Color (color2)
                TRUE
                  Color (color1)
              --}}}
        --}}}
        OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
        oitavo    := 8
        --{{{  Color
        IF
```

```
              TRUE
                SKIP
            color.by := BYTE(color.ayu)
            Color (color.by)
            --}}}
            --}}}
            oitavo    := 7
            OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
            --{{{  Color
            color.ayu := color.ayu + 5
            IF
              ( color.ayu > 255 ) OR ( color.ayu = 0 )
                color.ayu := 1
              TRUE
                SKIP
            color.by := BYTE(color.ayu)
            Color (color.by)
            --}}}
            --{{{  COMMENT Color
            --:::A COMMENT FOLD
            --{{{  Color
            color.ayu := color.ayu + 4
            IF
              ( color.ayu > 255 ) OR ( color.ayu = 0 )
                color.ayu := 1
              TRUE
                SKIP
            color.by := BYTE(color.ayu)
            Color (color.by)
            --}}}
            --}}}
            oitavo    := 8
            OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
            --{{{  Contorno
            Color ( BYTE( 255 - color.ayu ) )
            SemiCircunferencia ( xo.ayu.r, yo.int.r, r.ext.int.r, sup )
            Color (color.by)
            --}}}
            --}}}
        --}}}
        --{{{  Color
        color.ayu := color.ayu + 5
        IF
          ( color.ayu > 255 ) OR ( color.ayu = 0 )
            color.ayu := 1
          TRUE
            SKIP
        color.by := BYTE(color.ayu)
        Color (color.by)
        --}}}
        --{{{  Actualiza radio y contador
        r.ext.int.r := r.ext.int.r - h.r
        cont := cont + 1
        --}}}
        --}}}
    --}}}
:
--}}}
--
--{{{  Espiral Oitavo de Circ., sin Contorno y dos colores
PROC EspiralOitavoDosColores1  ( VAL REAL32 xo, VAL REAL32 yo,
                                 VAL REAL32 r.ext , VAL REAL32 ancho.int.r,
                                 VAL BYTE color1, VAL BYTE color2 )
  --{{{  Declaraciones
  REAL32 xo.int.r, yo.int.r, xo1.int.r, h.r, r.ext.int.r :
  REAL32 xo.ayu.r :
  INT cont, oitavo, sup :
  BOOL cambia :
  --}}}
  --{{{  Constantes
  VAL    dos.r   IS    REAL32 ROUND(2) :
  VAL   cero.r   IS    REAL32 ROUND(0) :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Asignaciones
    cont        :=  0
    sup         :=  0
    r.ext.int.r := r.ext
    xo.int.r    := xo
    yo.int.r    := yo
    h.r         := ancho.int.r / dos.r
    xo1.int.r   := xo.int.r - h.r
    cambia      := TRUE
    --}}}
    WHILE r.ext.int.r > cero.r
      SEQ
        --{{{  Proc
        --{{{  Prueba si es superior u inferior
        sup := cont \ 2
        IF
          sup=0
            SEQ
              --{{{  Proc/Dibuja
              xo.ayu.r  := xo.int.r
              oitavo    := 1
              --{{{  Color
              IF
                cambia = TRUE
                  SEQ
                    --{{{  Proc
                    IF
                      ( oitavo REM 2 ) = 0
                        Color (color1)
                      TRUE
                        Color (color2)
                    --}}}
                TRUE
                  SEQ
                    --{{{  Proc
                    IF
                      ( oitavo REM 2 ) = 0
                        Color (color2)
                      TRUE
                        Color (color1)
                    --}}}
              --}}}
              OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
              oitavo    := 2
              --{{{  Color
              IF
                cambia = TRUE
                  SEQ
                    --{{{  Proc
                    IF
                      ( oitavo REM 2 ) = 0
                        Color (color1)
                      TRUE
                        Color (color2)
                    --}}}
                TRUE
                  SEQ
```

```
          color.by := BYTE(color.ayu)
          Color (color.by)
          --}}}
          --{{{  Actualiza radio y contador
          r.ext.int.r := r.ext.int.r - h.r
          cont := cont + 1
          --}}}
        --}}}
    --}}}
:
--}}}
--
--}}}F
--
--{{{  Dibujar Coronas
PROC Corona ( VAL REAL32 xo, VAL REAL32 yo,
              VAL REAL32 r.int, VAL REAL32 r.ext )
  --{{{  Procesos
  --{{{  Declaraciones
  [1024]INT contorno.x.ext :
  [1024]INT contorno.x.int :
  [768]INT contorno.y.ext :
  [768]INT contorno.y.int :
  INT i, i.ayu, j, j.ayu :
  INT k.int, k.ayu.int, k.ext, k.ayu.ext :
  INT xo.ent, yo.ent :
  INT delta.l.x.int, delta.l.y.int :
  INT delta.l.x.ext, delta.l.y.ext :
  INT lim.inf.x.ent.int, lim.inf.y.ent.int :
  INT lim.inf.x.ent.ext, lim.inf.y.ent.ext :
  INT lim.sup.x.ent.int, lim.sup.y.ent.int :
  INT lim.sup.x.ent.ext, lim.sup.y.ent.ext :
  INT x.quad.ext, y.quad.ext, xq.mas.yq.ext :
  INT x.quad.int, y.quad.int, xq.mas.yq.int :
  INT r.ent.int, r.ent.ext, r.quad.int, r.quad.ext, cuanto.r.int, cuanto.r.ext :
  INT cuanto.x.int, cuanto.x.ext :
  INT x.arregla.int, y.arregla.int :
  INT x.arregla.ext, y.arregla.ext :
  --}}}
  SEQ
    --{{{  Procesos
    xo.ent := INT ROUND( xo * escala )
    yo.ent := INT ROUND( yo * escala )
    r.ent.int  := INT ROUND(  r.int * escala )
    r.ent.ext  := INT ROUND(  r.ext * escala )
    r.quad.int := r.ent.int * r.ent.int
    r.quad.ext := r.ent.ext * r.ent.ext
    --}}}
    --{{{  Arregla Limites de x Para r.int
    IF
      ( xo.ent-r.ent.int ) >= 0
        lim.inf.x.ent.int := xo.ent - r.ent.int
      TRUE
        lim.inf.x.ent.int := 0
    IF
      ( xo.ent+r.ent.int ) <= x.maximo.ent
        lim.sup.x.ent.int := xo.ent + r.ent.int
      TRUE
        lim.sup.x.ent.int := x.maximo.ent
    --}}}
    --{{{  Arregla Limites de x Para r.ext
    IF
      ( xo.ent-r.ent.ext ) >= 0
        lim.inf.x.ent.ext := xo.ent - r.ent.ext
      TRUE
        lim.inf.x.ent.ext := 0
    IF
      ( xo.ent+r.ent.ext ) <= x.maximo.ent
        lim.sup.x.ent.ext := xo.ent + r.ent.ext
      TRUE
        lim.sup.x.ent.ext := x.maximo.ent
    --}}}
    --{{{  Arregla Limites de y Para r.int
    IF
      ( yo.ent-r.ent.int ) >= 0
        lim.inf.y.ent.int := yo.ent - r.ent.int
      TRUE
        lim.inf.y.ent.int := 0
    IF
      ( yo.ent+r.ent.int ) <= y.maximo.ent
        lim.sup.y.ent.int := yo.ent + r.ent.int
      TRUE
        lim.sup.y.ent.int := y.maximo.ent
    --}}}
    --{{{  Arregla Limites de y Para r.ext
    IF
      ( yo.ent-r.ent.ext ) >= 0
        lim.inf.y.ent.ext := yo.ent - r.ent.ext
      TRUE
        lim.inf.y.ent.ext := 0
    IF
      ( yo.ent+r.ent.ext ) <= y.maximo.ent
        lim.sup.y.ent.ext := yo.ent + r.ent.ext
      TRUE
        lim.sup.y.ent.ext := y.maximo.ent
    --}}}
    IF
      ( lim.inf.x.ent.ext <= x.maximo.ent ) AND ( lim.sup.x.ent.ext >=0 ) AND
      ( lim.inf.y.ent.ext <= y.maximo.ent ) AND ( lim.sup.y.ent.ext >=0 )
        --{{{  Procesos
        SEQ
          --{{{  Asignaciones Para r.int
          cuanto.r.int := r.ent.int + 1
          k.int        := 0
          k.ayu.int    := 0
          i            := xo.ent
          i.ayu        := 0
          x.quad.int   := 0
          --}}}
          --{{{  Coge el Contorno Para r.int
          SEQ j = lim.inf.y.ent.int FOR cuanto.r.int
            SEQ
              --{{{  Procesos
              j.ayu          := j - yo.ent
              y.quad.int     := j.ayu * j.ayu
              xq.mas.yq.int  := x.quad.int + y.quad.int
              WHILE xq.mas.yq.int  <= r.quad.int
                SEQ
                  --{{{  Procesos
                  contorno.x.int[k.int] := xo.ent + i.ayu
                  contorno.y.int[k.int] := yo.ent + j.ayu
                  i                     := i + 1
                  k.int                 := k.int + 1
                  i.ayu                 := i - xo.ent
                  x.quad.int            := i.ayu * i.ayu
                  xq.mas.yq.int         := x.quad.int + y.quad.int
                  --}}}
              IF
                i > xo.ent
                  SEQ
                    --{{{  Procesos
                    i           := i - 1
```

```
              cambia = TRUE
                SEQ
                  --{{{  Proc
                  IF
                    ( oitavo REM 2 ) = 0
                      Color (color1)
                    TRUE
                      Color (color2)
                  --}}}
              TRUE
                SEQ
                  --{{{  Proc
                  IF
                    ( oitavo REM 2 ) = 0
                      Color (color2)
                    TRUE
                      Color (color1)
                  --}}}
            --}}}
            OitavoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, oitavo )
            --}}}
        --}}}
        --{{{  Actualiza Booleana "cambia"
        IF
          sup <> 0
            --{{{  Actualiza
            IF
              cambia = TRUE
                cambia  := FALSE
              TRUE
                cambia  := TRUE
            --}}}
          TRUE
            SKIP
        --}}}
        --{{{  Actualiza radio y contador
        r.ext.int.r := r.ext.int.r - h.r
        cont := cont + 1
        --}}}
        --}}}
    --}}}
:
--}}}
--
--{{{  Espiral con Cuarto de Circulos y Contorno (color.varia.lento)
PROC EspiralCuartoContorno ( VAL REAL32 xo, VAL REAL32 yo,
                             VAL REAL32 r.ext , VAL REAL32 ancho.int.r )
  --{{{  Declaraciones
  REAL32 xo.int.r, yo.int.r, xo1.int.r, h.r, r.ext.int.r :
  REAL32 xo.ayu.r :
  INT cont, cuadrante, sup, color.ayu :
  BYTE color.by :
  --}}}
  --{{{  Constantes
  VAL    dos.r   IS    REAL32 ROUND(2) :
  VAL   cero.r   IS    REAL32 ROUND(0) :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Asignaciones
    color.ayu   :=  INT(color)
    cont        :=  0'
    sup         :=  0
    cuadrante   :=  0
    r.ext.int.r := r.ext
    xo.int.r    := xo
    yo.int.r    := yo
    h.r         := ancho.int.r / dos.r
    xo1.int.r   := xo.int.r - h.r
    --}}}
    WHILE r.ext.int.r > cero.r
      SEQ
        --{{{  Proc
        --{{{  Prueba si es superior u inferior
        sup := cont \ 2
        IF
          sup=0
            SEQ
              --{{{  Proc/Dibuja
              xo.ayu.r  := xo.int.r
              cuadrante := 1
              CuartoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, cuadrante )
              --{{{  Color
              color.ayu := color.ayu + 2
              IF
                ( color.ayu > 255 ) OR ( color.ayu = 0 )
                  color.ayu := 1
                TRUE
                  SKIP
              color.by := BYTE(color.ayu)
              Color (color.by)
              --}}}
              cuadrante := 2
              CuartoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, cuadrante )
              --{{{  Contorno
              Color ( BYTE( 255 - color.ayu ) )
              SemiCircunferencia ( xo.ayu.r, yo.int.r, r.ext.int.r, sup )
              Color (color.by)
              --}}}
              --}}}
          TRUE
            SEQ
              --{{{  Proc/Dibuja
              xo.ayu.r  := xo1.int.r
              cuadrante := 3
              CuartoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, cuadrante )
              --{{{  Color
              color.ayu := color.ayu + 2
              IF
                ( color.ayu > 255 ) OR ( color.ayu = 0 )
                  color.ayu := 1
                TRUE
                  SKIP
              color.by := BYTE(color.ayu)
              Color (color.by)
              --}}}
              cuadrante := 4
              CuartoDeCirculo ( xo.ayu.r, yo.int.r, r.ext.int.r, cuadrante )
              --{{{  Contorno
              Color ( BYTE( 255 - color.ayu ) )
              SemiCircunferencia ( xo.ayu.r, yo.int.r, r.ext.int.r, sup )
              Color (color.by)
              --}}}
              --}}}'
        --}}}
        --{{{  Color
        color.ayu := color.ayu + 2
        IF
          ( color.ayu > 255 ) OR ( color.ayu = 0 )
            color.ayu := 1
          TRUE
            SKIP
```

```
PROC EscribirMensajePantallaGrafica ( VAL INT color.ent, VAL INT x.mens,
                                      VAL INT y.mens, VAL []INT cadena.ent )
  --{{{  Declaraciones
  BYTE fuente.byte.ayu, byte.ayu :
  INT i.ayu, j.ayu, k.ayu, fuente.ent.ayu, t.128 :
  INT ent.ayu, x.ayu, x.ayu.ayu, y.ayu, y.ayu.ayu, result.ayu :
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Asignaciones
    t.128          :=   128
    x.ayu          :=   x.mens
    x.ayu.ayu      :=   x.ayu
    y.ayu          :=   y.mens
    y.ayu.ayu      :=   y.ayu
    result.ayu     :=   0
    --}}}
    color.elegido := BYTE(color.ent)
    Color ( color.elegido )
    SEQ i=0 FOR ( longitud - 3 )
      SEQ
        i.ayu   :=  cadena.ent[i]
        SEQ j=0 FOR 12
          SEQ
            --{{{  Procesos
            fuente.byte.ayu :=  fuente.elegido.by[i.ayu][j]
            fuente.ent.ayu  :=  INT(fuente.byte.ayu)
            SEQ k=0 FOR 7
              SEQ
                --{{{  Procesos
                result.ayu    :=  fuente.ent.ayu /\ t.128
                IF
                  result.ayu = t.128
                    IF
                      y.ayu.ayu >= 0
                        --{{{  Dibuja Pixel ( x.ayu.ayu, y.ayu.ayu )
                        DibujaPuntos ( x.ayu.ayu, y.ayu.ayu )
                        --}}}
                      TRUE
                        SKIP
                  TRUE
                    SKIP
                fuente.ent.ayu  :=  fuente.ent.ayu << 1
                x.ayu.ayu       :=  x.ayu.ayu + 1
                --}}}
            x.ayu.ayu      :=  x.ayu
            y.ayu.ayu      :=  y.ayu.ayu - 1
            --}}}
        x.ayu      :=  x.ayu + 8
        y.ayu.ayu := y.ayu
        IF
          ( x.ayu + 8 ) >= x.maximo.ent
            SEQ
              x.ayu      := 0
              y.ayu      := y.ayu - 12
              y.ayu.ayu := y.ayu
          TRUE
            SKIP
  --}}}
:
--}}}
--
--{{{  Escribir Mensaje con Caracteres Doble en Pantalla Grafica
PROC EscribirMensajeCaracterDoble  ( VAL INT color.ent, VAL INT x.mens,
                                     VAL INT y.mens, VAL []INT cadena.ent )
  --{{{  Declaraciones
  BYTE fuente.byte.ayu, byte.ayu :
  INT i.ayu, j.ayu, k.ayu, fuente.ent.ayu, t.128 :
  INT ent.ayu, x.ayu, x.ayu.ayu, y.ayu, y.ayu.ayu, result.ayu :
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Asignaciones
    t.128          :=   128
    x.ayu          :=   x.mens
    x.ayu.ayu      :=   x.ayu
    y.ayu          :=   y.mens
    y.ayu.ayu      :=   y.ayu
    result.ayu     :=   0
    --}}}
    color.elegido := BYTE(color.ent)
    Color ( color.elegido )
    SEQ i=0 FOR ( longitud - 3 )
      SEQ
        i.ayu   :=  cadena.ent[i]
        SEQ j=0 FOR 12
          SEQ
            --{{{  Procesos
            fuente.byte.ayu :=  fuente.elegido.by[i.ayu][j]
            fuente.ent.ayu  :=  INT(fuente.byte.ayu)
            SEQ k=0 FOR 7
              SEQ
                --{{{  Procesos
                result.ayu    :=  fuente.ent.ayu /\ t.128
                IF
                  result.ayu = t.128
                    SEQ
                      IF
                        y.ayu.ayu >= 0
                          SEQ
                            --{{{  Dibuja Pixel ( x.ayu.ayu, y.ayu.ayu )
                            DibujaPuntos ( x.ayu.ayu, y.ayu.ayu )
                            DibujaPuntos ( (x.ayu.ayu+1), y.ayu.ayu )
                            DibujaPuntos ( x.ayu.ayu, (y.ayu.ayu-1) )
                            DibujaPuntos ( (x.ayu.ayu+1), (y.ayu.ayu-1) )
                            --}}}
                        TRUE
                          SKIP
                  TRUE
                    SKIP
                fuente.ent.ayu  :=  fuente.ent.ayu << 1
                x.ayu.ayu       :=  x.ayu.ayu + 2
                --}}}
            x.ayu.ayu      :=  x.ayu
            y.ayu.ayu      :=  y.ayu.ayu - 2
            --}}}
        x.ayu      :=  x.ayu + 16
        y.ayu.ayu := y.ayu
        IF
          ( x.ayu + 16 ) >= x.maximo.ent
            SEQ
              x.ayu      := 0
              y.ayu      := y.ayu - 24
              y.ayu.ayu := y.ayu
          TRUE
            SKIP
  --}}}
:
--}}}
--
--{{{  Escribir Mensaje con Caracteres de Ancho/Alto Variables
PROC EscribirMensajeCaracterVariable (VAL INT color.ent, VAL INT x.mens,
```

```
            i.ayu       := i - xo.ent
            x.quad.int  := i.ayu * i.ayu
            --}}}
        TRUE
          SKIP
      --}}}
  --}}}
  --{{{  Asignaciones Para r.ext
  cuanto.r.ext := r.ent.ext + 1
  k.ext        := 0
  k.ayu.ext    := 0
  i            := xo.ent
  i.ayu        := 0
  x.quad.ext   := 0
  --}}}
  --{{{  Coge el Contorno Para r.ext
  SEQ j = lim.inf.y.ent.ext FOR cuanto.r.ext
    SEQ
      --{{{  Procesos
      j.ayu          := j - yo.ent
      y.quad.ext     := j.ayu * j.ayu
      xq.mas.yq.ext  := x.quad.ext + y.quad.ext
      WHILE xq.mas.yq.ext  <= r.quad.ext
        SEQ
          --{{{  Procesos
          contorno.x.ext[k.ext] := xo.ent + i.ayu
          contorno.y.ext[k.ext] := yo.ent + j.ayu
          i                     := i + 1
          k.ext                 := k.ext + 1
          i.ayu                 := i - xo.ent
          x.quad.ext            := i.ayu * i.ayu
          xq.mas.yq.ext         := x.quad.ext + y.quad.ext
          --}}}
      IF
        i > xo.ent
          SEQ
            --{{{  Procesos
            i           := i - 1
            i.ayu       := i - xo.ent
            x.quad.ext  := i.ayu * i.ayu
            --}}}
        TRUE
          SKIP
      --}}}
  --}}}
  --{{{  Dibuja la Primera Parte de la Corona
  k.ayu.ext := 0
  WHILE contorno.y.ext[k.ayu.ext] <= contorno.y.int[0]
    SEQ
      --{{{  Procesos
      IF
        contorno.y.ext[(k.ayu.ext + 1)] <> contorno.y.ext[k.ayu.ext]
          SEQ
            --{{{  Procesos
            delta.l.x.ext   := contorno.x.ext[k.ayu.ext] - xo.ent
            delta.l.y.ext   := - ( contorno.y.ext[k.ayu.ext] - yo.ent
            cuanto.x.ext    := ( 2 * delta.l.x.ext ) + 1
            x.arregla.ext   := xo.ent - delta.l.x.ext
            IniciaVectorBytes ( x.arregla.ext,
                                contorno.y.ext[k.ayu.ext],
                                cuanto.x.ext, color )
            y.arregla.ext ' := yo.ent + delta.l.y.ext
            IniciaVectorBytes ( x.arregla.ext, y.arregla.ext,
                                cuanto.x.ext, color )
            --}}}
        TRUE
          SKIP
      k.ayu.ext := k.ayu.ext + 1
      --}}}
  --}}}
  --{{{  Dibuja la Segunda Parte de la Corona
  k.ayu.int := 1
  WHILE contorno.y.ext[k.ayu.ext] < yo.ent
    SEQ
      --{{{  Procesos
      IF
        contorno.y.int[k.ayu.int] <> contorno.y.int[(k.ayu.int - 1 )]
          SEQ
            IF
              contorno.y.ext[(k.ayu.ext + 1)] <> contorno.y.ext[k.ayu.
                SEQ
                  --{{{  Procesos
                  delta.l.x.int   := contorno.x.ext[k.ayu.ext] - conto
                  delta.l.x.ext   := contorno.x.ext[k.ayu.ext] - xo.en
                  delta.l.y.int   := - ( contorno.y.int[k.ayu.int] - y
                  cuanto.x.int    := delta.l.x.int  + 1
                  x.arregla.ext   := xo.ent - delta.l.x.ext
                  IniciaVectorBytes ( contorno.x.int[k.ayu.int], conto
                                      cuanto.x.int, color )
                  IniciaVectorBytes ( x.arregla.ext, contorno.y.ext[k.
                                      cuanto.x.int, color )
                  y.arregla.int   := yo.ent + delta.l.y.int
                  IniciaVectorBytes ( contorno.x.int[k.ayu.int], y.arr
                                      cuanto.x.int, color )
                  IniciaVectorBytes ( x.arregla.ext, y.arregla.int,
                                      cuanto.x.int, color )
                  --}}}
                  k.ayu.int := k.ayu.int + 1
                  k.ayu.ext := k.ayu.ext + 1
              TRUE
                k.ayu.ext := k.ayu.ext + 1
        TRUE
          k.ayu.int := k.ayu.int + 1
      --}}}
  --}}}
  --{{{  Dibuja la dos Ultimas Lineas
  x.arregla.ext := xo.ent - r.ent.ext
  cuanto.x.ext := ( r.ent.ext - r.ent.int ) + 1
  IniciaVectorBytes ( x.arregla.ext, yo.ent, cuanto.x.ext, color )
  x.arregla.int := xo.ent + r.ent.int
  IniciaVectorBytes ( x.arregla.int, yo.ent, cuanto.x.ext, color )
  --}}}

    --}}}
TRUE
  SKIP
--}}}
:
--}}}
--
--}}}
--
--}}}
--
--{{{F Otras Rutinas
--:::F Otragr03.tsr
--
--{{{  Caracterers
--
--{{{  Mensajes
--
--{{{  Escribir Mensaje en Pantalla Grafica
```

```
        color.ayu := color.ayu + delta.color
        WHILE color.ayu > 255
          color.ayu  := color.ayu - 128
        color.cambia := BYTE(color.ayu)
  --}}}
:
--}}}
--
--{{{  EJEMPLO 1 : Dibujar Cuadrados y Circulos
PROC DibujaEjemplo1 ( VAL BYTE color.inicial )
  --{{{  Procesos
  --{{{  Variables
  INT delta.color, dos.delta.color, color.ayu  :
  BYTE color.cambia, color.ayu1 :
  REAL32 delta.coord :
  REAL32 r.coord :
  REAL32 xi.coord, yi.coord, xf.coord, yf.coord, xo.coord, yo.coord :
  REAL32 x.minimo.coord, y.minimo.coord :
  REAL32 x.maximo.coord, y.maximo.coord :
  --}}}
  --{{{  Constantes
  VAL xi.coord.ent          IS   128 :
  VAL yi.coord.ent          IS     0 :
  VAL xf.coord.ent          IS   895 :
  VAL yf.coord.ent          IS   767 :
  VAL xo.coord.ent          IS   511 :
  VAL yo.coord.ent          IS   384 :
  VAL  r.coord.ent          IS   383 :
  VAL  x.minimo.coord.ent   IS     0 :
  VAL  x.maximo.coord.ent   IS  1023 :
  VAL  y.minimo.coord.ent   IS     0 :
  VAL  y.maximo.coord.ent   IS   767 :
  VAL  x.limite.coord.ent   IS   500 :
  --}}}
  SEQ
    --{{{  Asignacion de Variables
    xi.coord         :=  REAL32 ROUND(xi.coord.ent)
    yi.coord         :=  REAL32 ROUND(yi.coord.ent)
    xf.coord         :=  REAL32 ROUND(xf.coord.ent)
    yf.coord         :=  REAL32 ROUND(yf.coord.ent)
    xo.coord         :=  REAL32 ROUND(xo.coord.ent)
    yo.coord         :=  REAL32 ROUND(yo.coord.ent)
    r.coord          :=  REAL32 ROUND( r.coord.ent)
    x.minimo.coord   :=  REAL32 ROUND(x.minimo.coord.ent)
    x.maximo.coord   :=  REAL32 ROUND(x.maximo.coord.ent)
    y.minimo.coord   :=  REAL32 ROUND(y.minimo.coord.ent)
    y.maximo.coord   :=  REAL32 ROUND(y.maximo.coord.ent)
    delta.color      :=  13
    dos.delta.color  :=  2 * delta.color
    delta.coord      :=  REAL32 ROUND(20)
    color.cambia     :=  color.inicial
    color.ayu        :=  255
    color.ayu1       :=  BYTE(color.ayu)
    --}}}
    IniciaPantalla ( x.maximo.coord, y.maximo.coord )
    Color ( color.ayu1 )
    Rectangulo ( x.minimo.coord, y.minimo.coord, x.maximo.coord, y.maximo.coor
    WHILE xi.coord <= ( REAL32 ROUND(x.limite.coord.ent) )
      SEQ
        Color ( color.cambia )
        --{{{  Rectangulo
        Rectangulo ( xi.coord, yi.coord, xf.coord, yf.coord )
        xi.coord :=  xi.coord + delta.coord
        yi.coord :=  yi.coord + delta.coord
        xf.coord :=  xf.coord - delta.coord
        yf.coord :=  yf.coord - delta.coord
        --}}}
        color.ayu := INT(color.cambia)
        color.ayu := color.ayu + dos.delta.color
        WHILE color.ayu > 255
          color.ayu  := color.ayu - 1
        color.cambia := BYTE(color.ayu)
        Color ( color.cambia )
        --{{{  Circulo
        Circulo ( xo.coord, yo.coord, r.coord )
        r.coord := r.coord - delta.coord
        --}}}
        color.ayu    := color.ayu - delta.color
        color.cambia := BYTE(color.ayu)
  --}}}
:
--}}}
--
--{{{  EJEMPLO 2 : Dibujar Circulos de un Modo Dinamico
--
--{{{  Paleta Patron
PROC DibujaEjemplo2Patron ( VAL INT32 semilla )
  --{{{  Procesos
  --{{{  Variables/constantes
  --{{{  Variables
  INT parada.ayu :
  INT32 semilla.int :
  INT color.ayu, num.pseud.aleat :
  INT color.cambia.ent, delta.color.ent :
  BYTE color.cambia, color.ayu1 :
  BYTE delta.color.byte, color.inic.byte :
  REAL32 num.pseud.aleat.r, real.diez, real.cien, real.mil :
  REAL32 delta.coord.x, delta.coord.y :
  REAL32 delta.coord.inic.x, delta.coord.inic.y, r.delta.coord :
  REAL32 r.coord, r.coord.max, r.coord.min :
  REAL32 xo.coord, yo.coord :
  REAL32 xo.max , yo.max :
  REAL32 xo.min , yo.min :
  REAL32 x.minimo.coord, y.minimo.coord :
  REAL32 x.maximo.coord, y.maximo.coord :
  BOOL arriba, derecha, mayor, activo :
  --}}}
  --{{{  Constantes
  --{{{  x/y
  VAL     xo.coord.inic.ent    IS      512 :
  VAL     yo.coord.inic.ent    IS      384 :
  VAL     x.minimo.coord.ent   IS        0 :
  VAL     x.maximo.coord.ent   IS     1023 :
  VAL     y.minimo.coord.ent   IS        0 :
  VAL     y.maximo.coord.ent   IS      767 :
  VAL   dt.coord.inic.x.ent    IS       15 :
  VAL   dt.coord.inic.y.ent    IS       15 :
  --}}}
  --{{{  radio
  VAL  r.coord.inic.ent      IS           80 :
  VAL  r.coord.max.ent       IS          120 :
  VAL  r.coord.min.ent       IS           40 :
  VAL  r.dt.ent              IS            1 :
  --}}}
  --{{{  color
  VAL  color.max             IS    BYTE(255) :
  VAL  color.nigro           IS      BYTE(0) :
  VAL  delta.color.ent       IS            1 :
  VAL  delta.color.byte      IS      BYTE(1) :
  VAL  color.inic.byte       IS      BYTE(0) :
  --}}}
  --{{{  aleatoria
```

```
                                    VAL INT y.mens, VAL INT ancho.esc,
                                    VAL INT alto.esc, VAL []INT cadena.ent)
  --{{{  Declaraciones
  BYTE fuente.byte.ayu, byte.ayu :
  INT i.ayu, j.ayu, k.ayu, fuente.ent.ayu, t.128 :
  INT ent.ayu, x.ayu, x.ayu.ayu, y.ayu, y.ayu.ayu, result.ayu :
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Asignaciones
    t.128          :=   128
    x.ayu          :=   x.mens
    x.ayu.ayu      :=   x.ayu
    y.ayu          :=   y.mens
    y.ayu.ayu      :=   y.ayu
    result.ayu     :=   0
    --}}}
    color.elegido := BYTE(color.ent)
    Color ( color.elegido )
    SEQ i=0 FOR ( longitud - 5 )
      SEQ
        i.ayu   :=  cadena.ent[i]
        SEQ j=0 FOR 12
          SEQ
            --{{{  Procesos
            fuente.byte.ayu :=  fuente.elegido.by[i.ayu][j]
            fuente.ent.ayu  :=  INT(fuente.byte.ayu)
            SEQ k=0 FOR 7
              SEQ
                --{{{  Procesos
                result.ayu    :=  fuente.ent.ayu /\ t.128
                IF
                  result.ayu = t.128
                    SEQ
                      IF
                        ( y.ayu.ayu - alto.esc ) >= 0
                          SEQ i=x.ayu.ayu FOR ancho.esc
                            SEQ j=(y.ayu.ayu-alto.esc) FOR alto.esc
                              DibujaPuntos ( i, j )
                        TRUE
                          SKIP
                  TRUE
                    SKIP
                fuente.ent.ayu  :=  fuente.ent.ayu << 1
                x.ayu.ayu       :=  x.ayu.ayu + ancho.esc
                --}}}
            x.ayu.ayu      :=  x.ayu
            y.ayu.ayu      :=  y.ayu.ayu - alto.esc
            --}}}
        x.ayu      :=  x.ayu + ( ancho.esc * 8 )
        y.ayu.ayu := y.ayu
        IF
          ( x.ayu + ( ancho.esc * 8 ) ) >= x.maximo.ent
            SEQ
              x.ayu      := 0
              y.ayu      := y.ayu - ( alto.esc * 12 )
              y.ayu.ayu := y.ayu
          TRUE
            SKIP
    --}}}
:
--}}}
--
--}}}
--
--{{{  Escribir Fichero ASCII en la Pantalla

PROC EscribirFicherosASCII ( )
  SEQ
    --{{{  Procesos
    --}}}
:
--}}}
--
--{{{  Cargar Fuente de Caracteres
PROC CargaFuenteCaracteres ( VAL INT num.carac, VAL INT fuente.x,
                             VAL INT fuente.y, VAL INT longitud.fuente )
  SEQ
    --{{{  Procesos
    Org.Mp ? [ fuente.elegido.ent FROM 0 FOR longitud.fuente ]
    SEQ i=0 FOR num.carac
      SEQ j=0 FOR fuente.y
        fuente.elegido.by[i][j] := BYTE(fuente.elegido.ent[i][j])
    --[fuente.elegido.by FROM 0 FOR longitud.fuente] :=
    --            [ BYTE(fuente.elegido.ent) FROM 0 FOR longitud.fuente ]
    --}}}
:
--}}}
--
--}}}
--
--}}}F
--
--{{{  Ejemplos de Utilizacion
--
--{{{  EJEMPLO 0 : Dibujar Ventanas
PROC DibujaEjemplo0 ( VAL BYTE color.inicial )
  --{{{  Procesos
  INT delta.color, color.ayu :
  BYTE color.cambia :
  REAL32 delta.coord :
  REAL32 xi.coord, yi.coord, xf.coord, yf.coord :
  REAL32 x.maximo.coord, y.maximo.coord :
  VAL xi.coord.ent IS   0 :
  VAL yi.coord.ent IS   0 :
  VAL xf.coord.ent IS  1023 :
  VAL yf.coord.ent IS   767 :
  VAL y.maximo.coord.ent IS   767 :
  VAL x.maximo.coord.ent IS  1023 :
  VAL y.maximo.coord.ent IS   767 :
  VAL x.limite.coord.ent IS   380 :
  SEQ
    --{{{  Asignacion de Variables
    xi.coord       := REAL32 ROUND(xi.coord.ent)
    yi.coord       := REAL32 ROUND(yi.coord.ent)
    xf.coord       := REAL32 ROUND(xf.coord.ent)
    yf.coord       := REAL32 ROUND(yf.coord.ent)
    x.maximo.coord := REAL32 ROUND(x.maximo.coord.ent)
    y.maximo.coord := REAL32 ROUND(y.maximo.coord.ent)
    delta.color    := 13
    delta.coord    := REAL32 ROUND(20)
    color.cambia   := color.inicial
    --}}}
    IniciaPantalla ( x.maximo.coord, y.maximo.coord )
    WHILE xi.coord <= ( REAL32 ROUND(x.limite.coord.ent) )
      SEQ
        Color ( color.cambia )
        Rectangulo ( xi.coord, yi.coord, xf.coord, yf.coord )
        xi.coord  := xi.coord + delta.coord
        yi.coord  := yi.coord + delta.coord
        xf.coord  := xf.coord - delta.coord
        yf.coord  := yf.coord - delta.coord
        color.ayu := INT(color.cambia)
```

