

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
PÓS-GRADUAÇÃO EM CIENCIA DA COMPUTAÇÃO

28/407

Implementação de um Compilador para

Software Numérico

por

Javier García López



UFRGS

SABi



05232621

Dissertação submetida como requisito parcial  
a obtenção do grau de Mestre em  
Ciência da Computação

Prof. Paulo Alberto de Azeredo

Orientador

Porto Alegre, 27 de Setembro de 1987

UFRGS  
BIBLIOTECA  
CPD/PCCG

Garcia, Javier López

Implementação de um compilador para  
Software Numérico .

Porto Alegre, PGCC da UFRGS, 1987.

iv.

Diss(mestr. ci. comp.)

UFRGS - PGCC, Porto Alegre, BR-RS, 1987.

Dissertação : estruturas numéricas: Data  
Types: compilador.

## A G R A D E C I M E N T O S

Ao Professor Paulo Alberto de Azeredo, pela orientação recebida durante o desenvolvimento deste trabalho

Ao Professor Dalcídio M. Cláudio, pelo constante apoio e incentivo recebido desde minha chegada ao Curso .

A CAPES e CNPq pelo auxílio financeiro recebido através da Bolsa de Estudos .

Aos meus pais, irmãos e meu avô Cornelio, por acreditar sempre na minha pessoa .

À minha esposa Irene, pela compreensão e incentivo recebido durante a elaboração deste trabalho .

Aos meus amigos Carlos Dulanto Orellana, Mário Dávila Dávila e Armando Venero Baldeón, pelo incentivo e ajuda dados; sem os quais não teria sido possível chegar até aqui .

Ao pessoal administrativo e docente do Departamento de Informática do CPD e do Curso de Pós-Graduação em Ciência da Computação da UFRGS .

Aos meus colegas do curso pelas sugestões recebidas durante o desenvolvimento deste trabalho .

Aos meus colegas do grupo de Matemática Computacional, pelo constante apoio e companherismo .

" A los que luchan la vida entera ,  
porque son insustituibles "



## S U M Á R I O

---

Resumo

Abstract

Introdução.....	1
<b>1 O Matemático e o Computador</b>	
1.1 Introdução .....	5
1.2 O Computador e o Cálculo Numérico .....	5
1.3 O Usuário Matemático .....	6
1.4 As Linguagens de Programação e os Usuários	
Matemáticos .....	8
1.4.1 Fortran .....	9
1.4.2 Basic .....	12
1.4.3 Pascal .....	15
1.4.4 Pascal-SC .....	17
<b>2 Pascal-M</b>	
2.1 Introdução .....	18
2.2 Porque Pascal-M ? .....	18
2.3 Novas Estruturas Numéricas .....	21
2.3.1 Complexos .....	21
2.3.2 Intervalos .....	22
2.3.3 Matriz .....	24
2.3.4 Operações Relacionais .....	26
2.3.5 Operações Aritméticas .....	26
2.4 Criação de Novos Tipos .....	28
2.5 Outras Extensões .....	32

### 3 Complex, Segment e Matriz : Implementação

3.1	Introdução .....	33
3.2	Implentação.....	33
3.2.1	Complex .....	33
3.2.2	Segment .....	33
3.2.3	Matriz .....	35
3.3	Entradas e Saidas nos Novos Tipos Numéricos...	36
3.3.1	Entrada e Saida para Complexos .....	36
3.3.2	Entrada e Saida para Segment .....	37
3.3.3	Entrada e Saida para Matrizes .....	41
3.4	Operações nos Tipos COMPLEX, SEGMENT e MATRIZ .	47
3.4.1	Operações em Complex .....	49
3.4.2	Operações em Segment .....	50
3.4.3	Operações em Matriz .....	51
3.5	Funções de Biblioteca nos Tipos Complex, Segment e Matriz .....	60
3.5.1	Funções para Segment .....	61
3.5.2	Funções para Complex .....	64
3.5.3	Funções para Matriz .....	66

### 4 Implementação do DATA TYPE

4.1	Introdução .....	72
4.2	Tipos Abstratos de Dados .....	72
4.3	DATA TYPE .....	75
4.3.1	Sintaxe para <list-data-type> .....	76
4.3.2	Comentários sobre a Sintaxe .....	77

4.4	Compilação do DATA TYPE .....	78
4.4.1	Construção das Tabelas de Tipos e Constantes .....	78
4.4.2	Tabela de Operadores .....	79
4.4.3	Decomposição do DATA TYPE .....	79
4.5	Limitações do DATA TYPE .....	81
<b>5</b>	<b>Implementação</b>	
5.1	Introdução .....	82
5.2	Ambiente de Programação .....	82
5.3	O Programa do Usuário .....	92
5.4	Recuperação de Erros .....	94
5.5	Limitações .....	97
5.6	O Pascal-M e o Pascal-SC .....	98
	<b>Conclusões</b> .....	100
	<b>Anexo I : Manual do Editor</b> .....	102
	<b>Anexo II : Manual do Gerenciador</b> .....	104
	<b>Bibliografia</b> .....	108

## R E S U M O

---

Este trabalho apresenta proposta e implementação de uma linguagem de programação orientada para usuários de aplicações técnicas e matemáticas. São discutidas as características da linguagem desde o ponto de vista da aplicabilidade e da facilidade de utilização.

É feito um estudo genérico de como as linguagens mais usadas, por este tipo de usuário, abordam os problemas de manipulação de estruturas numéricas especiais como Complexos, Intervalos e Matrizes. Esse mesmo estudo mostra como é feita a implementação dos Tipos Abstratos de Dados e os problemas que isto acarreta nessas linguagens.

Finalmente é apresentada uma forma de implementar as estruturas numéricas acima, os Tipos Abstratos de Dados e o gerenciador da linguagem, o qual denominamos PASCAL-M.

## A B S T R A C T

This work presents a proposal and implementation of a programming language oriented to applied mathematics and engineering. The characteristics of the language are discussed from the point of view of both application and utilization aspects.

A generic study has been done to find out a way of implementing numerical structures such as complex numbers, intervals and matrices, as well as abstract data types on several programming languages.

Finally, an implementation of numerical and abstract data type structures on Pascal, and a language manager, is presented. The extended language is called Pascal-M.

## INTRODUÇÃO

O desenvolvimento dos computadores iniciou-se como um auxílio nas tarefas numéricas das áreas técnicas e científicas . Em consequência as primeiras linguagens de alto nível, FORTRAN e ALGOL, foram projetadas para esse fim . Aos poucos foi-se percebendo o potencial que teria o uso do computador nas aplicações comerciais, nas quais o problema fundamental não é a complexidade dos cálculos, mas sim o volume de informações . Assim, surgiu o COBOL e logo um conjunto de outras linguagens ou pacotes. Essas linguagens visavam a ser cada vez mais acessíveis ao usuário . Isto deu origem às chamadas linguagens de quarta geração [GHE 85], as quais permitem que o usuário programe em forma não-procedural, isto é, sem se preocupar como a tarefa é resolvida.

Esse desenvolvimento não foi acompanhado pelas linguagens usadas para as aplicações técnicas e científicas, as quais, nesse aspecto, têm permanecido quase inalteráveis. Assim, por exemplo, os usuários que têm que multiplicar duas matrizes A e B, em vez de fazer  $A*B$ , devem fazer uma rotina para implementar tal operação, isso se torna grave se levarmos em conta que, assim como nas aplicações comerciais, nem todos os usuários desse tipo são experientes em programação nem têm interesse nisso .

O objetivo de nosso trabalho é desenvolver um compilador que possa, nas estruturas básicas da Matemática, não atingidas pelos atuais compiladores, como é o caso de Matrizes, números Complexos e Intervalos, desenvolver a Aritmética fundamental associada a ela. Além disso, pretendemos que os TIPOS ABSTRATOS DE DADOS, sejam utilizados por esse tipo de usuário para construir as outras estruturas que não deixam de ser importantes por menos usadas, como é o caso de números racionais e tensores por exemplo. Neste tipo de dados o usuário teria a possibilidade de definir as operações associadas a essas estruturas. Isso também poderia ser aproveitado por outro tipo de usuário matemático, os Algebristas, estes últimos poderiam construir estruturas como anéis, grupos, etc. Desta maneira poderiam testar experimentalmente a estrutura e depois tentar a prova formal das propriedades destes conjuntos. Logo, nosso trabalho cumpriria a dupla finalidade de tentar uma linguagem não procedural para tarefas matemáticas, e por outro lado, incorporaria uma nova fatia de usuários de computadores. Ambas metas justificam amplamente a tarefa que aqui estamos começando .

O trabalho aqui desenvolvido está organizado da seguinte maneira : no Capítulo 1 é apresentado um estudo dos Recursos Computacionais das diversas linguagens que estão sendo usadas por usuários matemáticos, para os quais está orientado este trabalho, o objetivo desse Capítulo é fazer um levantamento das necessidades neste tipo de usuário que não estão sendo

satisfeitas pelas linguagens utilizados por ele .

No Capítulo II é apresentada uma proposta de uma nova linguagem destinada a usuários matemáticos . Ali são discutidas as exigências mínimas que essa linguagem deve possuir, visando a superar as inconveniências das linguagens atualmente utilizadas . Junto com essa apresentação será feito um estudo das possibilidades de implementar tais exigências. Discutido isso, será formulada uma proposta concreta. Esta pode se resumir na criação de novos tipos de dados os quais serão embutidos na linguagem. Estes tipos são os Intervalos, os Complexos e as Matrizes. O fato de incluir estes tipos se deve ao múltiplo uso que eles têm nas diversas aplicações, não só matemáticas como técnicas em geral . Como isso pode não ser suficiente, é oferecida a possibilidade de uma manipulação ampla de TIPOS DE DADOS ABSTRATOS [VEL 86]. Isso é feito pensando-se na possibilidade de deixar um espaço aberto à criatividade, tão comum neste tipo de usuários .

No Capítulo III é apresentada a forma como são implementadas as novas estruturas SEGMENT(Intervalos), COMPLEX e MATRIZES . Junto com isso é apresentado o conjunto de operadores destes tipos e as funções de biblioteca associadas a eles. Na sua implementação são apresentadas as suas características e as restrições no seu uso .

No Capítulo IV é apresentada a implementação do Gerenciador



do DATA TYPE, estrutura incorporada à linguagem com a finalidade de implementar uma manipulação ampla e quase irrestrita de TIPOS ABSTRATOS DE DADOS [VEL 86]. Basicamente será apresentada a sintaxe, a qual inclui a definição do tipo e a definição dos operadores sobre ele. A seguir é apresentada a forma como vão funcionar estes tipos e seus operadores internamente, tudo isso transparente ao usuário. Finalmente, serão apresentadas as limitações deste gerenciador e as suas causas.

No Capítulo V é apresentada a forma como as partes desenvolvidas nos Capítulos III e IV são ligadas. Será apresentado o PRÉ-PROCESSADOR e as razões pelas quais foi escolhida esta metodologia e também as suas limitações. Finalmente será apresentada a forma como os erros serão tratados dentro e fora do PRÉ-PROCESSADOR.

O trabalho inclui uma avaliação preliminar do compilador desenvolvido, na qual se faz um levantamento das metas alcançadas e as passíveis de se conseguirem em futuros trabalhos.

O trabalho inclui um apêndice, destinado aos usuários, no qual se mostra a forma como pode ser usado este compilador.

## C A P Í T U L O I

## O MATEMÁTICO E O COMPUTADOR

## 1.1 Introdução

Neste Capítulo vamos dar uma revisão geral sobre as necessidades de um usuário que lida com problemas matemáticos em geral e, paralelamente a isso, estudaremos as principais características das linguagens de programação que servem, ou deveriam servir, para atender essas necessidades. Neste estudo tentaremos estabelecer os alcances e limitações desses recursos.

## 1.2 O Computador e o Cálculo numérico

Os computadores foram inicialmente projetados para auxiliar ao homem em cálculos numéricos complexos, entendendo-se por complexos os cálculos que envolvem uma grande quantidade de operações e exige uma alta exatidão no cálculo dos resultados (accuracy) [REI 81]. É conveniente salientar que na época em que foram feitos tais projetos pensava-se que os usuários do computador seriam pessoas treinadas para esse fim.

Posteriormente, o uso do computador estendeu-se ao processamento de grande volume de dados. Nesse tipo de aplicação a dificuldade não era mais complexidade numérica. O volume de dados e a velocidade com a qual deviam ser processados passou a provocar mais atenção. A velocidade nas respostas é mais

importante que o número de dígitos decimais das mesmas, pois nessas aplicações nunca se trabalha com mais de dois dígitos de mantissa.

Com o surgimento dos microcomputadores e sua posterior "popularização", as linguagens de programação já existentes tiveram que se adequar à nova situação. Os usuários já não seriam sempre programadores experientes, mas sim usuários de todos os níveis (de leigos a pessoal altamente experiente). Os compiladores das diversas linguagens deveriam adequar-se a essa situação. Nesse processo de adequação, geralmente só se pensou nos usuários de aplicações comerciais, cujo número é consideravelmente maior que os usuários de aplicações técnicas. Para corroborar esse fato citaremos o caso do compilador FORTRAN. Quando foi adequado para microcomputadores, desconsiderou-se a declaração COMPLEX, e todas as funções associadas a ela, pois ao que se saiba, nenhuma aplicação de ordem comercial utilizaria tal estrutura numérica.

### 1.3 O Usuário Matemático

O usuário matemático que lida com problemas matemáticos tem algumas características e necessidades especiais que o distinguem de outro tipo de usuário. Compreendendo este tipo de usuário teremos mais subsídios para projetar uma linguagem que se enquadre dentro das suas expectativas.

Dentro das características mais marcantes, poderíamos assinalar as seguintes :

- Geralmente não é um programador experiente.
- Quando usa o computador, as linguagens de programação que mais usa são FORTRAN, BASIC e PASCAL .
- Utiliza o computador para realizar cálculos que envolvem um grande número de operações, as quais envolvem, geralmente, estruturas numéricas, como matrizes, intervalos, números reais de dupla precisão e números complexos.
- Sua capacidade para gerar novas estruturas numéricas e operações sobre elas é ilimitada e em muitos casos essas estruturas serão de seu uso exclusivo .
- Possui uma grande capacidade de abstração .
- Esse tipo de usuário gosta da formalização do conhecimento, o que o leva à exigência de linguagens bem definidas e sem ambigüidades .
- Em geral, os resultados que ele precisa devem ser calculados com uma alta exatidão .

Vistas essas características as necessidades do usuário matemático dentro de uma linguagem de programação seriam as seguintes :

- Uma linguagem que possua todos os tipos de dados que ele precisa e, junto a isso, os operadores e funções associadas a tais tipos . Entre esses tipos vamos citar REAL, INTEIROS, MATRIZ, INTERVALOS e COMPLEX . Não consideramos o caso de vetores, pois esta estrutura pode ser vista como uma Matriz Particular. A Matriz de ordem  $1 \times N$ (Matriz linha) .

- Recursos que permitam a criação de novos tipos de dados e operadores associados a eles . Isto deve se estender inclusive à possibilidade de mudar as operações usuais nos tipos tradicionalmente usados. Esta aplicação é muito freqüente entre os algebristas, fatia de usuários que ainda não descobriu o potencial do computador em suas pesquisas .

- Alta exatidão nos cálculos .

- Um editor de linguagem, no qual as tarefas de edição e correção de erros seja direta e acessível do mesmo modo que a compilação e execução dos programas . Isto é muito importante, pois nem todos os usuários serão programadores experientes .

#### **1.4 As Linguagens de Programação e os Usuários Matemáticos**

Nesta parte vamos revisar como as diversas linguagens usadas por usuários matemáticos conseguem ou tentam conseguir atingir as suas exigências vistas no parágrafo anterior. Para uma melhor compreensão desta parte, vamos seguir uma sequência histórica.

Assim vamos começar com o FORTRAN, e em seguida veremos, por causa de sua popularidade nos microcomputadores, o BASIC linguagem muito difundido (entre os usuários de microcomputadores de pequeno porte) e finalmente vamos ver o PASCAL uma linguagem que vem sendo muito usada pelos usuários técnicos e matemáticos nos últimos tempos.

#### 1.4.1 FORTRAN

Esta linguagem [IBM 82] é a mais antiga linguagem de alto nível, foi projetada para ser usada fundamentalmente em aplicações técnicas e científicas, mas também pensada para ser usada em equipamentos de grande porte. Ela possui os mais diversos tipos numéricos: REAL, INTEGER e COMPLEX. Por sua vez esses tipos podem ser tratados usando-se o critério de meia ou dupla palavra (REAL\*8, INTEGER\*2, COMPLEX\*16) para otimizar o uso da memória ou para aumentar o número de casas decimais nos dados. Quanto aos operadores para estes tipos numéricos ela possui operador Soma(+), diferença(-), Produto(\*), Divisão(/), Potencia(\*\*) e Raiz quadrada (só para REAL e COMPLEX, mas não para INTEGER). Além disso, para cada um destes tipos a linguagem possui uma vasta biblioteca de funções. Nela podemos achar desde funções trigonométricas (Seno(Sin), Coseno(Cos), Arctang(atan)), funções transcendentais (Ln, Log(base 10), Exp), Funções de Conversão de Tipo (FLOAT, IFIX, SNGL, DBLE), funções para máximos e mínimos (MAX0, MAX1, AMAX0, AMAX1, MIN0, MIN1, AMIN0, AMIN1), funções para Sinal (que não corresponde à função sinal do Análise

Matemático), Valor Absoluto para todos os seus tipos numéricos e, nas versões destinadas a equipamentos de grande porte, funções especiais, como é o caso da função GAMMA ou da função ERRO.

Estruturas como MATRIZES, VETORES e INTERVALOS podem ser construídas fazendo-se uso do comando DIMENSION, porém isso só serve para que os elementos de cada uma dessas estruturas seja distinguido um do outro no armazenamento na memória. Nada é dito sobre os elementos dessas estruturas. Isso se complica no caso dos INTERVALOS, nos quais deve cumprir-se uma certa relação de ordem. Quanto às operações estas terão que ser construídas elemento a elemento. Não estão previstas operações nas quais se trabalhe com a estrutura como um todo. Os Únicos casos previstos nesse sentido são para as operações de E/S, onde é possível fazer a leitura ou escrita do arranjo invocando simplesmente seu nome, porém isso só poderá ser feito no caso em que usemos matrizes de idêntica forma à definida no início do programa. O caso de entrada de dados para arranjos usando o comando DATA é similar.

Com respeito a TIPOS ABSTRATOS de dados nada está previsto. O uso deles vai depender da habilidade do programador para manipular os DIMENSION e estruturar operações usando FUNCTION ou SUBROUTINE. Mesmo para um programador experiente será muito difícil fazer os testes de consistência que garantam que as novas estruturas criadas, usando estes TIPOS ABSTRATOS, possam ser fechadas com respeito a seus operadores. [VEL 86]

Com o tempo o FORTRAN foi evoluindo, sendo a mudança mais notável dos últimos tempos, em termos de programação estruturada, a aparição do FORTRAN 77 [FRI 77], o qual incorpora estruturas de blocos através do :

```
WHILE( < condition > )  
  
  < lista-comandos >  
  
ENDWHILE
```

Adiciona o ELSE ao IF . Isso possibilita que o comando IF possa estabelecer duas listas de comandos mutuamente exclusivas para o caso de a condição ser verdadeira ou falsa . Para terminar o IF sera necessário o comando ENDIF, no fim dos comandos correspondentes ao ELSE. A sintaxe deste comando ficaria assim :

```
IF ( <condition> ) THEN  
  < list-comandos >  
ELSE  
  < list-comandos >  
ENDIF
```

O FORTRAN continua sendo ainda hoje uma linguagem muito usada pelos usuários das aplicações técnicas em geral. Isto pode ser facilmente conferido, se nós revisarmos a literatura e as bibliotecas de programas referentes a aplicações dessa natureza . Vamos encontrar que a maioria dos programas estão escritos em



FORTRAN . Por esta razão é que, quando o uso dos microcomputadores se massificou, foi necessário adequar um compilador FORTRAN para este tipo de equipamento . A versão da linguagem escolhida foi FORTRAN 77, mas se tomou cuidado para que o FORTRAN IV( a versão mais usada da linguagem) estivesse contido nela . Nesse traslado o FORTRAN perdeu o tipo COMPLEX, bem como as funções de Biblioteca especiais como é o caso da função ERRO e da função GAMMA . Portanto o usuário tem uma nova tarefa a implementar .

#### 1.4.2 BASIC

Esta linguagem [IBM 82a], que ressurgiu com os microcomputadores, foi projetada para ser usados em equipamentos de pequeno porte e sem muitos recursos. Por essa razão seu uso tornou-se frequente quando os microcomputadores passaram a ser uma ferramenta de trabalho nos afazeres técnicos e científicos . Apesar de os microcomputadores terem aumentado consideravelmente sua capacidade, o que permite o uso de outras linguagens de programação, o BASIC ainda é muito usado .

Quanto a tipos numéricos, ela possui os tipos REAL e INTEGER, os quais são definidos como tais a partir dos dados que vai recebendo durante a execução do programa . Essa declaração é automática, porém o usuário pode utilizar declarações específicas para indicar os tipos de dados. Alguns interpretadores incluem a possibilidade de criação de REAL de dupla precisão. Esses tipos numéricos possuem os mesmos operadores que o FORTRAN, porém, como

não têm tipos Pré-definidos, as funções podem ser usadas indistintamente por qualquer variável do tipo numérico. Assim, por exemplo, a função SQRT pode calcular a raiz quadrada de uma variável real ou inteira. Em ambos os casos o resultado será considerado do tipo REAL. No caso do FORTRAN, a tentativa de calcular a raiz de uma variável inteira usando SQRT seria considerada, pelo compilador, um erro.

Além disso possui funções de biblioteca, as quais tratam, em geral, seus argumentos como se fossem valores reais. As funções numéricas que a linguagem possui são : trigonométricas(Seno, Coseno, Tangente e Arco tangente), Transcendentes(Ln e Exp), Valor Absoluto, função Sinal e funções de truncamento. Como se pode perceber facilmente, essa biblioteca é notoriamente inferior(em quantidade) à biblioteca do FORTRAN. Quanto a Complexos, a linguagem não possui um tipo especial para eles. Este pode ser construído usando a declaração DIM. Os operadores e as funções dos Complexos terão que ser programados em separado por meio de rotinas.

Com relação aos INTERVALOS, MATRIZES e VETORES, eles podem ser usados, declarando-se-os previamente na declaração DIM. O comando DIM, como o comando DIMENSION do FORTRAN serve para a criação de variáveis com índice, porém permite o uso do índice 0 inferiores a 10, é dispensável a declaração DIM. Isso seria aplicável aos INTERVALOS e aos COMPLEXOS. Uma vez declarada a

variável no comando DIM, para usá-la implicitamente é necessário indicar seus índices entre parênteses . Com respeito aos operadores associados a esses tipos, o comportamento será similar ao caso do FORTRAN. Versões iniciais do BASIC, sobretudo as desenvolvidas para máquinas de médio porte, prevêem comandos como : MAT READ, MAT WRITE, MAT S = A + B, entre outros, os quais servem para se operar com matrizes, só que nem sempre essas operações correspondiam à realidade matemática, No caso dos INTERVALOS, os quais seriam tratados como se fosse uma matriz linha, não teríamos nem o teste de menor ou igual, tampouco os operadores associados .

Para o caso de TIPOS ABSTRATOS DE DADOS a linguagem também não prevê nada. Estes poderão ser criados, quando for possível, usando-se DIM e os operadores destes eventualmente serão criados usando-se DEF FN. Como se pode notar isto é muito restrito e vai depender da habilidade do programador, pois a linguagem não oferece muitas possibilidades. Além do mais, o trabalho dos tipos abstratos, que possam ser criados, tem um tratamento similar aos tipo REAIS e INTEIROS.

Uma restrição forte para o uso desta linguagem é o fato de suas rotinas não serem transportáveis, isso pelo fato de não trabalhar com variáveis "mudas". O único caso no qual isto é permitido é no caso das funções(DEF FN), porém essas só permitem um comando. A única exceção que o autor conhece neste sentido é o interpretador de BASIC do HP85 [HP-85], o qual permite a definição de funções de mais de um comando.

Outra restrição desta linguagem é o fato de que, na maioria dos casos, o BASIC é uma linguagem Interpretada e não compilada. Isso faz com que os programas de grande porte, que nessas aplicações são frequentes (Inversão Numérica de Laplace usa em torno de 600 comandos no programa da Biblioteca da ACM) [PIE 84], sejam muito demorados.

Finalmente poder-se-ia citar como uma das deficiências da linguagem o fato de ser, em geral, não apropriada para desenvolver programas estruturados, por não possuir comandos do tipo IF <Condition> THEN <Comando> ELSE <Comando> e quando os possui estes só podem ter comandos simples ou obrigam ao usuário ao uso de separadores de comandos ( ; ou @ ) o que fará com que os programas se tornem muito difíceis de ser entendidos, inclusive pela pessoa que o fez. Além do mais, a maioria de interpretadores carece de comandos do tipo WHILE .

#### 1.4.3 PASCAL

Linguagem [WEL 83] da linha sucessora do ALGOL ( Algorithm Language). Por essa razão foi pensada para desenvolver Programas a partir do algoritmo, que trata um problema grande como a decomposição em um conjunto de problemas pequenos, cada um dos quais será resolvido em um dos blocos do programa. Nesse sentido é uma linguagem ótima para o desenvolvimento de programas estruturados.

Quanto a tipos Numéricos, ela possui REAIS e INTEIROS. No caso dos Reais ela trabalha basicamente com dupla precisão. Para os inteiros, estes estão limitados pelas constantes pré-definidas MAXINT e MININT, cujos valores, em geral, não vão além de 6 cifras (No Turbo Pascal, uma versão muito usada do Pascal, são 32767 e -32768 respectivamente). A idéia é a de que as aplicações numéricas em geral sejam desenvolvidas em Aritmética Real. Os operadores para esses tipos incluem uma novidade nesta parte, que é o cancelamento do operador potência, o qual, no julgamento do projetista da linguagem, introduz muitos vícios no programador. Além disso possui uma biblioteca de funções, a qual contém funções trigonométricas (Sin, Cos e Arctg), transcendentais (Ln e Exp), potência (Sqrt(Raiz) e Sqr(quadrado)) e funções como Valor Absoluto, Truncamento (Parte Inteira e Parte Racional).

Quanto aos Complexos, não existe um tipo pré-definido e muito menos, operadores para eles. Esses podem ser usados a partir da declaração TYPE, sendo que os operadores deverão ser construídos usando-se rotinas. O mesmo acontece para funções sobre eles. Da mesma forma que no caso do tipo COMPLEX podemos, na seção TYPE, definir os tipos SEGMENT (Para Intervalos) e MATRIZ. No caso do tipo SEGMENT a criação do tipo não fornece nenhuma ferramenta para testar se o primeiro elemento do intervalo é menor ou igual ao segundo.

Com respeito aos TIPOS ABSTRATOS, ele fornece a possibilidade de criá-los como tais, porém não oferece

facilidades para implementar operadores sobre eles . Isto já representa um avanço sobre as outras linguagens vistas anteriormente, pois a partir dos tipos criados poderemos construir rotinas específicas. Essas rotinas poderiam ser gravadas em arquivos separados e usadas por diversos usuários através de inclusão de arquivos .