```
            --{{{  Atualiza Radio: derecha T, arriba T, mayor T.
            r.coord := r.coord + r.delta.coord
            IF
              r.coord > r.coord.max
                SEQ
                  r.coord := r.coord.max
                  mayor   := FALSE
              TRUE
                SKIP
            xo.max  :=  x.maximo.coord - r.coord
            yo.max  :=  y.maximo.coord - r.coord
            --}}}
            --{{{  Atualiza yo: derecha T, arriba T, mayor T.
            yo.coord := yo.coord + delta.coord.y
            IF
              yo.coord > yo.max
                SEQ
                  yo.coord := yo.max
                  arriba   := FALSE
              TRUE
                SKIP
            --}}}
            --{{{  Atualiza xo: derecha T, arriba T, mayor T.
            xo.coord := xo.coord + delta.coord.x
            IF
              xo.coord > xo.max
                SEQ
                  xo.coord := xo.max
                  derecha  := FALSE
              TRUE
                SKIP
            --}}}
          --}}}
        FALSE
          --{{{  case derecha T, arriba T, mayor F
          SEQ
            --{{{  Atualiza Radio: derecha T, arriba T, mayor F.
            r.coord := r.coord - r.delta.coord
            IF
              r.coord < r.coord.min
                SEQ
                  r.coord := r.coord.min
                  mayor   := TRUE
              TRUE
                SKIP
            xo.max  :=  x.maximo.coord - r.coord
            yo.max  :=  y.maximo.coord - r.coord
            --}}}
            --{{{  Atualiza yo: derecha T, arriba T, mayor F.
            yo.coord := yo.coord + delta.coord.y
            IF
              yo.coord > yo.max
                SEQ
                  yo.coord := yo.max
                  arriba   := FALSE
              TRUE
                SKIP
            --}}}
            --{{{  Atualiza xo: derecha T, arriba T, mayor F.
            xo.coord := xo.coord + delta.coord.x
            IF
              xo.coord > xo.max
                SEQ
                  xo.coord := xo.max
                  derecha  := FALSE
              TRUE
                SKIP
            --}}}
          --}}}
        FALSE
          --{{{  case derecha T, arriba F
          CASE mayor
            --{{{  case derecha T, arriba F, mayor
            TRUE
              --{{{  case derecha T, arriba F, mayor T
              SEQ
                --{{{  Atualiza Radio: derecha T, arriba F, mayor T
                r.coord := r.coord + r.delta.coord
                IF
                  r.coord > r.coord.max
                    SEQ
                      r.coord := r.coord.max
                      mayor   := FALSE
                  TRUE
                    SKIP
                xo.max  :=  x.maximo.coord - r.coord
                yo.min  :=  y.minimo.coord + r.coord
                --}}}
                --{{{  Atualiza yo: derecha T, arriba F, mayor T.
                yo.coord := yo.coord - delta.coord.y
                IF
                  yo.coord < yo.min
                    SEQ
                      yo.coord := yo.min
                      arriba   := TRUE
                  TRUE
                    SKIP
                --}}}
                --{{{  Atualiza xo: derecha T, arriba F, mayor T.
                xo.coord := xo.coord + delta.coord.x
                IF
                  xo.coord > xo.max
                    SEQ
                      xo.coord := xo.max
                      derecha  := FALSE
                  TRUE
                    SKIP
                --}}}
              --}}}
            FALSE
              --{{{  case derecha T, arriba F, mayor F
              SEQ
                --{{{  Atualiza Radio: derecha T, arriba F, mayor F.
                r.coord := r.coord - r.delta.coord
                IF
                  r.coord < r.coord.min
                    SEQ
                      r.coord := r.coord.min
                      mayor   := TRUE
                  TRUE
                    SKIP
                xo.max  :=  x.maximo.coord - r.coord
                yo.min  :=  y.minimo.coord + r.coord
                --}}}
                --{{{  Atualiza yo: derecha T, arriba F, mayor F.
                yo.coord := yo.coord - delta.coord.y
                IF
                  yo.coord < yo.min
                    SEQ
                      yo.coord := yo.min
```

```
VAL   entero.diez     IS     10 :
VAL   entero.cien     IS    100 :
VAL   entero.mil      IS   1000 :
--}}}
--}}}
--}}}
SEQ
  --{{{  Asignacion de Variables
  --{{{  Variables
  --{{{  logicas
  activo                 :=  TRUE
  --}}}
  --{{{  x/y
  xo.coord               :=  REAL32 ROUND(xo.coord.inic.ent)
  yo.coord               :=  REAL32 ROUND(yo.coord.inic.ent)
  x.minimo.coord         :=  REAL32 ROUND(x.minimo.coord.ent)
  x.maximo.coord         :=  REAL32 ROUND(x.maximo.coord.ent)
  y.minimo.coord         :=  REAL32 ROUND(y.minimo.coord.ent)
  y.maximo.coord         :=  REAL32 ROUND(y.maximo.coord.ent)
  delta.coord.inic.x     :=  REAL32 ROUND(dt.coord.inic.x.ent)
  delta.coord.inic.y     :=  REAL32 ROUND(dt.coord.inic.y.ent)
  delta.coord.x          :=  REAL32 ROUND(20)
  delta.coord.y          :=  REAL32 ROUND(20)
  --{{{  xo/yo, max/min
  xo.max                 :=  x.maximo.coord - r.coord
  yo.max                 :=  y.maximo.coord - r.coord
  xo.min                 :=  x.minimo.coord + r.coord
  yo.min                 :=  y.minimo.coord + r.coord
  --}}}
  --}}}
  --{{{  radio
  r.coord                :=  REAL32 ROUND( r.coord.inic.ent)
  r.coord.min            :=  REAL32 ROUND( r.coord.min.ent)
  r.coord.max            :=  REAL32 ROUND( r.coord.max.ent)
  r.delta.coord          :=  REAL32 ROUND(r.dt.ent)
  --}}}
  --{{{  color
  color.cambia           := color.inic.byte
  --}}}
  --{{{  aleatoria
  real.diez                      :=  REAL32 ROUND(entero.diez)
  real.cien                      :=  REAL32 ROUND(entero.cien)
  real.mil                       :=  REAL32 ROUND(entero.mil)
  semilla.int                    :=  semilla
  num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
  num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
  num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
  num.pseud.aleat                :=  num.pseud.aleat \ 10
  --}}}
  --}}}
  --{{{  Variables Pseudo Aleatorias
  --{{{  Genera Arriba
  IF
    num.pseud.aleat  >= 5
      arriba := TRUE
    TRUE
      arriba := FALSE
  --}}}
  --{{{  Genera Numero Pseudoaleatorio de 0 a 9 ( con RAN )
  num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
  num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
  num.pseud.aleat                :=  num.pseud.aleat \ 10
  --}}}
  --{{{  Genera Derecha
  IF
    num.pseud.aleat  >= 5
      derecha := TRUE
    TRUE
      derecha := FALSE
  --}}}
  --{{{  Genera Numero Pseudoaleatorio de 0 a 9 ( con RAN )
  num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
  num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
  num.pseud.aleat                :=  num.pseud.aleat \ 10
  --}}}
  --{{{  Genera Mayor
  IF
    num.pseud.aleat  >= 5
      mayor := TRUE
    TRUE
      mayor := FALSE
  --}}}
  --{{{  Genera Numero Pseudoaleatorio de 0 a 9 ( con RAN )
  num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
  num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
  num.pseud.aleat                :=  num.pseud.aleat \ 10
  --}}}
  --{{{  Genera delta.x
  delta.coord.x       := REAL32 ROUND( num.pseud.aleat )
  delta.coord.x       := delta.coord.inic.x -  delta.coord.x
  --}}}
  --{{{  Genera Numero Pseudoaleatorio de 0 a 9 ( con RAN )
  num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
  num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
  num.pseud.aleat                :=  num.pseud.aleat \ 10
  --}}}
  --{{{  Genera delta.y
  delta.coord.y        := REAL32 ROUND( num.pseud.aleat )
  delta.coord.y        := delta.coord.inic.y -  delta.coord.y
  --}}}
  --}}}
  --}}}
  IniciaPantalla ( x.maximo.coord, y.maximo.coord )
  BorraPantalla ()
  --{{{  Cambia Color
  color.cambia.ent := INT(color.cambia)
  color.cambia.ent := color.cambia.ent + delta.color.ent
  color.cambia := BYTE(color.cambia.ent)
  Color ( color.cambia )
  --}}}
  Circulo ( xo.coord, yo.coord, r.coord )
  WHILE activo
    PRI ALT
      Org.Mp ? parada.ayu
        activo := FALSE
      TRUE & SKIP
        --{{{  Procesos
        SEQ
          --{{{  Procesos
          CASE derecha
            --{{{  case derecha
            TRUE
              --{{{  case derecha T
              CASE arriba
                --{{{  case derecha T, arriba
                TRUE
                  --{{{  case derecha T, arriba T
                  CASE mayor
                    --{{{  case derecha T, arriba T, mayor
                    TRUE
                      --{{{  case derecha T, arriba T, mayor T
                      SEQ
```

```
                    IF
                      xo.coord < xo.min
                        SEQ
                          xo.coord := xo.min
                          derecha  := TRUE
                      TRUE
                        SKIP
                    --}}}
                  --}}}
                FALSE
                  --{{{  case derecha F, arriba F, mayor F
                  SEQ
                    --{{{  Atualiza Radio: derecha F, arriba F, mayor F.
                    r.coord := r.coord - r.delta.coord
                    IF
                      r.coord < r.coord.min
                        SEQ
                          r.coord := r.coord.min
                          mayor   := TRUE
                      TRUE
                        SKIP
                    xo.min  :=  x.minimo.coord + r.coord
                    yo.min  :=  y.minimo.coord + r.coord
                    --}}}
                    --{{{  Atualiza yo: derecha F, arriba F, mayor F.
                    yo.coord := yo.coord - delta.coord.y
                    IF
                      yo.coord < yo.min
                        SEQ
                          yo.coord := yo.min
                          arriba   := TRUE
                      TRUE
                        SKIP
                    --}}}
                    --{{{  Atualiza xo: derecha F, arriba F, mayor F.
                    xo.coord := xo.coord - delta.coord.x
                    IF
                      xo.coord < xo.min
                        SEQ
                          xo.coord := xo.min
                          derecha  := TRUE
                      TRUE
                        SKIP
                    --}}}
                  --}}}
                --}}}
              --}}}
            --}}}
          --}}}
        --}}}
      --}}}
      --{{{  Cambia Color
      color.cambia.ent := INT(color.cambia)
      color.cambia.ent := color.cambia.ent + delta.color.ent
      color.cambia := BYTE(color.cambia.ent)
      Color ( color.cambia )
      --}}}
      --{{{  Circulo
      Circulo ( xo.coord, yo.coord, r.coord )
      --}}}
      Color ( color.nigro )
      Circunferencia ( xo.coord, yo.coord, r.coord )
      --}}}
    --}}}
  --}}}
:
--}}}
```

```
--
--{{{  Paleta Suave ( con Contorno )
PROC DibujaEjemplo2Suave ( VAL INT32 semilla )
  --{{{  Procesos
  --{{{  Variables/constantes
  --{{{  Variables
  INT parada.ayu :
  INT32 semilla.int :
  INT color.ayu, num.pseud.aleat :
  INT color.cambia.ent, delta.color.ent :
  BYTE color.cambia, color.ayu1 :
  BYTE delta.color.byte, color.inic.byte :
  REAL32 num.pseud.aleat.r, real.diez, real.cien, real.mil :
  REAL32 delta.coord.x, delta.coord.y :
  REAL32 delta.coord.inic.x, delta.coord.inic.y, r.delta.coord :
  REAL32 r.coord, r.coord.max, r.coord.min :
  REAL32 xo.coord, yo.coord :
  REAL32 xo.max , yo.max :
  REAL32 xo.min , yo.min :
  REAL32 x.minimo.coord, y.minimo.coord :
  REAL32 x.maximo.coord, y.maximo.coord :
  BOOL arriba, derecha, mayor, activo :
  --}}}
  --{{{  Constantes
  --{{{  x/y
  VAL     xo.coord.inic.ent    IS      512 :
  VAL     yo.coord.inic.ent    IS      384 :
  VAL     x.minimo.coord.ent   IS        0 :
  VAL     x.maximo.coord.ent   IS     1023 :
  VAL     y.minimo.coord.ent   IS        0 :
  VAL     y.maximo.coord.ent   IS      767 :
  VAL    dt.coord.inic.x.ent   IS       15 :
  VAL    dt.coord.inic.y.ent   IS       15 :
  --}}}
  --{{{  radio
  VAL  r.coord.inic.ent      IS           80 :
  VAL  r.coord.max.ent       IS          120 :
  VAL  r.coord.min.ent       IS           40 :
  VAL  r.dt.ent              IS            1 :
  --}}}
  --{{{  color
  VAL  color.max             IS     BYTE(255) :
  VAL  color.nigro           IS       BYTE(0) :
  VAL  delta.color.ent       IS            1 :
  VAL  delta.color.byte      IS       BYTE(1) :
  VAL  color.inic.byte       IS       BYTE(0) :
  --}}}
  --{{{  aleatoria
  VAL   entero.diez      IS     10 :
  VAL   entero.cien      IS    100 :
  VAL   entero.mil       IS   1000 :
  --}}}
  --}}}
  --}}}
  SEQ
    --{{{  Asignacion de Variables
    --{{{  Variables
    --{{{  logicas
    activo                 :=  TRUE
    --}}}
    --{{{  x/y
    xo.coord               :=  REAL32 ROUND(xo.coord.inic.ent)
    yo.coord               :=  REAL32 ROUND(yo.coord.inic.ent)
    x.minimo.coord         :=  REAL32 ROUND(x.minimo.coord.ent)
    x.maximo.coord         :=  REAL32 ROUND(x.maximo.coord.ent)
    y.minimo.coord         :=  REAL32 ROUND(y.minimo.coord.ent)
```

```
                arriba   := TRUE
              TRUE
                SKIP
            --}}}
            --{{{  Atualiza xo: derecha T, arriba F, mayor F.
            xo.coord := xo.coord + delta.coord.x
            IF
              xo.coord > xo.max
                SEQ
                  xo.coord := xo.max
                  derecha  := FALSE
              TRUE
                SKIP
            --}}}
          --}}}
        --}}}
      --}}}
    --}}}
  FALSE
    --{{{  case derecha F
    CASE arriba
      --{{{  case derecha F, arriba
      TRUE
        --{{{  case derecha F, arriba T
        CASE mayor
          --{{{  case derecha F, arriba T, mayor
          TRUE
            --{{{  case derecha F, arriba T, mayor T
            SEQ
              --{{{  Atualiza Radio: derecha F, arriba T, mayor T.
              r.coord := r.coord + r.delta.coord
              IF
                r.coord > r.coord.max
                  SEQ
                    r.coord := r.coord.max
                    mayor   := FALSE
                TRUE
                  SKIP
              xo.min  :=  x.minimo.coord + r.coord
              yo.max  :=  y.maximo.coord - r.coord
              --}}}
              --{{{  Atualiza yo: derecha F, arriba T, mayor T.
              yo.coord := yo.coord + delta.coord.y
              IF
                yo.coord > yo.max
                  SEQ
                    yo.coord := yo.max
                    arriba   := FALSE
                TRUE
                  SKIP
              --}}}
              --{{{  Atualiza xo: derecha F, arriba T, mayor T.
              xo.coord := xo.coord - delta.coord.x
              IF
                xo.coord < xo.min
                  SEQ
                    xo.coord := xo.min
                    derecha  := TRUE
                TRUE
                  SKIP
              --}}}
            --}}}
          FALSE
            --{{{  case derecha F, arriba T, mayor F
            SEQ
              --{{{  Atualiza Radio: derecha F, arriba T, mayor F
              r.coord := r.coord - r.delta.coord
              IF
                r.coord < r.coord.min
                  SEQ
                    r.coord := r.coord.min
                    mayor   := TRUE
                TRUE
                  SKIP
              xo.min  :=  x.minimo.coord + r.coord
              yo.max  :=  y.maximo.coord - r.coord
              --}}}
              --{{{  Atualiza yo: derecha F, arriba T, mayor F.
              yo.coord := yo.coord + delta.coord.y
              IF
                yo.coord > yo.max
                  SEQ
                    yo.coord := yo.max
                    arriba   := FALSE
                TRUE
                  SKIP
              --}}}
              --{{{  Atualiza xo: derecha F, arriba T, mayor F.
              xo.coord := xo.coord - delta.coord.x
              IF
                xo.coord < xo.min
                  SEQ
                    xo.coord := xo.min
                    derecha  := TRUE
                TRUE
                  SKIP
              --}}}
            --}}}
        --}}}
      --}}}
    --}}}
  FALSE
    --{{{  case derecha F, arriba F
    CASE mayor
      --{{{  case derecha F, arriba F, mayor
      TRUE
        --{{{  case derecha F, arriba F, mayor T
        SEQ
          --{{{  Atualiza Radio: derecha F, arriba F, mayor T.
          r.coord := r.coord + r.delta.coord
          IF
            r.coord > r.coord.max
              SEQ
                r.coord := r.coord.max
                mayor   := FALSE
            TRUE
              SKIP
          xo.min  :=  x.minimo.coord + r.coord
          yo.min  :=  y.minimo.coord + r.coord
          --}}}
          --{{{  Atualiza yo: derecha F, arriba F, mayor T.
          yo.coord := yo.coord - delta.coord.y
          IF
            yo.coord < yo.min
              SEQ
                yo.coord := yo.min
                arriba   := TRUE
            TRUE
              SKIP
          --}}}
          --{{{  Atualiza xo: derecha F, arriba F, mayor T
          xo.coord := xo.coord - delta.coord.x
```

```
                arriba   := FALSE
            TRUE
              SKIP
          --}}}
          --{{{  Atualiza xo: derecha T, arriba T, mayor T.
          xo.coord := xo.coord + delta.coord.x
          IF
            xo.coord > xo.max
              SEQ
                xo.coord := xo.max
                derecha  := FALSE
            TRUE
              SKIP
          --}}}
        --}}}
      FALSE
        --{{{  case derecha T, arriba T, mayor F
        SEQ
          --{{{  Atualiza Radio: derecha T, arriba T, mayor F.
          r.coord := r.coord - r.delta.coord
          IF
            r.coord < r.coord.min
              SEQ
                r.coord := r.coord.min
                mayor   := TRUE
            TRUE
              SKIP
          xo.max  :=  x.maximo.coord - r.coord
          yo.max  :=  y.maximo.coord - r.coord
          --}}}
          --{{{  Atualiza yo: derecha T, arriba T, mayor F.
          yo.coord := yo.coord + delta.coord.y
          IF
            yo.coord > yo.max
              SEQ
                yo.coord := yo.max
                arriba   := FALSE
            TRUE
              SKIP
          --}}}
          --{{{  Atualiza xo: derecha T, arriba T, mayor F.
          xo.coord := xo.coord + delta.coord.x
          IF
            xo.coord > xo.max
              SEQ
                xo.coord := xo.max
                derecha  := FALSE
            TRUE
              SKIP
          --}}}
        --}}}
      --}}}
    --}}}
  FALSE
    --{{{  case derecha T, arriba F
    CASE mayor
      --{{{  case derecha T, arriba F, mayor
      TRUE
        --{{{  case derecha T, arriba F, mayor T
        SEQ
          --{{{  Atualiza Radio: derecha T, arriba F, mayor T.
          r.coord := r.coord + r.delta.coord
          IF
            r.coord > r.coord.max
              SEQ
                r.coord := r.coord.max
                mayor   := FALSE
            TRUE
              SKIP
          xo.max  :=  x.maximo.coord - r.coord
          yo.min  :=  y.minimo.coord + r.coord
          --}}}
          --{{{  Atualiza yo: derecha T, arriba F, mayor T.
          yo.coord := yo.coord - delta.coord.y
          IF
            yo.coord < yo.min
              SEQ
                yo.coord := yo.min
                arriba   := TRUE
            TRUE
              SKIP
          --}}}
          --{{{  Atualiza xo: derecha T, arriba F, mayor T.
          xo.coord := xo.coord + delta.coord.x
          IF
            xo.coord > xo.max
              SEQ
                xo.coord := xo.max
                derecha  := FALSE
            TRUE
              SKIP
          --}}}
        --}}}
      FALSE
        --{{{  case derecha T, arriba F, mayor F
        SEQ
          --{{{  Atualiza Radio: derecha T, arriba F, mayor F.
          r.coord := r.coord - r.delta.coord
          IF
            r.coord < r.coord.min
              SEQ
                r.coord := r.coord.min
                mayor   := TRUE
            TRUE
              SKIP
          xo.max  :=  x.maximo.coord - r.coord
          yo.min  :=  y.minimo.coord + r.coord
          --}}}
          --{{{  Atualiza yo: derecha T, arriba F, mayor F.
          yo.coord := yo.coord - delta.coord.y
          IF
            yo.coord < yo.min
              SEQ
                yo.coord := yo.min
                arriba   := TRUE
            TRUE
              SKIP
          --}}}
          --{{{  Atualiza xo: derecha T, arriba F, mayor F.
          xo.coord := xo.coord + delta.coord.x
          IF
            xo.coord > xo.max
              SEQ
                xo.coord := xo.max
                derecha  := FALSE
            TRUE
              SKIP
          --}}}
        --}}}
      --}}}
    --}}}
```

```
y.maximo.coord        :=  REAL32 ROUND(y.maximo.coord.ent)
delta.coord.inic.x    :=  REAL32 ROUND(dt.coord.inic.x.ent)
delta.coord.inic.y    :=  REAL32 ROUND(dt.coord.inic.y.ent)
delta.coord.x         :=  REAL32 ROUND(20)
delta.coord.y         :=  REAL32 ROUND(20)
--{{{  xo/yo, max/min
xo.max                :=  x.maximo.coord - r.coord
yo.max                :=  y.maximo.coord - r.coord
xo.min                :=  x.minimo.coord + r.coord
yo.min                :=  y.minimo.coord + r.coord
--}}}
--}}}
--{{{  radio
r.coord               :=  REAL32 ROUND( r.coord.inic.ent)
r.coord.min           :=  REAL32 ROUND( r.coord.min.ent)
r.coord.max           :=  REAL32 ROUND( r.coord.max.ent)
r.delta.coord         :=  REAL32 ROUND(r.dt.ent)
--}}}
--{{{  color
color.cambia          := color.inic.byte
--}}}
--{{{  aleatoria
real.diez                       :=  REAL32 ROUND(entero.diez)
real.cien                       :=  REAL32 ROUND(entero.cien)
real.mil                        :=  REAL32 ROUND(entero.mil)
semilla.int                     :=  semilla
num.pseud.aleat.r, semilla.int  :=  RAN(semilla.int)
num.pseud.aleat.r, semilla.int  :=  RAN(semilla.int)
num.pseud.aleat                 :=  INT ROUND(num.pseud.aleat.r * real.mil)
num.pseud.aleat                 :=  num.pseud.aleat \ 10
--}}}
--}}}
--{{{  Variables Pseudo Aleatorias
--{{{  Genera Arriba
IF
  num.pseud.aleat  >= 5
    arriba := TRUE
  TRUE
    arriba := FALSE
--}}}
--{{{  Genera Numero Pseudoaleatorio de 0 a 9 ( con RAN )
num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
num.pseud.aleat                :=  num.pseud.aleat \ 10
--}}}
--{{{  Genera Derecha
IF
  num.pseud.aleat  >= 5
    derecha := TRUE
  TRUE
    derecha := FALSE
--}}}
--{{{  Genera Numero Pseudoaleatorio de 0 a 9 ( con RAN )
num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
num.pseud.aleat                :=  num.pseud.aleat \ 10
--}}}
--{{{  Genera Mayor
IF
  num.pseud.aleat  >= 5
    mayor := TRUE
  TRUE
    mayor := FALSE
--}}}
--{{{  Genera Numero Pseudoaleatorio de 0 a 9 ( con RAN )
num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
num.pseud.aleat                :=  num.pseud.aleat \ 10
--}}}
--{{{  Genera delta.x
delta.coord.x       := REAL32 ROUND( num.pseud.aleat )
delta.coord.x       := delta.coord.inic.x -  delta.coord.x
--}}}
--{{{  Genera Numero Pseudoaleatorio de 0 a 9 ( con RAN )
num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
num.pseud.aleat                :=  num.pseud.aleat \ 10
--}}}
--{{{  Genera delta.y
delta.coord.y         := REAL32 ROUND( num.pseud.aleat )
delta.coord.y         := delta.coord.inic.y -  delta.coord.y
--}}}
--}}}
--}}}
IniciaPantalla ( x.maximo.coord, y.maximo.coord )
BorraPantalla ()
--{{{  Cambia Color
color.cambia.ent := INT(color.cambia)
color.cambia.ent := color.cambia.ent + delta.color.ent
color.cambia := BYTE(color.cambia.ent)
Color ( color.cambia )
--}}}
Circulo ( xo.coord, yo.coord, r.coord )
WHILE activo
  PRI ALT
    Org.Mp ? parada.ayu
      activo := FALSE
    TRUE & SKIP
      --{{{  Procesos
      SEQ
        --{{{  Procesos
        CASE derecha
          --{{{  case derecha
          TRUE
            --{{{  case derecha T
            CASE arriba
              --{{{  case derecha T, arriba
              TRUE
                --{{{  case derecha T, arriba T
                CASE mayor
                  --{{{  case derecha T, arriba T, mayor
                  TRUE
                    --{{{  case derecha T, arriba T, mayor T
                    SEQ
                      --{{{  Atualiza Radio: derecha T, arriba T, mayor T,
                      r.coord := r.coord + r.delta.coord
                      IF
                        r.coord > r.coord.max
                          SEQ
                            r.coord := r.coord.max
                            mayor   := FALSE
                        TRUE
                          SKIP
                      xo.max  :=  x.maximo.coord - r.coord
                      yo.max  :=  y.maximo.coord - r.coord
                      --}}}
                      --{{{  Atualiza yo: derecha T, arriba T, mayor T,
                      yo.coord := yo.coord + delta.coord.y
                      IF
                        yo.coord > yo.max
                          SEQ
                            yo.coord := yo.max
```

```
                    mayor   := TRUE
                  TRUE
                    SKIP
                xo.min  :=  x.minimo.coord + r.coord
                yo.min  :=  y.minimo.coord + r.coord
                --}}}
                --{{{  Atualiza yo: derecha F, arriba F, mayor F.
                yo.coord := yo.coord - delta.coord.y
                IF
                  yo.coord < yo.min
                    SEQ
                      yo.coord := yo.min
                      arriba   := TRUE
                  TRUE
                    SKIP
                --}}}
                --{{{  Atualiza xo: derecha F, arriba F, mayor F.
                xo.coord := xo.coord - delta.coord.x
                IF
                  xo.coord < xo.min
                    SEQ
                      xo.coord := xo.min
                      derecha  := TRUE
                  TRUE
                    SKIP
                --}}}
              --}}}
            --}}}
          --}}}
        --}}}
      --}}}
    --}}}
    --{{{  Cambia Color              MODIFICADA
    color.cambia.ent := INT(color.cambia)
    color.cambia.ent := color.cambia.ent + delta.color.ent
    IF
      color.cambia.ent = 256
        color.cambia.ent := 1
      TRUE
        SKIP

    color.cambia := BYTE(color.cambia.ent)
    Color ( color.cambia )
    --}}}
    --{{{  Circulo
    Circulo ( xo.coord, yo.coord, r.coord )
    --}}}
    Color ( BYTE (256-color.cambia.ent)  )
    Circunferencia ( xo.coord, yo.coord, r.coord )
    --}}}
  --}}}
:
--}}}
--
--{{{  Paleta Suave ( sin Contorno )
PROC DibujaEjemplo2Suave1 ( VAL INT32 semilla )
  --{{{  Procesos
  --{{{  Variables/constantes
  --{{{  Variables
  INT párada.ayu :
  INT32 semilla.int :
  INT color.ayu, num.pseud.aleat :
  INT color.cambia.ent, delta.color.ent :
  BYTE color.cambia, color.ayu1 :
  BYTE delta.color.byte, color.inic.byte :
  REAL32 num.pseud.aleat.r, real.diez, real.cien, real.mil :
  REAL32 delta.coord.x, delta.coord.y :
  REAL32 delta.coord.inic.x, delta.coord.inic.y, r.delta.coord :
  REAL32 r.coord, r.coord.max, r.coord.min :
  REAL32 xo.coord, yo.coord :
  REAL32 xo.max , yo.max :
  REAL32 xo.min , yo.min :
  REAL32 x.minimo.coord, y.minimo.coord :
  REAL32 x.maximo.coord, y.maximo.coord :
  BOOL arriba, derecha, mayor, activo :
  --}}}
  --{{{  Constantes
  --{{{  x/y
  VAL     xo.coord.inic.ent    IS      512 :
  VAL     yo.coord.inic.ent    IS      384 :
  VAL     x.minimo.coord.ent   IS        0 :
  VAL     x.maximo.coord.ent   IS     1023 :
  VAL     y.minimo.coord.ent   IS        0 :
  VAL     y.maximo.coord.ent   IS      767 :
  VAL    dt.coord.inic.x.ent   IS       15 :
  VAL    dt.coord.inic.y.ent   IS       15 :
  --}}}
  --{{{  radio
  VAL  r.coord.inic.ent     IS           80 :
  VAL  r.coord.max.ent      IS          120 :
  VAL  r.coord.min.ent      IS           40 :
  VAL  r.dt.ent             IS            1 :
  --}}}
  --{{{  color
  VAL  color.max            IS     BYTE(255) :
  VAL  color.nigro          IS       BYTE(0) :
  VAL  delta.color.ent      IS            1 :
  VAL  delta.color.byte     IS       BYTE(1) :
  VAL  color.inic.byte      IS       BYTE(0) :
  --}}}
  --{{{  aleatoria
  VAL   entero.diez     IS     10 :
  VAL   entero.cien     IS    100 :
  VAL   entero.mil      IS   1000 :
  --}}}
  --}}}
  --}}}
  SEQ
    --{{{  Asignacion de Variables
    --{{{  Variables
    --{{{  logicas
    activo                :=  TRUE
    --}}}
    --{{{  x/y
    xo.coord              :=  REAL32 ROUND(xo.coord.inic.ent)
    yo.coord              :=  REAL32 ROUND(yo.coord.inic.ent)
    x.minimo.coord        :=  REAL32 ROUND(x.minimo.coord.ent)
    x.maximo.coord        :=  REAL32 ROUND(x.maximo.coord.ent)
    y.minimo.coord        :=  REAL32 ROUND(y.minimo.coord.ent)
    y.maximo.coord        :=  REAL32 ROUND(y.maximo.coord.ent)
    delta.coord.inic.x    :=  REAL32 ROUND(dt.coord.inic.x.ent)
    delta.coord.inic.y    :=  REAL32 ROUND(dt.coord.inic.y.ent)
    delta.coord.x         :=  REAL32 ROUND(20)
    delta.coord.y         :=  REAL32 ROUND(20)
    --{{{  xo/yo, max/min
    xo.max                :=  x.maximo.coord - r.coord
    yo.max                :=  y.maximo.coord - r.coord
    xo.min                :=  x.minimo.coord + r.coord
    yo.min                :=  y.minimo.coord + r.coord
    --}}}
    --}}}
```

```
      --}}}
    FALSE
      --{{{  case derecha F
      CASE arriba
        --{{{  case derecha F, arriba
        TRUE
          --{{{  case derecha F, arriba T
          CASE mayor
            --{{{  case derecha F, arriba T, mayor
            TRUE
              --{{{  case derecha F, arriba T, mayor T
              SEQ
                --{{{  Atualiza Radio: derecha F, arriba T, mayor T.
                r.coord := r.coord + r.delta.coord
                IF
                  r.coord > r.coord.max
                    SEQ
                      r.coord := r.coord.max
                      mayor   := FALSE
                  TRUE
                    SKIP
                xo.min  :=  x.minimo.coord + r.coord
                yo.max  :=  y.maximo.coord - r.coord
                --}}}
                --{{{  Atualiza yo: derecha F, arriba T, mayor T.
                yo.coord := yo.coord + delta.coord.y
                IF
                  yo.coord > yo.max
                    SEQ
                      yo.coord := yo.max
                      arriba   := FALSE
                  TRUE
                    SKIP
                --}}}
                --{{{  Atualiza xo: derecha F, arriba T, mayor T.
                xo.coord := xo.coord - delta.coord.x
                IF
                  xo.coord < xo.min
                    SEQ
                      xo.coord := xo.min
                      derecha  := TRUE
                  TRUE
                    SKIP
                --}}}
              --}}}
            FALSE
              --{{{  case derecha F, arriba T, mayor F
              SEQ
                --{{{  Atualiza Radio: derecha F, arriba T, mayor F.
                r.coord := r.coord - r.delta.coord
                IF
                  r.coord < r.coord.min
                    SEQ
                      r.coord := r.coord.min
                      mayor   := TRUE
                  TRUE
                    SKIP
                xo.min  :=  x.minimo.coord + r.coord
                yo.max  :=  y.maximo.coord - r.coord
                --}}}
                --{{{  Atualiza yo: derecha F, arriba T, mayor F.
                yo.coord'= yo.coord + delta.coord.y
                IF
                  yo.coord > yo.max
                    SEQ
                      yo.coord := yo.max
                      arriba   := FALSE
                  TRUE
                    SKIP
                --}}}
                --{{{  Atualiza xo: derecha F, arriba T, mayor F
                xo.coord := xo.coord - delta.coord.x
                IF
                  xo.coord < xo.min
                    SEQ
                      xo.coord := xo.min
                      derecha  := TRUE
                  TRUE
                    SKIP
                --}}}
              --}}}
          --}}}
        FALSE
          --{{{  case derecha F, arriba F
          CASE mayor
            --{{{  case derecha F, arriba F, mayor
            TRUE
              --{{{  case derecha F, arriba F, mayor T
              SEQ
                --{{{  Atualiza Radio: derecha F, arriba F, mayor T.
                r.coord := r.coord + r.delta.coord
                IF
                  r.coord > r.coord.max
                    SEQ
                      r.coord := r.coord.max
                      mayor   := FALSE
                  TRUE
                    SKIP
                xo.min  :=  x.minimo.coord + r.coord
                yo.min  :=  y.minimo.coord + r.coord
                --}}}
                --{{{  Atualiza yo: derecha F, arriba F, mayor T.
                yo.coord := yo.coord - delta.coord.y
                IF
                  yo.coord < yo.min
                    SEQ
                      yo.coord := yo.min
                      arriba   := TRUE
                  TRUE
                    SKIP
                --}}}
                --{{{  Atualiza xo: derecha F, arriba F, mayor T
                xo.coord := xo.coord - delta.coord.x
                IF
                  xo.coord < xo.min
                    SEQ
                      xo.coord := xo.min
                      derecha  := TRUE
                  TRUE
                    SKIP
                --}}}
              --}}}
            FALSE
              --{{{  case derecha F, arriba F, mayor F
              SEQ
                --{{{  Atualiza Radio: derecha F, arriba F, mayor F
                r.coord := r.coord - r.delta.coord
                IF
                  r.coord < r.coord.min
                    SEQ
                      r.coord := r.coord.min
```

```
                SKIP
            --}}}
          --}}}
        FALSE
          --{{{  case derecha T, arriba T, mayor F
          SEQ
            --{{{  Atualiza Radio: derecha T, arriba T, mayor F.
            r.coord := r.coord - r.delta.coord
            IF
              r.coord < r.coord.min
                SEQ
                  r.coord := r.coord.min
                  mayor   := TRUE
              TRUE
                SKIP
            xo.max  :=  x.maximo.coord - r.coord
            yo.max  :=  y.maximo.coord - r.coord
            --}}}
            --{{{  Atualiza yo: derecha T, arriba T, mayor F.
            yo.coord := yo.coord + delta.coord.y
            IF
              yo.coord > yo.max
                SEQ
                  yo.coord := yo.max
                  arriba   := FALSE
              TRUE
                SKIP
            --}}}
            --{{{  Atualiza xo: derecha T, arriba T, mayor F.
            xo.coord := xo.coord + delta.coord.x
            IF
              xo.coord > xo.max
                SEQ
                  xo.coord := xo.max
                  derecha  := FALSE
              TRUE
                SKIP
            --}}}
          --}}}
      --}}}
    --}}}
  FALSE
    --{{{  case derecha T, arriba F
    CASE mayor
      --{{{  case derecha T, arriba F, mayor
      TRUE
        --{{{  case derecha T, arriba F, mayor T
        SEQ
          --{{{  Atualiza Radio: derecha T, arriba F, mayor T.
          r.coord := r.coord + r.delta.coord
          IF
            r.coord > r.coord.max
              SEQ
                r.coord := r.coord.max
                mayor   := FALSE
            TRUE
              SKIP
          xo.max  :=  x.maximo.coord - r.coord
          yo.min  :=  y.minimo.coord + r.coord
          --}}}
          --{{{  Atualiza yo: derecha T, arriba F, mayor T.
          yo.coord := yo.coord - delta.coord.y
          IF
            yo.coord < yo.min
              SEQ
                yo.coord := yo.min
                arriba   := TRUE
            TRUE
              SKIP
          --}}}
          --{{{  Atualiza xo: derecha T, arriba F, mayor T.
          xo.coord := xo.coord + delta.coord.x
          IF
            xo.coord > xo.max
              SEQ
                xo.coord := xo.max
                derecha  := FALSE
            TRUE
              SKIP
          --}}}
        --}}}
      --}}}
    --}}}
  --}}}
FALSE
  --{{{  case derecha F
  CASE arriba
    --{{{  case derecha F, arriba
    TRUE
      --{{{  case derecha F, arriba T
      CASE mayor
        --{{{  case derecha F, arriba T, mayor
        TRUE
          --{{{  case derecha F, arriba T, mayor T
          SEQ
            --{{{  Atualiza Radio: derecha F, arriba T, mayor T.
            r.coord := r.coord - r.delta.coord
            IF
              r.coord < r.coord.min
                SEQ
                  r.coord := r.coord.min
                  mayor   := TRUE
              TRUE
                SKIP
            xo.max  :=  x.maximo.coord - r.coord
            yo.max  :=  y.maximo.coord - r.coord
            --}}}
            --{{{  Atualiza yo: derecha F, arriba T, mayor T.
            yo.coord := yo.coord + delta.coord.y
            IF
              yo.coord > yo.max
                SEQ
                  yo.coord := yo.max
                  arriba   := TRUE
              TRUE
                SKIP
            --}}}
            --{{{  Atualiza xo: derecha F, arriba T, mayor T.
            xo.coord := xo.coord + delta.coord.x
            IF
              xo.coord > xo.max
                SEQ
                  xo.coord := xo.max
                  derecha  := FALSE
              TRUE
                SKIP
            --}}}
          --}}}
        --}}}
      FALSE
        --{{{  case derecha F, arriba F, mayor F
        SEQ
          --{{{  Atualiza Radio: derecha F, arriba F, mayor F.
          r.coord := r.coord - r.delta.coord
          IF
            r.coord < r.coord.min
              SEQ
                r.coord := r.coord.min
                mayor   := TRUE
            TRUE
              SKIP
          xo.max  :=  x.maximo.coord - r.coord
          yo.min  :=  y.minimo.coord + r.coord
          --}}}
          --{{{  Atualiza yo: derecha F, arriba F, mayor F.
          yo.coord := yo.coord - delta.coord.y
          IF
            yo.coord < yo.min
              SEQ
                yo.coord := yo.min
                arriba   := TRUE
            TRUE
              SKIP
          --}}}
          --{{{  Atualiza xo: derecha F, arriba F, mayor F.
          xo.coord := xo.coord + delta.coord.x
          IF
            xo.coord > xo.max
              SEQ
                xo.coord := xo.max
                derecha  := FALSE
            TRUE
              SKIP
          --}}}
        --}}}
      --}}}
    --}}}
  --}}}
```