#### **1.4.4 Pascal-SC**

Esta linguagem [NEA 85] foi desenvolvida em Karlsruhe, em Agosto de 1985, a partir do Pascal standard e, como nosso trabalho pretende se converter numa ferramenta para as pessoas que trabalham com programação de aplicações técnicas e científicas, para isso incorpora, entre outras coisas, os tipos SEGMENT para os intervalos, COMPLEX para os números complexos e MATRIZ para as matrizes. Esses tipos e as operações e funções associadas a elas são incorporadas através de bibliotecas, as quais para serem usadas, deverão ser incluídas usando-se o comando de inclusão do Pascal standard( (\$i <nome-biblioteca> ) ). Além disso desenvolve um sistema composto por um editor(SEP), um compilador(CMP) e um executor(XQP) para facilitar as tarefas do usuário. Voltaremos a este assunto na seção 5.6 .

## C A P Í T U L O   I I

## P A S C A L - M

## 2.1 INTRODUÇÃO

Neste capítulo, baseado no estudo feito no capítulo anterior, vamos fazer uma proposta de linguagem que permita ao usuário de aplicações técnicas e matemáticas programar usando todos os tipos numéricos que ele usa cotidianamente, que estes tipos possuam bibliotecas de funções que lhe forneçam resultados de uso importante. Além disso, essa linguagem deixa aberta a possibilidade não somente de criar novos tipos, como também operadores sobre eles. Este último recurso está ligado ao fato de que muitas estruturas numéricas podem não ser freqüentemente usadas por um tipo de usuário especial. Dentro dessa fatia de usuários poderiam ser citados os "algebristas", os quais freqüentemente trabalham com estruturas abstratas.

## 2.2 Por que PASCAL-M ?

No nosso estudo feito sobre as linguagens de programação desenvolvidas para microcomputadores e usadas pelos usuários de aplicações técnicas e/ou matemáticas, podemos observar que todas elas trabalham com dois tipos numéricos: REAL e INTEGER sendo que o FORTRAN, através da declaração REAL\*8 ou DOUBLE PRECISION, e o BASIC, através da declaração DEFDBL, permitem a distinção

explícita entre simples e dupla precisão . O PASCAL trabalha, no caso REAL, sempre com dupla precisão, porém armazena estes números em 6 bytes, diferente do BASIC e o do FORTRAN que utilizam 8 bytes para a mesma tarefa.

Além disso, temos que, para a criação de novas estruturas numéricas, via TIPOS ABSTRATOS DE DADOS, no BASIC e no FORTRAN esta é implícita e depende da habilidade do programador para criar rotinas, funções e arranjos para simular os tipos e seus operandos. Mesmo assim o usuário fica bastante limitado. No PASCAL existe uma seção para a criação explícita de novos tipos, o TYPE. Esses novos tipos poderão ser usados como tais nas PROCEDURES, FUNCTIONS e no programa principal, porém a criação de operações sobre eles dever ser feita usando FUNCTIONS e/ou PROCEDURES. Portanto, também dependerá da habilidade do programador

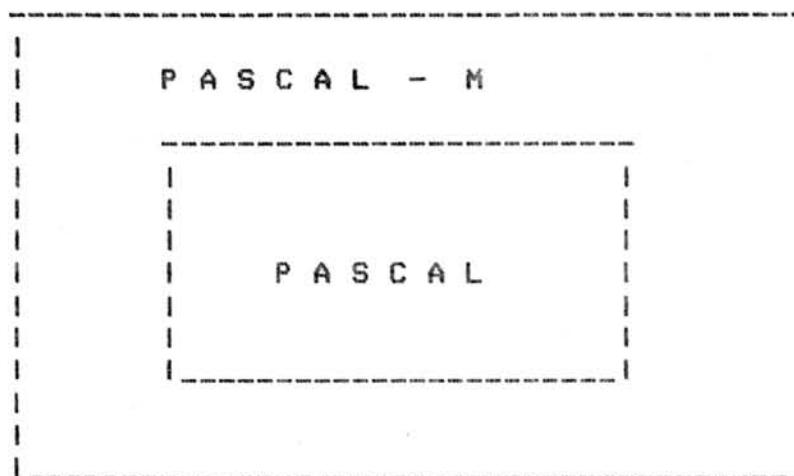
Finalmente o PASCAL, comparado com o BASIC e o FORTRAN, oferece uma quantidade maior de recursos computacionais para o desenvolvimento de programas modulares[ALA 78]. Isso não somente vai permitir a construção de programas de fácil leitura e manutenção, como também deixa aberta a possibilidade de ligação de programas com rotinas em ASSEMBLER ou no proprio PASCAL, ligação com rotinas prontas de outras aplicações ou inclusão e exclusão de arquivos(Usando INCLUDE). Este último recurso usado pelo Pascal-SC para usar as bibliotecas de Intervalos, Complexos e Matrizes. Além disso, é possível o uso de rotinas superpostas,



usando-se OVERLAY, este recurso permite um uso mais ótimo da memória.

Essas técnicas permitem, também, uma otimização no uso da memória. É preciso salientar que as inclusões podem também ser encontradas no FORTRAN 77. No BASIC pode ser simulada através do MERGE ou CHAIN, mas vê-se restringida por que as rotinas trabalham sem parâmetros.

Por estas razões é que nossa proposta vai tomar como base a linguagem PASCAL e estendê-la de maneira que incorpore as estruturas numéricas COMPLEX, SEGMENT e MATRIZ. Além disso devemos estender os limites do TYPE de maneira que permita, se necessário, a criação de operadores sobre os tipos que estão sendo criados. Essa versão de PASCAL será chamada PASCAL-M, e esquematicamente poderia ser vista assim :



Quer dizer que todo programa que "rodava" em PASCAL deve "rodar" em PASCAL-M .

## 2.3 Novas Estruturas Numéricas

Conforme já vimos anteriormente, nossa linguagem vai incorporar novos tipos numéricos e estender os limites na seção TYPE do Pascal standard. Dentro dos tipos numéricos a serem incluídos temos SEGMENT, para intervalos, COMPLEX, para números complexos e MATRIZ, para matrizes.

Não vamos considerar o tipo VETOR, pois este pode ser visto como uma matriz linha ou coluna [LUD 81].

### 2.3.1 Complexos

Para declarar uma variável do tipo número complexo, da forma  $a + ib$  usaremos a palavra chave COMPLEX.

Exemplo `VAR x,y: COMPLEX;`

Neste caso tanto  $x$  como  $y$  serão tratados como um par ordenado  $(a, b)$ , o qual representará o complexo  $a+ib$ . [HAU 72]

Para o caso de constantes do tipo COMPLEX usaremos a notação

`(cte, cte)`

onde  $cte$  é um real qualquer. No caso de precisarmos usar, numa expressão, uma parte do número complexo (sua parte real ou imaginária), a sintaxe será a seguinte:

<nome>. <ext>

onde        <ext> := real / imag

É preciso salientar que para o caso dos complexos não existe relação de ordem, entretanto essa pode ser necessária no PASCAL-M, seja no comando IF ou no comando WHILE, para avaliar expressões do tipo

<complex>    <op\_rel> <complex>.

onde        <op\_rel> ::= < | <= | > | >= | =

Para o caso = e <> seguiremos os critérios previstos em [HAU 72].

Para o caso das relações maior e menor, serão definidas da seguinte maneira:

Dado  $Z_1 = a_1 + ib_1$  e  $Z_2 = a_2 + ib_2$

$Z_1 > Z_2$     se     $a_1 > a_2$     e     $b_1 > b_2$

$Z_1 < Z_2$     se     $a_1 < a_2$     e     $b_1 < b_2$

### 2.3.2 Intervalos

Para declarar uma variável do tipo intervalo usaremos a palavra chave SEGMENT OF <tipo> onde <tipo> pode ser REAL / INTEGER / COMPLEX.

Esta nova estrutura terá que preservar as propriedades inerentes a todo intervalo [GAR 85], isto é que o primeiro elemento deve ser menor ou igual ao segundo. Sendo que no caso de complexo, usaremos o critério definido no parágrafo anterior.

Para escrever uma constante intervalar usaremos a seguinte notação:

$$[cte_1 ; cte_2]$$

Com as seguintes restrições:

1.  $cte_1 \leq cte_2$
2.  $cte_1$  e  $cte_2$  devem ambas ser do mesmo tipo.

No caso de precisarmos trabalhar com uma parte da constante intervalar, numa expressão por exemplo, usaremos a seguinte sintaxe:

$\langle nome \rangle . \langle ext \rangle$ ,

onde  $\langle ext \rangle := \text{inf} / \text{sup}$

Para o caso de relações de ordem: menor ( < ), menor ou igual ( <= ), maior ( > ), maior ou igual ( >= ) nos intervalos, vamos seguir os seguintes critérios:

$$A = [a_1 ; a_2] \text{ e } B = [b_1 , b_2]$$

A < B	a	<	b
	1		1
A > B	a	>	b
	2		2

Para o caso de igualdade e desigualdade usaremos o mesmo critério de [GAR 85].

### 2.3.3 Matrizes

No caso de Matrizes devemos ter um cuidado especial, dado que teremos que gerenciar índices e o PASCAL não permite a mudança dinâmica do valor máximo dos índices. Por essa razão, todas as matrizes serão tratadas tendo como base o valor máximo previsto para filas e colunas nas matrizes usadas. Isso ficará invisível ao usuário, entretanto pode causar um "estouro" na capacidade de armazenamento. Isso pode ser resolvido através dos DATA TYPE, seção do PASCAL-M na qual o usuário poderá construir um tipo associado a suas matrizes, e no qual ele mesmo gerenciará o uso da memória .

```
<lista_variaveis>: MATRIZ [<cte> , <cte>] OF <tipo>;
```

```
<cte> := será uma constante de tipo inteiro
```

```
<tipo> := REAL / INTEGER / COMPLEX / SEGMENT
```

```
Exemplo Var a,b : MATRIZ[10,20] OF SEGMENT
```

No caso de constante do tipo Matriz, só serão considerados os casos : da matriz identidade e da matriz nula, usando-se a seguinte notação

[cte \* <elemento unitário>]

[cte \* <elemento nulo>]]

<elemento unitário> := 1 / 1.0 / (1.0, 0.0) / [1.00 ; 1.00]

<elemento nulo> := 0 / 0.0 / (0.0, 0.0) / [0,0; 0.0]

<cte> := <inteiro positivo>

Exemplo

```
const ident = 10*1.0
```

Esta declaração produz uma Matriz identidade de 10x10 . Além disso quando declaramos, em CONST, ou usamos uma constante do tipo Matriz, essa é tratada automaticamente como uma matriz quadrada do tipo identidade ou nula. No caso de precisarmos trabalhar com um elemento da matriz, numa expressão, por exemplo, vamos usar a seguinte sintaxe:

<nome>.elemento[<cte>, <cte>]

Neste caso, as <cte> servirão para indicar a linha e coluna da matriz. Esta matriz deve estar previamente definida.

Exemplo A. elemento[2,4]

Refere-se ao elemento da linha 2 e da coluna 4 na matriz A.

A restrição de constantes do tipo MATRIZ só é considerada para os casos de matriz nula e identidade. Isto não é uma restrição muito forte, pois de fato, no cotidiano são as únicas constantes matriciais de uso regular [MUN 77].

#### 2.3.4 Operações Relacionais

Quanto às relações de ordem entre variáveis ou constantes do tipo matriz, somente serão considerados os caso de igualdade e desigualdade, nos quais o teste é feito termo a termo [MUN 77]. Dentro desses termos será adotado o critério de igualdade do tipo base da matriz (Real, Complexo ou Intervalo). Nesse caso seguiremos os critérios apresentados em [HAU 72], [GAR 85] e [CLA 84].

#### 2.3.5 Operações Aritméticas

Com respeito às operações aritméticas, tendo em conta a aritmética de cada um destes tipos, serão considerados os seguintes operadores:

Operador	Operando1	Operando2	Resultado
+, -, *, /	SEGMENT	SEGMENT	SEGMENT
+, -, X, /	COMPLEX	COMPLEX	COMPLEX

Operador	Operando1	Operando2	Resultado
*	REAL	MATRIZ(m,n)	MATRIZ(m,n)
+, -	MATRIZ (m,n)	MATRIZ(m,n)	MATRIZ(m,n)
*	MATRIZ(m,n)	MATRIZ(n,p)	MATRIZ(m,p)

Quer dizer que vamos usar os mesmos operadores que são usados no caso da aritmética inteira e real, porém esses trabalham na forma que corresponde a cada estrutura, sendo que o processo será invisível ao usuário.

Para as operações de entrada e saída serão usadas as seguintes funções:

```

READMAT (<LISTA_VARIAVEIS_TIPO MATRIZ>)
WRITEMAT(<LISTA_VARIAVEIS_TIPO MATRIZ>)
READLNSEG(list_variaveis_tipo_segment)
READSEG(list_variaveis_tipo_segment)
WRITESEG(list_variaveis_tipo_segment)
WRITELNSEG(list_variaveis_tipo_segment)
READLNCOM(list_variaveis_tipo_complex)
READCOM(list_variaveis_tipo_complex)
WRITECOM(list_variaveis_tipo_complex)
WRITELNCOM(list_variaveis_tipo_complex)

```

Para o caso do WRITEMAT ele fará a impressão da matriz na sua forma usual. Nesse sentido, cada vez que muda de matriz, automaticamente, muda de linha antes de começar a impressão da seguinte .