```
      ()
--{{{  radio
r.coord                 :=  REAL32 ROUND( r.coord.inic.ent)
r.coord.min             :=  REAL32 ROUND( r.coord.min.ent)
r.coord.max             :=  REAL32 ROUND( r.coord.max.ent)
r.delta.coord           :=  REAL32 ROUND(r.dt.ent)
--}}}
--{{{  color
color.cambia            := color.inic.byte
--}}}
--{{{  aleatoria
real.diez                       :=  REAL32 ROUND(entero.diez)
real.cien                       :=  REAL32 ROUND(entero.cien)
real.mil                        :=  REAL32 ROUND(entero.mil)
semilla.int                     :=  semilla
num.pseud.aleat.r, semilla.int  :=  RAN(semilla.int)
num.pseud.aleat.r, semilla.int  :=  RAN(semilla.int)
num.pseud.aleat                 :=  INT ROUND(num.pseud.aleat.r * real.mil)
num.pseud.aleat                 :=  num.pseud.aleat \ 10
--}}}
--}}}
--{{{  Variables Pseudo Aleatorias
--{{{  Genera Arriba
IF
  num.pseud.aleat  >= 5
    arriba := TRUE
  TRUE
    arriba := FALSE
--}}}
--{{{  Genera Numero Pseudoaleatorio de 0 a 9 ( con RAN )
num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
num.pseud.aleat                :=  num.pseud.aleat \ 10
--}}}
--{{{  Genera Derecha
IF
  num.pseud.aleat  >= 5
    derecha := TRUE
  TRUE
    derecha := FALSE
--}}}
--{{{  Genera Numero Pseudoaleatorio de 0 a 9 ( con RAN )
num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
num.pseud.aleat                :=  num.pseud.aleat \ 10
--}}}
--{{{  Genera Mayor
IF
  num.pseud.aleat  >= 5
    mayor := TRUE
  TRUE
    mayor := FALSE
--}}}
--{{{  Genera Numero Pseudoaleatorio de 0 a 9 ( con RAN )
num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
num.pseud.aleat                :=  num.pseud.aleat \ 10
--}}}
--{{{  Genera delta.x
delta.coord.x       := REAL32 ROUND( num.pseud.aleat )
delta.coord.x       := delta.coord.inic.x -  delta.coord.x
--}}}
--{{{  Genera Numero Pseudoaleatorio de 0 a 9 ( con RAN )
num.pseud.aleat.r, semilla.int :=  RAN(semilla.int)
num.pseud.aleat                :=  INT ROUND(num.pseud.aleat.r * real.mil)
num.pseud.aleat                :=  num.pseud.aleat \ 10
--}}}
```

```
--{{{  Genera delta.y
delta.coord.y         := REAL32 ROUND( num.pseud.aleat )
delta.coord.y         := delta.coord.inic.y -  delta.coord.y
--}}}
--}}}
--}}}
IniciaPantalla ( x.maximo.coord, y.maximo.coord )
BorraPantalla ()
--{{{  Cambia Color
color.cambia.ent := INT(color.cambia)
color.cambia.ent := color.cambia.ent + delta.color.ent
color.cambia := BYTE(color.cambia.ent)
Color ( color.cambia )
--}}}
Circulo ( xo.coord, yo.coord, r.coord )
WHILE activo
  PRI ALT
    Org.Mp ? parada.ayu
      activo := FALSE
    TRUE & SKIP
      --{{{  Procesos
      SEQ
        --{{{  Procesos
        CASE derecha
          --{{{  case derecha
          TRUE
            --{{{  case derecha T
            CASE arriba
              --{{{  case derecha T, arriba
              TRUE
                --{{{  case derecha T, arriba T
                CASE mayor
                  --{{{  case derecha T, arriba T, mayor
                  TRUE
                    --{{{  case derecha T, arriba T, mayor T
                    SEQ
                      --{{{  Atualiza Radio: derecha T, arriba T, mayor T.
                      r.coord := r.coord + r.delta.coord
                      IF
                        r.coord > r.coord.max
                          SEQ
                            r.coord := r.coord.max
                            mayor   := FALSE
                        TRUE
                          SKIP
                      xo.max  :=  x.maximo.coord - r.coord
                      yo.max  :=  y.maximo.coord - r.coord
                      --}}}
                      --{{{  Atualiza yo: derecha T, arriba T, mayor T.
                      yo.coord := yo.coord + delta.coord.y
                      IF
                        yo.coord > yo.max
                          SEQ
                            yo.coord := yo.max
                            arriba   := FALSE
                        TRUE
                          SKIP
                      --}}}
                      --{{{  Atualiza xo: derecha T, arriba T, mayor T.
                      xo.coord := xo.coord + delta.coord.x
                      IF
                        xo.coord > xo.max
                          SEQ
                            xo.coord := xo.max
                            derecha  := FALSE
                        TRUE
```

```
                        arriba   := TRUE
                      TRUE
                        SKIP
                    --}}}
                    --{{{  Atualiza xo: derecha F, arriba F, mayor F.
                    xo.coord := xo.coord - delta.coord.x
                    IF
                      xo.coord < xo.min
                        SEQ
                          xo.coord := xo.min
                          derecha  := TRUE
                      TRUE
                        SKIP
                    --}}}
                  --}}}
                --}}}
              --}}}
            --}}}
          --}}}
        --}}}
      --{{{  Cambia Color              MODIFICADA
      color.cambia.ent := INT(color.cambia)
      color.cambia.ent := color.cambia.ent + delta.color.ent
      IF
        color.cambia.ent = 256
          color.cambia.ent := 1
        TRUE
          SKIP

      color.cambia := BYTE(color.cambia.ent)
      Color ( color.cambia )
      --}}}
      --{{{  Circulo
      Circulo ( xo.coord, yo.coord, r.coord )
      --}}}
      --}}}
    --}}}
  --}}}
:
--}}}
--
--{{{  Elegir Entre Paleta Patron o Suave
PROC DibujaEjemplo2 ( VAL INT32 semilla )
  SEQ
    --{{{  Proc
    IF
      paleta.ant = 1
        DibujaEjemplo2Suave ( semilla )
      paleta.ant = 3
        DibujaEjemplo2Suave ( semilla )
      TRUE
        DibujaEjemplo2Patron ( semilla )
    --}}}
:
--}}}
--
--}}}
--
--{{{  EJEMPLO 3 : Mandelbroot
PROC Mandelbroot ()
  --{{{  Declaraciones
  VAL   real.1          IS   REAL32 ROUND(1)    :
  VAL   real.2          IS   REAL32 ROUND(2)    :
  VAL   real.4          IS   REAL32 ROUND(4)    :
  VAL   real.767        IS   REAL32 ROUND(767)  :
  VAL   real.1023       IS   REAL32 ROUND(1023) :
  VAL   m.x             IS   real.2/real.1023   :
  VAL   m.y             IS   real.2/real.767    :
  REAL32 mod.z, z.real, z.imag, z.real.ayu, z.imag.ayu :
  REAL32 i.r, j.r, i.r.inic, j.r.inic :
  INT n.inter, parada.ayu, i, j :
  --}}}
  SEQ i=0 FOR 1023
    --{{{  Proc
    SEQ j=0 FOR 768
      SEQ
        --{{{  Proc
        --{{{  Asignaciones
        i.r         :=  REAL32 ROUND(i)
        j.r         :=  REAL32 ROUND(j)
        z.real      :=  (m.x * i.r) - real.1
        z.imag      :=  real.1 - (m.y * j.r)
        i.r.inic    :=  z.real
        j.r.inic    :=  z.imag
        z.real.ayu  :=  z.real
        z.imag.ayu  :=  z.imag
        n.inter     :=  0
        mod.z       :=  (( z.real.ayu * z.real.ayu ) + ( z.imag.ayu * z.imag.ay
        --}}}
        WHILE ( n.inter < 255 ) AND ( ( INT ROUND(mod.z) ) < 4 )
          SEQ
            --{{{  Proc
            n.inter     :=  n.inter + 1
            z.real      :=  (((z.real.ayu*z.real.ayu)-(z.imag.ayu*z.imag.ayu))+
            z.imag      :=  (((real.2*z.real.ayu)*z.imag.ayu) + j.r.inic)
            z.real.ayu  :=  z.real
            z.imag.ayu  :=  z.imag
            mod.z       :=  (( z.real.ayu * z.real.ayu ) + ( z.imag.ayu * z.ima
            --}}}
        Color ( BYTE( 255 - n.inter ) )
        pantalla[ j ][ i ] := color
        --}}}
    --}}}
:
--}}}
--{{{  COMMENT EJEMPLO 3 : Mandelbroot
--:::A COMMENT FOLD
--{{{  EJEMPLO 3 : Mandelbroot
PROC Mandelbroot ()
  --{{{  Declaraciones
  VAL   real.1          IS   REAL32 ROUND(1)    :
  VAL   real.2          IS   REAL32 ROUND(2)    :
  VAL   real.4          IS   REAL32 ROUND(4)    :
  VAL   real.767        IS   REAL32 ROUND(767)  :
  VAL   real.1023       IS   REAL32 ROUND(1023) :
  VAL   m.x             IS   real.2/real.1023   :
  VAL   m.y             IS   real.2/real.767    :
  REAL32 mod.z, z.real, z.imag, z.real.ayu, z.imag.ayu :
  REAL32 i.r, j.r, i.r.inic, j.r.inic :
  INT n.inter, parada.ayu, i, j :
  --}}}
  SEQ
    --{{{  Proc
    i := 0
    j := 0
    WHILE  i <= 1023
      SEQ
        j := 0
        WHILE j <= 767
          SEQ
            PRI ALT
              Org.Mp ? parada.ayu
```

```
            --{{{  Atualiza Radio: derecha F, arriba T, mayor T.
            r.coord := r.coord + r.delta.coord
            IF
              r.coord > r.coord.max
                SEQ
                  r.coord := r.coord.max
                  mayor   := FALSE
              TRUE
                SKIP
            xo.min  :=  x.minimo.coord + r.coord
            yo.max  :=  y.maximo.coord - r.coord
            --}}}
            --{{{  Atualiza yo: derecha F, arriba T, mayor T.
            yo.coord := yo.coord + delta.coord.y
            IF
              yo.coord > yo.max
                SEQ
                  yo.coord := yo.max
                  arriba   := FALSE
              TRUE
                SKIP
            --}}}
            --{{{  Atualiza xo: derecha F, arriba T, mayor T.
            xo.coord := xo.coord - delta.coord.x
            IF
              xo.coord < xo.min
                SEQ
                  xo.coord := xo.min
                  derecha  := TRUE
              TRUE
                SKIP
            --}}}
          --}}}
        FALSE
          --{{{  case derecha F, arriba T, mayor F
          SEQ
            --{{{  Atualiza Radio: derecha F, arriba T, mayor F.
            r.coord := r.coord - r.delta.coord
            IF
              r.coord < r.coord.min
                SEQ
                  r.coord := r.coord.min
                  mayor   := TRUE
              TRUE
                SKIP
            xo.min  :=  x.minimo.coord + r.coord
            yo.max  :=  y.maximo.coord - r.coord
            --}}}
            --{{{  Atualiza yo: derecha F, arriba T, mayor F.
            yo.coord := yo.coord + delta.coord.y
            IF
              yo.coord > yo.max
                SEQ
                  yo.coord := yo.max
                  arriba   := FALSE
              TRUE
                SKIP
            --}}}
            --{{{  Atualiza xo: derecha F, arriba T, mayor F.
            xo.coord := xo.coord - delta.coord.x
            IF
              xo.coord < xo.min
                SEQ
                  xo.coord := xo.min
                  derecha  := TRUE
              TRUE
                SKIP
            --}}}
          --}}}
        --}}}
      --}}}
    --}}}
  FALSE
    --{{{  case derecha F, arriba F
    CASE mayor
      --{{{  case derecha F, arriba F, mayor
      TRUE
        --{{{  case derecha F, arriba F, mayor T
        SEQ
          --{{{  Atualiza Radio: derecha F, arriba F, mayor T
          r.coord := r.coord + r.delta.coord
          IF
            r.coord > r.coord.max
              SEQ
                r.coord := r.coord.max
                mayor   := FALSE
            TRUE
              SKIP
          xo.min  :=  x.minimo.coord + r.coord
          yo.min  :=  y.minimo.coord + r.coord
          --}}}
          --{{{  Atualiza yo: derecha F, arriba F, mayor T.
          yo.coord := yo.coord - delta.coord.y
          IF
            yo.coord < yo.min
              SEQ
                yo.coord := yo.min
                arriba   := TRUE
            TRUE
              SKIP
          --}}}
          --{{{  Atualiza xo: derecha F, arriba F, mayor T.
          xo.coord := xo.coord - delta.coord.x
          IF
            xo.coord < xo.min
              SEQ
                xo.coord := xo.min
                derecha  := TRUE
            TRUE
              SKIP
          --}}}
        --}}}
      FALSE
        --{{{  case derecha F, arriba F, mayor F
        SEQ
          --{{{  Atualiza Radio: derecha F, arriba F, mayor F
          r.coord := r.coord - r.delta.coord
          IF
            r.coord < r.coord.min
              SEQ
                r.coord := r.coord.min
                mayor   := TRUE
            TRUE
              SKIP
          xo.min  :=  x.minimo.coord + r.coord
          yo.min  :=  y.minimo.coord + r.coord
          --}}}
          --{{{  Atualiza yo: derecha F, arriba F, mayor F.
          yo.coord := yo.coord - delta.coord.y
          IF
            yo.coord < yo.min
              SEQ
                yo.coord := yo.min
```

```
IF
  ( yo.ent-r.ent ) >= 0
    lim.inf.y.ent := yo.ent - r.ent
  TRUE
    lim.inf.y.ent := y.minimo.ent
IF
  ( yo.ent+r.ent ) <= y.maximo.ent
    lim.sup.y.ent := yo.ent + r.ent
  TRUE
    lim.sup.y.ent := y.maximo.ent
--}}}
IF
  ( lim.sup.x.ent <= x.maximo.ent ) AND ( lim.inf.x.ent >= x.minimo.ent )
    ( lim.sup.y.ent <= y.maximo.ent ) AND ( lim.inf.y.ent >= y.minimo.ant
    --{{{  Procesos
    SEQ
      --{{{  Asignaciones
      cuanto.r := r.ent + 1
      k        := 0
      k.ayu    := 0
      l        := 0
      l.ayu    := 0
      i        := xo.ent
      i.ayu    := 0
      x.quad   := 0
      --}}}
      SEQ
        --{{{  Procesos
        --{{{  Coge el Contorno
        SEQ j = ( yo.ent - r.ent ) FOR cuanto.r
          SEQ
            --{{{  Procesos
            j.ayu      := j - yo.ent
            y.quad     := j.ayu * j.ayu
            xq.mas.yq  := x.quad + y.quad
            WHILE xq.mas.yq  <= r.quad
              IF
                i.ayu <= ( - j.ayu )
                  SEQ
                    --{{{  Procesos
                    contorno.x.inf[k] := xo.ent + i.ayu
                    contorno.y.inf[k] := yo.ent + j.ayu
                    i                 := i + 1
                    k                 := k + 1
                    i.ayu             := i - xo.ent
                    x.quad            := i.ayu * i.ayu
                    xq.mas.yq         := x.quad + y.quad
                    --}}}
                TRUE
                  SEQ
                    --{{{  Procesos
                    contorno.x.sup[l] := xo.ent + i.ayu
                    contorno.y.sup[l] := yo.ent + j.ayu
                    i                 := i + 1
                    l                 := l + 1
                    i.ayu             := i - xo.ent
                    x.quad            := i.ayu * i.ayu
                    xq.mas.yq         := x.quad + y.quad
                    --}}}
            IF
              i > xo.ent
                SEQ
                  --{{{  Procesos
                  i       := i - 1
                  i.ayu   := i - xo.ent
                  x.quad  := i.ayu * i.ayu
                  --}}}
              TRUE
                SKIP
            --}}}
--}}}
--{{{  Dibuja el Oitavo de Circulo
IF
  --{{{  De 5 hasta 8
  oitavo = 5
    SEQ
      --{{{  Dibuja el Oitavo 5
      --{{{  Almacena Puntos de Segmento
      AlmacenaPuntosSegmentoNA ( contorno.x.sup[0], contorno.y.sup
                                 xo.ent, yo.ent,
                                 tamano, contorno.x.inf,
                                 contorno.y.inf )
      --}}}
      --{{{  Dibuja el Oitavo
      k.ayu := tamano - 1
      l.ayu := 0
      WHILE ( contorno.y.sup[l.ayu] < yo.ent ) AND ( k.ayu > 0 )
        SEQ
          --{{{  Procesos
          IF
            contorno:y.inf[k.ayu] <> contorno.y.inf[(k.ayu - 1 )]
              SEQ
                IF
                  contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.
                    SEQ
                      --{{{  Procesos
                      delta.l.x   := contorno.x.sup[l.ayu] - conto
                      delta.l.x.1 := contorno.x.sup[l.ayu] - xo.en
                      cuanto.x    := delta.l.x  + 1
                      x.arregla   := xo.ent - delta.l.x.1
                      IniciaVectorBytes ( x.arregla, contorno.y.in
                                          cuanto.x, color )
                      --{{{  COMMENT Proc
                      --:::A COMMENT FOLD
                      --{{{  Proc
                      IF
                        ( x.arregla + cuanto.x ) <= x.maximo.ent
                          IniciaVectorBytes ( x.arregla, contorno.
                                              cuanto.x, color )
                        TRUE
                          SEQ
                            --{{{  Proc
                            cuanto.x := ( x.maximo.ent - x.arregla
                            IniciaVectorBytes ( x.arregla, contorn
                                                cuanto.x, color )
                            --}}}
                      --}}}
                      --}}}
                      --}}}
                      k.ayu := k.ayu - 1
                      l.ayu := l.ayu + 1
                  TRUE
                    l.ayu := l.ayu + 1
            TRUE
              k.ayu := k.ayu - 1
          --}}}
      --}}}
      --{{{  Dibuja la Ultima Linea
      cuanto.x  := r.ent + 1
      IniciaVectorBytes ( (xo.ent-r.ent), yo.ent, cuanto.x, color
      --}}}
      --}}}
```

```
            --{{{  Condicion 2
            ( x.arregla < x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 2
                delta.ayu := x.minimo.ent - x.arregla
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, y.arregla, cua
                --}}}
            --{{{  Condicion 3
            ( x.arregla >= x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 3
                delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.arregla, y.arregla, cuanto
                --}}}
            --{{{  Condicion 4
            ( x.arregla < x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 4
                delta.ayu := x.minimo.ent - x.arregla
                cuanto.x  := cuanto.x - delta.ayu
                delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, y.arregla, cua
                --}}}
            TRUE
              SKIP
          --}}}
          --}}}
      TRUE
        SKIP
    --}}}
--{{{  Dibuja la Ultima Linea
delta.l.y   := - ( contorno.y[x.ayu] - yo.ent )
x.arregla   := xo.ent - r.ent
y.arregla   := yo.ent + delta.l.y
cuanto.x    := r.ent + 1
IF
  condicion = 1
    SEQ
      --{{{  Dibuja Linea
      IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
      --}}}
  condicion = 2
    SEQ
      --{{{  Dibuja Linea
      delta.ayu := x.minimo.ent - x.arregla
      cuanto.x  := cuanto.x - delta.ayu
      IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
      --}}}
  condicion = 3
    SEQ
      --{{{  Dibuja Linea
      delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
      cuanto.x  := cuanto.x - delta.ayu
      IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
      --}}}
  condicion = 4
    SEQ
      --{{{  Dibuja Linea
      delta.ayu := x.minimo.ent - x.arregla
      cuanto.x  := cuanto.x - delta.ayu
      delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
      cuanto.x  := cuanto.x - delta.ayu
      IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
      --}}}
  TRUE
    SKIP
--}}}
--}}}
:
--}}}
--
--}}}F
--
--{{{  Dibujar Oitavo de Circulos
--
--{{{F Oitavo de Circulos
--:::F diocir04.tsr
PROC OitavoDeCirculo ( VAL REAL32 xo, VAL REAL32 yo,
                       VAL REAL32 r , VAL INT oitavo )
  --{{{  Declaraciones
  [(ancho*alto)]BYTE vec.pant RETYPES pantalla :
  [1024]INT contorno.x.inf :
  [1024]INT contorno.x.sup :
  [768]INT contorno.y.inf :
  [768]INT contorno.y.sup :
  INT i, i.ayu, j, j.ayu, k, k.ayu, l, l.ayu :
  INT xo.ent, yo.ent, delta.l.x, delta.l.x.i, delta.l.y :
  INT lim.inf.x.ent, lim.inf.y.ent, lim.sup.x.ent, lim.sup.y.ent :
  INT x.quad, y.quad, xq.mas.yq :
  INT r.ent, r.quad, cuanto.r :
  INT cuanto.x, tamano :
  INT x.arregla, y.arregla :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Procesos
    xo.ent      := INT ROUND( xo * escala )
    yo.ent      := INT ROUND( yo * escala )
    r.ent       := INT ROUND(  r * escala )
    r.quad := r.ent * r.ent
    --}}}
    --{{{  Arregla Limites de x
    IF
      ( xo.ent-r.ent ) >= 0
        lim.inf.x.ent := xo.ent - r.ent
      TRUE
        lim.inf.x.ent := x.minimo.ent
    IF
      ( xo.ent+r.ent ) <= x.maximo.ent
        lim.sup.x.ent := xo.ent + r.ent
      TRUE
        lim.sup.x.ent := x.maximo.ent
    --}}}
    --{{{  Arregla Limites de y
```

```
        --{{{  Dibuja el Oitavo 1
        --{{{  Almacena Puntos de Segmento
        AlmacenaPuntosSegmentoNA ( contorno.x.sup[0], contorno.y.sup
                                   xo.ent, yo.ent,
                                   tamano, contorno.x.inf,
                                   contorno.y.inf )
        --}}}
        --{{{  Dibuja el Oitavo
        k.ayu := tamano - 1
        l.ayu := 0
        WHILE ( contorno.y.sup[l.ayu] < yo.ent ) AND ( k.ayu > 0 )
          SEQ
            --{{{  Procesos
            IF
              contorno.y.inf[k.ayu] <> contorno.y.inf[(k.ayu - 1 )]
                SEQ
                  IF
                    contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.
                      SEQ
                        --{{{  Procesos
                        delta.l.x   := contorno.x.sup[l.ayu] - conto
                        delta.l.y   := - ( contorno.y.sup[l.ayu] - y
                        cuanto.x    := delta.l.x  + 1
                        y.arregla   := yo.ent + delta.l.y
                        IniciaVectorBytes ( contorno.x.inf[k.ayu], y
                                            cuantojx, color )
                        --}}}
                        k.ayu := k.ayu - 1
                        l.ayu := l.ayu + 1
                    TRUE
                      l.ayu := l.ayu + 1
              TRUE
                k.ayu := k.ayu - 1
            --}}}
        --}}}
        --{{{  Dibuja la Ultima Linea
        cuanto.x  := r.ent + 1
        IniciaVectorBytes ( xo.ent, yo.ent, cuanto.x, color )
        --}}}
        --}}}
      oitavo = 2
        SEQ
          --{{{  Dibuja el Oitavo 2
          AlmacenaPuntosSegmentoNA ( xo.ent, yo.ent,
                                     contorno.x.inf[(k-1)], contorno.y
                                     tamano, contorno.x.sup,
                                     contorno.y.sup )
          --{{{  Dibuja parte circular
          SEQ k.ayu = 0 FOR k
            SEQ
              --{{{  Procesos
              IF
                contorno.y.inf[(k.ayu + 1)] <> contorno.y.inf[k.ayu]
                  SEQ
                    --{{{  Procesos
                    delta.l.x   := contorno.x.inf[k.ayu] - xo.ent
                    delta.l.y   := - ( contorno.y.inf[k.ayu] - yo.ent
                    y.arregla   := yo.ent + delta.l.y
                    cuanto.x    := delta.l.x + 1
                    IniciaVectorBytes ( xo.ent, y.arregla,
                                        cuanto.x, color )
                    --}}}
                TRUE
                  SKIP
              --}}}
          --}}}
          --{{{  Dibuja parte triangular
          SEQ l.ayu = 0 FOR tamano
            SEQ
              --{{{  Procesos
              IF
                ( contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu]
                  ( contorno.y.sup[(l.ayu + 1)] <= yo.ent )
                  SEQ
                    --{{{  Procesos
                    delta.l.x   := contorno.x.sup[l.ayu] - xo.ent
                    delta.l.y   := - ( contorno.y.sup[l.ayu] - yo.ent
                    y.arregla   := yo.ent + delta.l.y
                    cuanto.x    := delta.l.x + 1
                    IniciaVectorBytes ( xo.ent, y.arregla,
                                        cuanto.x, color )
                    --}}}
                TRUE
                  SKIP
              --}}}
          --}}}
          --}}}
      oitavo = 3
        SEQ
          --{{{  Dibuja el Oitavo 3
          AlmacenaPuntosSegmentoNA ( xo.ent, yo.ent,
                                     contorno.x.inf[(k-1)], contorno.y
                                     tamano, contorno.x.sup,
                                     contorno.y.sup )
          --{{{  Dibuja parte circular
          SEQ k.ayu = 0 FOR k
            SEQ
              --{{{  Procesos
              IF
                contorno.y.inf[(k.ayu + 1)] <> contorno.y.inf[k.ayu]
                  SEQ
                    --{{{  Procesos
                    delta.l.x   := contorno.x.inf[k.ayu] - xo.ent
                    delta.l.y   := - ( contorno.y.inf[k.ayu] - yo.ent
                    x.arregla   := xo.ent - delta.l.x
                    y.arregla   := yo.ent + delta.l.y
                    cuanto.x    := delta.l.x + 1
                    IniciaVectorBytes ( x.arregla, y.arregla,
                                        cuanto.x, color )
                    --}}}
                TRUE
                  SKIP
              --}}}
          --}}}
          --{{{  Dibuja parte triangular
          SEQ l.ayu = 0 FOR tamano
            SEQ
              --{{{  Procesos
              IF
                ( contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu]
                  ( contorno.y.sup[(l.ayu + 1)] <= yo.ent )
                  SEQ
                    --{{{  Procesos
                    delta.l.x   := contorno.x.sup[l.ayu] - xo.ent
                    delta.l.y   := - ( contorno.y.sup[l.ayu] - yo.ent
                    x.arregla   := xo.ent - delta.l.x
                    y.arregla   := yo.ent + delta.l.y
                    cuanto.x    := delta.l.x + 1
                    IniciaVectorBytes ( x.arregla, y.arregla,
                                        cuanto.x, color )
                    --}}}
                TRUE
```

```
    oitavo = 6
      SEQ
        --{{{  Dibuja el Oitavo 6
        AlmacenaPuntosSegmentoNA ( xo.ent, yo.ent,
                                   contorno.x.inf[(k-1)], contorno.y
                                   tamano, contorno.x.sup,
                                   contorno.y.sup )
        --{{{  Dibuja parte circular
        SEQ k.ayu = 0 FOR k
          SEQ
            --{{{  Procesos
            IF
              contorno.y.inf[(k.ayu + 1)] <> contorno.y.inf[k.ayu]
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x.inf[k.ayu] - xo.ent
                  x.arregla   := xo.ent - delta.l.x
                  cuanto.x    := delta.l.x + 1
                  IniciaVectorBytes ( x.arregla, contorno.y.inf[k.ay
                                      cuanto.x, color )
                  --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        --{{{  Dibuja parte triangular
        SEQ l.ayu = 0 FOR tamano
          SEQ
            --{{{  Procesos
            IF
              ( contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu]
                ( contorno.y.sup[(l.ayu + 1)] <= yo.ent )
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x.sup[l.ayu] - xo.ent
                  x.arregla   := xo.ent - delta.l.x
                  cuanto.x    := delta.l.x + 1
                  IniciaVectorBytes ( x.arregla, contorno.y.sup[l.ay
                                      cuanto.x, color )
                  --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        --}}}
    oitavo = 7
      SEQ
        --{{{  Dibuja el Oitavo 7
        AlmacenaPuntosSegmentoNA ( xo.ent, yo.ent,
                                   contorno.x.inf[(k-1)], contorno.y
                                   tamano, contorno.x.sup,
                                   contorno.y.sup )
        --{{{  Dibuja parte circular
        SEQ k.ayu = 0 FOR k
          SEQ
            --{{{  Procesos
            IF
              contorno.y.inf[(k.ayu + 1)] <> contorno.y.inf[k.ayu]
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x.inf[k.ayu] - xo.ent
                  cuanto.x    := delta.l.x + 1
                  IniciaVectorBytes ( xo.ent, contorno.y.inf[k.ayu],
                                      cuanto.x, color )
                  --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        --{{{  Dibuja parte triangular
        SEQ l.ayu = 0 FOR tamano
          SEQ
            --{{{  Procesos
            IF
              ( contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu]
                ( contorno.y.sup[(l.ayu + 1)] <= yo.ent )
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x.sup[l.ayu] - xo.ent
                  cuanto.x    := delta.l.x + 1
                  IniciaVectorBytes ( xo.ent, contorno.y.sup[l.ayu],
                                      cuanto.x, color )
                  --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        --}}}
    oitavo = 8
      SEQ
        --{{{  Dibuja el Oitavo 8
        --{{{  Almacena Puntos de Segmento
        AlmacenaPuntosSegmentoNA ( contorno.x.sup[0], contorno.y.sup
                                   xo.ent, yo.ent,
                                   tamano, contorno.x.inf,
                                   contorno.y.inf )
        --}}}
        --{{{  Dibuja el Oitavo
        k.ayu := tamano - 1
        l.ayu := 0
        WHILE ( contorno.y.sup[l.ayu] < yo.ent ) AND ( k.ayu > 0 )
          SEQ
            --{{{  Procesos
            IF
              contorno.y.inf[k.ayu] <> contorno.y.inf[(k.ayu - 1 )]
                SEQ
                  IF
                    contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.
                      SEQ
                        --{{{  Procesos
                        delta.l.x   := contorno.x.sup[l.ayu] - conto
                        cuanto.x    := delta.l.x  + 1
                        IniciaVectorBytes ( contorno.x.inf[k.ayu], c
                                            cuanto.x, color )
                        --}}}
                        k.ayu := k.ayu - 1
                        l.ayu := l.ayu + 1
                    TRUE
                      l.ayu := l.ayu + 1
                TRUE
                  k.ayu := k.ayu - 1
            --}}}
        --}}}
        --{{{  Dibuja la Ultima Linea
        cuanto.x  := r.ent + 1
        IniciaVectorBytes ( xo.ent, yo.ent, cuanto.x, color )
        --}}}
        --}}}
    --}}}
    --{{{  De 1 hasta 4
    oitavo = 1
      SEQ
```

```
      IF
        i > xo.ent
          SEQ
            --{{{  Procesos
            i       := i - 1
            i.ayu   := i - xo.ent
            x.quad  := i.ayu * i.ayu
            --}}}
        TRUE
          SKIP
      --}}}
  --}}}
  --{{{  Dibuja el Oitavo de Circulo
  IF
    --{{{  De 5 hasta 8
    oitavo = 5
      SEQ
        --{{{  Dibuja el Oitavo 5
        --{{{  Almacena Puntos de Segmento
        AlmacenaPuntosSegmentoNA ( contorno.x.sup[0], contorno.y.sup[0],
                                   xo.ent, yo.ent,
                                   tamano, contorno.x.inf,
                                   contorno.y.inf )
        --}}}
        --{{{  Dibuja el Oitavo
        k.ayu := tamano - 1
        l.ayu := 0
        WHILE ( contorno.y.sup[l.ayu] < yo.ent ) AND ( k.ayu > 0 )
          SEQ
            --{{{  Procesos
            IF
              contorno.y.inf[k.ayu] <> contorno.y.inf[(k.ayu - 1 )]
                SEQ
                  IF
                    contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu]
                      SEQ
                        --{{{  Procesos
                        delta.l.x   := contorno.x.sup[l.ayu] - contorno.
                        delta.l.x.1 := contorno.x.sup[l.ayu] - xo.ent
                        cuanto.x    := delta.l.x  + 1
                        x.arregla   := xo.ent - delta.l.x.1
                        IniciaVectorBytes ( x.arregla, contorno.y.inf[k.
                                            cuanto.x, color )
                        --{{{  COMMENT Proc
                        --:::A COMMENT FOLD
                        --{{{  Proc
                        IF
                          ( x.arregla + cuanto.x ) <= x.maximo.ent
                            IniciaVectorBytes ( x.arregla, contorno.y.in
                                                cuanto.x, color )
                          TRUE
                            SEQ
                              --{{{  Proc
                              cuanto.x := ( x.maximo.ent - x.arregla ) +
                              IniciaVectorBytes ( x.arregla, contorno.y.
                                                  cuanto.x, color )
                              --}}}
                        --}}}
                        --}}}
                        --}}}
                        k.ayu := k.ayu - 1
                        l.ayu := l.ayu + 1
                    TRUE
                      l.ayu := l.ayu + 1
              TRUE
                k.ayu := k.ayu - 1
            --}}}
        --}}}
        --{{{  Dibuja la Ultima Linea
        cuanto.x  := r.ent + 1
        IniciaVectorBytes ( (xo.ent-r.ent), yo.ent, cuanto.x, color )
        --}}}
        --}}}
    oitavo = 6
      SEQ
        --{{{  Dibuja el Oitavo 6
        AlmacenaPuntosSegmentoNA ( xo.ent, yo.ent,
                                   contorno.x.inf[(l-1)], contorno.y.inf
                                   tamano, contorno.x.sup,
                                   contorno.y.sup )
        --{{{  Dibuja parte circular
        SEQ l.ayu = 0 FOR 1
          SEQ
            --{{{  Procesos
            IF
              contorno.y.inf[(l.ayu + 1)] <> contorno.y.inf[l.ayu]
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x.inf[l.ayu] - xo.ent
                  x.arregla   := xo.ent - delta.l.x
                  cuanto.x    := delta.l.x + 1
                  --{{{  Prueba se Esta en la Ventana Definida
                  IF
                    --{{{  Condicion 1
                    ( x.arregla >= x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 1
                        IniciaVectorBytes ( x.arregla, contorno.y.inf[].
                                            cuanto.x, color )
                        --}}}
                    --{{{  Condicion 2
                    ( x.arregla < x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 2
                        delta.ayu := x.minimo.ent - x.arregla
                        cuanto.x  := cuanto.x - delta.ayu
                        IniciaVectorBytes ( x.minimo.ent, contorno.y.int
                                            cuanto.x, color )
                        --}}}
                    --{{{  Condicion 3
                    ( x.arregla >= x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 3
                        delta.ayu := ( ( x.arregla + cuanto.x ) - 1 )
                        cuanto.x  := cuanto.x - delta.ayu
                        IniciaVectorBytes ( x.arregla, contorno.y.inf[].
                                            cuanto.x, color )
                        --}}}
                    --{{{  Condicion 4
                    ( x.arregla < x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                    --}}}
                      SEQ
```