Com respeito às funções de biblioteca para cada um dos tipos COMPLEX, SEGMENT e MATRIZ serão consideradas as mesmas que o PASCAL prevê para os tipos REAL e INTEGER. Além disso, deve se considerar algumas funções intrínsecas a cada tipo. Por exemplo:

Para COMPLEX : Conjugado

MATRIZ : Determinante e Inversa

SEGMENT : Diâmetro, Ponto médio, Pertença

#### 2.4 Criação de Novos tipos

Outra facilidade que iremos incorporar à linguagem é a possibilidade de o usuário criar novos tipos de dados, nos quais, além da estrutura de representação, especificará as operações e, se for o caso, relações de ordem. Essas operações ficarão disponíveis exclusivamente para variáveis declaradas sob esses tipos, podendo inclusive ser totalmente diferentes das definidas usualmente para eles. Como tínhamos citado anteriormente, esse recurso poderá ser usado pelos "algebristas", os quais frequentemente trabalham com estruturas abstratas.

A definição de um novo tipo é feita com uma unidade de programa denominado DATA TYPE. Cada DATA TYPE possui um nome, uma lista de parâmetros e outra de nomes de tipos, uma seção de definição de estrutura de dados e outra de definição de operações. Cada operação é implementada sob a forma de função ou procedure, e pode ser declarada, como visível ou não, ao usuário do tipo de dado sendo definido.

A seguir esboçaremos a sintaxe da seção DATA TYPE

```

<data type> := DATA TYPE <nome> IS
                <par list>
                <types names>;
                <data structure>;
                <operations>;
                END;

```

Onde :

```

<name> := <identificador>
<parlist> := PARAMETERS <parms>
<parms> := <par>; <par>; <parms>.
<par> := <list_identificadores> : <type>
<list_identificadores> := <identificador> <identificador>,
                        <list_identificadores>
<type_names> := TYPE NAMES (<name_list>)
<name_list> := <name> <name>, <name_list>
<operations> := OPERATION <oplist>
<oplist> := <operation> <operation>; <oplist>
<operation> := <procedure ou function declaration>
                EXPORT <procedure ou function declaration>

```

A definição de estrutura de dados é idêntica à seção <var <declaration part> do PASCAL [WEL 83]. A inclusão da palavra reservada EXPORT deve-se ao fato de algumas funções criadas no DATA TYPE serem visíveis ou invisíveis ao usuário.

Exemplo do uso do DATA TYPE :

DATA TYPE racional IS

TYPE NAMES (INTEGER)

DATA STRUCTURE

REGISTRO = RECORD

A, B: INTEGER

END;

OPERATIONS

FUNCTION MCD (A,B: INTEGER):INTEGER;

( Está função serve para expressar um racional  
na sua forma usual, digamos  $1/2$  em vez de  
 $2/4$  . )

VAR DIVIDENDO: INTEGER,

BEGIN

MCD := B;

DIVIDENDO := A;

R := DIVIDENDO MOD MCD;

WHILE R <> 0 DO

BEGIN

DIVIDENDO := MCD;

MCD := R;

R := DIVIDENDO MOD MCD

END;

END;

```

EXPORT PROCEDURE "+" (X,Y:RACIONAL;VAR Z:RACIONAL
VAR. F1, F2: INTEGER;
BEGIN
    F1 := X.A*Y.B + X.B+Y.B
    F2 := X.A*Y*B;
    Z.A := F1/MCD(F1,F2);
    Z.B := F2/MCD(F1,F2);
END;

```

Neste exemplo foi construído o tipo racional e criada a operação soma para ele. Observe-se que a operação MCD será absolutamente invisível no programa, mas é uma ferramenta importante no cálculo das operações entre racionais.

Para finalizar, vamos ver como fica a estrutura geral do PASCAL-M.

```

PROGRAM <nome>;
<list_const>
<list_types>
<list_datatype>
<list_VAR>;
<list_procedures ou funções>
BEGIN
    <list_comandos>
END.

```

Observe-se que o DATA TYPE vem logo após o TYPE. A idéia é manter a ordem natural do PASCAL, na qual é possível usarem-se as constantes ou tipos conforme estes vão sendo definidos. Assim, na seção DATA TYPE poderemos aproveitar as constantes e os tipos já criados e definidos e logo que precisarmos variáveis deste tipo, essas poderão ser declaradas na seção VAR, seja no Programa Principal ou em uma Procedure ou Função. Dessa forma, o processo de implementação do Pascal-M será facilitado, pois fica natural a conversão do Pascal standard para o Pascal-M.

## 2.5 Outras Extensões

Finalmente, queremos salientar que seria desejável incluir na linguagem rotinas de derivação de funções do tipo  $f(x)$  e quem sabe algumas de integração. No caso das integrais, pelo menos dando resultados de ordem numérica, dado que isso é muito usado nas aplicações de Ciências e Engenharia [BRE 85]. Mas julgamos que esse tema foge um pouco dos objetivos do presente trabalho, ficando como uma tarefa aberta para próximas extensões da linguagem proposta.

## CAPÍTULO III

## COMPLEX, SEGMENT e MATRIZ : IMPLEMENTAÇÃO

## 3.1 INTRODUÇÃO

Neste capítulo será apresentada a forma como serão implementados internamente os novos tipos numéricos, seus operadores intrínsecos e suas funções. Também serão estudados os tipos de parâmetros permitidos por estas e o tipo de resultado que elas devolvem. Quando a função estudada tem sua correspondente na aritmética real ou inteira, estabeleceremos as semelhanças e diferenças entre elas.

## 3.2 IMPLEMENTAÇÃO

A forma de implementar os tipos foi a seguinte : O tipo COMPLEX foi implementado como se fosse um RECORD da seguinte forma :

```
3.2.1   Complex = record
        real   : real ;
        imag   : real ;
        end ;
```

Dessa forma, por um lado, o COMPLEX seria tratado como um todo, o registro, e de outro, poderíamos distinguir facilmente cada uma de suas partes. Salienta-se o fato de que essas partes

estão sendo tratadas usando-se as formas usuais na Aritmética complexa REAL e IMAG .

Para o caso dos SEGMENT a forma será a seguinte :

Para o caso REAL

```
segment = record
    inf : real ;
    sup : real ;
end ;
```

Para o caso Complex :

```
segmentc = record
    inf : complex ;
    sup : complex ;
end ;
```

O caso de intervalos de inteiros não possui sentido matemático, portanto não será desenvolvido .

Para o caso do tipo MATRIZ nós vamos ter uma primeira parte de definição das linhas e colunas. Estas terão que ser definidas a partir de um limite constante, pois o Pascal não permite a mudança dinâmica dos limites. Por esta razão, as matrizes na realidade serão tratadas, todas elas, como se fossem quadradas, sendo a ordem atribuída o valor máximo usado pelo usuário para as linhas ou colunas das matrizes declaradas. Isto fica invisível para o usuário, quem trabalhará as matrizes como usualmente o faz

Além disso ao se definir uma matriz, os limites desta são armazenados numa tabela de matrizes para uso posterior. Logo que definimos os limites para as linhas e colunas, definiremos as estruturas propriamente ditas da seguinte maneira :

Para o caso Real :

```
MATRIZR = record
    numlinha,numcoluna : integer ;
    elemat           : array[1..maxlin,1..maxcol] of real;
end ;
```

Para o caso Integer :

```
MATRIZI = record
    numlinha,numcoluna : integer ;
    elemat : array[1..maxlin,1..maxcol] of integer;
end ;
```

Para o caso Complex :

```
MATRIZC = record
    numlinha,numcoluna : integer ;
    elemat : array[1..maxlin,1..maxcol] of complex;
end ;
```

Para o caso Segment :

```
MATRIZS = record
    numlinha,numcoluna : integer ;
    elemat : array[1..maxlin,1..maxcol] of segment;
end ;
```



### 3.3 Entradas e Saídas nos Novos Tipos Numéricos

Conforme tínhamos visto no Capítulo II, vamos criar funções especiais para a leitura destes tipos. Estas terão dois níveis : num primeiro nível, será estabelecida uma lista das variáveis a serem lidas. Logo elas serão introduzidas em uma primitiva do tipo correspondente . A seguir vamos ver como elas se comportam nos diversos casos:

#### 3.3.1 Entrada e Saída em Complexos

Para este caso serão desenvolvidas quatro rotinas, que efetuarão as leituras adequando-se aos padrões do Pascal, de forma que seja possível a tradução do PASCAL-M para o Pascal. Estas rotinas são as seguintes :

```

Procedure READCOMPLEX(Var a : COMPLEX);
  Begin
    Read(a.real,a.imag);
  end;
Procedure READLNCOMPLEX(Var a : COMPLEX);
  Begin
    readln(a.real,a.imag);
  end;
Procedure WRITELNCOMPLEX( a : COMPLEX);
  Begin
    writeln(a.real,a.imag);
  end;

```

```

Procedure WRITECOMPLEX( a : COMPLEX);
Begin
    write(a.real,a.imag);
end;

```

Como é fácil de se observar, cada uma delas simula as diversas formas de E/S do Pascal. Seguiremos esta mesma metodologia nos tipos seguintes, só que nesses casos os tipos poderão ser bem mais diversificados .

### 3.3.2 Entradas e Saídas para Segment

Para o caso do SEGMENT teremos que ter um cuidado especial, pois os intervalos definidos podem ser de diversos tipos. No caso temos definido para REAL e COMPLEX . Além disso terá que ser feita uma consistência sobre os dados, para que estes se ajustem à realidade matemática [GAR 85] . Para o caso de Reais, quando o elemento "superior" do intervalo for menor que o "inferior", nos vamos inverter os valores usando a seguinte rotina :

```

Procedure Trocaint(Var A: Segment)
( Esta função reordena um intervalo real mal condicionado )
Var temp : real ;
begin
    if a.sup < a.inf then
    begin
        temp := a.sup ;
        a.sup := a.inf ;

```

```

    a.inf := temp ;
end;
end ;

```

Um aviso do ocorrido deverá ser dado ao usuário.

No caso dos COMPLEX não poderemos adotar o mesmo critério, pois segundo a "relação de ordem" definida no Capítulo II, o fato de que o complexo 1 não seja menor ou igual que o complexo 2, não implica necessariamente o contrario. Por esta razão deverá ser dado como um erro .

A seguir mostraremos as rotinas para implementar as Entradas e Saídas para o tipo SEGMENT .

### **Leitura de Intervalos de Reais**

Para simular o READ e o READLN usaremos a seguintes funções :

```

Procedure Readint(var a : segment );
begin
    read(a.inf,a.sup);
    if a.inf > a.sup then trocaint(a)
end ;

```

```

Procedure Readlnint(var a : segment );
begin
    readln(a.inf,a.sup);
    if a.inf > a.sup then trocaint(a)
end ;

```

Para o uso destas funções o compilador em primeiro lugar deverá reconhecer o tipo correspondente ao SEGMENT . Sendo assim ativará estas funções, caso contrário :

### Leitura de intervalos de Complex

```

Procedure Readintc(var a : segmentc );
begin
  read(a.inf.real,a.inf.imag,a.sup.real,a.su.imag);
  if (a.inf.real > a.sup.real)
    or (a.inf.ima > a.sup.imag) then erro ;
end ;

```

```

Procedure Readintc(var a : segmentc );
begin
  readln(a.inf.real,a.inf.imag,a.sup.real,a.su.imag);
  if (a.inf.real > a.sup.real)
    or (a.inf.ima > a.sup.imag) then erro ;
end ;

```

### Impressão dos Intervalos

A impressão dos intervalos torna-se simples porque a consistência não será necessária, dado que esta é resultado de operações ou de leituras prévias .

**Impressão de Intervalo de Reais**

```

Procedure Writelnint( a : segment );
begin
  writeln('[' , a.inf , ';' , a.sup , ']' );
end;

```

```

Procedure Writeint( a : segment );
begin
  write('[' , a.inf , ';' , a.sup , ']' );
end;

```

**Impressão de Intervalos de Complexos**

```

Procedure Writelnint( a : segmentc );
begin
  writeln('[' , '(' , a.inf.real , ',' , a.inf.imag , ')' , ';' ,
        '(' , a.sup.real , ',' , a.sup.imag , ')' , ']' );
end;

```

```

Procedure Writeint( a : segment );
begin
  write('[' , '(' , a.inf.real , ',' , a.inf.imag , ')' , ';' ,
        '(' , a.sup.real , ',' , a.sup.imag , ')' , ']' );
end;

```

### 3.3.3 Entradas e Saídas para Matrizes

Para o caso das matrizes vamos ter que ter dois níveis de trabalho. No primeiro, procuraremos na tabela de matrizes a ordem da matriz e o tipo correspondente e, a seguir, a leitura ou escrita propriamente dita. Se os elementos da matriz forem intervalos, deve-se verificar que sejam cumpridas as condições estabelecidas em [GAR 85] e [DAL 84]. Isso será feito seguindo-se os mesmos critérios usados no parágrafo anterior.