```
                    SKIP
                --}}}
              --}}}
              --}}}
            octavo = 4
              SEQ
                --{{{  Dibuja el Octavo 4
                --{{{  Almacena Puntos de Segmento
                AlmacenaPuntosSegmentoNA ( contorno.x.sup[0], contorno.y.sup
                                           xo.ent, yo.ent,
                                           tamano, contorno.x.inf,
                                           contorno.y.inf )
                --}}}
                --{{{  Dibuja el Octavo
                k.ayu := tamano - 1
                l.ayu := 0
                WHILE ( contorno.y.sup[l.ayu] < yo.ent ) AND ( k.ayu > 0 )
                  SEQ
                    --{{{  Procesos
                    IF
                      contorno.y.inf[k.ayu] <> contorno.y.inf[(k.ayu - 1 )]
                        SEQ
                          IF
                            contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.
                              SEQ
                                --{{{  Procesos
                                delta.l.x   := contorno.x.sup[l.ayu] - conto
                                delta.l.x.1 := contorno.x.sup[l.ayu] - xo.en
                                delta.l.y   := - ( contorno.y.sup[l.ayu] - y
                                cuanto.x    := delta.l.x  + 1
                                x.arregla   := xo.ent - delta.l.x.1
                                y.arregla   := yo.ent + delta.l.y
                                IniciaVectorBytes ( x.arregla, y.arregla,
                                                    cuanto.x, color )
                                --}}}
                                k.ayu := k.ayu - 1
                                l.ayu := l.ayu + 1
                            TRUE
                              l.ayu := l.ayu + 1
                      TRUE
                        k.ayu := k.ayu - 1
                    --}}}
                --}}}
                --{{{  Dibuja la Ultima Linea
                cuanto.x  := r.ent + 1
                IniciaVectorBytes ( (xo.ent-r.ent), yo.ent, cuanto.x, color
                --}}}
                --}}}
            --}}}
            TRUE
              SKIP
          --}}}
          --}}}
      --}}}
    TRUE
      SKIP
  --}}}
:
--}}}F
--
--{{{F Octavo de Circulos con Verificacion de Limites
--:::F diocir05.tsr
PROC OctavoDeCirculo1 ( VAL REAL32 xo, VAL REAL32 yo,
                        VAL REAL32 r , VAL INT octavo )
  --{{{  Declaraciones
  [(ancho*alto)]BYTE vec.pant RETYPES pantalla :
  [1024]INT contorno.x.inf :
  [1024]INT contorno.x.sup :
  [768]INT contorno.y.inf :
  [768]INT contorno.y.sup :
  INT i, i.ayu, j, j.ayu, k, k.ayu, l, l.ayu :
  INT xo.ent, yo.ent, delta.l.x, delta.l.x.1, delta.l.y :
  INT lim.inf.x.ent, lim.inf.y.ent, lim.sup.x.ent, lim.sup.y.ent :
  INT x.quad, y.quad, xq.mas.yq :
  INT r.ent, r.quad, cuanto.r :
  INT x.arregla, y.arregla :
  INT cuanto.x, delta.ayu, cuanto.ayu, tamano :
  INT condicion :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Asignaciones Iniciales
    xo.ent := INT ROUND( xo * escala )
    yo.ent := INT ROUND( yo * escala )
    r.ent  := INT ROUND(  r * escala )
    r.quad := r.ent * r.ent
    --}}}
    --{{{  Procesos
    SEQ
      --{{{  Asignaciones
      cuanto.r := r.ent + 1
      k        := 0
      k.ayu    := 0
      l        := 0
      l.ayu    := 0
      i        := xo.ent
      i.ayu    := 0
      x.quad   := 0
      --}}}
      SEQ
        --{{{  Procesos
        --{{{  Coge el Contorno
        SEQ j = ( yo.ent - r.ent ) FOR cuanto.r
          SEQ
            --{{{  Procesos
            j.ayu      := j - yo.ent
            y.quad     := j.ayu * j.ayu
            xq.mas.yq  := x.quad + y.quad
            WHILE xq.mas.yq  <= r.quad
              IF
                i.ayu <= ( - j.ayu )
                  SEQ
                    --{{{  Procesos
                    contorno.x.inf[k] := xo.ent + i.ayu
                    contorno.y.inf[k] := yo.ent + j.ayu
                    i                 := i + 1
                    k                 := k + 1
                    i.ayu             := i - xo.ent
                    x.quad            := i.ayu * i.ayu
                    xq.mas.yq         := x.quad + y.quad
                    --}}}
                TRUE
                  SEQ
                    --{{{  Procesos
                    contorno.x.sup[l] := xo.ent + i.ayu
                    contorno.y.sup[l] := yo.ent + j.ayu
                    i                 := i + 1
                    l                 := l + 1
                    i.ayu             := i - xo.ent
                    x.quad            := i.ayu * i.ayu
                    xq.mas.yq         := x.quad + y.quad
                    --}}}
```

```
                ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 3
                    delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( xo.ent, contorno.y.inf[].ayu
                                        cuanto.x, color )
                    --}}}
                --{{{  Condicion 4
                ( x.arregla < x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 4
                    delta.ayu := x.minimo.ent - x.arregla
                    cuanto.x  := cuanto.x - delta.ayu
                    delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.minimo.ent, contorno.y.inf
                                        cuanto.x, color )
                    --}}}
                TRUE
                  SKIP
              --}}}
              --}}}
          TRUE
            SKIP
        --}}}
--}}}
--{{{  Dibuja parte triangular
SEQ k.ayu = 0 FOR tamano
  SEQ
    --{{{  Procesos
    IF
      ( contorno.y.sup[(k.ayu + 1)] <> contorno.y.sup[k.ayu] ) A
        ( contorno.y.sup[(k.ayu + 1)] <= yo.ent )
        SEQ
          --{{{  Procesos
          delta.l.x    := contorno.x.sup[k.ayu] - xo.ent
          cuanto.x     := delta.l.x + 1
          x.arregla    := xo.ent
          --{{{  Prueba se Esta en la Ventana Definida
          IF
            --{{{  Condicion 1
            ( x.arregla >= x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 1
                IniciaVectorBytes ( x.arregla, contorno.y.sup[k.
                                    cuanto.x, color )
                --}}}
            --{{{  Condicion 2
            ( x.arregla < x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 2
                delta.ayu := x.minimo.ent - x.arregla
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, contorno.y.sup
                                    cuanto.x, color )
                --}}}
            --{{{  Condicion 3
            ( x.arregla >= x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 3
                delta.ayu  := ( ( x.arregla + cuanto.x ) - 1 ) -
                cuanto.ayu := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.arregla, contorno.y.sup[k.
                                    cuanto.ayu, color )
                --}}}
            --{{{  Condicion 4
            ( x.arregla < x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 4
                delta.ayu := x.minimo.ent - x.arregla
                cuanto.x  := cuanto.x - delta.ayu
                delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, contorno.y.sup
                                    cuanto.x, color )
                --}}}
            TRUE
              SKIP
          --}}}
          --}}}
      TRUE
        SKIP
    --}}}
--}}}
--{{{  Dibuja parte cuadrada
IF
  --{{{  Condicion 3
  ( x.arregla >= x.minimo.ent ) AND
  ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
  --}}}
    SEQ
      --{{{  Dibuja Linea
      condicion  := 3
      cuanto.x     := contorno.x.sup[(tamano-1)] - xo.ent
      delta.l.y    := ( contorno.y.sup[(tamano-1)] - contorno...
      SEQ i.ayu=contorno.y.inf[(1-1)] FOR delta.l.y
        IniciaVectorBytes ( xo.ent, i.ayu, cuanto.x, color )
      --}}}
  --{{{  Condicion 4
  ( x.arregla < x.minimo.ent ) AND
  ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
  --}}}
    SEQ
      --{{{  Dibuja Linea
      condicion    := 4
      cuanto.x     := x.maximo.ent - x.minimo.ent
      delta.l.y    := ( contorno.y.sup[0] - contorno.y.inf[k.ayu]
      SEQ i.ayu=contorno.y.inf[k.ayu] FOR delta.l.y
        IniciaVectorBytes ( x.minimo.ent, i.ayu, cuanto.x, color
      --}}}
  TRUE
    SKIP
--}}}
--}}}
```

```
                --{{{  Dibuja Linea
                condicion  := 4
                delta.ayu := x.minimo.ent - x.arregla
                cuanto.x  := cuanto.x - delta.ayu
                delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, contorno.y.inf
                                    cuanto.x, color )
                --}}}
              TRUE
                SKIP
            --}}}
            --}}}
        TRUE
          SKIP
      --}}}
--}}}
--{{{  Dibuja parte triangular
SEQ k.ayu = 0 FOR tamano
  SEQ
    --{{{  Procesos
    IF
      ( contorno.y.sup[(k.ayu + 1)] <> contorno.y.sup[k.ayu] ) A
        ( contorno.y.sup[(k.ayu + 1)] <= yo.ent )
        SEQ
          --{{{  Procesos
          delta.l.x   := contorno.x.sup[k.ayu] - xo.ent
          x.arregla   := xo.ent - delta.l.x
          cuanto.x    := delta.l.x + 1
          --{{{  Prueba se Esta en la Ventana Definida
          IF
            --{{{  Condicion 1
            ( x.arregla >= x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 1
                IniciaVectorBytes ( x.arregla, contorno.y.sup[k.
                                    cuanto.x, color )
                --}}}
            --{{{  Condicion 2
            ( x.arregla < x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 2
                delta.ayu := x.minimo.ent - x.arregla
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, contorno.y.sup
                                    cuanto.x, color )
                --}}}
            --{{{  Condicion 3
            ( x.arregla >= x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 3
                delta.ayu  := ( ( x.arregla + cuanto.x ) - 1 ) -
                cuanto.ayu := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.arregla, contorno.y.sup[k.
                                    cuanto.ayu, color )
                --}}}
            --{{{  Condicion 4
            ( x.arregla < x.minimo.ent ) AND
            ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
            --}}}
              SEQ
                --{{{  Dibuja Linea
                condicion  := 4
                delta.ayu := x.minimo.ent - x.arregla
                cuanto.x  := cuanto.x - delta.ayu
                delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                cuanto.x  := cuanto.x - delta.ayu
                IniciaVectorBytes ( x.minimo.ent, contorno.y.sup
                                    cuanto.x, color )
                --}}}
            TRUE
              SKIP
          --}}}
          --}}}
      TRUE
        SKIP
    --}}}
  --}}}
  --}}}
oitavo = 7
  SEQ
    --{{{  Dibuja el Oitavo 7
    AlmacenaPuntosSegmentoNA ( xo.ent, yo.ent,
                               contorno.x.inf[(1-1)], contorno.y.inf
                               tamano, contorno.x.sup,
                               contorno.y.sup )
    --{{{  Dibuja parte circular
    SEQ l.ayu = 0 FOR l
      SEQ
        --{{{  Procesos
        IF
          contorno.y.inf[(l.ayu + 1)] <> contorno.y.inf[l.ayu]
            SEQ
              --{{{  Procesos
              delta.l.x   := contorno.x.inf[l.ayu] - xo.ent
              cuanto.x    := delta.l.x + 1
              x.arregla   := xo.ent
              --{{{  Prueba se Esta en la Ventana Definida
              IF
                --{{{  Condicion 1
                ( x.arregla >= x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 1
                    IniciaVectorBytes ( xo.ent, contorno.y.inf[l.ayu
                                        cuanto.x, color )
                    --}}}
                --{{{  Condicion 2
                ( x.arregla < x.minimo.ent ) AND
                ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                --}}}
                  SEQ
                    --{{{  Dibuja Linea
                    condicion  := 2
                    delta.ayu := x.minimo.ent - x.arregla
                    cuanto.x  := cuanto.x - delta.ayu
                    IniciaVectorBytes ( x.minimo.ent, contorno.y.inf
                                        cuanto.x, color )
                    --}}}
                --{{{  Condicion 3
                ( x.arregla >= x.minimo.ent ) AND
```

```
      --}}}
    --}}}
    --{{{  De 1 hasta 4
    oitavo = 1
      SEQ
        --{{{  Dibuja el Oitavo 1
        --{{{  Almacena Puntos de Segmento
        AlmacenaPuntosSegmentoNA ( contorno.x.sup[0], contorno.y.sup[0],
                                   xo.ent, yo.ent,
                                   tamano, contorno.x.inf,
                                   contorno.y.inf )
        --}}}
        --{{{  Dibuja el Oitavo
        k.ayu := tamano - 1
        l.ayu := 0
        WHILE ( contorno.y.sup[l.ayu] < yo.ent ) AND ( k.ayu > 0 )
          SEQ
            --{{{  Procesos
            IF
              contorno.y.inf[k.ayu] <> contorno.y.inf[(k.ayu - 1 )]
                SEQ
                  IF
                    contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu]
                      SEQ
                        --{{{  Procesos
                        delta.l.x   := contorno.x.sup[l.ayu] - contorno.
                        delta.l.y   := - ( contorno.y.sup[l.ayu] - yo.en
                        cuanto.x    := delta.l.x  + 1
                        y.arregla   := yo.ent + delta.l.y
                        IniciaVectorBytes ( contorno.x.inf[k.ayu], y.arr
                                            cuanto.x, color )
                        --}}}
                        k.ayu := k.ayu - 1
                        l.ayu := l.ayu + 1
                    TRUE
                      l.ayu := l.ayu + 1
              TRUE
                k.ayu := k.ayu - 1
            --}}}
        --}}}
        --{{{  Dibuja la Ultima Linea
        cuanto.x  := r.ent + 1
        IniciaVectorBytes ( xo.ent, yo.ent, cuanto.x, color )
        --}}}
      --}}}
    oitavo = 2
      SEQ
        --{{{  Dibuja el Oitavo 2
        AlmacenaPuntosSegmentoNA ( xo.ent, yo.ent,
                                   contorno.x.inf[(k-1)], contorno.y.inf
                                   tamano, contorno.x.sup,
                                   contorno.y.sup )
        --{{{  Dibuja parte circular
        SEQ k.ayu = 0 FOR k
          SEQ
            --{{{  Procesos
            IF
              contorno.y.inf[(k.ayu + 1)] <> contorno.y.inf[k.ayu]
                SEQ
                  --{{{  Procesos
                  delta.l.x  := contorno.x.inf[k.ayu] - xo.ent
                  delta.l.y   := - ( contorno.y.inf[k.ayu] - yo.ent )
                  y.arregla   := yo.ent + delta.l.y
                  cuanto.x    := delta.l.x + 1
                  IniciaVectorBytes ( xo.ent, y.arregla,
                                      cuanto.x, color )
                  --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        --{{{  Dibuja parte triangular
        SEQ l.ayu = 0 FOR tamano
          SEQ
            --{{{  Procesos
            IF
              ( contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu] ) A
                ( contorno.y.sup[(l.ayu + 1)] <= yo.ent )
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x.sup[l.ayu] - xo.ent
                  delta.l.y   := - ( contorno.y.sup[l.ayu] - yo.ent )
                  y.arregla   := yo.ent + delta.l.y
                  cuanto.x    := delta.l.x + 1
                  IniciaVectorBytes ( xo.ent, y.arregla,
                                      cuanto.x, color )
                  --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        --}}}
    oitavo = 3
      SEQ
        --{{{  Dibuja el Oitavo 3
        AlmacenaPuntosSegmentoNA ( xo.ent, yo.ent,
                                   contorno.x.inf[(k-1)], contorno.y.inf
                                   tamano, contorno.x.sup,
                                   contorno.y.sup )
        --{{{  Dibuja parte circular
        SEQ k.ayu = 0 FOR k
          SEQ
            --{{{  Procesos
            IF
              contorno.y.inf[(k.ayu + 1)] <> contorno.y.inf[k.ayu]
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x.inf[k.ayu] - xo.ent
                  delta.l.y   := - ( contorno.y.inf[k.ayu] - yo.ent )
                  x.arregla   := xo.ent - delta.l.x
                  y.arregla   := yo.ent + delta.l.y
                  cuanto.x    := delta.l.x + 1
                  IniciaVectorBytes ( x.arregla, y.arregla,
                                      cuanto.x, color )
                  --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        --{{{  Dibuja parte triangular
        SEQ l.ayu = 0 FOR tamano
          SEQ
            --{{{  Procesos
            IF
              ( contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu] ) A
                ( contorno.y.sup[(l.ayu + 1)] <= yo.ent )
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x.sup[l.ayu] - xo.ent
                  delta.l.y   := - ( contorno.y.sup[l.ayu] - yo.ent )
                  x.arregla   := xo.ent - delta.l.x
                  y.arregla   := yo.ent + delta.l.y
```

```
octavo = 8
  SEQ
    --{{{  Dibuja el Octavo 8
    --{{{  Almacena Puntos de Segmento
    --AlmacenaPuntosSegmentoNA ( contorno.x.sup[0], contorno.y.sup[0
    i.ayu  := contorno.x.inf[(1-1)]
    k.ayu  := contorno.y.inf[(1-1)]
    AlmacenaPuntosSegmentoNA ( i.ayu, k.ayu,
                               xo.ent, yo.ent,
                               tamano, contorno.x.inf,
                               contorno.y.inf )
    --}}}
    --{{{  Dibuja el Octavo
    i.ayu := tamano - 1
    k.ayu := 0
    WHILE ( contorno.y.sup[k.ayu] <= yo.ent ) AND ( i.ayu >= 0 )
      SEQ
        --{{{  Procesos
        IF
          contorno.y.inf[i.ayu] <> contorno.y.inf[(i.ayu - 1 )]
            SEQ
              IF
                contorno.y.sup[(k.ayu + 1)] <> contorno.y.sup[k.ayu]
                  SEQ
                    --{{{  Procesos
                    delta.l.x   := contorno.x.sup[k.ayu] - contorno.
                    cuanto.x    := delta.l.x  + 1
                    x.arregla   := contorno.x.inf[i.ayu]
                    --{{{  Prueba se Esta en la Ventana Definida
                    IF
                      --{{{  Condicion 1
                      ( x.arregla >= x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 1
                          IniciaVectorBytes ( x.arregla, contorno.y.
                                              cuanto.x, color )
                          --}}}
                      --{{{  Condicion 2
                      ( x.arregla < x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 2
                          delta.ayu := x.minimo.ent - x.arregla
                          cuanto.x  := cuanto.x - delta.ayu
                          IniciaVectorBytes ( x.minimo.ent, contorno
                                              cuanto.x, color )
                          --}}}
                      --{{{  Condicion 3
                      ( x.arregla >= x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 3
                          delta.ayu := ( ( contorno.x.inf[i.ayu] + c
                          cuanto.x  := cuanto.x - delta.ayu
                          IF
                            x.arregla <= x.maximo.ent
                              IniciaVectorBytes ( x.arregla, contorn
                                                  cuanto.x, color )
                            TRUE
                              SKIP
                          --}}}
                      --{{{  Condicion 4
                      ( x.arregla < x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 4
                          delta.ayu := x.minimo.ent - x.arregla
                          cuanto.x  := cuanto.x - delta.ayu
                          delta.ayu := ( ( contorno.x.inf[i.ayu] + c
                          cuanto.x  := cuanto.x - delta.ayu
                          IniciaVectorBytes ( x.minimo.ent, contorno
                                              cuanto.x, color )
                          --}}}
                      TRUE
                        SKIP
                    --}}}
                    --}}}
                    i.ayu := i.ayu - 1
                    k.ayu := k.ayu + 1
                TRUE
                  k.ayu := k.ayu + 1
            TRUE
              i.ayu := i.ayu - 1
        --}}}
--}}}
--{{{  Dibuja la Ultima Linea
cuanto.x  := r.ent + 1
--{{{  Dibuja la Ultima Linea
x.arregla := xo.ent
cuanto.x  := r.ent + 1
IF
  condicion = 1
    SEQ
      --{{{  Dibuja Linea
      IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
      --}}}
  condicion = 2
    SEQ
      --{{{  Dibuja Linea
      delta.ayu := x.minimo.ent - x.arregla
      cuanto.x  := cuanto.x - delta.ayu
      IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
      --}}}
  condicion = 3
    SEQ
      --{{{  Dibuja Linea
      delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
      cuanto.x  := cuanto.x - delta.ayu
      IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
      --}}}
  condicion = 4
    SEQ
      --{{{  Dibuja Linea
      delta.ayu := x.minimo.ent - x.arregla
      cuanto.x  := cuanto.x - delta.ayu
      delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
      cuanto.x  := cuanto.x - delta.ayu
      IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
      --}}}
  TRUE
    SKIP
--}}}
--}}}
```

```
        x.arregla := xo.ent - r.ent
        cuanto.x  := r.ent + 1
        IF
          condicion = 1
            SEQ
              --{{{  Dibuja Linea
              IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
              --}}}
          condicion = 2
            SEQ
              --{{{  Dibuja Linea
              delta.ayu := x.minimo.ent - x.arregla
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
              --}}}
          condicion = 3
            SEQ
              --{{{  Dibuja Linea
              delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
              --}}}
          condicion = 4
            SEQ
              --{{{  Dibuja Linea
              delta.ayu := x.minimo.ent - x.arregla
              cuanto.x  := cuanto.x - delta.ayu
              delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
              --}}}
          TRUE
            SKIP
        --}}}
        --}}}
    cuadrante = 4
      SEQ
        --{{{  Dibuja el Cuadrante 4
        SEQ k.ayu = 0 FOR k
          SEQ
            --{{{  Procesos
            IF
              ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
                ( contorno.y[(k.ayu + 1)] <= yo.ent )
                SEQ
                  --{{{  Procesos
                  delta.l.x    := contorno.x[k.ayu] - xo.ent
                  cuanto.x     := delta.l.x + 1
                  x.arregla    := xo.ent
                  --{{{  Prueba se Esta en la Ventana Definida
                  IF
                    --{{{  Condicion 1
                    ( x.arregla >= x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 1
                        IniciaVectorBytes ( x.arregla, contorno.y[k.ayu]
                                            cuanto.x, color )
                        --}}},
                    --{{{  Condicion 2
                    ( x.arregla < x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 2
                        delta.ayu := x.minimo.ent - x.arregla
                        cuanto.x  := cuanto.x - delta.ayu
                        IniciaVectorBytes ( x.minimo.ent, contorno.y[k.a
                                            cuanto.x, color )
                        --}}}
                    --{{{  Condicion 3
                    ( x.arregla >= x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 3
                        delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                        cuanto.x  := cuanto.x - delta.ayu
                        IniciaVectorBytes ( x.arregla, contorno.y[k.ayu]
                                            cuanto.x, color )
                        --}}}
                    --{{{  Condicion 4
                    ( x.arregla < x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 4
                        delta.ayu := x.minimo.ent - x.arregla
                        cuanto.x  := cuanto.x - delta.ayu
                        delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                        cuanto.x  := cuanto.x - delta.ayu
                        IniciaVectorBytes ( x.minimo.ent, contorno.y[k.a
                                            cuanto.x, color )
                        --}}}
                    TRUE
                      SKIP
                  --}}}
                  --}}}
            TRUE
              SKIP
        --}}}
    --{{{  Dibuja la Ultima Linea
    x.arregla := xo.ent
    cuanto.x := r.ent + 1
    IF
      condicion = 1
        SEQ
          --{{{  Dibuja Linea
          IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
          --}}}
      condicion = 2
        SEQ
          --{{{  Dibuja Linea
          delta.ayu := x.minimo.ent - x.arregla
          cuanto.x  := cuanto.x - delta.ayu
          IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
          --}}}
      condicion = 3
        SEQ
          --{{{  Dibuja Linea
          delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
          cuanto.x  := cuanto.x - delta.ayu
          IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
          --}}}
      condicion = 4
        SEQ
          --{{{  Dibuja Linea
```

```
                cuanto.x    := delta.l.x + 1
                IniciaVectorBytes ( x.arregla, y.arregla,
                                    cuanto.x, color )
              --}}}
            TRUE
              SKIP
          --}}}
      --}}}
      --}}}
  octavo = 4
    SEQ
      --{{{  Dibuja el Octavo 4
      --{{{  Almacena Puntos de Segmento
      AlmacenaPuntosSegmentoNA ( contorno.x.sup[0], contorno.y.sup[0],
                                 xo.ent, yo.ent,
                                 tamano, contorno.x.inf,
                                 contorno.y.inf )
      --}}}
      --{{{  Dibuja el Octavo
      k.ayu := tamano - 1
      l.ayu := 0
      WHILE ( contorno.y.sup[l.ayu] < yo.ent ) AND ( k.ayu > 0 )
        SEQ
          --{{{  Procesos
          IF
            contorno.y.inf[k.ayu] <> contorno.y.inf[(k.ayu - 1 )]
              SEQ
                IF
                  contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu]
                    SEQ
                      --{{{  Procesos
                      delta.l.x   := contorno.x.sup[l.ayu] - contorno.
                      delta.l.x.1 := contorno.x.sup[l.ayu] - xo.ent
                      delta.l.y   := - ( contorno.y.sup[l.ayu] - yo.en
                      cuanto.x    := delta.l.x  + 1
                      x.arregla   := xo.ent - delta.l.x.1
                      y.arregla   := yo.ent + delta.l.y
                      IniciaVectorBytes ( x.arregla, y.arregla,
                                          cuanto.x, color )
                      --}}}
                      k.ayu := k.ayu - 1
                      l.ayu := l.ayu + 1
                  TRUE
                    l.ayu := l.ayu + 1
            TRUE
              k.ayu := k.ayu - 1
          --}}}
      --}}}
      --{{{  Dibuja la Ultima Linea
      cuanto.x  := r.ent + 1
      IniciaVectorBytes ( (xo.ent-r.ent), yo.ent, cuanto.x, color )
      --}}}
      --}}}
  --}}}
  TRUE
    SKIP
--{{{  COMMENT Dibuja el Cuarto de Circulo
--:::A COMMENT FOLD
--{{{  Dibuja el Cuarto de Circulo
IF
  cuadrante = 3
    SEQ
      --{{{  Dibuja el Cuadrante 3
      SEQ k.ayu = 0 FOR k
        SEQ
          --{{{  Procesos
          IF
            ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
              ( contorno.y[(k.ayu + 1)] <= yo.ent )
              SEQ
                --{{{  Procesos
                delta.l.x   := contorno.x[k.ayu] - xo.ent
                cuanto.x    := delta.l.x + 1
                x.arregla   := xo.ent - delta.l.x
                --{{{  Prueba se Esta en la Ventana Definida
                IF
                  --{{{  Condicion 1
                  ( x.arregla >= x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 1
                      IniciaVectorBytes ( x.arregla, contorno.y[k.ayu]
                                          cuanto.x, color )
                      --}}}
                  --{{{  Condicion 2
                  ( x.arregla < x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 2
                      delta.ayu := x.minimo.ent - x.arregla
                      cuanto.x  := cuanto.x - delta.ayu
                      IniciaVectorBytes ( x.minimo.ent, contorno.y[k.
                                          cuanto.x, color )
                      --}}}
                  --{{{  Condicion 3
                  ( x.arregla >= x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 3
                      delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                      cuanto.x  := cuanto.x - delta.ayu
                      IniciaVectorBytes ( x.arregla, contorno.y[k.ayu]
                                          cuanto.x, color )
                      --}}}
                  --{{{  Condicion 4
                  ( x.arregla < x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 4
                      delta.ayu := x.minimo.ent - x.arregla
                      cuanto.x  := cuanto.x - delta.ayu
                      delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                      cuanto.x  := cuanto.x - delta.ayu
                      IniciaVectorBytes ( x.minimo.ent, contorno.y[k.
                                          cuanto.x, color )
                      --}}}
                  TRUE
                    SKIP
                --}}}
                --}}}
            TRUE
              SKIP
          --}}}
      --{{{  Dibuja la Ultima Linea
```

```
                y.arregla   := yo.ent + delta.l.y
                --{{{  Prueba se Esta en la Ventana Definida
                IF
                  --{{{  Condicion 1
                  ( x.arregla >= x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 1
                      IniciaVectorBytes ( x.arregla, y.arregla, cuanto
                      --}}}
                  --{{{  Condicion 2
                  ( x.arregla < x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 2
                      delta.ayu := x.minimo.ent - x.arregla
                      cuanto.x  := cuanto.x - delta.ayu
                      IniciaVectorBytes ( x.minimo.ent, y.arregla, cua
                      --}}}
                  --{{{  Condicion 3
                  ( x.arregla >= x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 3
                      delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                      cuanto.x  := cuanto.x - delta.ayu
                      IniciaVectorBytes ( x.arregla, y.arregla, cuanto
                      --}}}
                  --{{{  Condicion 4
                  ( x.arregla < x.minimo.ent ) AND
                  ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                  --}}}
                    SEQ
                      --{{{  Dibuja Linea
                      condicion  := 4
                      delta.ayu := x.minimo.ent - x.arregla
                      cuanto.x  := cuanto.x - delta.ayu
                      delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                      cuanto.x  := cuanto.x - delta.ayu
                      IniciaVectorBytes ( x.minimo.ent, y.arregla, cua
                      --}}}
                  TRUE
                    SKIP
                --}}}
                --}}}
          TRUE
            SKIP
        --}}}
        --{{{  Dibuja la Ultima Linea
        delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
        x.arregla   := xo.ent - r.ent
        y.arregla   := yo.ent + delta.l.y
        cuanto.x    := r.ent + 1
        IF
          condicion = 1
            SEQ
              --{{{  Dibuja Linea
              IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
              --}}}
          condicion = 2
            SEQ
              --{{{  Dibuja Linea
              delta.ayu := x.minimo.ent - x.arregla
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
              --}}}
          condicion = 3
            SEQ
              --{{{  Dibuja Linea
              delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
              --}}}
          condicion = 4
            SEQ
              --{{{  Dibuja Linea
              delta.ayu := x.minimo.ent - x.arregla
              cuanto.x  := cuanto.x - delta.ayu
              delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
              --}}}
          TRUE
            SKIP
      --}}}
      --}}}
    TRUE
      SKIP
  --}}}
  --}}}
:
--}}}F
--{{{  COMMENT Oitavo de Circulos con Verificacion de Limites
--:::A COMMENT FOLD
--{{{F Oitavo de Circulos con Verificacion de Limites
--:::F diocir06.tsr
PROC OitavoDeCirculo1 ( VAL REAL32 xo, VAL REAL32 yo,
                        VAL REAL32 r , VAL INT oitavo )
  --{{{  Declaraciones
  [(ancho*alto)]BYTE vec.pant RETYPES pantalla :
  [1024]INT contorno.x.inf :
  [1024]INT contorno.x.sup :
  [768]INT contorno.y.inf :
  [768]INT contorno.y.sup :
  INT i, i.ayu, j, j.ayu, k, k.ayu, l, l.ayu :
  INT xo.ent, yo.ent, delta.l.x, delta.l.x.1, delta.l.y :
  INT lim.inf.x.ent, lim.inf.y.ent, lim.sup.x.ent, lim.sup.y.ent :
  INT x.quad, y.quad, xq.mas.yq :
  INT r.ent, r.quad, cuanto.r :
  INT x.arregla, y.arregla :
  INT cuanto.x, delta.ayu, cuanto.ayu, tamano :
  INT condicion :
  --}}}
  SEQ
    --{{{  Proc
    --{{{  Asignaciones Iniciales
    xo.ent := INT ROUND( xo * escala )
    yo.ent := INT ROUND( yo * escala )
    r.ent  := INT ROUND(  r * escala )
    r.quad := r.ent * r.ent
    --}}}
    --{{{  Procesos
```

```
            delta.ayu := x.minimo.ent - x.arregla
            cuanto.x  := cuanto.x - delta.ayu
            delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
            cuanto.x  := cuanto.x - delta.ayu
            IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
            --}}}
          TRUE
            SKIP
        --}}}
        --}}}
    cuadrante = 1
      SEQ
        --{{{  Dibuja el Cuadrante 1
        SEQ k.ayu = 0 FOR k
          SEQ
            --{{{  Procesos
            IF
              ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
                ( contorno.y[(k.ayu + 1)] <= yo.ent )
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x[k.ayu] - xo.ent
                  delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
                  cuanto.x    := delta.l.x + 1
                  x.arregla   := xo.ent
                  y.arregla   := yo.ent + delta.l.y
                  --{{{  Prueba se Esta en la Ventana Definida
                  IF
                    --{{{  Condicion 1
                    ( x.arregla >= x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 1
                        IniciaVectorBytes ( x.arregla, y.arregla, cuanto
                        --}}}
                    --{{{  Condicion 2
                    ( x.arregla < x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 2
                        delta.ayu := x.minimo.ent - x.arregla
                        cuanto.x  := cuanto.x - delta.ayu
                        IniciaVectorBytes ( x.minimo.ent, y.arregla, cua
                        --}}}
                    --{{{  Condicion 3
                    ( x.arregla >= x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 3
                        delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                        cuanto.x  := cuanto.x - delta.ayu
                        IniciaVectorBytes ( x.arregla, y.arregla, cuanto
                        --}}}
                    --{{{  Condicion 4
                    ( x.arregla < x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 4
                        delta.ayu := x.minimo.ent - x.arregla
                        cuanto.x  := cuanto.x - delta.ayu
                        delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                        cuanto.x  := cuanto.x - delta.ayu
                        IniciaVectorBytes ( x.minimo.ent, y.arregla, cua
                        --}}}
                  TRUE
                    SKIP
                --}}}
                --}}}
            TRUE
              SKIP
          --}}}
        --{{{  Dibuja la Ultima Linea
        delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
        x.arregla   := xo.ent
        y.arregla   := yo.ent + delta.l.y
        cuanto.x    := r.ent + 1
        IF
          condicion = 1
            SEQ
              --{{{  Dibuja Linea
              IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
              --}}}
          condicion = 2
            SEQ
              --{{{  Dibuja Linea
              delta.ayu := x.minimo.ent - x.arregla
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
              --}}}
          condicion = 3
            SEQ
              --{{{  Dibuja Linea
              delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
              --}}}
          condicion = 4
            SEQ
              --{{{  Dibuja Linea
              delta.ayu := x.minimo.ent - x.arregla
              cuanto.x  := cuanto.x - delta.ayu
              delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
              cuanto.x  := cuanto.x - delta.ayu
              IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
              --}}}
          TRUE
            SKIP
        --}}}
        --}}}
    cuadrante = 2
      SEQ
        --{{{  Dibuja el Cuadrante 2
        SEQ k.ayu = 0 FOR k
          SEQ
            --{{{  Procesos
            IF
              ( contorno.y[(k.ayu + 1)] <> contorno.y[k.ayu] ) AND
                ( contorno.y[(k.ayu + 1)] <= yo.ent )
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x[k.ayu] - xo.ent
                  delta.l.y   := - ( contorno.y[k.ayu] - yo.ent )
                  cuanto.x    := delta.l.x + 1
                  x.arregla   := xo.ent - delta.l.x
```

```
                delta.l.x   := contorno.x.inf[k.ayu] - xo.ent
                x.arregla   := xo.ent - delta.l.x
                cuanto.x    := delta.l.x + 1
                IniciaVectorBytes ( x.arregla, contorno.y.inf[k.ayu],
                                    cuanto.x, color )
                --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        --{{{  Dibuja parte triangular
        SEQ l.ayu = 0 FOR tamano
          SEQ
            --{{{  Procesos
            IF
              ( contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu] ) A
                ( contorno.y.sup[(l.ayu + 1)] <= yo.ent )
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x.sup[l.ayu] - xo.ent
                  x.arregla   := xo.ent - delta.l.x
                  cuanto.x    := delta.l.x + 1
                  IniciaVectorBytes ( x.arregla, contorno.y.sup[l.ayu],
                                      cuanto.x, color )
                  --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        --}}}
    oitavo = 7
      SEQ
        --{{{  Dibuja el Oitavo 7
        AlmacenaPuntosSegmentoNA ( xo.ent, yo.ent,
                                   contorno.x.inf[(l-1)], contorno.y.inf
                                   tamano, contorno.x.sup,
                                   contorno.y.sup )
        --{{{  Dibuja parte circular
        SEQ l.ayu = 0 FOR 1
          SEQ
            --{{{  Procesos
            IF
              contorno.y.inf[(l.ayu + 1)] <> contorno.y.inf[l.ayu]
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x.inf[l.ayu] - xo.ent
                  cuanto.x    := delta.l.x + 1
                  x.arregla   := xo.ent
                  --{{{  Prueba se Esta en la Ventana Definida
                  IF
                    --{{{  Condicion 1
                    ( x.arregla >= x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 1
                        IniciaVectorBytes ( xo.ent, contorno.y.inf[l.ayu
                                            cuanto.x, color )
                        --}}}
                    --{{{  Condicion 2
                    ( x.arregla < x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 2
                        delta.ayu := x.minimo.ent - x.arregla
                        cuanto.x  := cuanto.x - delta.ayu
                        IniciaVectorBytes ( x.minimo.ent, contorno.y.inf
                                            cuanto.x, color )
                        --}}}
                    --{{{  Condicion 3
                    ( x.arregla >= x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 3
                        delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                        cuanto.x  := cuanto.x - delta.ayu
                        IniciaVectorBytes ( xo.ent, contorno.y.inf[l.ayu
                                            cuanto.x, color )
                        --}}}
                    --{{{  Condicion 4
                    ( x.arregla < x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 4
                        delta.ayu := x.minimo.ent - x.arregla
                        cuanto.x  := cuanto.x - delta.ayu
                        delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                        cuanto.x  := cuanto.x - delta.ayu
                        IniciaVectorBytes ( x.minimo.ent, contorno.y.inf
                                            cuanto.x, color )
                        --}}}
                    TRUE
                      SKIP
                  --}}}
                  --}}}
              TRUE
                SKIP
            --}}}
        --}}}
        --{{{  Dibuja parte triangular
        SEQ k.ayu = 0 FOR tamano
          SEQ
            --{{{  Procesos
            IF
              ( contorno.y.sup[(k.ayu + 1)] <> contorno.y.sup[k.ayu] ) A
                ( contorno.y.sup[(k.ayu + 1)] <= yo.ent )
                SEQ
                  --{{{  Procesos
                  delta.l.x   := contorno.x.sup[k.ayu] - xo.ent
                  cuanto.x    := delta.l.x + 1
                  x.arregla   := xo.ent
                  --{{{  Prueba se Esta en la Ventana Definida
                  IF
                    --{{{  Condicion 1
                    ( x.arregla >= x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion  := 1
                        IniciaVectorBytes ( x.arregla, contorno.y.sup[k.
                                            cuanto.x, color )
                        --}}}
                    --{{{  Condicion 2
                    ( x.arregla < x.minimo.ent ) AND
```

```
SEQ
  --{{{  Asignaciones
  cuanto.r := r.ent + 1
  k        := 0
  k.ayu    := 0
  l        := 0
  l.ayu    := 0
  i        := xo.ent
  i.ayu    := 0
  x.quad   := 0
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Coge el Contorno
    SEQ j = ( yo.ent - r.ent ) FOR cuanto.r
      SEQ
        --{{{  Procesos
        j.ayu      := j - yo.ent
        y.quad     := j.ayu * j.ayu
        xq.mas.yq  := x.quad + y.quad
        WHILE xq.mas.yq  <= r.quad
          IF
            i.ayu <= ( - j.ayu )
              SEQ
                --{{{  Procesos
                contorno.x.inf[k] := xo.ent + i.ayu
                contorno.y.inf[k] := yo.ent + j.ayu
                i                 := i + 1
                k                 := k + 1
                i.ayu             := i - xo.ent
                x.quad            := i.ayu * i.ayu
                xq.mas.yq         := x.quad + y.quad
                --}}}
            TRUE
              SEQ
                --{{{  Procesos
                contorno.x.sup[l] := xo.ent + i.ayu
                contorno.y.sup[l] := yo.ent + j.ayu
                i                 := i + 1
                l                 := l + 1
                i.ayu             := i - xo.ent
                x.quad            := i.ayu * i.ayu
                xq.mas.yq         := x.quad + y.quad
                --}}}
        IF
          i > xo.ent
            SEQ
              --{{{  Procesos
              i       := i - 1
              i.ayu   := i - xo.ent
              x.quad  := i.ayu * i.ayu
              --}}}
          TRUE
            SKIP
        --}}}
    --}}}
    --{{{  Dibuja el Octavo de Circulo
    IF
      --{{{  De 5 hasta 8
      octavo = 5
        SEQ
          --{{{  Dibuja el Octavo 5
          --{{{  Almacena Puntos de Segmento
          AlmacenaPuntosSegmentoNA ( contorno.x.sup[0], contorno.y.sup[0],
                                     xo.ent, yo.ent,
                                     tamano, contorno.x.inf,
                                     contorno.y.inf )
          --}}}
          --{{{  Dibuja el Octavo
          k.ayu := tamano - 1
          l.ayu := 0
          WHILE ( contorno.y.sup[l.ayu] < yo.ent ) AND ( k.ayu > 0 )
            SEQ
              --{{{  Procesos
              IF
                contorno.y.inf[k.ayu] <> contorno.y.inf[(k.ayu - 1 )]
                  SEQ
                    IF
                      contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu]
                        SEQ
                          --{{{  Procesos
                          delta.l.x   := contorno.x.sup[l.ayu] - contorno.
                          delta.l.x.1 := contorno.x.sup[l.ayu] - xo.ent
                          cuanto.x    := delta.l.x  + 1
                          x.arregla   := xo.ent - delta.l.x.1
                          IniciaVectorBytes ( x.arregla, contorno.y.inf[l.
                                              cuanto.x, color )
                          --{{{  COMMENT Proc
                          --:::A COMMENT FOLD
                          --{{{  Proc
                          IF
                            ( x.arregla + cuanto.x ) <= x.maximo.ent
                              IniciaVectorBytes ( x.arregla, contorno.y.in
                                                  cuanto.x, color )
                            TRUE
                              SEQ
                                --{{{  Proc
                                cuanto.x := ( x.maximo.ent - x.arregla ) +
                                IniciaVectorBytes ( x.arregla, contorno.y.
                                                    cuanto.x, color )
                                --}}}
                          --}}}
                          --}}}
                          --}}}
                          k.ayu := k.ayu - 1
                          l.ayu := l.ayu + 1
                      TRUE
                        l.ayu := l.ayu + 1
                TRUE
                  k.ayu := k.ayu - 1
              --}}}
          --}}}
          --{{{  Dibuja la Ultima Linea
          cuanto.x  := r.ent + 1
          IniciaVectorBytes ( (xo.ent-r.ent), yo.ent, cuanto.x, color )
          --}}}
          --}}}
      octavo = 6
        SEQ
          --{{{  Dibuja el Octavo 6
          AlmacenaPuntosSegmentoNA ( xo.ent, yo.ent,
                                     contorno.x.inf[(k-1)], contorno.y.inf
                                     tamano, contorno.x.sup,
                                     contorno.y.sup )
          --{{{  Dibuja parte circular
          SEQ k.ayu = 0 FOR k
            SEQ
              --{{{  Procesos
              IF
                contorno.y.inf[(k.ayu + 1)] <> contorno.y.inf[k.ayu]
                  SEQ
                    --{{{  Procesos
```

```
                        condicion := 3
                        delta.ayu := ( ( contorno.x.inf[i.ayu] + c
                        cuanto.x  := cuanto.x - delta.ayu
                        IF
                          x.arregla <= x.maximo.ent
                            IniciaVectorBytes ( x.arregla, contorn
                                                cuanto.x, color )
                          TRUE
                            SKIP
                        --}}}
                    --{{{  Condicion 4
                    ( x.arregla < x.minimo.ent ) AND
                    ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                    --}}}
                      SEQ
                        --{{{  Dibuja Linea
                        condicion := 4
                        delta.ayu := x.minimo.ent - x.arregla
                        cuanto.x  := cuanto.x - delta.ayu
                        delta.ayu := ( ( contorno.x.inf[i.ayu] + c
                        cuanto.x  := cuanto.x - delta.ayu
                        IniciaVectorBytes ( x.minimo.ent, contorno
                                            cuanto.x, color )
                        --}}}
                    TRUE
                      SKIP
                  --}}}
                  --}}}
                  i.ayu := i.ayu - 1
                  k.ayu := k.ayu + 1
              TRUE
                k.ayu := k.ayu + 1
        TRUE
          i.ayu := i.ayu - 1
      --}}}
  --}}}
  --{{{  Dibuja la Ultima Linea
  cuanto.x  := r.ent + 1
  --{{{  Dibuja la Ultima Linea
  x.arregla := xo.ent
  cuanto.x  := r.ent + 1
  IF
    condicion = 1
      SEQ
        --{{{  Dibuja Linea
        IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
        --}}}
    condicion = 2
      SEQ
        --{{{  Dibuja Linea
        delta.ayu := x.minimo.ent - x.arregla
        cuanto.x  := cuanto.x - delta.ayu
        IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
        --}}}
    condicion = 3
      SEQ
        --{{{  Dibuja Linea
        delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
        cuanto.x  := cuanto.x - delta.ayu
        IniciaVectorBytes ( x.arregla, yo.ent, cuanto.x, color )
        --}}}
    condicion = 4
      SEQ
        --{{{  Dibuja Linea
        delta.ayu := x.minimo.ent - x.arregla
        cuanto.x  := cuanto.x - delta.ayu
```

```
        delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) - x.maximo.e
        cuanto.x  := cuanto.x - delta.ayu
        IniciaVectorBytes ( x.minimo.ent, yo.ent, cuanto.x, color
        --}}}
      TRUE
        SKIP
    --}}}
    --}}}
    --}}}
--}}}
--{{{  De 1 hasta 4
octavo = 1
  SEQ
    --{{{  Dibuja el Octavo 1
    --{{{  Almacena Puntos de Segmento
    AlmacenaPuntosSegmentoNA ( contorno.x.sup[0], contorno.y.sup[0],
                               xo.ent, yo.ent,
                               tamano, contorno.x.inf,
                               contorno.y.inf )
    --}}}
    --{{{  Dibuja el Octavo
    k.ayu := tamano - 1
    l.ayu := 0
    WHILE ( contorno.y.sup[l.ayu] < yo.ent ) AND ( k.ayu > 0 )
      SEQ
        --{{{  Procesos
        IF
          contorno.y.inf[k.ayu] <> contorno.y.inf[(k.ayu - 1 )]
            SEQ
              IF
                contorno.y.sup[(l.ayu + 1)] <> contorno.y.sup[l.ayu]
                  SEQ
                    --{{{  Procesos
                    delta.l.x   := contorno.x.sup[l.ayu] - contorno.
                    delta.l.y   := - ( contorno.y.sup[l.ayu] - yo.en
                    cuanto.x    := delta.l.x  + 1
                    y.arregla   := yo.ent + delta.l.y
                    IniciaVectorBytes ( contorno.x.inf[k.ayu], y.arr
                                        cuanto.x, color )
                    --}}}
                    k.ayu := k.ayu - 1
                    l.ayu := l.ayu + 1
                TRUE
                  l.ayu := l.ayu + 1
          TRUE
            k.ayu := k.ayu - 1
        --}}}
    --}}}
    --{{{  Dibuja la Ultima Linea
    cuanto.x  := r.ent + 1
    IniciaVectorBytes ( xo.ent, yo.ent, cuanto.x, color )
    --}}}
    --}}}
octavo = 2
  SEQ
    --{{{  Dibuja el Octavo 2
    AlmacenaPuntosSegmentoNA ( xo.ent, yo.ent,
                               contorno.x.inf[(k-1)], contorno.y.inf
                               tamano, contorno.x.sup,
                               contorno.y.sup )
    --{{{  Dibuja parte circular
    SEQ k.ayu = 0 FOR k
      SEQ
        --{{{  Procesos
        IF
          contorno.y.inf[(k.ayu + 1)] <> contorno.y.inf[k.ayu]
```

```
              ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
              --}}}
                SEQ
                  --{{{  Dibuja Linea
                  condicion  := 2
                  delta.ayu := x.minimo.ent - x.arregla
                  cuanto.x  := cuanto.x - delta.ayu
                  IniciaVectorBytes ( x.minimo.ent, contorno.y.sup
                                      cuanto.x, color )
                  --}}}
              --{{{  Condicion 3
              ( x.arregla >= x.minimo.ent ) AND
              ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
              --}}}
                SEQ
                  --{{{  Dibuja Linea
                  condicion  := 3
                  delta.ayu  := ( ( x.arregla + cuanto.x ) - 1 ) -
                  cuanto.ayu := cuanto.x - delta.ayu
                  IniciaVectorBytes ( x.arregla, contorno.y.sup[k.
                                      cuanto.ayu, color )
                  --}}}
              --{{{  Condicion 4
              ( x.arregla < x.minimo.ent ) AND
              ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
              --}}}
                SEQ
                  --{{{  Dibuja Linea
                  condicion  := 4
                  delta.ayu := x.minimo.ent - x.arregla
                  cuanto.x  := cuanto.x - delta.ayu
                  delta.ayu := ( ( x.arregla + cuanto.x ) - 1 ) -
                  cuanto.x  := cuanto.x - delta.ayu
                  IniciaVectorBytes ( x.minimo.ent, contorno.y.sup
                                      cuanto.x, color )
                  --}}}
            TRUE
              SKIP
          --}}}
          --}}}
      TRUE
        SKIP
    --}}}
--}}}
--{{{  Dibuja parte cuadrada
IF
  --{{{  Condicion 3
  ( x.arregla >= x.minimo.ent ) AND
  ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
  --}}}
    SEQ
      --{{{  Dibuja Linea
      condicion  := 3
      cuanto.x    := contorno.x.sup[(tamano-1)] - xo.ent
      delta.l.y   := ( contorno.y.sup[(tamano-1)] - contorno.y.i
      SEQ i.ayu=contorno.y.inf[(1-1)] FOR delta.l.y
        IniciaVectorBytes ( xo.ent, i.ayu, cuanto.x, color )
      --}}}
  --{{{  Condicion 4
  ( x.arregla < x.minimo.ent ) AND
  ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
  --}}}
    SEQ
      --{{{  Dibuja Linea
      condicion   := 4
      cuanto.x    := x.maximo.ent - x.minimo.ent
      delta.l.y   := ( contorno.y.sup[0] - contorno.y.inf[k.ayu]
      SEQ i.ayu=contorno.y.inf[k.ayu] FOR delta.l.y
        IniciaVectorBytes ( x.minimo.ent, i.ayu, cuanto.x, color
      --}}}
  TRUE
    SKIP
--}}}
--}}}
octavo = 8
  SEQ
    --{{{  Dibuja el Octavo 8
    --{{{  Almacena Puntos de Segmento
    --AlmacenaPuntosSegmentoNA ( contorno.x.sup[0], contorno.y.sup[0
    i.ayu  := contorno.x.inf[(1-1)]
    k.ayu  := contorno.y.inf[(1-1)]
    AlmacenaPuntosSegmentoNA ( i.ayu, k.ayu,
                               xo.ent, yo.ent,
                               tamano, contorno.x.inf,
                               contorno.y.inf )
    --}}}
    --{{{  Dibuja el Octavo
    i.ayu := tamano - 1
    k.ayu := 0
    WHILE ( contorno.y.sup[k.ayu] <= yo.ent ) AND ( i.ayu >= 0 )
      SEQ
        --{{{  Procesos
        IF
          contorno.y.inf[i.ayu] <> contorno.y.inf[(i.ayu - 1 )]
            SEQ
              IF
                contorno.y.sup[(k.ayu + 1)] <> contorno.y.sup[k.ayu]
                  SEQ
                    --{{{  Procesos
                    delta.l.x   := contorno.x.sup[k.ayu] - contorno.
                    cuanto.x    := delta.l.x  + 1
                    x.arregla   := contorno.x.inf[i.ayu]
                    --{{{  Prueba se Esta en la Ventana Definida
                    IF
                      --{{{  Condicion 1
                      ( x.arregla >= x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 1
                          IniciaVectorBytes ( x.arregla, contorno.y.
                                              cuanto.x, color )
                          --}}}
                      --{{{  Condicion 2
                      ( x.arregla < x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) <= x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
                          condicion  := 2
                          delta.ayu := x.minimo.ent - x.arregla
                          cuanto.x  := cuanto.x - delta.ayu
                          IniciaVectorBytes ( x.minimo.ent, contorno
                                              cuanto.x, color )
                          --}}}
                      --{{{  Condicion 3
                      ( x.arregla >= x.minimo.ent ) AND
                      ( ((x.arregla+cuanto.x)-1) > x.maximo.ent )
                      --}}}
                        SEQ
                          --{{{  Dibuja Linea
```

```
        color.ayu    := color.ayu - delta.color
        color.cambia := BYTE(color.ayu)
  --}}}
:
--}}}
--
--}}}
--
--{{{  Verificacion de ejecucion/Errores
--
--{{{  Envia Entero Para Verificacion de Ejecucion
PROC VerificacionEjecucion ()
  SEQ
    --{{{  Procesos
    a.ver  := 0
    Mp.Org !  a.ver
    --}}}
:
--}}}
--
--}}}
--
--}}}
--
--
--
-- -- Programa de LLamada de los Procedimientos Graficos --
--
--{{{  Programa principal
SEQ
  --{{{  Asignaciones
  color.elegido.ent := 0
  color := BYTE(color.elegido.ent)
  activo := TRUE
  x.maximo.ant.ent := 1023
  y.maximo.ant.ent := 767
  --}}}
  --{{{  Paleta Inicial
  Org.Mp ? paleta
  Mp.Cp ! paleta
  paleta.ant := paleta
  --}}}
  --{{{  Cargar Fuente
  Org.Mp ?  origen; destino; tipo; longitud
  --{{{  Prueba de Longitud
  IF
    longitud > 0
      SEQ
        Org.Mp ? [ parametros FROM 0  FOR longitud ]
    TRUE
      SKIP
  --}}}
  --{{{  Cargar Fuente de Caracteres
  longitud.fuente := parametros[0]
  num.carac       := parametros[1]
  fuente.x        := parametros[2]
  fuente.y        := parametros[3]
  IF
    longitud.fuente > 0
      CargaFuenteCaracteres ( num.carac, fuente.x, fuente.y,
                              longitud.fuente )
    TRUE
      SKIP
  --}}}
  --}}}
  WHILE activo
```

```
    --{{{  Procesos
    SEQ
      Org.Mp ?  origen; destino; tipo; longitud
      --{{{  Prueba de Longitud
      IF
        longitud > 0
          Org.Mp ? [ parametros FROM 0  FOR longitud ]
        TRUE
          SKIP
      --}}}
      --{{{  Elegir Rotinas
      IF
        --{{{  Fundamentales
        --{{{  Iniciales
        tipo = t.borrar
          --{{{  Borra Pantalla
          SEQ
            BorraPantalla ()
            VerificacionEjecucion ()
          --}}}
        tipo = t.x
          --{{{  Inicia Pantalla
          SEQ
            --{{{  Procesos
            x.maximo :=  REAL32 ROUND(parametros[0])
            x.maximo.ant.ent := parametros[0]
            y.maximo :=  REAL32 ROUND(parametros[1])
            y.maximo.ant.ent := parametros[1]
            IniciaPantalla ( x.maximo, y.maximo )
            VerificacionEjecucion ()
            --}}}
          --}}}
        tipo = t.color
          --{{{  Color
          SEQ
            --{{{  Procesos
            color.elegido.ent := parametros[0]
            color.elegido := ( BYTE( color.elegido.ent ) )
            Color ( color.elegido )
            VerificacionEjecucion ()
            --}}}
          --}}}
        tipo = t.iniciar
          --{{{  Inicia Pantalla
          SEQ
            --{{{  Procesos
            x.maximo :=  REAL32 ROUND(parametros[0])
            x.maximo.ant.ent := parametros[0]
            y.maximo :=  REAL32 ROUND(parametros[1])
            y.maximo.ant.ent := parametros[1]
            IniciaPantalla ( x.maximo, y.maximo )
            VerificacionEjecucion ()
            --}}}
          --}}}
        tipo = t.y
          --{{{  Cambia Paleta
          SEQ
            IF
              paleta.ant = 2
                SEQ
                  --{{{  No Puede Cambiar Paleta
                  -- Esta Rutina Corre Directamente En Controle Pantalla
                  VerificacionEjecucion ()
                  --}}}
              TRUE
                SEQ
```

```
            SEQ
              i := 1024
              j := 768
          TRUE & SKIP
            SEQ
              --{{{  Proc
              --{{{  Asignaciones
              i.r         :=  REAL32 ROUND(i)
              j.r         :=  REAL32 ROUND(j)
              z.real      :=  (m.x * i.r) - real.1
              z.imag      :=  real.1 - (m.y * j.r)
              i.r.inic    :=  z.real
              j.r.inic    :=  z.imag
              z.real.ayu  :=  z.real
              z.imag.ayu  :=  z.imag
              n.inter     :=  0
              mod.z       :=  (( z.real.ayu * z.real.ayu ) + ( z.imag.ayu *
              --}}}
              WHILE ( n.inter < 255 ) AND ( ( INT ROUND(mod.z) ) < 4 )
                SEQ
                  --{{{  Proc
                  n.inter     :=  n.inter + 1
                  z.real      :=  (((z.real.ayu*z.real.ayu)-(z.imag.ayu*z.i
                  z.imag      :=  (((real.2*z.real.ayu)*z.imag.ayu) + j.r.i
                  z.real.ayu  :=  z.real
                  z.imag.ayu  :=  z.imag
                  mod.z       :=  (( z.real.ayu * z.real.ayu ) + ( z.imag.a
                  --}}}
              Color ( BYTE( 255 - n.inter ) )
              pantalla[ j ][ i ] := color
              --}}}
        j := j + 1
      i := i + 1
  --}}}
:
--}}}
--}}}
--
--{{{  EJEMPLO 4 : Julia
PROC Julia ()
  --{{{  Declaraciones
  --}}}
  SEQ
    --{{{  Proc
    --}}}
:
--}}}
--
--{{{  EJEMPLO 5 : Mostrar Paleta
PROC MostraPaleta ()
  --{{{  Declaraciones
  REAL32 xi, yi, xf, yf :
  INT xi.e, yi.e, xf.e, yf.e :
  INT pos.x :
  --}}}
  SEQ i=0 FOR 256
    --{{{  Proc
    SEQ
      pos.x := ( i + 1 ) \ 19
    --}}}
:
--}}}
--
--{{{  EJEMPLO 9 : Dibujar Cuadrados y Circulos Lentos
PROC DibujaEjemplo9 ( VAL BYTE color.inicial )
  --{{{  Procesos
  --{{{  Variables
  INT delta.color, dos.delta.color, color.ayu  :
  BYTE color.cambia, color.ayu1 :
  REAL32 delta.coord :
  REAL32 r.coord :
  REAL32 xi.coord, yi.coord, xf.coord, yf.coord, xo.coord, yo.coord :
  REAL32 x.minimo.coord, y.minimo.coord :
  REAL32 x.maximo.coord, y.maximo.coord :
  --}}}
  --{{{  Constantes
  VAL xi.coord.ent          IS    128 :
  VAL yi.coord.ent          IS      0 :
  VAL xf.coord.ent          IS    895 :
  VAL yf.coord.ent          IS    767 :
  VAL xo.coord.ent          IS    511 :
  VAL yo.coord.ent          IS    384 :
  VAL  r.coord.ent          IS    383 :
  VAL  x.minimo.coord.ent   IS      0 :
  VAL  x.maximo.coord.ent   IS   1023 :
  VAL  y.minimo.coord.ent   IS      0 :
  VAL  y.maximo.coord.ent   IS    767 :
  VAL  x.limite.coord.ent   IS    500 :
  --}}}
  SEQ
    --{{{  Asignacion de Variables
    xi.coord          :=  REAL32 ROUND(xi.coord.ent)
    yi.coord          :=  REAL32 ROUND(yi.coord.ent)
    xf.coord          :=  REAL32 ROUND(xf.coord.ent)
    yf.coord          :=  REAL32 ROUND(yf.coord.ent)
    xo.coord          :=  REAL32 ROUND(xo.coord.ent)
    yo.coord          :=  REAL32 ROUND(yo.coord.ent)
    r.coord           :=  REAL32 ROUND( r.coord.ent)
    x.minimo.coord    :=  REAL32 ROUND(x.minimo.coord.ent)
    x.maximo.coord    :=  REAL32 ROUND(x.maximo.coord.ent)
    y.minimo.coord    :=  REAL32 ROUND(y.minimo.coord.ent)
    y.maximo.coord    :=  REAL32 ROUND(y.maximo.coord.ent)
    delta.color       :=  13
    dos.delta.color   :=  2 * delta.color
    delta.coord       :=  REAL32 ROUND(20)
    color.cambia      :=  color.inicial
    color.ayu         :=  255
    color.ayu1        :=  BYTE(color.ayu)
    --}}}
    IniciaPantalla ( x.maximo.coord, y.maximo.coord )
    Color ( color.ayu1 )
    Rectangulo1 ( x.minimo.coord, y.minimo.coord, x.maximo.coord, y.maximo.coo
    WHILE xi.coord <= ( REAL32 ROUND(x.limite.coord.ent) )
      SEQ
        Color ( color.cambia )
        --{{{  Rectangulo
        Rectangulo1 ( xi.coord, yi.coord, xf.coord, yf.coord )
        xi.coord :=  xi.coord + delta.coord
        yi.coord :=  yi.coord + delta.coord
        xf.coord :=  xf.coord - delta.coord
        yf.coord :=  yf.coord - delta.coord
        --}}}
        color.ayu := INT(color.cambia)
        color.ayu := color.ayu + dos.delta.color
        WHILE color.ayu > 255
          color.ayu  := color.ayu - 1
        color.cambia := BYTE(color.ayu)
        Color ( color.cambia )
        --{{{  Circulo
        Circulo1 ( xo.coord, yo.coord, r.coord )
        r.coord := r.coord - delta.coord
        --}}}
```

```
      ancho.int.r  := REAL32 ROUND(parametros[3])
      EspiralCuarto ( xo, yo, r.ext, ancho.int.r )
      VerificacionEjecucion ()
    --}}}
  tipo = t.siete
    --{{{  Espiral con Contorno
    SEQ
      xo           := REAL32 ROUND(parametros[0])
      yo           := REAL32 ROUND(parametros[1])
      r.ext        := REAL32 ROUND(parametros[2])
      ancho.int.r  := REAL32 ROUND(parametros[3])
      EspiralOitavoContorno ( xo, yo, r.ext, ancho.int.r )
      --EspiralCuartoContorno1 ( xo, yo, r.ext, ancho.int.r )
      VerificacionEjecucion ()
    --}}}
  tipo = t.nueve
    --{{{  Espiral de dos Colores
    SEQ
      xo           := REAL32 ROUND(parametros[0])
      yo           := REAL32 ROUND(parametros[1])
      r.ext        := REAL32 ROUND(parametros[2])
      ancho.int.r  := REAL32 ROUND(parametros[3])
      color1       := BYTE(parametros[4])
      color2       := BYTE(parametros[5])
      EspiralSemiDosColores ( xo, yo, r.ext, ancho.int.r, color1, color2
      VerificacionEjecucion ()
    --}}}
--}}}
tipo = t.d
  --{{{  Elipse
  SEQ
    VerificacionEjecucion ()
  --}}}
tipo = t.e
  --{{{  Arco
  SEQ
    VerificacionEjecucion ()
  --}}}
tipo = t.g
  --{{{  Corona
  SEQ
    xo      := REAL32 ROUND(parametros[0])
    yo      := REAL32 ROUND(parametros[1])
    r.int   := REAL32 ROUND(parametros[2])
    r.ext   := REAL32 ROUND(parametros[3])
    Corona ( xo, yo, r.int, r.ext )
    VerificacionEjecucion ()
  --}}}
tipo = t.h
  --{{{  Circulo Lento
  SEQ
    xo := REAL32 ROUND(parametros[0])
    yo := REAL32 ROUND(parametros[1])
    r  := REAL32 ROUND(parametros[2])
    Circulo1 ( xo, yo, r )
    VerificacionEjecucion ()
  --}}}
--}}}
--}}}
--{{{  Zoom de Pantalla
tipo = t.z
  --{{{  Zoom
  SEQ
    VerificacionEjecucion ()
  --}}}
--}}}
--}}}
--{{{  Caracteres/Ficheros
tipo = t.otras
  --{{{  Cargar Fuente de Caracteres
  SEQ
    longitud.fuente := parametros[0]
    num.carac       := parametros[1]
    fuente.x        := parametros[2]
    fuente.y        := parametros[3]
    CargaFuenteCaracteres ( num.carac, fuente.x, fuente.y,
                            longitud.fuente )
    VerificacionEjecucion ()
  --}}}
tipo = t.q
  --{{{  Escribir Ficheros ASCII en la Pantalla
  SEQ
    EscribirFicherosASCII ( )
    VerificacionEjecucion ()
  --}}}
tipo = t.s
  --{{{  Escribir Cadenas de Caracteres en la Pantalla
  SEQ
    --{{{  Coge los Parametros y Mensaje
    color.elegido.ent := parametros[0]
    x.mensaje         := parametros[1]
    y.mensaje         := parametros[2]
    SEQ i=3 FOR ( longitud - 3 )
      cadena.mensaje.ent[(i-3)] := parametros[i]
    --}}}
    EscribirMensajePantallaGrafica ( color.elegido.ent, x.mensaje,
                                     y.mensaje, cadena.mensaje.ent )
    VerificacionEjecucion ()
  --}}}
tipo = t.t
  --{{{  Escribir Cadenas de Caracteres Dobles en la Pantalla
  SEQ
    --{{{  Coge los Parametros y Mensaje
    color.elegido.ent := parametros[0]
    x.mensaje         := parametros[1]
    y.mensaje         := parametros[2]
    SEQ i=3 FOR ( longitud - 3 )
      cadena.mensaje.ent[(i-3)] := parametros[i]
    --}}}
    EscribirMensajeCaracterDoble ( color.elegido.ent, x.mensaje,
                                   y.mensaje, cadena.mensaje.ent )
    VerificacionEjecucion ()
  --}}}
tipo = t.w
  --{{{  Escribir Cadenas de Caracteres con Ancho/Alto Variables
  SEQ
    --{{{  Coge los Parametros y Mensaje
    color.elegido.ent := parametros[0]
    x.mensaje         := parametros[1]
    y.mensaje         := parametros[2]
    ancho.esc         := parametros[3]
    alto.esc          := parametros[4]
    SEQ i=5 FOR ( longitud - 5 )
      cadena.mensaje.ent[(i-5)] := parametros[i]
    --}}}
    EscribirMensajeCaracterVariable ( color.elegido.ent,   x.mensaje,
                                      y.mensaje, ancho.esc, alto.esc,
                                      cadena.mensaje.ent )
    VerificacionEjecucion ()
  --}}}
--}}}
--{{{  Ejemplos
```

```
          --{{{  Puede Cambiar Paleta
          -- Esta Rutina Corre Directamente En Controle Pantalla
          paleta := parametros[0]
          Mp.Cp ! paleta
          paleta.ant := paleta
          VerificacionEjecucion ()
          --}}}
    --}}}
  --}}}
  --{{{  Que Dibujan
  --{{{  Rectilineas
  tipo = t.linea
    --{{{  Tira Linea
    SEQ
      --{{{  Procesos
      TiraLineaEntero ( parametros[0], parametros[1],
                        parametros[2], parametros[3]  )
      VerificacionEjecucion ()
      --}}}
    --}}}
  tipo = t.rectang
    --{{{  Rectangulo
    SEQ
      --{{{  Procesos
      xi := ( REAL32 ROUND(parametros[0]) )
      yi := ( REAL32 ROUND(parametros[1]) )
      xf := ( REAL32 ROUND(parametros[2]) )
      yf := ( REAL32 ROUND(parametros[3]) )
      Rectangulo ( xi, yi, xf, yf )
      VerificacionEjecucion ()
      --}}}
    --}}}
  tipo = t.k
    --{{{  Flechas
    SEQ
      VerificacionEjecucion ()
    --}}}
  tipo = t.m
    --{{{  Paralelogramos
    SEQ
      Paralelogramo ( parametros[0], parametros[1], parametros[2],
                      parametros[3], parametros[4], parametros[5] )
      VerificacionEjecucion ()
    --}}}
  tipo = t.n
    --{{{  Poligonos
    SEQ
      VerificacionEjecucion ()
    --}}}
  tipo = t.p
    --{{{  Rectangulo Lento
    SEQ
      --{{{  Procesos
      xi := ( REAL32 ROUND(parametros[0]) )
      yi := ( REAL32 ROUND(parametros[1]) )
      xf := ( REAL32 ROUND(parametros[2]) )
      yf := ( REAL32 ROUND(parametros[3]) )
      Rectangulo1 ( xi, yi, xf, yf )
      VerificacionEjecucion ()
      --}}}
    --}}}
  --}}}
  --{{{  Curvilineas
  --{{{  Circulos
  tipo = t.circulares
    --{{{  Circulo
    SEQ
      xo := REAL32 ROUND(parametros[0])
      yo := REAL32 ROUND(parametros[1])
      r  := REAL32 ROUND(parametros[2])
      Circulo ( xo, yo, r )
      VerificacionEjecucion ()
    --}}}
  tipo = t.dos
    --{{{  Semicirculo
    SEQ
      xo      := REAL32 ROUND(parametros[0])
      yo      := REAL32 ROUND(parametros[1])
      r       := REAL32 ROUND(parametros[2])
      sup     := parametros[3]
      SemiCirculo ( xo, yo, r, sup )
      VerificacionEjecucion ()
    --}}}
  tipo = t.tres
    --{{{  Cuarto de circulo
    SEQ
      xo            := REAL32 ROUND(parametros[0])
      yo            := REAL32 ROUND(parametros[1])
      r             := REAL32 ROUND(parametros[2])
      cuadrante     := parametros[3]
      CuartoDeCirculo ( xo, yo, r, cuadrante )
      VerificacionEjecucion ()
    --}}}
  --}}}
  --{{{  Circunferencias
  tipo = t.a
    --{{{  Circunferencia
    SEQ
      xo := ( REAL32 ROUND(parametros[0]) )
      yo := ( REAL32 ROUND(parametros[1]) )
      r  := ( REAL32 ROUND(parametros[2]) )
      Circunferencia ( xo, yo, r )
      VerificacionEjecucion ()
    --}}}
  tipo = t.cinco
    --{{{  Semicircunferencia
    SEQ
      xo      := REAL32 ROUND(parametros[0])
      yo      := REAL32 ROUND(parametros[1])
      r       := REAL32 ROUND(parametros[2])
      sup     := parametros[3]
      SemiCircunferencia ( xo, yo, r, sup )
      VerificacionEjecucion ()
    --}}}
  tipo = t.seis
    --{{{  Cuarto de Circunferencia
    SEQ
      xo            := REAL32 ROUND(parametros[0])
      yo            := REAL32 ROUND(parametros[1])
      r             := REAL32 ROUND(parametros[2])
      cuadrante     := parametros[3]
      CuartoDeCircunferencia ( xo, yo, r, cuadrante )
      VerificacionEjecucion ()
    --}}}
  --}}}
  --{{{  Espirales
  tipo = t.cuatro
    --{{{  Espiral sin Contorno
    SEQ
      xo            := REAL32 ROUND(parametros[0])
      yo            := REAL32 ROUND(parametros[1])
      r.ext         := REAL32 ROUND(parametros[2])
```