#### Leitura de Matrizes

Em primeiro lugar será pesquisado o tipo da matriz. De acordo com cada caso, usaremos uma das seguintes funções :

#### Matrizes de Reais

```
Procedure READMAT(Var a : matrizr );
Var i,j : Integer ;
Begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  for i := 1 to a.numlinha do
  for j := 1 to a.numcoluna do
    read(a.elemat[i,j]);
end;
```

```
Procedure READLNMAT(Var a : matrizr );
Var i,j : Integer ;
Begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  writeln;
  for i := 1 to a.numlinha do
    for j := 1 to a.numcoluna do
      read(a.elemat[i,j]);
    end;
  end;
```

### **Matrizes de Complexos**

```
Procedure READMAT(Var a : matrizc );
Var i,j : Integer ;
Begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  for i := 1 to a.numlinha do
    for j := 1 to a.numcoluna do
      read(a.elemat[i,j].real,a.elemat[i,j].imag);
    end;
  end;
```

```
Procedure READLNMAT(Var a : matrizr );
Var i,j : Integer ;
Begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  writeln;
```

```
for i := 1 to a.numlinha do
for j := 1 to a.numcoluna do
  read(a.elemat[i,j].real,a.elemat[i,j].imag);
end;
```

### Matriz de Intervalos

```
Procedure READMAT(Var a : matriz);
Var i,j : Integer ;
Begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  for i := 1 to a.numlinha do
  for j := 1 to a.numcoluna do
  begin
    read(a.elemat[i,j].inf,a.elemat[i,j].sup);
    if a.elemat[i,j].inf > a.elemat[i,j].sup then erro
  end;
```

```
Procedure READLNMAT(Var a : matriz);
Var i,j : Integer ;
Begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  writeln;
  for i := 1 to a.numlinha do
  for j := 1 to a.numcoluna do
  begin
    read(a.elemat[i,j].inf,a.elemat[i,j].sup);
```



```

    if a.elemat[i,j].inf > a.elemat[i,j].sup then erro ;
end;
writeln ;
end;

```

### Escrita de Matrizes

Em primeiro lugar será pesquisado o tipo da matriz. Dependendo do tipo dos seus elementos (Reais, Inteiros, Intervalos ou Complexos), será usada uma das seguintes funções para escrevê-la :

#### Matrizes de Reais

```

Procedure WRITEMAT(Var a : matrizr );
Var i,j : Integer ;
Begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  for i := 1 to a.numlinha do
  begin
    writeln ;
    for j := 1 to a.numcoluna do
      write(a.elemat[i,j]);
    end;
  end;
end;

```

```

Procedure WRITELNMAT(Var a : matrizr );
Var i,j : Integer ;

```

```

end;
end;
read(('',a[emati,j].real,',' ,a[emati,j].imag,','));
for j := 1 to a.numcoluna do
  writeln
begin
  for i := 1 to a.numlinha do
    Procuratabelamtriz(a,a.numlinha,a.numcoluna);
  begin
    Var i,j : Integer ;
    Procedure WRITEMAT(Var a : matriz);

```

**Matrizes de Complexos**

```

end;
writeln;
end;
write(a[emati,j]);
for j := 1 to a.numcoluna do
  writeln;
begin
  for i := 1 to a.numlinha do
    writeln;
    Procuratabelamtriz(a,a.numlinha,b.numlinha);
  begin

```

```

Procedure WRITELNMAT(Var a : matrizc );
Var i,j : Integer ;
Begin
  Procuratabelamatriz(a,a.numlinha,b.numlinha);
  for i := 1 to a.numlinha do
  begin
    writeln ;
    for j := 1 to a.numcoluna do
      write('(',a.elemat[i,j].real,',',a.elemat[i,j].imag,')');
    end;
    writeln
  end ;

```

### Matrizes de Intervalos

```

Procedure WRITEMAT(Var a : matrizs );
Var i,j : Integer ;
Begin
  Procuratabelamatriz(a,a.numlinha,b.numlinha);
  for i := 1 to a.numlinha do
  begin
    writeln
    for j := 1 to a.numcoluna do
      write('[',a.elemat[i,j].inf,',',a.elemat[i,j].sup,']');
    end;
  end;
end;

```

```
Procedure WRITELNMAT(Var a : matriz);  
Var i,j : Integer ;  
Begin  
  Procuratabelamatriz(a,a.numlinha,b.numlinha);  
  for i := 1 to a.numlinha do  
    begin  
      writeln;  
      for j := 1 to a.numcoluna do  
        write('[' ,a.elema[i,j].inf,',' ,a.elema[i,j].sup,']');  
      end;  
    end;  
end;
```

### 3.4 Operações nos tipos COMPLEX, SEGMENT e MATRIZ

Para se efetuar qualquer uma das operações que envolvem variáveis ou constantes dos tipos COMPLEX, SEGMENT ou MATRIZ serão usados os símbolos usuais dessas operações. Além disso, as operações manterão as prioridades usuais do Pascal. Porém, ao ser avaliada, cada expressão, internamente será decomposta numa expressão equivalente em forma polonesa [TAN 83]. Nela ficarão estabelecidas todas as operações que devem ser feitas e a ordem na qual serão desenvolvidas. Além disso, será verificado se a operação é permitida e o tipo que resultará de tal expressão. Por exemplo, a soma de uma variável complexa com uma matriz deve ser rejeitada. Por outro lado no caso do produto de um real por uma matriz a operação deve ser aceita e o resultado será do tipo

Matriz . Para fazer este teste o Compilador tem uma tabela de operadores na qual constam todos os tipos existentes e as operações neles permitidas, bem como o resultado que devolve.

Exemplo :

Operador	Tipo1	Tipo2	Result
+	I	I	I
+	I	R	R
+	R	I	R
+	R	R	R
+	IR	IR	IR
+	M(m,n)	M(m,n)	M(m,n)
*	R	M(m,n)	M(m,n)
*	M(m,n)	M(n,p)	M(m,p)
*	C	C	C
+	C	C	C

E assim por diante . Esta tabela estará organizada para ser "atualizada", conforme veremos no Capítulo seguinte. É por esta razão que, ao se procurar um operador, a busca será feita de "trás" para "frente", tomando-se o primeiro elemento onde os tipos das variáveis coincidam e também o operador. Neste caso vai ficar armazenado o tipo de resultado da operação. Se não for encontrado nenhum, será produzida uma mensagem de ERRO .

### 3.4.1 Operações em Complex

```
Procedure SOMACOMPLEX(a,b : COMPLEX; var c : COMPLEX );
```

```
Begin
```

```
  c.real := a.real + b.real ;
```

```
  c.imag := a.imag + b.imag ;
```

```
end;
```

```
Procedure DIFCOMPLEX(a,b : COMPLEX; var c : COMPLEX );
```

```
Begin
```

```
  c.real := a.real - b.real ;
```

```
  c.imag := a.imag - b.imag ;
```

```
end;
```

```
Procedure MULCOMPLEX(a,b : COMPLEX; Var c : COMPLEX );
```

```
Begin
```

```
  c.real := a.real*b.real - a.imag*b.imag ;
```

```
  c.imag := a.real*b.imag + b.real*a.imag ;
```

```
end;
```

```
Procedure DIVCOMPLEX(a,b : COMPLEX; VAR c : COMPLEX );
```

```
Var
```

```
  c2 : REAL;
```

```
  c1 : COMPLEX ;
```

```
begin
```

```
  c1.real := b.real ;
```

```
  c1.imag :=-b.imag ;
```

```

MULCOMPLEX(a,c1,c);
c2 := SQR(b.real) + SQR(b.imag);
c.real := c.real/c2 ;
c.imag := c.imag/c2 ;
end;

```

### 3.4.2 Operações em Segment

```

Procedure Somainterv(a,b : Segment ; Var c : Segment) ;
Begin
  c.inf := a.inf + b.inf;
  c.sup := a.sup + b.sup;
end;

```

```

Procedure Difinterv(a,b : Segment ; Var c : Segment) ;
Begin
  c.inf := a.inf - b.sup;
  c.sup := a.sup - b.inf;
end;

```

```

Procedure Prodinterv(a,b : Segment ; Var c : Segment) ;
Var test          : array[1..4] of real;
    maxtest ,mintest : real;
    i              : integer ;
Begin
  test[1] := a.inf*b.inf ;   test[3] := a.sup*b.inf ;
  test[2] := a.inf*b.sup ;   Test[4] := a.sup*b.sup ;

```

```

mintest := test[1];
maxtest := test[2];
for i := 2 to 4 do
begin
  if test[i] > maxtest then maxtest := test[i];
  if test[i] < mintest then mintest := test[i];
end;
c.inf := mintest ;
c.sup := maxtest ;
end;

Procedure Divinterv(a,b : Segment ; Var c : Segment) ;
Var cc : Segment ;
Begin
  if b.inf*b.sup >= 0 then
  begin
    cc.inf := 1/b.sup;
    cc.sup := 1/b.inf;
    Prodinterv(a,cc,c);
  end else ERRO;
end;

```

### 3.4.3 Operações em Matrizes

Para o caso das operações com matrizes, em primeiro lugar deverá ser feito um levantamento sobre o tipo de seus elementos (Real, Inteiro, Segment ou Complex). Conforme o caso, deverá ser usada uma das seguintes funções.



**Matriz de Reais**

```
Procedure Somatrizr(a,b : matrizr ; Var c : matrizr);
var i ,j : integer;
Begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  Procuratabelamatriz(b,b.numlinha,b.numcoluna);
  if ( a.numlinha <> b.numlinha ) or
     ( a.numcoluna <> b.numcoluna) then ERRO else
  begin
    c.numlinha := a.numlinha ;
    c.numcoluna := a.numcoluna ;
    for i := 1 to a.numlinha do
      for j := 1 to a.numcoluna do
        c.elemat[i,j] := a.elemat[i,j] + b.elemat[i,j] ;
      end;
    end;
```

```
Procedure Difmatrizr(a,b : matrizr ; Var c : matrizr);
var i ,j : integer ;
Begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  Procuratabelamatriz(b,b.numlinha,b.numcoluna);
  if ( a.numlinha <> b.numlinha ) or
     ( a.numcoluna <> b.numcoluna) then ERRO else
```

```

begin
  c.numlinha := a.numlinha ;
  c.numcoluna := a.numcoluna ;
  for i := 1 to a.numlinha do
    for j := 1 to a.numcoluna do
      c.elema[i,j] := a.elema[i,j] - b.elema[i,j] ;
    end;
  end;
end;

Procedure Prodmatriz(a,b : matriz ; Var c : matriz);
var i,j,k : integer ;
    soma : real ;
Begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  Procuratabelamatriz(b,b.numlinha,b.numcoluna);
  if a.numcoluna <> b.linha then ERRO else
  begin
    c.numlinha := a.numlinha ;
    c.numcoluna := b.numcoluna ;
    for i := 1 to a.numlinha do
      for j := 1 to b.numcoluna do
        begin
          soma := 0.0 ;
          for k := 1 to a.numlinha do
            soma := soma + a.elema[i,k]*b.elema[k,j];
          c.elema[i,j] := soma ;
        end;
      end;
    end;
  end;
end;

```

**Matriz de Inteiros**

```
Procedure Somatrizi(a,b : matrizi ; Var c : matrizi);
```

```
var i ,j : integer ;
```

```
Begin
```

```
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
```

```
  Procuratabelamatriz(b,b.numlinha,b.numcoluna);
```

```
  if ( a.numlinha <> b.numlinha ) or
```

```
      ( a.numcoluna <> b.numcoluna) then ERRO else
```

```
  begin
```

```
    c.numlinha := a.numlinha ;
```

```
    c.numcoluna := a.numcoluna ;
```

```
    for i := 1 to a.numlinha do
```

```
      for j := 1 to a.numcoluna do
```

```
        c.elemat[i,j] := a.elemat[i,j] + b.elemat[i,j] ;
```

```
    end;
```

```
end;
```

```
Procedure Difmatrizi(a,b : matrizi ; Var c : matrizi);
```

```
var i ,j : integer;
```

```
Begin
```

```
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
```

```
  Procuratabelamatriz(b,b.numlinha,b.numcoluna);
```

```
  if ( a.numlinha <> b.numlinha ) or
```

```
      ( a.numcoluna <> b.numcoluna) then ERRO else
```

```
  begin
```

```
    c.numlinha := a.numlinha ;
```

```
c.numcoluna := a.numcoluna ;
for i := 1 to a.numlinha do
  for j := 1 to a.numcoluna do
    c.elemat[i,j] := a.elemat[i,j] - b.elemat[i,j] ;
  end;
end;

end;

Procedure Prodmatrizi(a,b : matrizi ; Var c : matrizi);
var i,j,k ,soma : integer ;
Begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  Procuratabelamatriz(b,b.numlinha,b.numcoluna);
  if a.numcoluna <> b.linha then ERRO else
  begin
    c.numlinha := a.numlinha ;
    c.numcoluna := b.numcoluna ;
    for i := 1 to a.numlinha do
      for j := 1 to b.numcoluna do
        begin
          soma := 0 ;
          for k := 1 to a.numlinha do
            soma := soma + a.elemat[i,k]*b.elemat[k,j];
          c.elemat[i,j] := soma ;
        end;
      end;
    end;
  end;
end;
```

### Matrizes de Intervalos

```

Procedure Somatriza(a,b : matriz ; Var c : matriz);
var i ,j : integer;
Begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  Procuratabelamatriz(b,b.numlinha,b.numcoluna);
  if ( a.numlinha <> b.numlinha ) or
      ( a.numcoluna <> b.numcoluna) then ERRO else
  begin
    c.numlinha := a.numlinha ;
    c.numcoluna := a.numcoluna ;
    for i := 1 to a.numlinha do
      for j := 1 to a.numcoluna do
        SOMAINTERV(a.elemat[i,j] , b.elemat[i,j],c.elemat[i,j]) ;
      end;
    end;
end;

Procedure Difmatriz(a,b : matriz ; Var c : matriz);
var i ,j : integer;
begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  Procuratabelamatriz(b,b.numlinha,b.numcoluna);
  if ( a.numlinha <> b.numlinha ) or
      ( a.numcoluna <> b.numcoluna) then ERRO else
  begin
    c.numlinha := a.numlinha ;

```

```

c.numcoluna := a.numcoluna ;
for i := 1 to a.numlinha do
for j := 1 to a.numcoluna do
DIFINTERV(a.elemat[i,j] , b.elemat[i,j],c.elemat[i,j]) ;
end;
end;

Procedure Prodmatrizs(a,b : matrizs ; Var c : matrizs);
var i,j,k      : integer ;
    soma,prod  : segment ;
Begin
Procuratabelamatriz(a,a.numlinha,a.numcoluna);
Procuratabelamatriz(b,b.numlinha,b.numcoluna);
if a.numcoluna <> b.linha then ERRO else
begin
c.numlinha := a.numlinha ;
c.numcoluna := b.numcoluna ;
for i := 1 to a.numlinha do
for j := 1 to b.numcoluna do
begin
soma.inf := 0.0 ;
soma.sup := 0.0 ;
for k := 1 to a.numlinha do
Begin
PRODINTERV(A.elemat[i,j],B.elemat[i,j],prod);
SOMAINTERV(soma,prod,soma);
end ;