```
  VAL     fifo.llena     IS   2 :
  --}}}
  --
  --{{{  Declaracion de Canales y Variables y sus Direciones
  --
  --{{{  Timer
  TIMER tiempo :
  --}}}
  --
  --{{{  Canales
  CHAN OF ANY event:
  --}}}
  --
  --{{{  Variables
  --
  INT ahora, retraso :
  INT registrador.mandato, registrador.estado, parametro.fifo :
  INT pixel.mask, pixel.addr.write:
  INT selec.modo.canal :
  --
  [768]BYTE colour.val.block :
  [768]BYTE tabla.colores :
  [768]BYTE tabla.ayu :
  --
  BYTE nada :
  --
  --}}}
  --
  --{{{  Direciones
  PLACE  registrador.estado  AT   (#A000 / 2)   PLUS   #4000 :
  PLACE registrador.mandato  AT   (#A002 / 2)   PLUS   #4000 :
  PLACE       parametro.fifo  AT   (#A000 / 2)   PLUS   #4000 :
  PLACE     pixel.addr.write  AT   (#0000 / 2)   PLUS   #4000 :
  PLACE           pixel.mask  AT   (#0800 / 2)   PLUS   #4000 :
  PLACE     colour.val.block  AT   (#4400 / 2)   PLUS   #4000 :
  PLACE                event  AT                        8 :
  PLACE     selec.modo.canal  AT   (#B000 / 2)   PLUS   #4000 :
  --}}}
  --
  --}}}
  --
  --{{{  Declaracion de Procedimientos ( del manual de la B409 y otros )
  --
  --{{{  Procedimientos Para Escribir/Leer Mandatos/Estado
  --
  --{{{  PROC EscribeMandato ( VAL INT mandato )
  PROC EscribeMandato ( VAL INT mandato )
    TIMER tiempo:
    INT ahora:
    SEQ
      tiempo ? ahora
      tiempo ? AFTER (ahora PLUS 1)
      --espera fifo quedar vacia
      WHILE (registrador.estado /\ fifo.vacia) <> fifo.vacia
        SEQ
          tiempo ? ahora
          tiempo ? AFTER (ahora PLUS 1)
      tiempo ? ahora
      tiempo ? AFTER (ahora PLUS 1)
      registrador.mandato := mandato
  :
  --}}}
  --
  --{{{  PROC EscribeParametro ( VAL INT dato )
  PROC EscribeParametro (VAL INT dato)
    TIMER tiempo :
    INT ahora :
    SEQ
      tiempo ? ahora
      tiempo ? AFTER (ahora PLUS 1)
      --espera se fifo esta llena
      WHILE (registrador.estado /\ fifo.llena) = fifo.llena
        SEQ
          tiempo ? ahora
          tiempo ? AFTER (ahora PLUS 1)
      tiempo ? ahora
      tiempo ? AFTER (ahora PLUS 1)
      parametro.fifo := dato
  :
  --}}}
  --
  --{{{  PROC Activar ()
  PROC Activar ()
    EscribeMandato (#0D)
  :
  --}}}
  --
  --{{{  PROC PonerModo ( VAL INT modo )
  PROC PonerModo (VAL INT modo)
    SEQ
      tiempo ? ahora
      tiempo ? AFTER (ahora PLUS 1)
      selec.modo.canal := modo
      tiempo ? ahora
      tiempo ? AFTER (ahora PLUS 1)
  :
  --}}}
  --
  --{{{  PROC Reiniciar ()
  PROC Reiniciar ()
    --{{{  Definiciones para programar el Timing-Generator  1024x768
    VAL aw   IS   30:
    VAL hfp  IS    1:
    VAL hs   IS    3:
    VAL hbp  IS    7:
    VAL al   IS  768:
    VAL vfp  IS   27:
    VAL vs   IS    7:
    VAL vbp  IS   34:
    --}}}
    --{{{  Construccion de la secuencia de bytes para la orden RESET
    VAL b1 IS #12 :
    VAL b2 IS aw :
    VAL b3 IS (((vs /\ 7) << 5) \/ hs) :
    VAL b4 IS ((hfp << 2) \/ ((vs /\ #18) >> 3)) :
    VAL b5 IS hbp :
    VAL b6 IS vfp :
    VAL b7 IS (al /\ #FF) :
    VAL b8 IS ((vbp << 2) \/ ((al /\ #300) >> 8)) :
    VAL []INT param IS [b1, b2, b3, b4, b5, b6, b7, b8]:
    --}}}
    SEQ
      registrador.mandato := 0   --- Prueba para resetear
      EscribeMandato ( 0 )
      SEQ i=0 FOR 8
        EscribeParametro (param[i])
  :
  --}}}
  --
  --{{{  PROC LeeEstado ( INT32 estado )
  PROC LeeEstado ( INT32 estado )
    INT a :
```

```
      tipo = t.ventanas
        --{{{  Ventanas
        SEQ
          color.elegido.ent := parametros[0]
          color.elegido := ( BYTE( color.elegido.ent ) )
          DibujaEjemplo0 ( color.elegido )
          VerificacionEjecucion ()
        --}}}
      tipo = t.cero
        --{{{  Dibuja Circulos Cambiando xo, yo y radio
        SEQ
          ayu.real := REAL32 ROUND(parametros[0])
          semilla.ayu := INT32 ROUND(ayu.real)
          DibujaEjemplo2 ( semilla.ayu )
          VerificacionEjecucion ()
        --}}}
      tipo = t.uno
        --{{{  Cuadrados y Circulos
        SEQ
          color.elegido.ent := parametros[0]
          color.elegido := ( BYTE( color.elegido.ent ) )
          DibujaEjemplo1 ( color.elegido )
          VerificacionEjecucion ()
        --}}}
      tipo = t.ejemplos
        --{{{  Cuadrados y Circulos Lentos
        SEQ
          color.elegido.ent := parametros[0]
          color.elegido := ( BYTE( color.elegido.ent ) )
          DibujaEjemplo7 ( color.elegido )
          VerificacionEjecucion ()
        --}}}
      tipo = t.A
        --{{{  Mandelbroot/Julia
        SEQ
          --{{{  Procesos
          opcion  := parametros[0]
          IF
            opcion = 0
              Mandelbroot ()
            opcion = 1
              Julia ()
            TRUE
              Mandelbroot ()
          VerificacionEjecucion ()
          --}}}
        --}}}
      tipo = t.B
        --{{{  Mostrar Paleta
        SEQ
          MostraPaleta ()
          VerificacionEjecucion ()
        --}}}
    --}}}
    --{{{  Rutina en Prueba
    tipo = t.ocho
      --{{{  Prueba
      SEQ
        --{{{  COMMENT Prueba Anterior
        --:::A COMMENT FOLD
        --{{{  Prueba Anterior
        xo           := REAL32 ROUND(parametros[0])
        yo           := REAL32 ROUND(parametros[1])
        r.ext        := REAL32 ROUND(parametros[2])
        ancho.int.r  := REAL32 ROUND(parametros[3])
        color1       := BYTE(parametros[4])
        color2       := BYTE(parametros[5])
        EspiralOitavoDosColores1 ( xo, yo, r.ext, ancho.int.r, color1, col
        --}}}
        --}}}
        --{{{  COMMENT Prueb.
        --:::A COMMENT FOLD
        --{{{  Prueb.
        xo      := REAL32 ROUND(parametros[0])
        yo      := REAL32 ROUND(parametros[1])
        r       := REAL32 ROUND(parametros[2])
        sup     := parametros[3]
        SemiCirculo1 ( xo, yo, r, sup )
        VerificacionEjecucion ()
        --}}}
        --}}}
        xo              := REAL32 ROUND(parametros[0])
        yo              := REAL32 ROUND(parametros[1])
        r               := REAL32 ROUND(parametros[2])
        cuadrante       := parametros[3]
        OitavoDeCirculo1 ( xo, yo, r, cuadrante )
        VerificacionEjecucion ()
      --}}}
    --}}}
    --{{{  Salir
    tipo = t.fin
      --{{{  Salida
      SEQ
        activo := FALSE
      --}}}
    --}}}
    TRUE
      SKIP
  --}}}
--}}}
--
--}}}
:
--}}}F
--...F code
--:::A  1 2
--:::F Memori27.dcd
--...F descriptor
--:::A  1 4
--:::F Memori27.dds
--...F link
--:::A  1 9
--:::F Memori27.dlk
--...F debug
--:::A  1 5
--:::F Memori27.ddb
--...F CODE SC Operaciones Graficas - Memoria de la Pantalla en el B408
--:::A  2 10
--:::F Memori27.csc
--}}}
--{{{  SC Operaciones Graficas - Controle de la Pantalla en el B409
--:::A  3 10
--{{{F ControlPantalla
--:::F Contro27.tsr
PROC ControlPantalla ( CHAN OF ANY Mp.Cp, Cp.Mp, Org.Cp, Cp.Org )
  --{{{  Procesos
  --
  -- --- Implementacion de Rutinas de Control de la Pantalla ---
  --
  --{{{  Declaracion de constantes
  VAL     fifo.vacia     IS   4 :
```

```
        [tabla.colores FROM punt FOR 3] := [BYTE (rojo), BYTE (verde), BYTE (a
        punt := punt - 3
        --}}}
      --}}}
    LoadTableA (tabla.colores)
:
--}}},
--
--{{{  PROC HaceClut8Suave ([768]BYTE tabla.colores)
PROC HaceClut8Suave ([768]BYTE tabla.colores)
  INT punt, n :
  --{{{  Declaraciones
  VAL BYTE cero IS 0 (BYTE):
  VAL R IS 0:
  VAL G IS 1:
  VAL B IS 2:
  INT rojo, verde, azul:
  INT i, g1, g2:
  [3]INT crece, max, decr:
  --}}}
  SEQ
    --{{{  Primer dato
    punt := 765
    --{{{  Calculo del primer dato
    n  := 1
    SEQ
      --{{{  Calculo de crece y decr
      SEQ j=0 FOR 3
        crece[j], max[j], decr[j] := 0, 0, 0
      i := (n-1) REM 85
      g1 := (n-1) / 85
      g2 := i / 43
      i := i REM 43
      crece[((g1+1) REM 3)] := 1 - g2
      decr [g1] := g2
      max[((g1+g2) REM 3)] := 1
      --}}}
      rojo  := ((((63*i)/42) * crece[R]) +
               ((63 - ((63*i)/43)) * decr[R] )) +
               (63 * max[R])
      verde := ((((63*i)/42) * crece[G]) +
               ((63 - ((63*i)/43)) * decr[G] )) +
               (63 * max[G])
      azul  := ((((63*i)/42) * crece[B]) +
               ((63 - ((63*i)/43)) * decr[B] )) +
               (63 * max[B])
      --{{{  Dato n = 1
      [tabla.colores FROM punt FOR 3] := [BYTE (rojo), BYTE (verde), BYTE (azu
      punt := punt - 3
      --}}}
    --}}}
    --}}}
    SEQ n=1 FOR 255
      --{{{  Resto de la tabla
      SEQ
        --{{{  Calculo de crece y decr
        SEQ j=0 FOR 3
          crece[j], max[j], decr[j] := 0, 0, 0
        i := (n-1) REM 85
        g1 := (n-1) / 85
        g2 := i / 43
        i := i REM 43
        crece[((g1+1) REM 3)] := 1 - g2
        decr [g1] := g2
        max[((g1+g2) REM 3)] := 1
        --}}}
        rojo  := ((((63*i)/42) * crece[R]) +
                 ((63 - ((63*i)/43)) * decr[R] )) +
                 (63 * max[R])
        verde := ((((63*i)/42) * crece[G]) +
                 ((63 - ((63*i)/43)) * decr[G] )) +
                 (63 * max[G])
        azul  := ((((63*i)/42) * crece[B]) +
                 ((63 - ((63*i)/43)) * decr[B] )) +
                 (63 * max[B])
        --{{{  Dato n
        [tabla.colores FROM punt FOR 3] := [BYTE (rojo), BYTE (verde), BYTE (
        punt := punt - 3
        --}}}
      --}}}
:
--}}}
--
--{{{  COMMENT PROC HaceOtroClut8 ( )
--:::A COMMENT FOLD
--{{{  PROC HaceOtroClut8 ( )
PROC HaceOtroClut8 ( )
  INT punt, n :
  --{{{  Declaraciones
  VAL BYTE cero IS 0 (BYTE):
  VAL R IS 0:
  VAL G IS 1:
  VAL B IS 2:
  INT rojo, verde, azul:
  INT i, g1, g2:
  [3]INT crece, max, decr:
  --}}}
  SEQ
    --{{{  Primer dato
    punt := 381
    n  := 1
    --{{{  Dato n = 1
    [tabla.colores FROM punt FOR 3] := [ cero, cero, cero ]
    punt := punt - 3
    --}}}
    --}}}
    SEQ n=1 FOR 127
      --{{{  Resto de la tabla
      SEQ
        --{{{  Calculo de crece y decr
        SEQ j=0 FOR 3
          crece[j], max[j], decr[j] := 0, 0, 0
        i := (n-1) REM 41
        g1 := (n-1) / 41
        g2 := i / 21
        i := i REM 21
        crece[((g1+1) REM 3)] := 1 - g2
        decr [g1] := g2
        max[((g1+g2) REM 3)] := 1
        --}}}
        rojo  := ((((31*i)/21) * crece[R]) +
                 ((31 - ((31*i)/21)) * decr[R] )) +
                 (31 * max[R])
        verde := ((((31*i)/21) * crece[G]) +
                 ((31 - ((31*i)/21)) * decr[G] )) +
                 (31 * max[G])
        azul  := ((((31*i)/21) * crece[B]) +
                 ((31 - ((31*i)/21)) * decr[B] )) +
                 (31 * max[B])
        --{{{  Dato n
        [tabla.colores FROM punt FOR 3] := [BYTE (rojo), BYTE (verde), BYTE (
        punt := punt - 3
```

```
    SEQ
      tiempo ? ahora
      tiempo ? AFTER (ahora PLUS 1)
      a := registrador.estado
      estado := INT32 (a)
:
--}}}
--
--}}}
--
--{{{  Procedimientos de Carga
--
--{{{  PROC LoadTable ([768]BYTE colours)
--{{{  Channel A ( con event )
PROC LoadTableA ([768]BYTE tabla.colores)
  SEQ
    event ? nada
    event ? nada
    pixel.mask := #FF
    pixel.addr.write := 0
    colour.val.block := tabla.colores
:
--}}}
--}}}
--
--{{{  PROC CargaClut8 ()
PROC CargaClut8 ()
  --{{{  Definiciones de constantes
  --{{{  Numeros en BYTEs
  VAL BYTE ff IS #FF (BYTE):
  VAL BYTE cero IS #00 (BYTE):
  VAL BYTE siete IS #07 (BYTE):
  VAL BYTE uno.cinco IS #15 (BYTE):
  VAL BYTE tres.uno IS #31 (BYTE):
  VAL BYTE cuatro.siete IS #47 (BYTE):
  VAL BYTE cinco.cinco IS #47 (BYTE):
  VAL BYTE seis.tres IS #63 (BYTE):
  --}}}
  --{{{  Tabla de mezclas
  VAL [][]BYTE tab IS [ [seis.tres, cero, cero],
   [seis.tres, cero, siete], [seis.tres, siete, siete],
   [seis.tres, cero, uno.cinco], [seis.tres, uno.cinco, uno.cinco],
   [seis.tres, cero, tres.uno], [seis.tres, tres.uno, tres.uno],
   [seis.tres, cero, cuatro.siete], [seis.tres, cuatro.siete, cuatro.siete],
   [seis.tres, cero, cinco.cinco], [seis.tres, cinco.cinco, cinco.cinco],
   [seis.tres, siete, uno.cinco], [seis.tres, siete, tres.uno],
   [seis.tres, siete, cuatro.siete], [seis.tres, siete, cinco.cinco],
   [seis.tres, uno.cinco, tres.uno], [seis.tres, uno.cinco, cuatro.siete],
   [seis.tres, uno.cinco, cinco.cinco], [seis.tres, tres.uno, cuatro.siete],
   [seis.tres, tres.uno, cinco.cinco], [seis.tres, cuatro.siete, cinco.cinco]
  --}}}
  --}}}
  INT punt:
  SEQ
    --{{{  Proc
    [tabla.colores FROM 765 FOR 3] := [cero, cero, cero]
    punt := 762
    SEQ i1=0 FOR 21
      VAL a IS tab[i1][0]:
      VAL b IS tab[i1][1]:
      VAL c IS tab[i1][2]:
      SEQ
        --{{{  Asignacion
        IF
          b <> c
            SEQ
              [tabla.colores FROM punt FOR 3] := [a, b, c]
              [tabla.colores FROM punt-3 FOR 3] := [a, c, b]
              [tabla.colores FROM punt-6 FOR 3] := [b, a, c]
              [tabla.colores FROM punt-9 FOR 3] := [c, a, b]
              [tabla.colores FROM punt-12 FOR 3] := [b, c, a]
              [tabla.colores FROM punt-15 FOR 3] := [c, b, a]
          b = c
            SEQ
              [tabla.colores FROM punt FOR 3] := [a, b, b]
              [tabla.colores FROM punt-3 FOR 3] := [a, a, b]
              [tabla.colores FROM punt-6 FOR 3] := [b, a, b]
              [tabla.colores FROM punt-9 FOR 3] := [b, a, a]
              [tabla.colores FROM punt-12 FOR 3] := [b, b, a]
              [tabla.colores FROM punt-15 FOR 3] := [a, b, a]
        --}}}
        punt := punt - 18
    [tabla.colores FROM punt FOR 3] := [seis.tres, seis.tres, seis.tres]
    SEQ i=0 FOR 384
      tabla.colores[i] := BYTE ((((INT (tabla.colores[(i+384)]))+7)/2)-1)
    LoadTableA (tabla.colores)
    --}}}
:
--}}}
--
--{{{  PROC CargaClut8Suave ()   -- Variacion suave de colores
PROC CargaClut8Suave ()
  INT punt:
  --{{{  Declaraciones
  VAL BYTE cero IS 0 (BYTE):
  VAL R IS 0:
  VAL G IS 1:
  VAL B IS 2:
  INT rojo, verde, azul:
  INT i, g1, g2:
  [3]INT crece, max, decr:
  --}}}
  SEQ
    --{{{  Primer dato
    punt := 765
    [tabla.colores FROM punt FOR 3] := [cero, cero, cero]
    punt := punt - 3
    --}}}
    SEQ n=1 FOR 255
      --{{{  Resto de la tabla
      SEQ
        --{{{  Calculo de crece y decr
        SEQ j=0 FOR 3
          crece[j], max[j], decr[j] := 0, 0, 0
        i := (n-1) REM 85
        g1 := (n-1) / 85
        g2 := i / 43
        i := i REM 43
        crece[((g1+1) REM 3)] := 1 - g2
        decr [g1] := g2
        max[((g1+g2) REM 3)] := 1
        --}}}
        rojo  := ((((63*i)/42) * crece[R]) +
                 ((63 - ((63*i)/43)) * decr[R] )) +
                 (63 * max[R])
        verde := ((((63*i)/42) * crece[G]) +
                 ((63 - ((63*i)/43)) * decr[G] )) +
                 (63 * max[G])
        azul  := ((((63*i)/42) * crece[B]) +
                 ((63 - ((63*i)/43)) * decr[B] )) +
                 (63 * max[B])
        --{{{  Dato n
```

```
        ()
        --}}}
    TRUE
      SEQ
        --{{{  CargaClut8 ( inicial  )
        Reiniciar ()
        CargaClut8 ()
        PonerModo (1)
        Activar()
        LeeEstado ( estado )
        --}}}
--}}}
WHILE activo
  SEQ
    --{{{  Proc
    Mp.Cp ? paleta
    IF
      paleta = cero
        --{{{  Proc
        SEQ
          CargaClut8 ()
        --}}}
      paleta = uno
        --{{{  Proc
        SEQ
          CargaClut8Suave ()
        --}}}
      paleta = dos
        --{{{  Proc
        SEQ
          CargaClut8 ()
          activo := FALSE
        --}}}
      paleta = tres
        SEQ
          --{{{  Proc
          HaceClut8Suave ( tabla.colores )
          WHILE activo.ayu
            PRI ALT
              Mp.Cp ? paleta.ayu
                SEQ
                  --{{{  Proc
                  IF
                    paleta.ayu = cero
                      CargaClut8 ()
                    paleta.ayu = uno
                      CargaClut8Suave ()
                    paleta.ayu = dos
                      --{{{  Proc
                      SEQ
                        CargaClut8 ()
                        activo := FALSE
                      --}}}
                    TRUE
                      CargaClut8 ()
                  activo.ayu := FALSE
                  --}}}
              TRUE & SKIP
                SEQ
                  --{{{  Proc
                  --CambiaCargaClut8SuaveA ( tabla.colores )
                  CambiaCargaClut8SuaveALento ( tabla.colores )
                  --}}}
          activo.ayu := TRUE
          --}}}
      paleta = cuatro
        SEQ
          --{{{  Proc
          WHILE activo.ayu
            PRI ALT
              Mp.Cp ? paleta.ayu
                SEQ
                  --{{{  Proc
                  IF
                    paleta.ayu = cero
                      CargaClut8 ()
                    paleta.ayu = uno
                      CargaClut8Suave ()
                    paleta.ayu = dos
                      --{{{  Proc
                      SEQ
                        CargaClut8 ()
                        activo := FALSE
                      --}}}
                    TRUE
                      CargaClut8 ()
                  activo.ayu := FALSE
                  --}}}
              TRUE & SKIP
                SEQ
                  --{{{  Proc
                  ComutaCargaClut8SuaveA ( )
                  --}}}
          activo.ayu := TRUE
          --}}}
      paleta = cinco
        SEQ
          --{{{  Proc
          CargaClut8 ()
          WHILE activo.ayu
            PRI ALT
              Mp.Cp ? paleta.ayu
                SEQ
                  --{{{  Proc
                  IF
                    paleta.ayu = cero
                      CargaClut8 ()
                    paleta.ayu = uno
                      CargaClut8Suave ()
                    paleta.ayu = dos
                      --{{{  Proc
                      SEQ
                        CargaClut8 ()
                        activo := FALSE
                      --}}}
                    TRUE
                      CargaClut8 ()
                  activo.ayu := FALSE
                  --}}}
              TRUE & SKIP
                SEQ
                  --{{{  Proc
                  CambiaCargaClut8SuaveALento ( tabla.colores )
                  --}}}
          activo.ayu := TRUE
          --}}}
      TRUE
        --{{{  Proc
        SEQ
          CargaClut8 ()
        --}}}
    --}}}
--}}}
```

```
      --}}}
    --}}}
    [tabla.colores FROM 384 FOR 384] := [tabla.colores FROM 0 FOR 384]
:
--}}}
--}}}
--
--{{{  PROC CambiaCargaClut8SuaveA ([768]BYTE tabla.colores)(con event)
PROC CambiaCargaClut8SuaveA ([768]BYTE tabla.colores)
  --{{{  Declaraciones/Direcciones
  BYTE color.ayu, color.ayu1, color.ayu2 :
  --}}}
  SEQ
    --{{{  Proc
    color.ayu  := tabla.colores[0]
    color.ayu1 := tabla.colores[1]
    color.ayu2 := tabla.colores[2]
    [tabla.colores FROM 0 FOR 765] := [tabla.colores FROM 3 FOR 765]
    tabla.colores[765] := color.ayu
    tabla.colores[766] := color.ayu1
    tabla.colores[767] := color.ayu2
    event ? nada
    event ? nada
    pixel.mask := #FF
    pixel.addr.write := 0
    colour.val.block := tabla.colores
    --}}}
:
--}}}
--
--{{{  COMMENT PROC CambiaCargaClut8SuaveALento ([]BYTE tabla.colores)(event/l
--:::A COMMENT FOLD
--{{{  PROC CambiaCargaClut8SuaveALento ([]BYTE tabla.colores)(event/lento)
PROC CambiaCargaClut8SuaveALento ([768]BYTE tabla.colores)
  --{{{  Declaraciones/Direcciones
  TIMER reloj:
  BYTE color.ayu, color.ayu1, color.ayu2 :
  INT retraso, ahora:
  --}}}
  SEQ
    --{{{  Proc
    retraso := 1800
    reloj ? ahora
    reloj ? AFTER ahora PLUS retraso
    color.ayu  := tabla.colores[0]
    color.ayu1 := tabla.colores[1]
    color.ayu2 := tabla.colores[2]
    [tabla.colores FROM 0 FOR 765] := [tabla.colores FROM 3 FOR 765]
    tabla.colores[765] := color.ayu
    tabla.colores[766] := color.ayu1
    tabla.colores[767] := color.ayu2
    event ? nada
    event ? nada
    pixel.mask := #FF
    pixel.addr.write := 0
    colour.val.block := tabla.colores
    --}}}
:
--}}}
--}}}
--
--{{{  PROC ComutaCargaClut8SuaveA ( )(con event)
PROC ComutaCargaClut8SuaveA ( )
  SEQ
    --{{{  Proc
    retraso := 1200
    [tabla.ayu FROM 0 FOR 768]     := [tabla.colores FROM 0 FOR 768]
    SEQ i=0 FOR 256
      [tabla.colores FROM (3*i) FOR 3] :=
                          [tabla.ayu FROM (768-(3*(i+1))) FOR 3]
    event ? nada
    event ? nada
    pixel.mask := #FF
    pixel.addr.write := 0
    colour.val.block := tabla.colores
    tiempo ? ahora
    tiempo ? AFTER ( ahora  PLUS retraso )
    --}}}
:
--}}}
--
--
--}}}
--
--}}}
--
--
--
-- -- Programa de Prueba --
--
--{{{  COMMENT Programa Principal
--:::A COMMENT FOLD
--{{{  Programa Principal
--{{{  Version normal 8 bits/pixel
--{{{  Declaraciones/Asignaciones
--{{{  VAL
VAL  cero   IS   0 (INT32) :
VAL  uno    IS   1 (INT32) :
VAL  dos    IS   2 (INT32) :
VAL  tres   IS   3 (INT32) :
VAL  cuatro IS   4 (INT32) :
VAL  cinco  IS   5 (INT32) :
--}}}
--{{{  INT
INT32 num, estado :
INT32 paleta, paleta.ayu :
--}}}
BOOL activo, activo.ayu :
--}}}
SEQ
  activo     := TRUE
  activo.ayu := TRUE
  --{{{  Carga Paleta Inicial
  Mp.Cp ? paleta
  IF
    paleta = cero
      SEQ
        --{{{  CargaClut8 ( inicial  )
        Reiniciar ()
        CargaClut8 ()
        PonerModo (1)
        Activar()
        LeeEstado ( estado )
        --}}}
    paleta = uno
      SEQ
        --{{{  CargaClut8Suave ( inicial  )
        Reiniciar ()
        CargaClut8Suave ()
        PonerModo (1)
        Activar()
        LeeEstado ( estado )
```

```
              activo.ayu := TRUE
              --}}}
            TRUE
              --{{{  Proc
              SEQ
                CargaClut8 ()
              --}}}
        --}}}
    --}}}
    --}}}
    --
    --
    --}}}
:
--}}}F
--...F code
--:::A  1 2
--:::F Contro27.dcd
--...F descriptor
--:::A  1 4
--:::F Contro27.dds
--...F debug
--:::A  1 5
--:::F Contro27.ddb
--...F CODE SC Operaciones Graficas - Controle de la Pantalla en el B409
--:::A  2 10
--:::F Contro27.csc
--}}}
--}}}
--
PLACED PAR
  PROCESSOR 0 T8
    --{{{  Definiciones MemoriaPantalla
    PLACE Org.Mp          AT link.e[1]:
    PLACE Mp.Org          AT link.s[1]:
    PLACE Mp.Cp           AT link.s[2]:
    PLACE Cp.Mp           AT link.e[2]:
    --}}}
    MemoriaPantalla (Org.Mp, Mp.Org, Mp.Cp, Cp.Mp)
  PROCESSOR 1 T2
    --{{{  Definiciones ControlePantalla
    PLACE Mp.Cp           AT link.e[1]:
    PLACE Cp.Mp           AT link.s[1]:
    PLACE Org.Cp          AT link.e[2]:
    PLACE Cp.Org          AT link.s[2]:
    --}}}
    ControlPantalla (Mp.Cp, Cp.Mp, Org.Cp, Cp.Org)
--
--}}}F
--...F code
--:::A  1 2
--:::F Config27.dcd
--...F descriptor
--:::A  1 4
--:::F Config27.dds
--}}}
--{{{  EXE Grafic01 - Organizacion de las Operaciones Graficas
--:::A  3 13
--{{{F Organizador
--:::F Organi28.tsr
--
-- ------ Programa de Ativacion de las Rutinas Graficas ------
--
--{{{  Configuracion
--
--{{{  Definiciones de la configuracion
--
--{{{  Definicion de canales fisicos
VAL link.e IS [4, 5, 6, 7]:
VAL link.s IS [0, 1, 2, 3]:
--}}}
--
--{{{  Definicion de canales (Organizador y MemoriaPantalla )
--
CHAN OF ANY Org.Mp, Mp.Org, Org.Cp, Cp.Org :
--
--}}}
--
--}}}
--
PLACE Org.Mp          AT link.s[2]:
PLACE Mp.Org          AT link.e[2]:
PLACE Org.Cp          AT link.s[1]:
PLACE Cp.Org          AT link.e[1]:
--
--}}}
--
--{{{  Declaracion de Librerias Usadas
#USE userio
#USE interf
#USE userhdr
#USE filerhdr
#USE krnlhdr
#USE msdos
#USE extrio
#USE ioconv
#USE strings
#USE uservals
--}}}
--
--{{{  Declaracion de Constantes
--{{{  Parametros
VAL  retorno          IS   INT('*c')    :
VAL  espacio          IS   INT('*s')    :
VAL  nueva.linea      IS   INT('*n')    :
VAL  ancho.max.ent    IS        1024    :
VAL  alto.max.ent     IS         768    :
--}}}
--{{{  Para LLamada de Rutinas Graficas
--{{{  De a Hasta z
VAL  t.a              IS   INT('a')     :
VAL  t.borrar         IS   INT('b')     :
VAL  t.color          IS   INT('c')     :
VAL  t.d              IS   INT('d')     :
VAL  t.e              IS   INT('e')     :
VAL  t.fin            IS   INT('f')     :
VAL  t.g              IS   INT('g')     :
VAL  t.h              IS   INT('h')     :
VAL  t.iniciar        IS   INT('i')     :
VAL  t.ejemplos       IS   INT('j')     :
VAL  t.k              IS   INT('k')     :
VAL  t.linea          IS   INT('l')     :
VAL  t.m              IS   INT('m')     :
VAL  t.n              IS   INT('n')     :
VAL  t.otras          IS   INT('o')     :
VAL  t.p              IS   INT('p')     :
VAL  t.q              IS   INT('q')     :
VAL  t.rectang        IS   INT('r')     :
VAL  t.s              IS   INT('s')     :
VAL  t.t              IS   INT('t')     :
VAL  t.circulares     IS   INT('u')     :
VAL  t.ventanas       IS   INT('v')     :
```

```
--}}}
--}}}
--{{{  Programa Principal
--{{{  Version normal 8 bits/pixel
--{{{  Declaraciones/Asignaciones
--{{{  VAL
VAL  cero   IS  0 (INT32) :
VAL  uno    IS  1 (INT32) :
VAL  dos    IS  2 (INT32) :
VAL  tres   IS  3 (INT32) :
VAL  cuatro IS  4 (INT32) :
--}}}
--{{{  INT
INT32 num, estado :
INT32 paleta, paleta.ayu :
--}}}
BOOL activo, activo.ayu :
--}}}
SEQ
  activo     := TRUE
  activo.ayu := TRUE
  --{{{  Carga Paleta Inicial
  Mp.Cp ? paleta
  IF
    paleta = cero
      SEQ
        --{{{  CargaClut8 ( inicial  )
        Reiniciar ()
        CargaClut8 ()
        PonerModo (1)
        Activar()
        LeeEstado ( estado )
        --}}}
    paleta = uno
      SEQ
        --{{{  CargaClut8Suave ( inicial  )
        Reiniciar ()
        CargaClut8Suave ()
        PonerModo (1)
        Activar()
        LeeEstado ( estado )
        --}}}
    TRUE
      SEQ
        --{{{  CargaClut8 ( inicial  )
        Reiniciar ()
        CargaClut8 ()
        PonerModo (1)
        Activar()
        LeeEstado ( estado )
        --}}}
  --}}}
  WHILE activo
    SEQ
      --{{{  Proc
      Mp.Cp ? paleta
      IF
        paleta = cero
          --{{{  Proc
          SEQ
            CargaClut8 ()
          --}}}
        paleta = uno
          --{{{  Proc
          SEQ
            CargaClut8Suave ()
          --}}}
        paleta = dos
          --{{{  Proc
          SEQ
            CargaClut8 ()
            activo := FALSE
          --}}}
        paleta = tres
          SEQ
            --{{{  Proc
            HaceClut8Suave ( tabla.colores )
            WHILE activo.ayu
              PRI ALT
                Mp.Cp ? paleta.ayu
                  SEQ
                    --{{{  Proc
                    IF
                      paleta.ayu = cero
                        CargaClut8 ()
                      paleta.ayu = uno
                        CargaClut8Suave ()
                      paleta.ayu = dos
                        --{{{  Proc
                        SEQ
                          CargaClut8 ()
                          activo := FALSE
                        --}}}
                      TRUE
                        CargaClut8 ()
                    activo.ayu := FALSE
                    --}}}
                TRUE & SKIP
                  SEQ
                    --{{{  Proc
                    CambiaCargaClut8SuaveA ( tabla.colores )
                    --}}}
            activo.ayu := TRUE
            --}}}
        paleta = cuatro
          SEQ
            --{{{  Proc
            WHILE activo.ayu
              PRI ALT
                Mp.Cp ? paleta.ayu
                  SEQ
                    --{{{  Proc
                    IF
                      paleta.ayu = cero
                        CargaClut8 ()
                      paleta.ayu = uno
                        CargaClut8Suave ()
                      paleta.ayu = dos
                        --{{{  Proc
                        SEQ
                          CargaClut8 ()
                          activo := FALSE
                        --}}}
                      TRUE
                        CargaClut8 ()
                    activo.ayu := FALSE
                    --}}}
                TRUE & SKIP
                  SEQ
                    --{{{  Proc
                    ComutaCargaClut8SuaveA ( )
                    --}}}
```

```
    keyboard ? caracter
  :
  --}}}
  --
  --{{{  Teclear Mandato
  PROC TeclearMandato ()
    SEQ
      --{{{  Procesos
      --{{{  Mandato.antes.teclear
      SalidaCadena ( "*C*N" )
      SalidaCadena ( " Aguardando.teclear.mandato ")
      --}}}
      EntradaCaracter ()
      tipo := caracter
      --{{{  Mandato.despues.teclear
      SalidaCadena ( "*S*S" )
      SalidaCadena ( " Despues.teclear.mandato ")
      SalidaCadena ( "*C*N" )
      --}}}
      --}}}
  :
  --}}}
  --
  --{{{  Teclear Parada
  PROC TeclearParada ( VAL []BYTE cadena )
    INT parada :
    SEQ
      --{{{  Procesos
      --{{{  Pantalla Inicial
      SalidaCadena ( "*C*N" )
      SalidaCadena ( " Teclee p Para Parar El Proceso de " )
      SalidaCadena ( "*C*N" )
      --}}}
      SEQ i=0 FOR SIZE cadena
        write.char ( screen, cadena[i] )
      --{{{  Parada.antes.teclear
      SalidaCadena ( "*C*N" )
      SalidaCadena ( " Aguardando.teclear.parada ")
      --}}}
      EntradaCaracter ()
      parada := caracter
      Org.Mp ! parada
      --{{{  Parada.despues.teclear
      SalidaCadena ( "*S*S" )
      SalidaCadena ( " Despues.teclear.parada ")
      SalidaCadena ( "*C*N" )
      --}}}
      --}}}
  :
  --}}}
  --
  --}}}
  --
  --{{{  Generar Tablas/Pantallas
  --
  --{{{  Pantalla de Numeros
  PROC PantallaCogerNumero ( VAL []BYTE cadena )
    --{{{  Procesos
    SEQ
      SalidaCadena ( "*C*N*N" )
      SalidaCadena ( " Teclee " )
      SEQ i=0 FOR SIZE cadena
        write.char ( screen, cadena[i] )
      SalidaCadena ( " y Despues Teclee " )
      SalidaCadena ( "*C*N" )
      SalidaCadena ( " Introducir Para El Ordenador Aceptar El Numero " )
      SalidaCadena ( "*C*N*N" )
      SEQ i=0 FOR SIZE cadena
        write.char ( screen, cadena[i] )
      SalidaCadena ( "= " )
    --}}}
  :
  --}}}
  --
  --{{{  Pantalla de Parametros
  PROC PantallaCogerParametros ( VAL INT n )
    SEQ i=0 FOR n
      SEQ
        --{{{  Proc
        SalidaCadena ( "*C*N*N" )
        SalidaCadena ( " Teclee Parametro[" )
        SalidaEntero ( i )
        SalidaCadena ( "] y Despues Teclee " )
        SalidaCadena ( "*C*N" )
        SalidaCadena ( " Introducir Para El Ordenador Aceptar El Numero " )
        SalidaCadena ( "*C*N*N" )
        SalidaCadena ( " Parametro[" )
        SalidaEntero ( i )
        SalidaCadena ( "] =" )
        caracter := INT(' ')
        read.echo.int ( keyboard, screen, numero.ent, caracter )
        parametros[i] := numero.ent
        --}}}
  :
  --}}}
  --
  --{{{  Cabecera Inicial
  PROC CabeceraInicial ()
    SEQ
      --{{{  Procesos
      SalidaCadena ( "*C*N" )
      SalidaCadena ( "  Teclee uno Caracter Para Elegir una Rutina Grafica" )
      SalidaCadena ( "*C*N" )
      SalidaCadena ( "de Acuerdo con la Seguiente Tabla:" )
      SalidaCadena ( "*C*N" )
      --}}}
  :
  --}}}
  --
  --{{{  Mandatos
  --
  --{{{  Mandatos
  PROC MandatosDisponibles ()
    SEQ
      --{{{  Procesos
      --{{{  Fundamentales
      SalidaCadena ( "*C*N" )
      --{{{  Iniciales
      SalidaCadena ( " Borrar la Pantalla              : b " )
      SalidaCadena ( "*S*S*S*S" )
      SalidaCadena ( " Iniciar Pantalla (x.max, y.max): x " )
      SalidaCadena ( "*C*N" )
      SalidaCadena ( " Assignar Color                  : c " )
      SalidaCadena ( "*S*S*S*S" )
      SalidaCadena ( " Iniciar Pantalla ( 1024 X 768 ): i " )
      SalidaCadena ( "*C*N" )
      SalidaCadena ( " Camb. Paleta (Pad, Suav, Fijar): y " )
      SalidaCadena ( "*C*N" )
      --}}}
      --{{{  Que Dibujan
      --{{{  Rectilineas
      SalidaCadena ( " Dibujar Tira Linea              : l " )
```

```
VAL  t.w              IS   INT('w')     :
VAL  t.x              IS   INT('x')     :
VAL  t.y              IS   INT('y')     :
VAL  t.z              IS   INT('z')     :
--}}}
--{{{  De A Hasta Z
VAL  t.A              IS   INT('A')     :
--}}}
--{{{  De 0 Hasta 9
VAL  t.cero           IS   INT('0')     :
VAL  t.uno            IS   INT('1')     :
VAL  t.dos            IS   INT('2')     :
VAL  t.tres           IS   INT('3')     :
VAL  t.cuatro         IS   INT('4')     :
VAL  t.cinco          IS   INT('5')     :
VAL  t.seis           IS   INT('6')     :
VAL  t.siete          IS   INT('7')     :
VAL  t.ocho           IS   INT('8')     :
VAL  t.nueve          IS   INT('9')     :
--}}}
--}}}
--{{{  Para Ficheros
VAL    va.bien              IS          fi.ok   :
VAL    fin.de.fich.ent      IS          fi.eof  :
VAL    signal.de.restar     IS          '-'     :
--}}}
--}}}
--
--{{{  Declaracion de Variables y sus Direciones
--
--{{{  Variables
--
--{{{  Variables Enteras
--{{{  Enteros Para Ficheros
INT numero.bytes.fich :
INT num.carac, fuente.x, fuente.y :
INT long, resultado :
INT long.bloque :
INT longitud.fuente :
INT terminador :
INT x.mensaje, y.mensaje :
INT ancho.esc, alto.esc :
--}}}
INT a.ver, paleta, paleta.ant, paleta.trace :
INT caracter :
INT destino, origen, longitud :
INT no.borra.pantalla :
INT numero.ent :
INT tamano, color.elegido.ent :
INT tipo :
--}}}
--
--{{{  Variables Logicas
--{{{  Para Ficheros
BOOL existe, fich.abierto :
BOOL fin.de.fichero :
BOOL no.leer :
BOOL error.de.lectura :
BOOL ha.leido.fichero :
--}}}
BOOL activo :
BOOL negativo :
--}}}
--
--{{{  Variables Reales
--}}}
--
--{{{  Variables Tipo Vector o Array de Enteros
[256][12]INT fuente.elegido.ent :
[100]INT parametros :
[80]INT cadena.mensaje.ent :
--}}}
--
--{{{  Variables Tipo Vector o Array de Bytes
--{{{  Para Ficheros
[256][12]BYTE fuente.elegido.by :
[80]BYTE cadena.mensaje.by :
[abs.id.size]BYTE nombre :
[12]BYTE nomb.fich.plie :
[max.record.size]BYTE bloque :
--}}}
--}}}
--
--}}}
--
--}}}
--
--{{{  Declaracion de Procedimientos
--
--{{{  Entrada/Salida Pantalla/Teclado
--
--{{{  Salida Pantalla
--
--{{{  Salida de Caracteres Para Pantalla
PROC SalidaCaracter ( VAL INT caracter )
  write.char ( screen, BYTE(caracter) )
:
--}}}
--
--{{{  Salida de Enteros Para Pantalla
PROC SalidaEntero ( VAL INT entero )
  write.int ( screen, INT(entero), 0 )
:
--}}}
--
--{{{  Salida de Mensagem Para Pantalla
PROC SalidaCadena ( VAL []BYTE mensaje )
  SEQ i=0 FOR SIZE mensaje
    write.char ( screen, mensaje[i] )
:
--}}}
--
--{{{  Volver Para TDS2 sin Borrar Pantalla
PROC VolverTds2 ()
  --{{{  Procesos
  SEQ
    SalidaCadena ( "*C*N*N" )
    SalidaCadena ( " Teclee una Tecla Cualquiera Para Volver a TDS2 " )
    SalidaCadena ( "*C*N*N" )
    SalidaCadena ( "                        Gracias                        " )
    SalidaCadena ( "*C*N" )
    keyboard ? no.borra.pantalla
  --}}}
:
--}}}
--
--}}}
--
--{{{  Entrada Teclado
--
--{{{  Entrada de Caracteres por Teclado
PROC EntradaCaracter ( )
```

```
    SalidaCadena ( " e, Para Esto, Hay Que Teclear " )
    SEQ i=0 FOR SIZE cadena2
      write.char ( screen, cadena2[i] )
    SalidaCadena ( " que es el numero de " )
    SalidaCadena ( "*C*N" )
    SalidaCadena ( " parametros que va Utilizar Para Hacer la Prueba. " )
    --}}}
:
--}}}
--
--{{{  Distancia Entre Tablas
PROC DistanciaEntreTablas ( VAL INT n )
  SEQ
    --{{{  Procesos
    SalidaCadena ( "*C*N" )
    SEQ i=0 FOR n
      SalidaCadena ( "*N" )
    --}}}
:
--}}}
--
--}}}
--
--}}}
--
--{{{  Ficheros/Mensajes
--
--{{{  Procedimientos Auxiliares
PROC LeeBloque ()
  SEQ
    --{{{  Procesos
    read.tkf.block (from.filer, to.filer, long.bloque, bloque, resultado)
    --{{{  Prueba de fin de Fichero
    IF
      resultado = fin.de.fich.ent
        fin.de.fichero := TRUE
      TRUE
        SKIP
    --}}}
    --}}}
:
--}}}
--
--{{{  Lectura de Ficheros
--
--{{{  Lectura de Fich. con Nombre Fijo (fuente de caracteres via "pantalla")
PROC LeeFicheroFuenteNombreFijo ( []BYTE nombre.fichero, VAL INT tamano,
                                  VAL BOOL otra.vez, CHAN OF ANY pantalla )
  --{{{  Declaraciones
  BOOL repetir :
  INT cont.hoja :
  INT i.ayu, j.ayu :
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Verificacion del nombre del fichero y apertura del mismo
    --{{{  Asignaciones
    cont.hoja          :=    0
    i.ayu              :=    0
    j.ayu              :=    0
    numero.bytes.fich  :=    0
    fich.abierto       :=    FALSE
    fin.de.fichero     :=    FALSE
    no.leer            :=    FALSE
    existe             :=    FALSE
    error.de.lectura   :=    FALSE
    long               :=    tamano
    repetir            :=    TRUE
    --}}}
    WHILE ( ( NOT existe ) AND repetir )
      SEQ
        newline (pantalla)
        test.exists (from.filer, to.filer, long, nombre.fichero, existe)
        IF
          existe
            SEQ
              --{{{  Apertura del Fichero
              open.tkf.file (from.filer, to.filer, tkf.open.read, long,
                             nombre.fichero, resultado)
              --}}}
              --{{{  Indica el resultado de la apertura
              IF
                resultado = va.bien
                  SEQ
                    fich.abierto := TRUE
                    write.full.string (pantalla, "Leyendo el fichero")
                    newline (pantalla)
                    no.leer := FALSE
                TRUE
                  SEQ
                    write.full.string (pantalla, "Error en la apertura del fiche
                    newline (pantalla)
                    write.full.string (pantalla, "Resultado: ")
                    write.int (pantalla, resultado, 0)
                    newline (pantalla)
                    error.de.lectura := FALSE
                    --{{{  Cierra el fichero e indica el resultado
                    close.tkf.file (from.filer, to.filer, resultado)
                    IF
                      resultado = va.bien
                        SEQ
                          write.full.string (pantalla, "*C*NCierre correcto de "
                          write.full.string (pantalla, [nombre.fichero FROM 0 FO
                          fich.abierto := FALSE
                          newline (pantalla)
                      TRUE
                        --{{{  Escribe el resultado en la pantalla
                        SEQ
                          write.full.string (pantalla, "ERROR en el cierre de ")
                          write.full.string (pantalla, [nombre.fichero FROM 0 FO
                          newline (pantalla)
                          write.full.string (pantalla, "Resultado : ")
                          write.int (pantalla, resultado, 0)
                          newline (pantalla)
                        --}}}
                    --}}}
                    no.leer := TRUE
              --}}}
          NOT existe
            SEQ
              write.full.string (pantalla, "ERROR: el fichero no existe")
              newline (pantalla)
              no.leer  :=  TRUE
              repetir  :=  otra.vez
    --}}}
    --{{{  Datos del Fuente, a sacar de un fichero
    IF
      NOT no.leer
        SEQ
          --{{{  Entrada de los datos
          LeeBloque ()
```

```
    SalidaCadena ( "*S*S*S*S" )
    SalidaCadena ( " Dibujar Rectangulo              : r " )
    SalidaCadena ( "*C*N" )
    SalidaCadena ( " Flechas                         : k " )
    SalidaCadena ( "*S*S*S*S" )
    SalidaCadena ( " Paralelogramos                  : m " )
    SalidaCadena ( "*C*N" )
    SalidaCadena ( " Poligonos                       : n " )
    SalidaCadena ( "*S*S*S*S" )
    SalidaCadena ( " Rectangulos Lentos              : p " )
    SalidaCadena ( "*C*N" )
    --}}}
    --{{{  Curvilineas
    --{{{  Circulos
    SalidaCadena ( " Dibujar Circulos                : u " )
    SalidaCadena ( "*S*S*S*S" )
    SalidaCadena ( " Semicirculos                    : 2 " )
    SalidaCadena ( "*C*N" )
    SalidaCadena ( " Cuarto de Circulo               : 3 " )
    SalidaCadena ( "*S*S*S*S" )
    --}}}
    --{{{  Circunferencias
    SalidaCadena ( " Circunferencias                 : a " )
    SalidaCadena ( "*C*N" )
    SalidaCadena ( " Semicircunferencias             : 5 " )
    SalidaCadena ( "*S*S*S*S" )
    SalidaCadena ( " Cuarto de Circunferencia        : 6 " )
    SalidaCadena ( "*C*N" )
    --}}}
    SalidaCadena ( " Elipses                         : d " )
    SalidaCadena ( "*S*S*S*S" )
    SalidaCadena ( " Arcos                           : e " )
    SalidaCadena ( "*C*N" )
    SalidaCadena ( " Espiral sin Contorno            : 4 " )
    SalidaCadena ( "*S*S*S*S" )
    SalidaCadena ( " Espiral con Contorno            : 7 " )
    SalidaCadena ( "*C*N" )
    SalidaCadena ( " Espiral con dos Colores         : 9 " )
    SalidaCadena ( "*S*S*S*S" )
    SalidaCadena ( " Coronas                         : g " )
    SalidaCadena ( "*C*N" )
    SalidaCadena ( " Dibujar Circulos Lentos         : h " )
    SalidaCadena ( "*C*N" )
    --}}}
    --}}}
    --{{{  Zoom de Pantalla
    SalidaCadena ( " Zoom                            : z " )
    SalidaCadena ( "*C*N" )
    --}}}
    --}}}
    --{{{  Caracteres/Ficheros
    SalidaCadena ( " Cargar Fuente de Caracteres     : o " )
    SalidaCadena ( "*S*S*S*S" )
    SalidaCadena ( " Poner Fichero ASCII en Pant.    : q " )
    SalidaCadena ( "*C*N" )
    SalidaCadena ( " Escribir Cadena ASCII en Pant. : s " )
    SalidaCadena ( "*S*S*S*S" )
    SalidaCadena ( " Escr. Cad. ASCII Carac. Doble   : t " )
    SalidaCadena ( "*C*N" )
    SalidaCadena ( " Escr. Cad. ASCII Carac. Varia. : w " )
    SalidaCadena ( "*C*N" )
    --}}}
    --{{{  Ejemplos
    SalidaCadena ( " Ventanas                        : v " )
    SalidaCadena ( "*S*S*S*S" )
    SalidaCadena ( " Circulos Cambiando xo, yo, r    : 0 " )
    SalidaCadena ( "*C*N" )
    SalidaCadena ( " Cuadrados y Circulos            : 1 " )
    SalidaCadena ( "*S*S*S*S" )
    SalidaCadena ( " Cuadrados y Circulos Lentos     : j " )
    SalidaCadena ( "*C*N" )
    SalidaCadena ( " Mandelbroot/Julia               : A " )
    SalidaCadena ( "*C*N" )
    --}}}
    --{{{  Rutina en Prueba
    SalidaCadena ( " Rutina en Prueba                : 8 " )
    SalidaCadena ( "*S*S*S*S" )
    --}}}
    --{{{  Salir
    SalidaCadena ( " Terminar Programa               : f " )
    SalidaCadena ( "*C*N" )
    --}}}
    --}}}
  :
  --}}}
  --
  --{{{  Pantalla de Mandatos Simples
  PROC PantallaMandatos ( VAL []BYTE cadena )
    SEQ
      --{{{  Procesos
      SalidaCadena ( "*C*N" )
      SalidaCadena ( " Ha Elegido " )
      SEQ i=0 FOR SIZE cadena
        write.char ( screen, cadena[i] )
      SalidaCadena ( ", " )
      SalidaCadena ( "*C*N" )
      SalidaCadena ( " y Para Esto, " )
      IF
        longitud = 0
          SalidaCadena ( "No" )
        longitud <> 0
          SalidaCadena ( " " )
      SalidaCadena ( " Hay Que Teclear Parametros : " )
      --}}}
  :
  --}}}
  --
  --}}}
  --
  --{{{  Pantalla de No Implementado Todavia
  PROC PantallaNoImplementadoTodavia ( VAL []BYTE cadena )
    SEQ
      --{{{  Procesos
      SalidaCadena ( "*C*N" )
      SalidaCadena ( "      Lo   Siento,   Pero   . . . " )
      SalidaCadena ( "*C*N" )
      SEQ i=0 FOR SIZE cadena
        write.char ( screen, cadena[i] )
      SalidaCadena ( "  No Esta Implementado Todavia. " )
      --}}}
  :
  --}}}
  --
  --{{{  Pantalla Para Elegir Longitud Variable
  PROC PantallaElegirLongitud ( VAL []BYTE cadena1, VAL []BYTE cadena2 )
    SEQ
      --{{{  Procesos
      SalidaCadena ( "*C*N*N" )
      SalidaCadena ( " Ha Elegido " )
      SEQ i=0 FOR SIZE cadena1
        write.char ( screen, cadena1[i] )
      SalidaCadena ( "*C*N" )
```

```
                write.int ( pantalla, ent.ayu, 0 )
                write.full.string ( pantalla, "*S*S*S*S*S" )
                --}}}
            TRUE
              SKIP
          --}}}
      TRUE
        SKIP
    --{{{  Cargar Paleta Inicial
    paleta      :=  ent.ayu
    Org.Mp ! paleta
    paleta.ant := paleta
    --}}}
    --}}}
    --{{{  Lectura/Envio del Fuente de Caracteres Inicial
    read.text.line (fich.e, lon, linea, dato)
    IF
      ( dato >= 0 ) AND ( linea[0] <> signal.de.restar )
        SEQ
          --{{{  Proc
          write.full.string ( pantalla, "*C*NNombre Fichero Fuente = " )
          SEQ i=0 FOR ( lon - 1 )
            write.char ( pantalla, linea[i] )
          write.full.string ( pantalla, "*C*N" )
          --}}}
      ( dato >= 0 ) AND ( linea[0] = signal.de.restar )
        SEQ
          --{{{  Proc
          read.text.line (fich.e, lon, linea, dato)
          IF
            ( dato >= 0 ) AND ( linea[0] <> signal.de.restar )
              SEQ
                --{{{  Proc
                write.full.string ( pantalla, "Nombre Fichero Fuente = " )
                SEQ i=0 FOR ( lon - 1 )
                  write.char ( pantalla, linea[i] )
                write.full.string ( pantalla, "*C*N" )
                --}}}
            TRUE
              SKIP
          --}}}
      TRUE
        SKIP
    --{{{  Cargar Fuente de Caracteres
    longitud                  :=    4
    LeeFicheroFuenteNombreFijo ( linea, ( lon - 1 ), FALSE,
                                 pantalla )
    IF
      ha.leido.fichero = TRUE
        SEQ
          --{{{  Procesos
          --{{{  Coger Numeros
          fuente.y          := 12
          num.carac         := 256
          longitud.fuente   := num.carac * fuente.y
          parametros[0]     := longitud.fuente
          parametros[1]     := 256
          parametros[2]     := 8
          parametros[3]     := 12
          --}}}
          --{{{  Envia Cabecera .
          Org.Mp !  origen; destino; tipo; longitud
          --}}}
          --{{{  Envia Parametros
          Org.Mp !  [ parametros FROM 0  FOR longitud ]
          --}}}
          --{{{  Envia Fichero de Fuente
          SEQ i=0 FOR num.carac
            SEQ j=0 FOR fuente.y
              fuente.elegido.ent[i][j] := INT(fuente.elegido.by[i][j])
          Org.Mp !  [ fuente.elegido.ent FROM 0 FOR longitud.fuente ]
          --}}}
          ha.leido.fichero := FALSE
          --}}}
        TRUE
          SEQ
            --{{{  Procesos
            --{{{  Coger Numeros
            fuente.y          := 12
            num.carac         := 256
            longitud.fuente   := 0
            parametros[0]     := longitud.fuente
            parametros[1]     := 256
            parametros[2]     := 8
            parametros[3]     := 12
            --}}}
            --{{{  Envia Cabecera
            Org.Mp !  origen; destino; tipo; longitud
            --}}}
            --{{{  Envia Parametros
            Org.Mp !  [ parametros FROM 0  FOR longitud ]
            --}}}
            --}}}
      --}}}
      --}}}
      --{{{  Detecta Fin de Pliegue
      read.text.line (fich.e, lon, linea, dato)
      IF
        ( dato <= 0 )
          SEQ
            write.full.string ( pantalla, "Fin de Lectura de " )
            write.full.string ( pantalla, "Pliegue de Parametros Iniciales" )
        TRUE
          SKIP
      --}}}
      --}}}
    --}}}
    --{{{  Lectura en un fichero
    --{{{  Declaraciones
    INT res:
    --}}}
    keystream.from.file (de.uf[0], a.uf[0], fich.e, 0, res)
    --}}}
  --}}}
:
--}}}
--
--{{{  Lectura de Fich. Bin. con Nombre Fijo (Fuente de Caracteres via "screen")
PROC LeeFicheroFuenteCaracterFijo ( []BYTE nombre.fichero, VAL INT tamano,
                                    VAL BOOL otra.vez )
  --{{{  Declaraciones
  BOOL repetir :
  INT cont.hoja :
  INT i.ayu, j.ayu :
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Verificacion del nombre del fichero y apertura del mismo
    --{{{  Asignaciones
    cont.hoja           :=    0
    i.ayu               :=    0
    j.ayu               :=    0
```

```
        numero.bytes.fich := numero.bytes.fich + long.bloque
        SEQ i=0 FOR long.bloque
          --{{{  Actualiza Matriz de Fuente
          SEQ
            fuente.elegido.by[i.ayu][j.ayu] := bloque[i]
            j.ayu := j.ayu + 1
            IF
              j.ayu >= 12
                SEQ
                  j.ayu := 0
                  i.ayu := i.ayu + 1
              TRUE
                SKIP
          --}}}
        WHILE NOT fin.de.fichero
          SEQ
            --{{{  Continua Lectura
            LeeBloque ()
            numero.bytes.fich := numero.bytes.fich + long.bloque
            SEQ i=0 FOR long.bloque
              --{{{  Actualiza Matriz de Fuente
              SEQ
                fuente.elegido.by[i.ayu][j.ayu] := bloque[i]
                j.ayu := j.ayu + 1
                IF
                  j.ayu >= 12
                    SEQ
                      j.ayu := 0
                      i.ayu := i.ayu + 1
                  TRUE
                    SKIP
              --}}}
            --}}}
        --{{{  Cierra el fichero e indica el resultado
        close.tkf.file (from.filer, to.filer, resultado)
        IF
          resultado = va.bien
            SEQ
              write.full.string (pantalla, "*C*NCierre correcto de ")
              write.full.string (pantalla, [nombre.fichero FROM 0 FOR long])
              newline (pantalla)
              fich.abierto := FALSE
          TRUE
            --{{{  Escribe el resultado en la pantalla
            SEQ
              write.full.string (pantalla, "ERROR en el cierre de ")
              write.full.string (pantalla, [nombre.fichero FROM 0 FOR long])
              newline (pantalla)
              write.full.string (pantalla, "Resultado : ")
              write.int (pantalla, resultado, 0)
              newline (pantalla)
            --}}}
        --}}}
        ha.leido.fichero  := TRUE
        --}}}
      TRUE
        SKIP
    --}}}
    --}}}
:
--}}}
--
--{{{  Lectura de Pliegue con Param. Iniciales ( Apuntado por Cursor )
PROC CargaParamInicDePliegue (CHAN OF INT teclado,
                              CHAN OF ANY pantalla,
                              [4]CHAN OF ANY de.uf, a.uf)
```

```
SEQ
  --{{{  Procesos
  --{{{  COMMENT Escritura en Pliegues
  --:::A COMMENT FOLD
  --{{{  Escritura en Pliegues
  CHAN OF ANY fich.s:
  PAR
    --{{{  Programa principal (escritura)
    INT dato, una:
    SEQ
      write.full.string (pantalla, "Pulsa una tecla para empezar*C*N")
      read.char (teclado, una)
      write.full.string (fich.s, "Hola, esta es la primera prueba*C*N")
      write.full.string (fich.s, "esta es la segunda linea*C*N")
      write.full.string (fich.s, "esta es la tercera linea*C*N")
      write.full.string (fich.s, "ya es la ultima linea*C*N")
      write.endstream (fich.s)
      write.full.string (pantalla, "La escritura del fichero se ha completado*
    --}}}
    --{{{  Escritura en un fichero
    INT res, l.nom:
    SEQ
      l.nom := 1
      scrstream.to.file(fich.s, de.uf[0], a.uf[0], "Pliegue de prueba", l.nom,
    --}}}
  --}}}
  --}}}
  CHAN OF ANY fich.e:
  PAR
    --{{{  Programa principal (lectura)
    --{{{  Declaraciones
    INT dato, lon:
    INT ent.ayu :
    INT32 ayu32 :
    BOOL negativo :
    [80]BYTE linea:
    --}}}
    SEQ
      --{{{  Proc
      --{{{  Asignaciones
      negativo := FALSE
      dato := 0
      --}}}
      --{{{  Lectura/Envio de la Paleta Inicial
      read.text.line (fich.e, lon, linea, dato)
      IF
        ( dato >= 0 ) AND ( linea[0] <> signal.de.restar )
          SEQ
            --{{{  Proc
            write.full.string ( pantalla, "*C*NPaleta Inicial = " )
            STRINGTOINT32 ( negativo, ayu32, linea )
            ent.ayu := INT(ayu32)
            write.int ( pantalla, ent.ayu, 0 )
            write.full.string ( pantalla, "*S*S*S*S*S" )
            --}}}
        ( dato >= 0 ) AND ( linea[0] = signal.de.restar )
          SEQ
            --{{{  Proc
            read.text.line (fich.e, lon, linea, dato)
            IF
              ( dato >= 0 ) AND ( linea[0] <> signal.de.restar )
                SEQ
                  --{{{  Proc
                  write.full.string ( pantalla, "*C*NPaleta Inicial = " )
                  STRINGTOINT32 ( negativo, ayu32, linea )
                  ent.ayu := INT(ayu32)
```

```
          ha.leido.fichero  := TRUE
          --}}}
      TRUE
        SKIP
    --}}}
    --}}}
:
--}}}
--
--{{{  Lectura de Ficheros Binarios con Fuente de Caracteres
PROC LeeFicheroFuenteCaracter ( )
  --{{{  Declaraciones
  INT cont.hoja :
  INT i.ayu, j.ayu :
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Lectura del nombre del fichero y apertura del mismo
    --{{{  Asignaciones
    cont.hoja          :=    0
    i.ayu              :=    0
    j.ayu              :=    0
    numero.bytes.fich  :=    0
    fich.abierto       :=    FALSE
    fin.de.fichero     :=    FALSE
    no.leer            :=    FALSE
    existe             :=    FALSE
    error.de.lectura   :=    FALSE
    --}}}
    WHILE NOT existe
      SEQ
        write.full.string (screen, "*C*NNombre del fichero: ")
        read.echo.text.line (keyboard, screen, long, nombre, terminador)
        long := long - 1
        newline (screen)
        test.exists (from.filer, to.filer, long, nombre, existe)
        IF
          existe
            SEQ
              --{{{  Apertura del Fichero
              open.tkf.file (from.filer, to.filer, tkf.open.read, long, nombre,
              --}}}
              --{{{  Indica el resultado de la apertura
              IF
                resultado = va.bien
                  SEQ
                    fich.abierto := TRUE
                    write.full.string (screen, "Leyendo el fichero")
                    newline (screen)
                    no.leer := FALSE
                TRUE
                  SEQ
                    write.full.string (screen, "Error en la apertura del fichero
                    newline (screen)
                    write.full.string (screen, "Resultado: ")
                    write.int (screen, resultado, 0)
                    newline (screen)
                    error.de.lectura := FALSE
                    --{{{  Cierra el fichero e indica el resultado
                    close.tkf.file (from.filer, to.filer, resultado)
                    IF
                      resultado = va.bien
                        SEQ
                          write.full.string (screen, "*C*NCierre correcto de ")
                          write.full.string (screen, [nombre FROM 0 FOR long])
                          fich.abierto := FALSE
                          newline (screen)
                    TRUE
                      --{{{  Escribe el resultado en la pantalla
                      SEQ
                        write.full.string (screen, "ERROR en el cierre de ")
                        write.full.string (screen, [nombre FROM 0 FOR long])
                        newline (screen)
                        write.full.string (screen, "Resultado : ")
                        write.int (screen, resultado, 0)
                        newline (screen)
                      --}}}
                  --}}}
                  no.leer := TRUE

          --}}}
        NOT existe
          SEQ
            write.full.string (screen, "ERROR: el fichero no existe")
            newline (screen)
            no.leer := TRUE
  --}}}
  --{{{  Datos del Fuente, a sacar de un fichero
  IF
    NOT no.leer
      SEQ
        --{{{  Entrada de los datos
        LeeBloque ()
        numero.bytes.fich := numero.bytes.fich + long.bloque
        SEQ i=0 FOR long.bloque
          --{{{  Actualiza Matriz de Fuente
          SEQ
            fuente.elegido.by[i.ayu][j.ayu] := bloque[i]
            j.ayu := j.ayu + 1
            IF
              j.ayu >= 12
                SEQ
                  j.ayu := 0
                  i.ayu := i.ayu + 1
              TRUE
                SKIP
          --}}}
        WHILE NOT fin.de.fichero
          SEQ
            --{{{  Continua Lectura
            LeeBloque ()
            numero.bytes.fich := numero.bytes.fich + long.bloque
            SEQ i=0 FOR long.bloque
              --{{{  Actualiza Matriz de Fuente
              SEQ
                fuente.elegido.by[i.ayu][j.ayu] := bloque[i]
                j.ayu := j.ayu + 1
                IF
                  j.ayu >= 12
                    SEQ
                      j.ayu := 0
                      i.ayu := i.ayu + 1
                  TRUE
                    SKIP
              --}}}
            --}}}
        --{{{  COMMENT Mostra Fichero de Fuente en Pantalla
        --:::A COMMENT FOLD
        --{{{  Mostra Fichero de Fuente en Pantalla
        --{{{  Pantalla
        write.full.string ( pantalla,  "*C*N" )
        SalidaCadena(" Fichero de Fuente Leido :")
```

```
numero.bytes.fich :=    0
fich.abierto      :=    FALSE
fin.de.fichero    :=    FALSE
no.leer           :=    FALSE
existe            :=    FALSE
error.de.lectura  :=    FALSE
long              :=    tamano
SalidaCadena ("*C*Nlong=")
SalidaEntero (long)
SalidaCadena ("*C*Nnombre.fichero.fuente=<")
SEQ i=0 FOR tamano
  write.char ( screen, nombre.fichero[i] )
SalidaCadena (">")
SalidaCadena ("*C*N")
repetir           :=    TRUE
--}}}
WHILE ( ( NOT existe ) AND repetir )
  SEQ
    newline (screen)
    test.exists (from.filer, to.filer, long, nombre.fichero, existe)
    IF
      existe
        SEQ
          --{{{  Apertura del Fichero
          open.tkf.file (from.filer, to.filer, tkf.open.read, long,
                         nombre.fichero, resultado)
          --}}}
          --{{{  Indica el resultado de la apertura
          IF
            resultado = va.bien
              SEQ
                fich.abierto := TRUE
                write.full.string (screen, "Leyendo el fichero")
                newline (screen)
                no.leer := FALSE
            TRUE
              SEQ
                write.full.string (screen, "Error en la apertura del fichero
                newline (screen)
                write.full.string (screen, "Resultado: ")
                write.int (screen, resultado, 0)
                newline (screen)
                error.de.lectura := FALSE
                --{{{  Cierra el fichero e indica el resultado
                close.tkf.file (from.filer, to.filer, resultado)
                IF
                  resultado = va.bien
                    SEQ
                      write.full.string (screen, "*C*NCierre correcto de ")
                      write.full.string (screen, [nombre.fichero FROM 0 FOR
                      fich.abierto := FALSE
                      newline (screen)
                  TRUE
                    --{{{  Escribe el resultado en la pantalla
                    SEQ
                      write.full.string (screen, "ERROR en el cierre de ")
                      write.full.string (screen, [nombre.fichero FROM 0 FOR
                      newline (screen)
                      write.full.string (screen, "Resultado : ")
                      write.int (screen, resultado, 0)
                      newline (screen)
                    --}}}
                --}}}
                no.leer := TRUE

          --}}}
    NOT existe
      SEQ
        write.full.string (screen, "ERROR: el fichero no existe")
        newline (screen)
        no.leer  :=  TRUE
        repetir  :=  otra.vez
--}}}
--{{{  Datos del Fuente, a sacar de un fichero
IF
  NOT no.leer
    SEQ
      --{{{  Entrada de los datos
      LeeBloque ()
      numero.bytes.fich := numero.bytes.fich + long.bloque
      SEQ i=0 FOR long.bloque
        --{{{  Actualiza Matriz de Fuente
        SEQ
          fuente.elegido.by[i.ayu][j.ayu] := bloque[i]
          j.ayu := j.ayu + 1
          IF
            j.ayu >= 12
              SEQ
                j.ayu := 0
                i.ayu := i.ayu + 1
            TRUE
              SKIP
        --}}}
      WHILE NOT fin.de.fichero
        SEQ
          --{{{  Continua Lectura
          LeeBloque ()
          numero.bytes.fich := numero.bytes.fich + long.bloque
          SEQ i=0 FOR long.bloque
            --{{{  Actualiza Matriz de Fuente
            SEQ
              fuente.elegido.by[i.ayu][j.ayu] := bloque[i]
              j.ayu := j.ayu + 1
              IF
                j.ayu >= 12
                  SEQ
                    j.ayu := 0
                    i.ayu := i.ayu + 1
                TRUE
                  SKIP
            --}}}
          --}}}
      --{{{  Cierra el fichero e indica el resultado
      close.tkf.file (from.filer, to.filer, resultado)
      IF
        resultado = va.bien
          SEQ
            write.full.string (screen, "*C*NCierre correcto de ")
            write.full.string (screen, [nombre.fichero FROM 0 FOR long])
            newline (screen)
            fich.abierto := FALSE
        TRUE
          --{{{  Escribe el resultado en la pantalla
          SEQ
            write.full.string (screen, "ERROR en el cierre de ")
            write.full.string (screen, [nombre.fichero FROM 0 FOR long])
            newline (screen)
            write.full.string (screen, "Resultado : ")
            write.int (screen, resultado, 0)
            newline (screen)
          --}}}
      --}}}
```

```
                resultado = va.bien
                  SEQ
                    write.full.string (screen, "*C*NCierre correcto de ")
                    write.full.string (screen, [nombre.fichero.pliegue FRO
                    fich.abierto := FALSE
                    newline (screen)
                TRUE
                  --{{{  Escribe el resultado en la pantalla
                  SEQ
                    write.full.string (screen, "ERROR en el cierre de ")
                    write.full.string (screen, [nombre.fichero.pliegue FRO
                    newline (screen)
                    write.full.string (screen, "Resultado : ")
                    write.int (screen, resultado, 0)
                    newline (screen)
                  --}}}
              --}}}
              no.leer := TRUE

        --}}}
      NOT existe
        SEQ
          write.full.string (screen, "ERROR: el fichero no existe")
          newline (screen)
          no.leer  :=  TRUE
          repetir  :=  FALSE
--}}}
--{{{  Datos a sacar del fichero del pliegue
IF
  NOT no.leer
    SEQ
      --{{{  Entrada de los datos
      LeeBloque ()
      numero.bytes.fich := numero.bytes.fich + long.bloque
      SEQ i=0 FOR long.bloque
        --{{{  Actualiza Matriz de Fuente
        SEQ
          fuente.elegido.by[i.ayu][j.ayu] := bloque[i]
          j.ayu := j.ayu + 1
          IF
            j.ayu >= 12
              SEQ
                j.ayu := 0
                i.ayu := i.ayu + 1
            TRUE
              SKIP
        --}}}
      WHILE NOT fin.de.fichero
        SEQ
          --{{{  Continua Lectura
          LeeBloque ()
          numero.bytes.fich := numero.bytes.fich + long.bloque
          SEQ i=0 FOR long.bloque
            --{{{  Actualiza Matriz de Fuente
            SEQ
              fuente.elegido.by[i.ayu][j.ayu] := bloque[i]
              j.ayu := j.ayu + 1
              IF
                j.ayu >= 12
                  SEQ
                    j.ayu := 0
                    i.ayu := i.ayu + 1
                TRUE
                  SKIP
            --}}}
          --}}}
```

```
          --{{{  Cierra el fichero e indica el resultado
          close.tkf.file (from.filer, to.filer, resultado)
          IF
            resultado = va.bien
              SEQ
                write.full.string (screen, "*C*NCierre correcto de ")
                write.full.string (screen, [nombre.fichero.pliegue FROM 0 FOR lo
                newline (screen)
                fich.abierto := FALSE
            TRUE
              --{{{  Escribe el resultado en la pantalla
              SEQ
                write.full.string (screen, "ERROR en el cierre de ")
                write.full.string (screen, [nombre.fichero.pliegue FROM 0 FOR lo
                newline (screen)
                write.full.string (screen, "Resultado : ")
                write.int (screen, resultado, 0)
                newline (screen)
              --}}}
          --}}}
          ha.leido.fichero  := TRUE
          --}}}
      TRUE
        SKIP
    --}}}
  --}}}
:
--}}}
--
--}}}
--
--{{{  Lectura de Mensajes
--
--{{{  Lectura de Mensajes con Caracteres Patron
PROC  EscribirMensajePantallaGrafica (  )
  SEQ
    --{{{  Procesos
    --{{{  Color
    SalidaCadena ( "*N*C Teclee la Color de la Mensaje *N*C" )
    PantallaCogerNumero ( " Color " )
    caracter := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    color.elegido.ent := numero.ent
    --}}}
    --{{{  X.mensaje
    SalidaCadena ( "*N*C Teclee la Coordenada x de la Mensaje *N*C" )
    PantallaCogerNumero ( " x.mensaje " )
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    x.mensaje := numero.ent
    --}}}
    --{{{  Y.mensaje
    SalidaCadena ( "*N*C Teclee la Coordenada y de la Mensaje *N*C" )
    PantallaCogerNumero ( " y.mensaje " )
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    y.mensaje := numero.ent
    --}}}
    --{{{  Mensaje
    SalidaCadena ( "*N*C Teclear Mensaje *N*C" )
    read.echo.text.line ( keyboard, screen, tamano, cadena.mensaje.by,
                          caracter )
    --}}}
    --}}}
:
--}}}
--
--{{{  Lectura de Mensajes con Caracteres Doble
```