```

```

    c.eleamat[i,j].inf := soma.inf ;
    c.eleamat[i,j].sup := soma.sup ;
end;
end;

```

### Matrizes de Complexos

```

Procedure Somatrizc(a,b : matrizc ; Var c : matrizc);
var i ,j : integer ;
Begin
    Procuratabelamatriz(a,a.numlinha,a.numcoluna);
    Procuratabelamatriz(b,b.numlinha,b.numcoluna);
    if ( a.numlinha <> b.numlinha ) or
        ( a.numcoluna <> b.numcoluna) then ERRO else
begin
    c.numlinha := a.numlinha ;
    c.numcoluna := a.numcoluna ;
    for i := 1 to a.numlinha do
    for j := 1 to a.numcoluna do
        SOMACOMPLEX(a.eleamat[i,j] , b.eleamat[i,j],c.eleamat[i,j]) ;
    end;
end;
end;

```

```

Procedure Difmatrizc(a,b : matrizc ; Var c : matrizc);
var i ,j : integer ;
Begin
    Procuratabelamatriz(a,a.numlinha,a.numcoluna);

```

```

Procuratabelamatrix(b,b.numlinha,b.numcoluna);
if ( a.numlinha <> b.numlinha ) or
    ( a.numcoluna <> b.numcoluna) then ERRO else
begin
    c.numlinha := a.numlinha ;
    c.numcoluna := a.numcoluna ;
    for i := 1 to a.numlinha do
    for j := 1 to a.numcoluna do
        DIFCOMPLEX(A.elemat[i,j] , B.elemat[i,j] , c.elemat[i,j]) ;
    end ;
end;

Procedure Prodmatrizc(a,b : matrizc ; Var c : matrizc);
var i,j,k      : integer ;
    soma,prod  : complex ;
Begin
    Procuratabelamatrix(a,a.numlinha,a.numcoluna);
    Procuratabelamatrix(b,b.numlinha,b.numcoluna);
    if a.numcoluna <> b.linha then ERRO else
    begin
        c.numlinha := a.numlinha ;
        c.numcoluna := b.numcoluna ;
        for i := 1 to a.numlinha do
        for j := 1 to b.numcoluna do

        begin
            soma.real := 0.0 ;

```



```

soma.imag := 0.0 ;
for k := 1 to a.numlinha do
  Begin
    Mulcomplex(A.elemat[i,j],B.elemat[i,j],prod);
    Somacomplex(soma,prod,soma);
  end ;
  c.elemat[i,j].real := soma.real ;
  c.elemat[i,j].imag := soma.imag ;
end;
end;

```

### 3.5 Funções de Biblioteca nos tipos SEGMENT, COMPLEX E MATRIZ

Para o caso das funções de Biblioteca do PASCAL-M, nos tipos numéricos recentemente criados, ira se implementar, quando for o caso, todos os existentes para REAL ou INTEGER. Além disso, serão incorporados aqueles intrínsecos a cada um dos tipos desenvolvidos. Assim, por exemplo, Inversa para o caso do tipo Matriz, Conjugado para Complex e assim por diante .

Para fazer a avaliação destas funções, em primeiro lugar se fará um empilhamento, similar ao caso dos operadores, para estabelecer antes se pode-se aplicar ou não a função e assim poder estabelecer a ordem na qual devem ser avaliadas . Para poder fazer tal avaliação temos que implementar uma tabela de funções, tipos que são permitidos nelas e resultados que estas devolvem Adicionalmente podem-se incluir os limites de seus

argumentos. Quando uma função for requerida, essa tabela nos dará a informação necessária. Conforme seja o tipo das variáveis, poderemos aplicar uma das seguintes funções.

### 3.5.1 Funções para SEGMENT

```
Procedure Senointerv(x:segment ; Var ssen: Segment);
```

```
Begin
```

```
  If x.sup < pi/2 then
```

```
    Begin
```

```
      ssen.inf := sin(x.inf);
```

```
      ssen.sup := sin(x.sup);
```

```
    end else
```

```
    Begin
```

```
      if x.inf < pi/2 then
```

```
        begin
```

```
          ssen.sup := 1 ;
```

```
          if x.sup >= 3*pi/2 then ssen.inf := -1
```

```
              else ssen.inf := sin(x.sup) ;
```

```
        end else
```

```
          if x.sup <= 3*pi/2 then
```

```
            begin
```

```
              ssen.sup := Sin(x.inf);
```

```
              ssen.inf := Sin(x.sup);
```

```
            end else
```

```
            Begin
```

```
              if x.inf < 3*pi/2 then
```

```
begin
  ssen.inf := -1 ;
  if sin(x.inf) > sin(x.sup) then ssen.sup := sin(x.inf)
    else ssen.inf := sin(x.sup);
end else
Begin
  ssen.inf := sin(x.inf);
  ssen.sup := sin(x.sup);
end;
end;
end;
end;

Procedure COSENDINTERV(x : SEGMENT ; VAR ccos : SEGMENT );
Begin
  if x.sup < pi then
begin
  ccos.inf := cos(x.sup) ;
  ccos.sup := cos(x.inf) ;
end else
  if x.inf > pi then
begin
  ccos.inf := cos(x.inf);
  ccos.sup := cos(x.sup);
end else
```

```
begin
  ccos.inf := - 1 ;
  if cos(x.inf) > cos(x.sup) then ccos.sup := cos(x.inf)
    else ccos.sup := cos(x.sup);
end;

end;

Procedure expinterv(x : SEGMENT ;VAR Sexp : SEGMENT );
Begin
  sexp.inf := exp(x.inf) ;
  sexp.sup := exp(x.sup) ;
end;

Procedure lninterv(x : SEGMENT ;VAR Sln : SEGMENT );
Begin
  sln.inf := ln(x.inf) ;
  sln.sup := ln(x.sup) ;
end;

Procedure sqrtinterv(x : SEGMENT ;VAR Ssqrt : SEGMENT );
Begin
  ssqrt.inf := SQRT(x.inf) ;
  ssqrt.sup := SQRT(x.sup) ;
end;

Function absinterv(x : SEGMENT ) : real ;
Begin
  if abs(x.sup) > abs(x.inf) then absinterv := abs(x.sup)
    else absinterv := abs(x.inf) ;
end;
```

```

Function Pertenca( xo : real ; x : SEGMENT ) : boolean ;
{ Funcao para determinar se xo pertence ao intervalo x }
Begin
  if ( xo <= x.sup ) and ( xo >= x.inf )
  then pertenca := true
  else pertenca := false ;
end;

```

### 3.5.2 Funções para os Complex

```

Procedure SINCOMPLEX(z : COMPLEX ; VAR sinz : COMPLEX);
Begin
  sinz.real := sin(z.real)*(exp(z.imag)+exp(-z.imag))/2;
  sinz.imag := cos(z.real)*(exp(z.imag)-exp(-z.imag))/2;
end;

Procedure COSCOMPLEX(z : COMPLEX ; VAR cosz : COMPLEX);
Begin
  cosz.real := cos(z.real)*(exp(z.imag)+exp(-z.imag))/2;
  cosz.imag := -sin(z.real)*(exp(z.imag)-exp(-z.imag))/2;
end;

Procedure EXPCOMPLEX(z : COMPLEX ; VAR expz : COMPLEX);
Begin
  expz.real := exp(z.real)*cos(z.imag);
  expz.imag := exp(z.real)*sin(z.imag);
end;

```

```
Procedure LNCOMPLEX(z : COMPLEX ; VAR lnz : COMPLEX);
```

```
Var r : real ;
```

```
Begin
```

```
  r := SQRT(sqr(z.real) + sqr(z.imag));
```

```
  lnz.real := ln(r);
```

```
  lnz.imag := arctan(z.real/z.imag);
```

```
end;
```

```
Procedure Conjugado( z : Complex ; Var conjz : Complex );
```

```
begin
```

```
  conjz.real := z.real ;
```

```
  conjz.imag := - z.imag ;
```

```
end ;
```

```
Function abscomplex( z : Complex ) : real ;
```

```
begin
```

```
  abscomplex := sqrt( z.real*z.real + z.imag*z.imag) ;
```

```
end ;
```

```
Procedure Sqrtcomplex( z : Complex ; Var sqrtz : complex );
```

```
Begin
```

```
  sqrtz.real := sqrt( z.real + abscomplex(z))/sqrt(2.0) ;
```

```
  sqrtz.imag := sqrt(-z.real + abscomplex(z))/sqrt(2.0) ;
```

```
end ;
```

### 3.5.3 Funções para MATRIZ

Função para calcular o valor absoluto de uma matriz, Para este caso estamos usando a "norma euclidiana" [DIU 82] .

```
Function absmatriz( a :matrizr ) : Real ;
Var soma : real ;
    i,j : integer ;
Begin
    procuratabelamatriz(a,a.numlinha,a.numcoluna);
    if a.numlinha <> a.numcoluna then ERRO else
    begin
        soma := 0.0 ;
        for i := 1 to a.numlinha do
            for j := 1 to a.numcoluna do
                soma := soma + sqr(a.elema[i,j]) ;
            absmatriz := sqrt(soma) ;
        end;
```

**Procedimento para calcular a forma triangular equivalente de uma matriz quadrada qualquer, Isto será muito usado no caso de sistema lineares .**

```
procedure trianmatrizr(a2 : matrizr ; Var a1 : matrizr) ;
var i,j,k : Integer ;
    xx : real ;
begin
    for i := 1 to a2.numlinha - 1 do
        begin
```

```

for j := i+1 to a.numlinha do
begin
  xx := a2.elemat[j,i]/a2.elemat[i,i];
  for k := 1 to a.numlinha do
  begin
    a1.elemat[j,k] := a1.elemat[j,k] - a1.elemat[i,k]*xx;
    a2.elemat[j,k] := a2.elemat[j,k] - a2.elemat[i,k]*xx;
  end;
end;
end;
end;
end;

```

**Procedimento auxiliar nos cálculos de Matriz triangular e Matriz inversa . Este procedimento será invisível ao usuário .**

```

procedure trocar(i,j : integer ; Var a: matrizr);
var
  k : integer ;
  x : real ;
begin
  for k := 1 to a.numlinha do
  begin
    with a do
    begin
      x := elemat[i,k] ;
      elemat[i,k] := elemat[j,k] ;

```



```

    elemat[j,k] := x ;
end;
end;
end;

```

**Procedimento para calcular a inversa de uma matriz.**

```

procedure inver( a : matrizr ; Var ai : Matrizr );
Var
    i,j,k,l,p : integer ;
    xx : real ;
    a2 : matrizr ;
begin
    procuratabelamatriz(a,a.numlinha,a.numcoluna);
    if a.numlinha <> a.numcoluna then ERRO ;
Begin
    a2.numlinha := a.numlinha ;
    a2.numcoluna := a.numcoluna ;
    ai.numcoluna := a.numcoluna ;
    ai.numlinha := a.numlinha ;
    for i := 1 to a.numlinha do
    for j := 1 to a.numcoluna do
begin
    if i <> j then ai.elemat[i,j] := 0.0 else ai.elemat[i,j] := 1.0;
    a2.elemat [i,j] := a.elemat[i,j];
end;
for i := 1 to a.numlinha - 1 do

```

```

begin
  for j := i+1 to a.numlinha do
    begin
      if a2.elemat[i,i] <> 0.0 then xx := a2.elemat[j,i]/a2.elemat[i,i]
        else begin
          p := i ;
          repeat
            if a2.elemat[l,i] <> 0.0 then p := l
              l := l + 1 ;
          until (p <> i) or ( l > a2.numlinha);
          if l <= a2.numlinha then
            begin
              trocar(i,p,a2);
              trocar(i,p,a1);
            end else ERRO ;
          end;
        for k := 1 to a.numlinha do
          begin
            a1.elemat[j,k] := a1.elemat[j,k] - a1.elemat[i,k]*xx;
            a2.elemat[j,k] := a2.elemat[j,k] - a2.elemat[i,k]*xx;
          end;
        end;
      end;
    end;
  for i := a.numlinha downto 2 do
    begin
      for j := i - 1 downto 1 do

```

```

begin
  xx := a2.elemat[j,i]/a2.elemat[i,i];
  for k := 1 to a.numlinha do
    begin
      a1.elemat[j,k] := a1.elemat[j,k] - a1.elemat[i,k]*xx;
      a2.elemat[j,k] := a2.elemat[j,k] - a2.elemat[i,k]*xx;
    end;
  end;
end;
for i := 1 to a.numlinha do
  begin
    xx := a2.elemat[i,i] ;
    for j := 1 to a.numlinha do
      begin
        a1.elemat[i,j] := a1.elemat[i,j]/xx ;
        a2.elemat[i,j] := a2.elemat[i,j]/xx ;
      end;
    end;
  end;
end;
end;

```

#### Function para calcular o Determinante de uma Matriz

```

Function Determinante(a : Matrizr) : Real ;
var i : integer ;
    aux : matrizr ;
    pro : real ;

```

```
begin
  Procuratabelamatriz(a,a.numlinha,a.numcoluna);
  if a.numlinha <> a.numcoluna then ERRO else
  begin
    trianmatrizr(a , aux);
    pro := 1 ;
    for i := 1 to a.numlinha do pro := pro * aux.elema[i,i];
    determinante := pro ;
  end;
end;
```

C A P I T U L O      I V  
-----

IMPLEMENTAÇÃO DOS DATA TYPE

#### 4.1 Introdução

Neste capítulo vamos apresentar a forma pela qual serão implementados os Tipos Abstratos de Dados e os operadores aritméticos e/ou relacionais ligados a eles . O objetivo é fornecer ao usuário uma ferramenta, a qual lhe permita a criação dos mais variados tipos ou inclusive mudar os operadores já existentes. Isto será muito útil para que possamos trabalhar com estruturas "abstratas" ou para implementar estruturas numéricas de uso particular, como por exemplo elementos de "Álgebra Tensorial".