```
    --}}}
  SEQ i=0 FOR 256
    SEQ
      --{{{  Formatacion de la Pantalla
      IF
        cont.hoja >= 24
          SEQ
            cont.hoja  := 0
            SalidaCadena( "*C*NTeclee Espacio Para Continuar " )
            keyboard ? caracter
            write.full.string ( pantalla,  "*C*N" )
        TRUE
          SKIP
      --}}}
      write.full.string ( pantalla,  "*C*NCARACTER ")
      SalidaEntero ( i )
      --{{{  Formatacion
      IF
        i <= 9
          SEQ
            write.full.string ( pantalla,  "*S*S" )
        ( i > 9 ) AND ( i <= 99 )
          write.full.string ( pantalla,  "*S" )
        TRUE
          SKIP
      --}}}
      write.full.string ( pantalla,  " : ")
      cont.hoja := cont.hoja + 1
      SEQ j=0 FOR 12
        --{{{  Procesos
        SEQ
          k := INT ( fuente.elegido.by[i][j] )
          --{{{  Formatacion de los Numeros
          IF
            k <= 9
              write.full.string ( pantalla,  "*S*S*S" )
            ( k > 9 ) AND ( k <= 99 )
              write.full.string ( pantalla,  "*S*S" )
            TRUE
              SalidaCaracter ( "*S" )
          --}}}
          write.int ( screen, k, 0 )
          SalidaCaracter ( "*S" )
        --}}}
--}}}
--}}}
--{{{  Cierra el fichero e indica el resultado
close.tkf.file (from.filer, to.filer, resultado)
IF
  resultado = va.bien
    SEQ
      write.full.string (screen, "*C*NCierre correcto de ")
      write.full.string (screen, [nombre FROM 0 FOR long])
      newline (screen)
      fich.abierto := FALSE
  TRUE
    --{{{  Escribe el resultado en la pantalla
    SEQ
      write.full.string (screen, "ERROR en el cierre de ")
      write.full.string (screen, [nombre FROM 0 FOR long])
      newline (screen)
      write.full.string (screen, "Resultado : ")
      write.int (screen, resultado, 0)
      newline (screen)
    --}}}
--}}}
```

```
        --}}}
      TRUE
        SKIP
    --}}}
    --}}}
:
--}}}
--
--{{{  Lectura de Fich. DOS de Pliegues con Nombre Fijo
PROC LeeFicheroDOSPliegueFijo ( []BYTE nombre.fichero.pliegue )
  --{{{  Declaraciones
  INT cont.hoja :
  INT i.ayu, j.ayu :
  BOOL repetir :
  --}}}
  SEQ
    --{{{  Procesos
    --{{{  Verificacion del nombre del fichero y apertura del mismo
    --{{{  Asignaciones
    cont.hoja         :=    0
    i.ayu             :=    0
    j.ayu             :=    0
    numero.bytes.fich :=    0
    fich.abierto      :=    FALSE
    fin.de.fichero    :=    FALSE
    no.leer           :=    FALSE
    existe            :=    FALSE
    error.de.lectura  :=    FALSE
    SalidaCadena ( "*C*NNombre fichero =<")
    SEQ i=0 FOR 12
      write.char ( screen, nombre.fichero.pliegue[i] )
    SalidaCadena ( ">")
    SalidaCadena ( "*C*N")
    repetir           :=    TRUE
    --}}}
    WHILE ( ( NOT existe ) AND repetir )
      SEQ
        newline (screen)
        test.exists (from.filer, to.filer, long, nombre.fichero.pliegue,
                     existe)
        IF
          existe
            SEQ
              --{{{  Apertura del Fichero
              open.tkf.file (from.filer, to.filer, tkf.open.read, long,
                             nombre.fichero.pliegue, resultado)
              --}}}
              --{{{  Indica el resultado de la apertura
              IF
                resultado = va.bien
                  SEQ
                    fich.abierto := TRUE
                    write.full.string (screen, "Leyendo el fichero")
                    newline (screen)
                    no.leer := FALSE
                TRUE
                  SEQ
                    write.full.string (screen, "Error en la apertura del fichero
                    newline (screen)
                    write.full.string (screen, "Resultado: ")
                    write.int (screen, resultado, 0)
                    newline (screen)
                    error.de.lectura := FALSE
                    --{{{  Cierra el fichero e indica el resultado
                    close.tkf.file (from.filer, to.filer, resultado)
                    IF
```

```
      PantallaMandatos ( " Borra Pantalla " )
      Org.Mp !  origen; destino; tipo; longitud
      VerificacionEjecucion ()
    --}}}
  tipo = t.x
    --{{{  Inicia Pantalla
    SEQ
      longitud := 2
      PantallaMandatos ( " Inicia Pantalla (x.maximo, y.maximo) " )
      --{{{  Coge x.maximo/y.maximo
      --{{{  X.maximo
      PantallaCogerNumero ( " x.maximo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Y.maximo
      PantallaCogerNumero ( " y.maximo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.color
    --{{{  Color
    SEQ
      longitud := 1
      PantallaMandatos ( " Color " )
      --{{{  Coger Color
      PantallaCogerNumero ( " color " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametro
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.iniciar
    --{{{  Inicia Pantalla con 1024 X 768
    SEQ
      -- Para no Pedir Parametros en la Pantalla
      longitud := 0
      PantallaMandatos ( " Inicia Pantalla ( 1024 X 768 ) " )
      -- Para Enviar Parametros Fijos
      longitud := 2
      --{{{  Coge 1023 X 767
      parametros[0] := ancho.max.ent - 1
      parametros[1] := alto.max.ent - 1
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.y
    --{{{  Cambiar Paleta ( Patron=0, Suave=1, Fijar Paleta elegido=2 )
    SEQ
      IF
        paleta.ant = 2
          SEQ
            --{{{  No Puede Cambiar Paleta
            longitud := 0
            PantallaMandatos ( " Cambiar Paleta " )
            --{{{  Envia Cabecera
            Org.Mp !  origen; destino; tipo; longitud
            --}}}
            VerificacionEjecucion ()
            --}}}
        TRUE
          SEQ
            --{{{  Puede Cambiar Paleta
            longitud := 1
            PantallaMandatos ( " Cambiar Paleta " )
            --{{{  Coger/Enviar Paleta Para Controle Pantalla
            PantallaCogerNumero ( " paleta " )
            caracter      := INT(' ')
            read.echo.int ( keyboard, screen, numero.ent, caracter )
            parametros[0] := numero.ent
            paleta        := numero.ent
            paleta.ant    := paleta
            --}}}
            --{{{  Envia Cabecera
            Org.Mp !  origen; destino; tipo; longitud
            --}}}
            --{{{  Envia Parametro
            Org.Mp !  [ parametros FROM 0  FOR longitud ]
            --}}}
            VerificacionEjecucion ()
            --}}}
    --}}}
  --}}}
--{{{  Que Dibujan
--{{{  Rectilineas
tipo = t.linea
  --{{{  Tira Linea
  SEQ
    longitud := 4
    PantallaMandatos ( " TiraLinea ( xi, yi, xf, yf ) " )
    --{{{  Coger Numeros
    --{{{  Xi
    PantallaCogerNumero ( " xi " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[0] := numero.ent
    --}}}
    --{{{  Yi
    PantallaCogerNumero ( " yi " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[1] := numero.ent
    --}}}
    --{{{  Xf
    PantallaCogerNumero ( " xf " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[2] := numero.ent
```

```
PROC  EscribirMensajeCaracterDoble ( )
  SEQ
    --{{{  Procesos
    --{{{  Color
    SalidaCadena ( "*N*C Teclee la Color de la Mensaje *N*C" )
    PantallaCogerNumero ( " Color " )
    caracter := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    color.elegido.ent := numero.ent
    --}}}
    --{{{  X.mensaje
    SalidaCadena ( "*N*C Teclee la Coordenada x de la Mensaje *N*C" )
    PantallaCogerNumero ( " x.mensaje " )
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    x.mensaje := numero.ent
    --}}}
    --{{{  Y.mensaje
    SalidaCadena ( "*N*C Teclee la Coordenada y de la Mensaje *N*C" )
    PantallaCogerNumero ( " y.mensaje " )
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    y.mensaje := numero.ent
    --}}}
    --{{{  Mensaje
    SalidaCadena ( "*N*C Teclear Mensaje *N*C" )
    read.echo.text.line ( keyboard, screen, tamano, cadena.mensaje.by,
                          caracter )
    --}}}
    --}}}
:
--}}}
--
--{{{  Lectura de Mensajes con Caracteres de Ancho/Alto Variables
PROC  EscribirMensajeCaracterVariable ( )
  SEQ
    --{{{  Procesos
    --{{{  Color
    SalidaCadena ( "*N*C Teclee la Color de la Mensaje *N*C" )
    PantallaCogerNumero ( " Color " )
    caracter := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    color.elegido.ent := numero.ent
    --}}}
    --{{{  X.mensaje
    SalidaCadena ( "*N*C Teclee la Coordenada x de la Mensaje *N*C" )
    PantallaCogerNumero ( " x.mensaje " )
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    x.mensaje := numero.ent
    --}}}
    --{{{  Y.mensaje
    SalidaCadena ( "*N*C Teclee la Coordenada y de la Mensaje *N*C" )
    PantallaCogerNumero ( " y.mensaje " )
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    y.mensaje := numero.ent
    --}}}
    --{{{  Ancho.esc
    SalidaCadena ( "*N*C Teclee la Escala del Ancho del Caracter *N*C" )
    PantallaCogerNumero ( " ancho.esc " )
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    ancho.esc := numero.ent
    --}}}
    --{{{  Alto.esc
    SalidaCadena ( "*N*C Teclee la Escala del Alto del Caracter *N*C" )
    PantallaCogerNumero ( " alto.esc " )
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    alto.esc := numero.ent
    --}}}
    --{{{  Mensaje
    SalidaCadena ( "*N*C Teclear Mensaje *N*C" )
    read.echo.text.line ( keyboard, screen, tamano, cadena.mensaje.by,
                          caracter )
    --}}}
    --}}}
:
--}}}
--
--}}}
--
--}}}
--{{{  Verificacion de ejecucion/Errores
PROC VerificacionEjecucion ()
  SEQ
    --{{{  Procesos
    Mp.Org ?  a.ver
    --{{{  A ver.
    SalidaCadena ( "*C*N" )
    IF
      a.ver = 0
        SalidaCadena ( "OK." )
      TRUE
        SEQ
          SalidaCadena ( "ERROR numero " )
          SalidaEntero ( a.ver )
    SalidaCadena ( "*C*N". "
    --}}}
    --}}}
:
--}}}
--
--}}}
--
--
--
--
-- -- Programa de Prueba --
--
--{{{  Programa Principal
SEQ
  --{{{  Asignacion de Variables
  activo             :=  TRUE
  ha.leido.fichero   :=  FALSE
  negativo           :=  FALSE
  destino            :=  0
  origen             :=  2
  [ nomb.fich.plie FROM 0 FOR 12 ] :=    "paramini.tsr"
  --}}}
  --{{{  Carga Parametros Iniciales de Pliegue
  CargaParamInicDePliegue ( keyboard, screen, from.user.filer, to.user.filer )
  --}}}
  WHILE activo
    SEQ
      CabeceraInicial ()
      MandatosDisponibles ()
      TeclearMandato ()
      --{{{  Elegir Mandato
      IF
        --{{{  Fundamentales
        --{{{  Iniciales
        tipo = t.borrar
          --{{{  Borra Pantalla
          SEQ
            longitud := 0
```

```
      --}}}
      --{{{  Yf
      PantallaCogerNumero ( " yf " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[3] := numero.ent
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.rectang
    --{{{  Rectangulo
    SEQ
      longitud := 4
      PantallaMandatos ( " Rectangulo ( xi, yi, xf, yf ) " )
      --{{{  Coger Numeros
      --{{{  Xi
      PantallaCogerNumero ( " xi " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Yi
      PantallaCogerNumero ( " yi " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --{{{  Xf
      PantallaCogerNumero ( " xf " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[2] := numero.ent
      --}}}
      --{{{  Yf
      PantallaCogerNumero ( " yf " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[3] := numero.ent
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.k
    --{{{  Flechas
    SEQ
      longitud := 0
      --longitud := 4
      PantallaMandatos ( " Flecha ( xi, yi, xf, yf ) " )
      PantallaNoImplementadoTodavia ( " Flechas " )
      --{{{  COMMENT Coger Numeros
      --:::A COMMENT FOLD
      --{{{  Coger Numeros
      --{{{  Xi
      PantallaCogerNumero ( " xi " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Yi
      PantallaCogerNumero ( " yi " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --{{{  Xf
      PantallaCogerNumero ( " xf " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[2] := numero.ent
      --}}}
      --{{{  Yf
      PantallaCogerNumero ( " yf " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[3] := numero.ent
      --}}}
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  COMMENT Envia Parametros
      --:::A COMMENT FOLD
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.m
    --{{{  Paralelogramos
    SEQ
      --longitud := 0
      longitud := 6
      PantallaMandatos ( " Paralelogramo ( xi, yi, xf, yf, xo, yo ) " )
      --PantallaNoImplementadoTodavia ( " Paralelogramo " )
      --{{{  Coger Numeros
      --{{{  Xi
      PantallaCogerNumero ( " xi " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Yi
      PantallaCogerNumero ( " yi " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --{{{  Xf
      PantallaCogerNumero ( " xf " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[2] := numero.ent
      --}}}
      --{{{  Yf
      PantallaCogerNumero ( " yf " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
```

```
      parametros[3] := numero.ent
      --}}}
      --{{{  Xo
      PantallaCogerNumero ( " xo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[4] := numero.ent
      --}}}
      --{{{  Yo
      PantallaCogerNumero ( " yo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[5] := numero.ent
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.n
    --{{{  Poligonos
    SEQ
      PantallaElegirLongitud ( "Poligono", "n" )
      --{{{  Longitud
      longitud := 0
      -- Porque No Esta Implementado Todavia
      --}}}
      --{{{  COMMENT Coger Longitud
      --:::A COMMENT FOLD
      --{{{  Coger Longitud
      CogerNumero ( " n" )
      longitud  := INT ROUND(numero.real)
      numero.real := REAL32 ROUND(0)
      --}}}
      --}}}
      PantallaMandatos ( " Poligono ( n vertices ) " )
      PantallaNoImplementadoTodavia ( " Poligonos " )
      --{{{  COMMENT Coger Numeros
      --:::A COMMENT FOLD
      --{{{  Coger Numeros
      SEQ i = 0 FOR longitud
        SEQ
          SalidaCaracter ( retorno )
          SalidaCaracter ( nueva.linea )
          SalidaCadena ( " Vertice[" )
          SalidaCaracter ( i )
          SalidaCadena ( "] : " )
          --{{{  Xi
          PantallaCogerNumero ( " x " )
          caracter  := INT(' ')
          read.echo.int ( keyboard, screen, numero.ent, caracter )
          parametros[(2 * i)] := numero.ent
          --}}}
          --{{{  Yi
          PantallaCogerNumero ( " yi " )
          caracter  := INT(' ')
          read.echo.int ( keyboard, screen, numero.ent, caracter )
          parametros[((2 * i) + 1)] := numero.ent
          --}}}
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  COMMENT Envia Parametros
      --:::A COMMENT FOLD
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.p
    --{{{  Rectangulo Lento
    SEQ
      longitud := 4
      PantallaMandatos ( " Rectangulo Lento ( xi, yi, xf, yf ) " )
      --{{{  Coger Numeros
      --{{{  Xi
      PantallaCogerNumero ( " xi " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Yi
      PantallaCogerNumero ( " yi " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --{{{  Xf
      PantallaCogerNumero ( " xf " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[2] := numero.ent
      --}}}
      --{{{  Yf
      PantallaCogerNumero ( " yf " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[3] := numero.ent
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  --}}}
  --{{{  Curvilineas
  --{{{  Circulos
  tipo = t.circulares
    --{{{  Circulo
    SEQ
      longitud := 3
      PantallaMandatos ( " Circulo ( xo, yo, r ) " )
      --{{{  Coger Numeros
      --{{{  Xo
      PantallaCogerNumero ( " xo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Yo
      PantallaCogerNumero ( " yo " )
```

```
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --{{{  R
      PantallaCogerNumero ( " radio " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[2] := numero.ent
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.dos
    --{{{  Semicirculo
    SEQ
      longitud := 4
      PantallaMandatos ( " Semicirculo ( xo, yo, r, sup/inf ) " )
      --{{{  Coger Numeros
      --{{{  Xo
      PantallaCogerNumero ( " xo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Yo
      PantallaCogerNumero ( " yo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --{{{  R
      PantallaCogerNumero ( " radio " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[2] := numero.ent
      --}}}
      --{{{  Superior(0)/Inferior(1)
      PantallaCogerNumero ( " superior(0)/inferior(1) " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[3] := numero.ent
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.tres
    --{{{  Cuarto de Circulo
    SEQ
      longitud := 4
      PantallaMandatos ( " Cuarto de circulo ( xo, yo, r, cuadrante ) " )
      --{{{  Coger Numeros
      --{{{  Xo
      PantallaCogerNumero ( " xo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Yo
      PantallaCogerNumero ( " yo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --{{{  R
      PantallaCogerNumero ( " radio " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[2] := numero.ent
      --}}}
      --{{{  Cuadrante
      PantallaCogerNumero ( " cuadrante " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[3] := numero.ent
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  --}}}
  --{{{  Circunferencias
  tipo = t.a
    --{{{  Circunferencia
    SEQ
      longitud := 3
      PantallaMandatos ( " Circunferencia ( xo, yo, r ) " )
      --{{{  Coger Numeros
      --{{{  Xo
      PantallaCogerNumero ( " xo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Yo
      PantallaCogerNumero ( " yo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --{{{  R
      PantallaCogerNumero ( " radio " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[2] := numero.ent
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
```

```
tipo = t.cinco
  --{{{  Semicircunferencia
  SEQ
    longitud := 4
    PantallaMandatos ( " Semicircunferencia ( xo, yo, r, sup/inf ) " )
    --{{{  Coger Numeros
    --{{{  Xo
    PantallaCogerNumero ( " xo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[0] := numero.ent
    --}}}
    --{{{  Yo
    PantallaCogerNumero ( " yo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[1] := numero.ent
    --}}}
    --{{{  R
    PantallaCogerNumero ( " radio " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[2] := numero.ent
    --}}}
    --{{{  Superior(0)/Inferior(1)
    PantallaCogerNumero ( " superior(0)/inferior(1) " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[3] := numero.ent
    --}}}
    --}}}
    --{{{  Envia Cabecera
    Org.Mp !  origen; destino; tipo; longitud
    --}}}
    --{{{  Envia Parametros
    Org.Mp !  [ parametros FROM 0  FOR longitud ]
    --}}}
    VerificacionEjecucion ()
  --}}}
tipo = t.seis
  --{{{  Cuarto de Circunferencia
  SEQ
    longitud := 4
    PantallaMandatos ( " Cuarto de circunferencia ( xo, yo, r, cuadrante
    --{{{  Coger Numeros
    --{{{  Xo
    PantallaCogerNumero ( " xo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[0] := numero.ent
    --}}}
    --{{{  Yo
    PantallaCogerNumero ( " yo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[1] := numero.ent
    --}}}
    --{{{  R
    PantallaCogerNumero ( " radio " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[2] := numero.ent
    --}}}
    --{{{  Cuadrante
    PantallaCogerNumero ( " cuadrante " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[3] := numero.ent
    --}}}
    --}}}
    --{{{  Envia Cabecera
    Org.Mp !  origen; destino; tipo; longitud
    --}}}
    --{{{  Envia Parametros
    Org.Mp !  [ parametros FROM 0  FOR longitud ]
    --}}}
    VerificacionEjecucion ()
  --}}}
--{{{  Espirales
tipo = t.cuatro
  --{{{  Espiral  sin Contorno
  SEQ
    longitud := 4
    PantallaMandatos ( " Espiral ( xo, yo, radio.ext, ancho.int ) " )
    --{{{  Coger Numeros
    --{{{  Xo
    PantallaCogerNumero ( " xo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[0] := numero.ent
    --}}}
    --{{{  Yo
    PantallaCogerNumero ( " yo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[1] := numero.ent
    --}}}
    --{{{  Radio Externo
    PantallaCogerNumero ( " radio externo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[2] := numero.ent
    --}}}
    --{{{  Ancho Interno
    PantallaCogerNumero ( " ancho interno " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[3] := numero.ent
    --}}}
    --}}}
    --{{{  Envia Cabecera
    Org.Mp !  origen; destino; tipo; longitud
    --}}}
    --{{{  Envia Parametros
    Org.Mp !  [ parametros FROM 0  FOR longitud ]
    --}}}
    VerificacionEjecucion ()
  --}}}
tipo = t.siete
  --{{{  Espiral  con Contorno
  SEQ
    longitud := 4
    PantallaMandatos ( " Espiral ( xo, yo, radio.ext, ancho.int ) " )
    --{{{  Coger Numeros
    --{{{  Xo
    PantallaCogerNumero ( " xo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[0] := numero.ent
    --}}}
    --{{{  Yo
```

```
    PantallaCogerNumero ( " yo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[1] := numero.ent
    --}}}
    --{{{  Radio Externo
    PantallaCogerNumero ( " radio externo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[2] := numero.ent
    --}}}
    --{{{  Ancho Interno
    PantallaCogerNumero ( " ancho interno " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[3] := numero.ent
    --}}}
    --}}}
    --{{{  Envia Cabecera
    Org.Mp !  origen; destino; tipo; longitud
    --}}}
    --{{{  Envia Parametros
    Org.Mp !  [ parametros FROM 0  FOR longitud ]
    --}}}
    VerificacionEjecucion ()
  --}}}
tipo = t.nueve
  --{{{  Espiral  sin Contorno con dos Colores
  SEQ
    longitud := 6
    PantallaMandatos (" Espiral (xo, yo, r.ext, ancho.int, colori, color
    --{{{  Coger Numeros
    --{{{  Xo
    PantallaCogerNumero ( " xo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[0] := numero.ent
    --}}}
    --{{{  Yo
    PantallaCogerNumero ( " yo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[1] := numero.ent
    --}}}
    --{{{  Radio Externo
    PantallaCogerNumero ( " radio externo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[2] := numero.ent
    --}}}
    --{{{  Ancho Interno
    PantallaCogerNumero ( " ancho interno " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[3] := numero.ent
    --}}}
    --{{{  Color 1
    PantallaCogerNumero ( " color 1 " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[4] := numero.ent
    --}}}
    --{{{  Color 2
    PantallaCogerNumero ( " color 2 " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
```

```
    parametros[5] := numero.ent
    --}}}
    --}}}
    --{{{  Envia Cabecera
    Org.Mp !  origen; destino; tipo; longitud
    --}}}
    --{{{  Envia Parametros
    Org.Mp !  [ parametros FROM 0  FOR longitud ]
    --}}}
    VerificacionEjecucion ()
  --}}}
--}}}
tipo = t.d
  --{{{  Elipse
  SEQ
    longitud := 0
    --longitud := 4
    PantallaMandatos ( " Elipse ( xo, yo, r.a, r.b ) " )
    PantallaNoImplementadoTodavia ( " Elipses " )
    --{{{  COMMENT Coger Numeros
    --:::A COMMENT FOLD
    --{{{  Coger Numeros
    --{{{  Xo
    PantallaCogerNumero ( " xo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[0] := numero.ent
    --}}}
    --{{{  Yo
    PantallaCogerNumero ( " yo " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[1] := numero.ent
    --}}}
    --{{{  Ra
    PantallaCogerNumero ( " radio a " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[2] := numero.ent
    --}}}
    --{{{  Rb
    PantallaCogerNumero ( " radio b " )
    caracter  := INT(' ')
    read.echo.int ( keyboard, screen, numero.ent, caracter )
    parametros[3] := numero.ent
    --}}}
    --}}}
    --}}}
    --{{{  Envia Cabecera
    Org.Mp !  origen; destino; tipo; longitud
    --}}}
    --{{{  COMMENT Envia Parametros
    --:::A COMMENT FOLD
    --{{{  Envia Parametros
    Org.Mp !  [ parametros FROM 0  FOR longitud ]
    --}}}
    --}}}
    VerificacionEjecucion ()
  --}}}
tipo = t.e
  --{{{  Arco
  SEQ
    longitud := 0
    --longitud := 5
    PantallaMandatos ( " Arco ( xo, yo, xi, yi, angulo ) " )
    PantallaNoImplementadoTodavia ( " Arcos " )
```

```
      --{{{  COMMENT Coger Numeros
      --:::A COMMENT FOLD
      --{{{  Coger Numeros
      --{{{  Xo
      PantallaCogerNumero ( " xo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Yo
      PantallaCogerNumero ( " yo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --{{{  Xi
      PantallaCogerNumero ( " xi " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[2] := numero.ent
      --}}}
      --{{{  Yi
      PantallaCogerNumero ( " yi " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[3] := numero.ent
      --}}}
      --{{{  Angulo
      PantallaCogerNumero ( " angulo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[4] := numero.ent
      --}}}
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  COMMENT Envia Parametros
      --:::A COMMENT FOLD
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.g
    --{{{  Corona
    SEQ
      longitud := 4
      PantallaMandatos ( " Corona ( xo, yo, r.int, r.ext ) " )
      --{{{  Coger Numeros
      --{{{  Xo
      PantallaCogerNumero ( " xo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Yo
      PantallaCogerNumero ( " yo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --{{{  R.int
      PantallaCogerNumero ( " radio.int " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[2] := numero.ent
      --}}}
      --{{{  R.ext
      PantallaCogerNumero ( " radio.ext " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[3] := numero.ent
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.h
    --{{{  Circulo ( Lento )
    SEQ
      longitud := 3
      PantallaMandatos ( " Circulo Lento ( xo, yo, r ) " )
      --{{{  Coger Numeros
      --{{{  Xo
      PantallaCogerNumero ( " xo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Yo
      PantallaCogerNumero ( " yo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --{{{  R
      PantallaCogerNumero ( " radio " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[2] := numero.ent
      --}}}
      --}}}
      --{{{  Envia Cabecera Opcion
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  --}}}
  --}}}
  --{{{  Zoom de Pantalla
  tipo = t.z
    --{{{  Zoom
    SEQ
      longitud := 0
      --longitud := 3
      PantallaMandatos ( " Zoom ( xo, yo, n ) " )
      PantallaNoImplementadoTodavia ( " Zoom " )
      --{{{  COMMENT Coger Numeros
      --:::A COMMENT FOLD
      --{{{  Coger Numeros
      --{{{  Xo
```

```
      PantallaCogerNumero ( " xo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Yo
      PantallaCogerNumero ( " yo " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[1] := numero.ent
      --}}}
      --{{{  N
      PantallaCogerNumero ( " n " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[2] := numero.ent
      --}}}
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  COMMENT Envia Parametros
      --:::A COMMENT FOLD
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      --}}}
      VerificacionEjecucion ()
    --}}}
  --}}}
  --}}}
  --{{{  Caracteres/Ficheros
  tipo = t.otras
    --{{{  Cargar Fuente de Caracteres
    SEQ
      longitud      := 4
      PantallaMandatos ( " Cargar Fuente " )
      LeeFicheroFuenteCaracter ( )
      --{{{  Coger Numeros
      --{{{  COMMENT Numero de Caracteres
      --:::A COMMENT FOLD
      --{{{  Numero de Caracteres
      PantallaCogerNumero ( " numero.caracteres " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      num.carac     := numero.ent
      parametros[1] := numero.ent
      --}}}
      --}}}
      parametros[1]     := 256
      num.carac         := 256
      --{{{  COMMENT Fuente.x
      --:::A COMMENT FOLD
      --{{{  Fuente.x
      PantallaCogerNumero ( " fuente.x " )
      caracter        := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      fuente.x        := numero.ent
      parametros[2]   := numero.ent
      --}}}
      --}}}
      parametros[2]       := 8
      fuente.x            := 8
      --{{{  COMMENT Fuente.y
      --:::A COMMENT FOLD
      --{{{  Fuente.y
      PantallaCogerNumero ( " fuente.y " )
      caracter        := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      fuente.y        := numero.ent
      parametros[3]   := numero.ent
      --}}}
      --}}}
      parametros[3]       := 12
      fuente.y            := 12
      --{{{  COMMENT Longitud.fuente
      --:::A COMMENT FOLD
      --{{{  Longitud.fuente
      PantallaCogerNumero ( " longitud.fuente " )
      caracter        := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      longitud.fuente := numero.ent
      parametros[0]   := numero.ent
      --}}}
      --}}}
      longitud.fuente     := num.carac * fuente.y
      parametros[0]       := longitud.fuente
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      --{{{  Envia Fichero de Fuente
      SEQ i=0 FOR num.carac
        SEQ j=0 FOR fuente.y
          fuente.elegido.ent[i][j] := INT(fuente.elegido.by[i][j])
      Org.Mp !  [ fuente.elegido.ent FROM 0 FOR longitud.fuente ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.q
    --{{{  Escribir Ficheros ASCII en la Pantalla
    SEQ
      longitud := 0
      --longitud := 6
      PantallaMandatos ( " Escribir Fichero ASCII  en Pantalla " )
      --{{{  COMMENT Coger Numeros
      --:::A COMMENT FOLD
      --{{{  Coger Numeros
      --{{{  Numero de Caracteres
      PantallaCogerNumero ( " numero.caracteres " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      num.carac     := numero.ent
      parametros[1] := numero.ent
      --}}}
      --{{{  Fuente.x
      PantallaCogerNumero ( " fuente.x " )
      caracter        := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      fuente.x        := numero.ent
      parametros[2]   := numero.ent
      --}}}
      --{{{  Fuente.y
      PantallaCogerNumero ( " fuente.y " )
      caracter        := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      fuente.y        := numero.ent
      parametros[3]   := numero.ent
```

```
      --}}}
      --{{{  Longitud.fuente
      PantallaCogerNumero ( " longitud.fuente " )
      caracter        := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      longitud.fuente := numero.ent
      parametros[0]   := numero.ent
      --}}}
      --}}}
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  COMMENT Envia Parametros
      --:::A COMMENT FOLD
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.s
    --{{{  Escribir Mensajes de Caracteres en la Pantalla
    SEQ
      PantallaMandatos ( " Escribir Mensaje en Pantalla " )
      EscribirMensajePantallaGrafica ( )
      longitud := tamano + 2
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      SEQ i=0 FOR ( tamano - 1 )
        cadena.mensaje.ent [i] := INT(cadena.mensaje.by[i])
      parametros[0] := color.elegido.ent
      parametros[1] := x.mensaje
      parametros[2] := y.mensaje
      SEQ i=3 FOR ( longitud - 3 )
        parametros[i] := cadena.mensaje.ent[(i-3)]
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.t
    --{{{  Escribir Mensajes de Caracteres Dobles en la Pantalla
    SEQ
      PantallaMandatos ( " Escribir Mensaje con Caracteres Doble " )
      EscribirMensajeCaracterDoble ( )
      longitud := tamano + 2
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      SEQ i=0 FOR ( tamano - 1 )
        cadena.mensaje.ent [i] := INT(cadena.mensaje.by[i])
      parametros[0] := color.elegido.ent
      parametros[1] := x.mensaje
      parametros[2] := y.mensaje
      SEQ i=3 FOR ( longitud - 3 )
        parametros[i] := cadena.mensaje.ent[(i-3)]
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.w
    --{{{  Escrib. Mensaj. con Caracteres de Ancho/Alto Variables
    SEQ
      PantallaMandatos ( "Escrib. Mens. con Caracteres de Ancho/Alto Varia
      EscribirMensajeCaracterVariable ( )
      longitud := tamano + 4
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      SEQ i=0 FOR ( tamano - 1 )
        cadena.mensaje.ent [i] := INT(cadena.mensaje.by[i])
      parametros[0] := color.elegido.ent
      parametros[1] := x.mensaje
      parametros[2] := y.mensaje
      parametros[3] := ancho.esc
      parametros[4] := alto.esc
      SEQ i=5 FOR ( longitud - 5 )
        parametros[i] := cadena.mensaje.ent[(i-5)]
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  --}}}
  --{{{  Ejemplos
  tipo = t.ventanas
    --{{{  Ventanas
    SEQ
      longitud := 1
      PantallaMandatos ( " Dibujar Ventanas  ( color.inicial ) " )
      --{{{  Pantalla Para Teclear los Parametros
      SalidaCaracter ( retorno )
      SalidaCaracter ( nueva.linea )
      SalidaCaracter ( nueva.linea )
      SalidaCaracter ( retorno )
      SalidaCaracter ( nueva.linea )
      SalidaCadena ( " e, Para Esto, Hay Que Teclear un Parametro " )
      SalidaCaracter ( retorno )
      SalidaCaracter ( nueva.linea )
      SalidaCaracter ( nueva.linea )
      --}}}
      --{{{  Coger Color Inicial
      PantallaCogerNumero ( " color " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.cero
    --{{{  Circulos Cambiando xo, yo y radio
    SEQ
      longitud := 1
      PantallaMandatos ( " Dibujar Circulos Cambiando xo, yo y radio. " )
      --{{{  Coger Numeros
      PantallaCogerNumero ( " semilla " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
```

Antonio 20/09/90 8C

```
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      TeclearParada ( " Dibujar Circulos Cambiando xo, yo y radio " )
      VerificacionEjecucion ()
    --}}}
  tipo = t.uno
    --{{{  Cuadrados y Circulos
    SEQ
      longitud := 1
      PantallaMandatos ( " Dibujar Cuadrados y Circulos ( color.inicial )
      --{{{  Coger Color Inicial
      PantallaCogerNumero ( " color " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.ejemplos
    --{{{  Cuadrados y Circulos Lentos
    SEQ
      longitud := 1
      PantallaMandatos ( "Dibujar Cuadrados y Circulos Lentos ( color.inic
      --{{{  Coger Color Inicial
      PantallaCogerNumero ( " color " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  tipo = t.A
    --{{{  Mandelbroot/Julia
    SEQ
      longitud := 1
      PantallaMandatos ( " Mandelbroot/julia " )
      --{{{  Coger Opcion
      PantallaCogerNumero ( " Mandelbroot/Julia " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, numero.ent, caracter )
      parametros[0] := numero.ent
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametro
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      TeclearParada ( " Mandelbroot/Julia " )
      VerificacionEjecucion ()
    --}}}
  --}}}
  --{{{  Rutina en Prueba
  tipo = t.ocho
    --{{{  Prueba
    SEQ
      PantallaElegirLongitud ( "Rutina en Prueba", "num.param" )
      --{{{  Coger longitud
      PantallaCogerNumero ( " longitud " )
      caracter  := INT(' ')
      read.echo.int ( keyboard, screen, longitud, caracter )
      --}}}
      --{{{  Coger Parametros
      PantallaCogerParametros ( longitud )
      --}}}
      --{{{  Envia Cabecera
      Org.Mp !  origen; destino; tipo; longitud
      --}}}
      --{{{  Envia Parametros
      Org.Mp !  [ parametros FROM 0  FOR longitud ]
      --}}}
      VerificacionEjecucion ()
    --}}}
  --}}}
  --{{{  Salir
  tipo = t.fin
    --{{{  Salida
    SEQ
      longitud := 0
      Org.Mp !  origen; destino; tipo; longitud
      SalidaCaracter ( nueva.linea )
      SalidaCadena ( " Ha Elegido Salir del Programa, " )
      SalidaCaracter ( retorno )
      SalidaCaracter ( nueva.linea )
      SalidaCadena ( " no Hay Que Teclear Parametros " )
      activo := FALSE
    --}}}
  --}}}
  TRUE
    --{{{  Mandato No Definido
    SEQ
      SalidaCaracter (retorno)
      SalidaCaracter (nueva.linea)
      SalidaCadena ( " No ha tecleado un mandato definido " )
      SalidaCaracter (retorno)
      SalidaCaracter (nueva.linea)
      SalidaCadena ( " Intente otra vez " )
    --}}}
--}}}
DistanciaEntreTablas ( 2 )
VolverTds2 ()
--}}}
--
--}}}F
--...F code
--:::A  1 2
--:::F Organi28.dcd
--...F descriptor
--:::A  1 4
--:::F Organi28.dds
--...F link
--:::A  1 9
--:::F Organi28.dlk
--...F debug
--:::A  1 5
--:::F Organi28.ddb
--...F CODE EXE Grafic01 - Organizacion de las Operaciones Graficas
--:::A  2 13
--:::F Organi28.cex
```

```
--}}}
--{{{F paramini
--:::F parami01.tsr
-- Paleta inicial de colores
0
--Nombre del fichero de Fuente de caraceteres
8x12
--}}}F
```