#### 4.2 Tipos Abstratos de Dados

Na linguagem Pascal podemos achar os seguintes tipos primitivos básicos : REAL, INTEGER, BOOLEAN, CHAR etc . A partir deles existe a possibilidade de gerar Arranjos (ARRAYs), Registros (RECORDs), Conjuntos (SETs) e inclusive constantes desses tipos, tipos enumeráveis (Exceção de REAL). A grande restrição é que os operadores definidos para estes tipos ficam restritos aos operadores válidos para a estrutura básica que deu origem ao tipo.

Estes "novos" tipos ficam definidos na seção TYPE e nosso objetivo é atingir com eles todas características definidas para

TADs em [GUT 78] .

Para poder usar os TADs em toda sua potencialidade, em primeiro lugar vamos fazer um levantamento das necessidades para que isto seja possível . Podemos constatar que um tipo qualquer, para que esteja bem definido, precisa de um Domínio, sobre o qual definiremos as operações aritméticas e, se for o caso, relações de ordem, através de operadores relacionais . Estas exigências partem da idéia de estruturas algébricas : Grupos, Anéis, etc [GIL 77].

Para poder implementar os TADs, em primeiro lugar deveremos identificar o tipo criado com um "nome", que o distinguirá dos tipos já existentes e o caracterizará como um "token" [AHO 77] . Além disso gerará uma aritmética para as variáveis declaradas como pertencentes a esse tipo . Definido o "nome", será necessário associar a ele uma estrutura de dados para representá-lo, e se for preciso, parâmetros para a estrutura .

Exemplo :

----- Tipo -----	----- Estrutura -----
Natural	Integer Positivo
Matriz Esparsa	Arranjo de Registros

Podemos observar que, para o segundo caso, por ser arranjo, estes deverão ter um limite . Além disso será necessário definir o registro correspondente, no qual estará indicado o valor e a posição relativa dele na matriz .

Uma vez identificado o tipo, por um nome, e a estrutura de dados associada, temos a necessidade de definir as operações aritméticas e/ou relacionais. Nelas vamos distinguir entre as internas e as externas. Vamos considerar como internas aquelas que servem como uma ferramenta auxiliar no cálculo das operações desenvolvidas para o tipo. Por exemplo, se estivéssemos criando o tipo RACIONAL e as operações aritméticas associadas a esse tipo, teríamos que, para chegar ao resultado final dado um resultado parcial da forma  $p/q$ , será necessário eliminar os divisores comuns, dado que esse procedimento se repetirá nas quatro operações aritméticas fundamentais. Portanto será conveniente criar uma rotina para tal fim. Essa rotina ficará para uso interno do tipo. Chamaremos de externas aquelas rotinas que implementam os operadores do tipo criado.

Exemplo :

Tipo : Natural  
 Operações - Internas : Eliminação de Fatores comuns  
 - Externas : + | - | / | \*

Para implementar estas operações o usuário poderá usar rotinas ou funções, segundo a estrutura usada para definir o tipo. Essa operação será reconhecida no Programa Principal pelo nome dado à rotina que deu origem a ela. Para tal efeito, os nomes dessas rotinas, quando a operação for aritmética, deverá ser formada por um conjunto de símbolos não alfabéticos e não numéricos, isso

possibilitará que essas operações possam ser usadas na forma usual dos tipos numéricos . Quando os TADs não forem de tipo numérico esse requisito não será necessário e seu uso será através da chamada à rotina, da mesma maneira como acontece no Pascal standard .

Exemplo :

TAD	OPERAÇÃO	ROTINAS	SINAL
Racional	Soma	'+'	+
Pilha	Pop	Pop	Não possui

#### 4.3 DATA TYPE

Com a finalidade de implementar os detalhes vistos na seção anterior, vamos criar uma seção adicional no Pascal standard. Essa seção é o DATA TYPE e servirá para definir os TADs . Essa seção ficará depois da seção TYPE e antes da seção VAR . Sendo, do mesmo modo que TYPE e CONST, uma seção optativa . É claro que um programa escrito em PASCAL-M, que não possua a seção DATA TYPE, só se diferencia de um programa escrito em Pascal standard pelo uso dos tipos numéricos MATRIZ, SEGMENT e COMPLEX .

A partir disso a estrutura geral da Linguagem PASCAL-M ficará assim :



```

PROGRAM <nome>( <param-input>,<param-output>);

<list-const>

<list-types>

<list-data-types>

<list-var>

<list-proced>

BEGIN

<list-comandos>

END.

```

A única coisa que necessita ser desenvolvida é <list-data-types>, pois as demais seções já existem em Pascal standard.

#### 4.3.1 Sintaxe para <list-data-type>

Conforme vimos na seção 4.2, a criação de um TAD implicará um nome, uma estrutura de dados, parametros e operações. Além disso, devemos garantir que a seção DATA TYPE seja opcional. A partir disso usaremos a seguinte sintaxe :

- 1.- <list-data-type> := e | DATA TYPE <list-d-types>
- 2.- <list-d-types> := <data-type>; | <data-type>;<list-d-types>
- 3.- <data-type> := <nome> IS
  - <par-list>
  - <types-names>
  - <data-structure>
  - <operations>
  - END;

- 4.- <nome> := <identificador>
- 5.- <par-list> := e | PARAMETERS <parms>
- 6.- <parms> := <par>; | <par> ; <parms>
- 7.- <par> := <list-identif> : <type>
- 8.- <types-names> := TYPES NAMES( <list-identif> )
9. <operations> := OPERATION <op-list>
- 10.- <op-list> := <operation>; | <operation>; <op-list>
- 11.- <operation> := <declar> | EXPORT <declar>
- 12.- <declar> := <declaration-procedure> |  
<declaration-fuction>

#### 4.3.2 Comentários sobre a Sintaxe dos DATA TYPE

O fato de que em ( 1 ) e ( 5 ) se incluía a palavra vazia (e) é devido ao fato de que tanto a seção DATA TYPE como os parâmetros dentro dela são opcionais. Observe-se que no caso de ( 10 ) , referente a OPERATION, é obrigatória a inclusão, no mínimo, de um operador. Isso porque careceria de sentido a criação da seção DATA TYPE, simplesmente para criar um novo tipo sem operadores associados, dado que isso pode ser feito na seção TYPE .

O tratamento da lista de identificadores será idêntico ao

tratamento feito no Pascal standard . Outra coisa que funciona de modo similar é a criação de novos tipos, onde poderemos usar todos os tipos e constantes definidos previamente .

Para o caso de ( 12 )( declaração de procedures ou funções ) a sintaxe vai funcionar de modo igual ao PASCAL standard, porém o tratamento de nomes deverá permitir o uso de símbolos, pois esses, no caso de tipos numéricos, servirão de sinal para suas operações .

#### 4.4 Compilação do DATA TYPE

Quando um programa escrito em PASCAL-M for compilado, dado que vamos usar um compilador de Pascal standard, teremos que adequar esta nova seção de Pascal a esse contexto .

##### 4.4.1 Construção das Tabelas de Tipos e Constantes .

Dado que nesta seção poderemos usar todas as definições feitas previamente, teremos necessidade de fazer uma tabela de símbolos com as constantes e uma tabela de tipos com os types previamente definidos. A tabela de constantes ficaria definida como um arranjo de registros da forma :

```
-----
| Num   | Valor |
-----
```

E a tabela de tipos será uma arranjo de registros do tipo :

```
-----
| Num | Tipo-base | Estrutura |
-----
```

O número (Num) referido nas tabelas será retirado da tabela de símbolos. Esta tabela tem dois níveis :

Nível 1 : Cadeia de caracteres contendo todos os nomes lidos .

Nível 2 : Arranjo de Registros do tipo :

Num : Número de símbolo

Posi : Posição Inicial na cadeia total .

Posf : Posição Final na cadeia total .

Tipo : <cte> | <tipo> | <var>

#### 4.4.2 Tabela de Operadores

Além das tabelas acima, vamos precisar da tabela de operadores, pois as operações desenvolvidas para dar origem às operações associadas aos novos tipos terão quase sempre como base as operações já existentes, e inclusive existe a possibilidade de que uma delas seja mudada . Nesta tabela, é claro, estarão incluídas as operações definidas para os tipos COMPLEX, SEGMENT e MATRIZ . O formato desta tabela é o descrito no Capítulo III .

#### 4.4.3 Decomposição do DATA TYPE

Com a informação sobre os operadores do Pascal-M pronta, procederemos à análise da seção DATA TYPE, onde, em primeiro lugar, será analisada a sintaxe básica, já que as operações geradas nessa seção se agregarão à tabela de operadores de

Pascal-M . A sintaxe dos comandos dos Procedimentos será analisada no momento da compilação do conjunto dos tipos abstratos. No caso de não existirem erros, a definição de tipo será anexada ao TYPE do programa em Pascal standard equivalente ao nosso. O nome deste programa será TEMP.PAS.

Para cada operação será implementada uma Função ou Procedure, de acordo com o caso. Estas rotinas formarão o arquivo ABSTRAC.PAS, o qual será compilado em separado. Para se poder recuperar a informação de cada operação, estas serão incluídas numa tabela de operadores especiais. Nesta tabela será incluída a seguinte informação :

- 1.- Sinal : Símbolo Usado para indicar a operação .
- 2.- Operando 1 : Tipo do primeiro Operando .
- 3.- Operando 2 : Tipo do segundo Operando .
- 5.- Resultado : Tipo do Resultado .
- 6.- Rotina : Procedure ou Função associada .
- 7.- Tipo de Rotina : Function ou Procedure .
- 8.- Prioridade : Para decidir a ordem das operações .

Quando uma linha contendo tipos Abstratos for analisada e um operador for achado nelas, esta tabela será percorrida até se achar a descrição do operador. Com informação fornecida por ela

será possível montar o comando em Pascal standard, através de chamadas às rotinas pré-definidas, que faça uma tarefa equivalente .

#### 4.5 Limitações do DATA TYPE

A principal limitação dos DATA TYPEs refere-se à construção das operações, as quais terão que ser feitas através de Funções e Procedures, o que exigirá uma certa experiência do usuário . Além disso a montagem destas só poderá ser feita a partir das já definidas para o Pascal standard. Isso poderá restringir o nível de abstração delas .

Outra das limitações refere-se ao fato de que estas operações serão compiladas antes do programa todo, e que mesmo que sintaticamente corretas, poderiam levar, em casos particulares, a erros de execução, os quais não seriam facilmente resolvidos.

Finalmente poderíamos acrescentar o fato de que, se forem redefinidos muitos tipos, estes terão que ser localizados no programa todo para que sejam redefinidos seus operadores. Isso fará da compilação uma tarefa mais demorada. Também o fato de partir o programa em "pedaços", para sua compilação, fará com que a informação sobre os erros não seja suficientemente esclarecedora. Além disso exigirá um maior espaço de memória nos disquetes, para o armazenamento dos OBJs correspondentes .

## CAPÍTULO V

### IMPLEMENTAÇÃO

#### 5.1 Introdução

Neste capítulo vamos mostrar como será implementado o gerenciamento dos seguintes programas :

- Programa do usuário escrito em Pascal-M .
- Biblioteca de tipos Numéricos(Complex, Segment e Matriz ) .
- Biblioteca dos Tipos Abstratos gerados pelo usuário .

Isto será feito tendo-se em conta que este processo deve ficar invisível ao usuário . Além do mais, devemos garantir que todo programa escrito em Pascal standard deve rodar em Pascal-M e, por último, devemos gerar um ambiente de programação de fácil uso para os usuários de Pascal-M .

#### 5.2 Ambiente de Programação

O ambiente de trabalho do Pascal-M está desenvolvido de tal forma que permite um fácil acesso ao editor, ao compilador e uma fácil execução dos programas . Além disso, considerou-se conveniente fornecer ao usuário um nível de auxílio. Para isso foi providenciado um arquivo contendo um manual de usuário, o qual pode ser usado durante a fase de trabalho . Outra das facilidades que tem o usuário é a consulta aos diretórios e a

possibilidade de listar seus arquivos em impressora ou, se for necessário criar espaço nos disquetes, o usuário poderá remover arquivos.

Ao iniciar seu trabalho o usuário receberá o seguinte menu :

---

P A S C A L - M

(E)ditar	(C)ompilar	(R)un	(I)mprimir
-			
(A)pagar	(D)iretorio	(M)anual	(T)erminar

---

Junto com esse "menu" aparecerá o cursor piscando na posição correspondente a (E)ditar . Caso o usuário queira executar uma operação diferente poderá deslocar o cursor usando as setas ou pressionar a tecla correspondente à letra inicial da tarefa desejada .

A seguir vamos detalhar a função de cada um dos utilitários.

E : Ingressará no editor da Linguagem . Este editor foi desenvolvido especialmente para o nosso trabalho e está feito seguindo a linha do Wordstar[RAM 86], porém, ao contrário do uso da tecla CTRL seguida de outra tecla para indicar um comando especial(Gravar, Sair, etc) da edição, usaremos as teclas programáveis(F1,F2,F3...,F8) para as tarefas inerentes a um editor, como Gravar, Apagar e Copiar, Pular no início ou no fim



do arquivo etc. Para nos movimentar-nos pelo arquivo vamos usar as setas que se encontram no lado direito do teclado . Do mesmo modo, será usada a tecla DEL quando precisarmos apagar um caracter, cuja opção não está aparecendo no Menu do editor . Quando o usuário inicializar o editor, será solicitado o nome do arquivo com o qual vai trabalhar . Recebido o nome, o editor informará se se trata de um arquivo novo ou não, afixando tal informação na tela Para continuar, deve ser pressionada qualquer tecla. Caso desista de trabalhar nesse arquivo, pressione a tecla ESC . Ao ingressar no editor a tela fica dividida em duas partes, onde na parte superior existe um "menu" com o seguinte formato :

---

[F1] Gravar	[F2] Terminar	[F3] Inicio	[F4] Apagar Palavra
[F5] Inserir(N)	[F6] Copiar	[F7] Fim	[F8] Apagar Linha

---

Vamos agora descrever brevemente o funcionamento destas opções do Menu do Editor :

[F1] Gravar : Grava o arquivo corrente e retorna ao Editor .

[F2] Terminar : Sai do Editor, mas antes verifica se foram feitas modificações no arquivo. Em caso afirmativo, perguntará, se deseja atualiza-lo ou não

[F3] Inicio : Leva a linha corrente ao Inicio do Arquivo .

[F4] Apagar Palavra : Apaga uma palavra a partir da posição corrente.

[F5] Inserir : Muda o estado de inserção de ativo(S) para desativo(N) ou vice-versa .

[F6] Cópia : Cópia o conteúdo do arquivo corrente em outro, cujo nome será solicitado .

[F7] Fim : Leva a linha corrente ao Fim do Arquivo .

[F8] Apagar Linha : Apaga a linha corrente .

A parte inferior do vídeo fica reservada para a leitura e escrita do texto que se deseja atualizar .

Vamos agora continuar com as opções do Gerenciador :

C : Servirá para compilar o programa . Este utilitário terá a seguinte função :

Fará o processo de Pré-Compilação do programa escrito em Pascal-M gerando um programa equivalente em Pascal standard . Além disso gerará uma biblioteca de rotinas para os tipos Abstratos de dados gerados no DATA TYPE . Nos casos em que estas rotinas ou o uso dos tipos numéricos Complex, Segment ou Matriz estejam errados, impossibilitando a geração dos programas Pascal equivalentes, o processo será detido e uma mensagem de erro referindo-se ao uso indevido dos tipos numéricos ou abstratos será dado. O usuário será advertido, também, de outros erros que

A escolha desta opção em detrimento da opção oferecida pelo compilador TURBO PASCAL 3.0 é em razão de que esta última não oferece a possibilidade de uso de ligação de objetos [SC0 83] [W00 87]. Em vez disso, oferece o \$include, o qual faria com que o processo não ficasse invisível para o usuário, pois o que ele faz é anexar os rotinas correspondentes aos tipos abstratos ou à biblioteca numérica. Além disso, no caso de serem detectados erros, o editor faria referências a um programa desconhecido pelo usuário, o que dificultaria seu trabalho. Por último, o tempo que seria ganho pela velocidade do TURBO seria perdido em

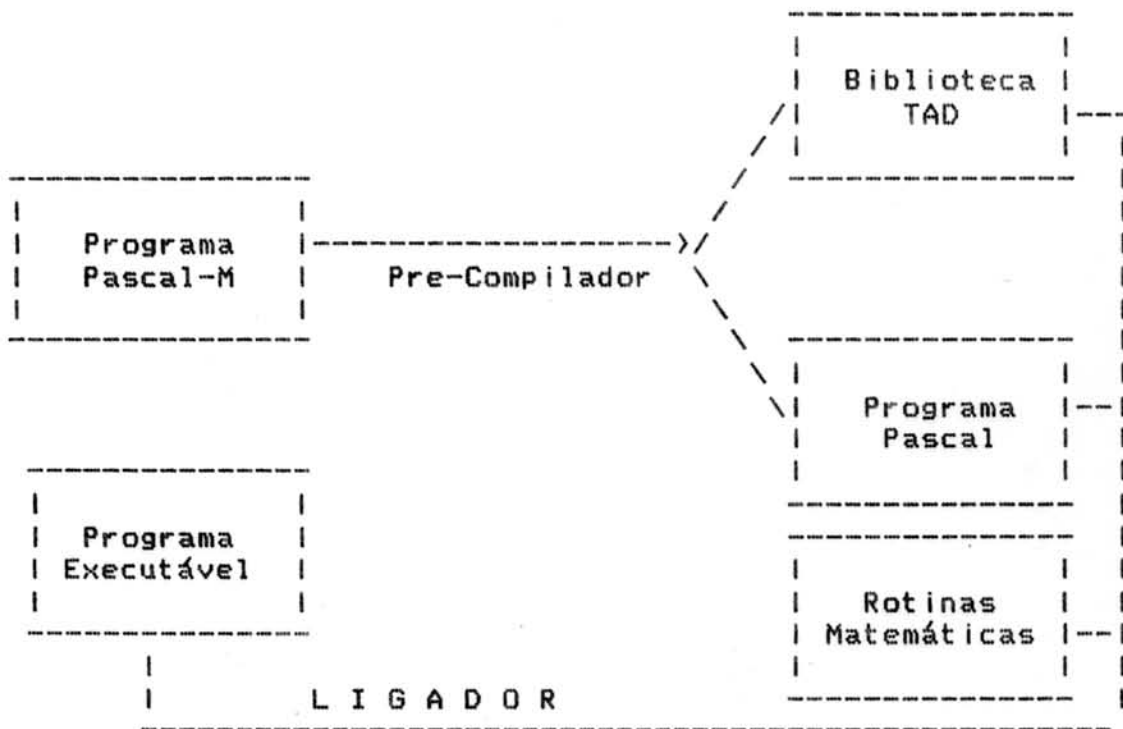
LINK.EXE PASCAL.LIB

PAS1.COM PASKEY PAS2.EXE

Pascal da Microsoft [MIC 83]. tarefas serão usados os seguintes utilitários do compilador pelos tipos abstratos de dados. Para a realização dessas rotinas numéricas do Pascal-M e à biblioteca de rotinas geradas existem erros, o Programa Objeto será ligado à biblioteca de programa original e a lista de erros correspondente. Se não gerado. Se ainda subsistirem erros, será devolvida a listagem do abstratos. Nesse momento será efetuada a compilação do programa teremos junto a ele os "programas objeto" das rotinas dos tipos programa em Pascal standard equivalente ao escrito em Pascal-M, poderão vir a ocorrer. No caso de que tenha sido possível gerar o

entradas nele e saídas sucessivas dele para correção de erros, pois o programa gerenciador da compilação é gerado em tipo ".COM" e portanto será executado fora do TURBO. O preço a pagar por essas ligações é uma compilação mais demorada.

O processo geral da compilação pode ser expresso usando-se o seguinte ESQUEMA :



A partir do esquema acima pode-se notar que o Pré-Compilador vai dividir o programa original em dois: um que contém as rotinas correspondentes aos TADs e outro que contém a parte restante, no qual todos os comandos de Pascal-M são substituídos, conforme veremos adiante, por rotinas que executam a operação

equivalente . Logo após desse processo, ambos os programas serão compilados gerando os programas objeto correspondentes, os quais serão ligados junto com as rotinas matemáticas . O resultado dessa ligação é o programa executável equivalente ao escrito em PASCAL-M . As tarefas necessárias para implementar este processo são feitas pelo seguinte arquivo de comandos :

```

Echo off { Isto fará que o processo geral fique invisível
b:      { Drive contendo os arquivos do usuário
a:pas1 <rotinas TAD>,,xx.lst,; {xx.lst traz os erros }
rem Testando se o passo 1 foi bem sucedido
if exist pasibf.bin goto siga
A: imprime {Mostrará os erros ao usuário}
echo -----
echo Para continuar a Compilação deve corrigir os
echo      no Data type
echo -----
a:
exit
siga : a:pas2
a:pas1 <fonte-programa pascal>,,xx.lst,;
if exist pasibf.bin goto siga1
A: imprime {Mostrará os erros ao usuário}
a:
exit

```

```

sigal a:pas2

erase xx.lst ( Apagando os XX.lst

a:link <programa-principal>+<rotinas-matematicas>+<TAD>;;

echo -----
echo      Seu programa está pronto para ser executado
echo -----

exit

```

Vamos continuar agora vendo os outros utilitários do nosso gerenciador :

R : Serve para executar o programa corrente ou qualquer outro de tipo ".EXE" ou de tipo ".COM" .

I : Serve para imprimir uma listagem total ou parcial do programa corrente ou de qualquer outro que se encontre no diretório .

A : Serve para apagar arquivos, se o usuário precisar de eliminar algum arquivo .

D : Serve para consultar o diretório de qualquer um dos disquetes. Por "default" trabalha sobre o disquete B, no caso de estarmos trabalhando com dois drives, e com o drive A se estivermos trabalhando com disco rígido . Além disso, o usuário poderá usar todas as opções que são oferecidas para um "dir" desde o sistema operacional .

M : Serve para auxiliar o usuário durante o seu trabalho. Terá dois níveis de auxílio :

- Nível Editor
- Nível utilitários do Gerenciador

Adicionalmente, pode-se incluir uma referência ao Pascal-M. Entretanto, problemas de espaço nos disquetes dificultam, por enquanto, a instalação dessa facilidade.

T : Serve para terminar o processo e retorna o controle para o sistema Operacional .

Para terminar esta parte, torna-se conveniente salientar algumas coisas :

1.- No caso de estar trabalhando com dois drives, todo o trabalho do usuário será feito no drive B . Se estiver trabalhando com "winchester", o usuário poderá escolher entre o drive A: ou C: . Por "default" assume-se A:

2.- Os utilitários correspondentes ao Editor, Impressor, Consulta ao Manual e Apagar são parte do programa gerenciador. Para a implementação dos utilitários Compila, Executa e Diretório é gerado um arquivo de comandos chamado "WORK.BAT", o qual fará todas as tarefas correspondientes a cada uma dessas atividades, para logo retornar o controle ao programa gerenciador .

3.- Para ativar o programa gerenciador, foi desenvolvido o arquivo de comandos : PASM.BAT. Este arquivo foi construído usando a linguagem de comandos do DOS 2.1 . A seguir, vamos descrever o arquivo :

```
Echo Off { para tornar invisíveis os comandos }
Cls      { Limpando a tela }

: Inicio

    b: { Passando o drive ativo para B }

    If exist a:work.bat erase work.bat
    if exist C:work.bat erase c:work.bat

    { Apagando arquivos de trabalho anteriores }

    if exist a:pascal-m.com goto continua
    if not exist C:pascal-m.com goto erro
    c:pascal-m
    goto siga
: continua
    a:pascal-m
: siga
    if exist a:work.bat goto execa
    if not exist c:work.bat goto fim { Testando o Fim }
    command /c c:work
    goto inicio
: execa
    command/c a:work
    goto inicio
: Fim
a: { Voltando o drive ativo para A }
exit
: erro
rem O Pascal-M não foi achado
```



```

echo -----
echo O gerenciador de Pascal-m nao foi achado
echo -----
exit

```

Nota : Os utilitários PAS1, PAS2 e LINK correspondem aos passos usuais da compilação de um programa Pascal, no Compilador Microsoft[MIC 83]. Pascal-M corresponde ao programa gerenciador e WORK é o arquivo de comandos gerado pelo gerenciador .

### 5.3 O programa do Usuário

Com a finalidade de geração de um programa escrito em Pascal a partir de um escrito em Pascal-M, vamos usar um Pré-compilador. Este se encarregará de analisar o programa original, o qual para facilitar essa tarefa, deverá "marcar" com "cerquinhos"("#") a coluna 1 das linhas que contêm comandos exclusivos Pascal-M e/ou operadores gerados para os tipos abstratos de dados . O Pré-compilador analisará inicialmente as listas de constantes, types e Variáveis, dado que as linhas com "#" poderiam fazer referências a uma variável qualquer. Logo serão analisadas as linhas marcadas com "#". Numa primeira passagem só será analisada sua sintaxe. A seguir, se não tiver erros, será gerado um conjunto de comandos que farão tarefas equivalentes usando rotinas da biblioteca numérica ou da biblioteca de rotinas correspondentes aos tipos abstratos de dados . Essas rotinas

estão descritas no Capítulo III e IV . Convém salientar que os DATA TYPE, por ser uma parte do programa exclusiva do Pascal-M não precisa dessa marca especial. Sua existencia gerará uma biblioteca de rotinas para os tipos abstratos, cada uma das quais implementará a operação correspondente. Essas rotinas gerarão o arquivo ABSTRAC.PAS, o qual será compilado em separado. Um erro nele significará um erro associado aos DATA TYPE e gerará uma mensagem nesse sentido, daí a razão pela qual está sendo compilado em separado.

O novo programa equivalente ao escrito em Pascal-M terá as seguintes diferenças em relação ao programa original :

No novo programa os DATA TYPEs passam a ser somente referenciais. No novo programa tem-se incluído os tipos numéricos vistos no Capítulo III.

Nas linhas onde existiam "#" são feitas substituições por chamadas a rotinas, sejam da biblioteca numérica, sejam da biblioteca dos tipos abstratos .

Para concluir é necessário salientar dois aspectos : primeiro, que o processo de Pré-compilação aqui descrito é totalmente invisível ao usuário, o qual nem conhecerá o programa em Pascal derivado do seu. Além disso, deve-se observar que a política escolhida de geração de objetos e sua posterior ligação fará com que muitas das transformações de comandos de PASCAL-M para Pascal sejam simplesmente uma única chamada à rotina

corespondente, especialmente no caso do uso dos novos tipos numéricos .

#### 5.4 Recuperação de Erros

Um dos aspectos mais cruciais do uso de um Pré-compilador é o fato de que a compilação final será feita sobre um programa que o usuário não conhece, Portanto os erros achados serão não do programa que ele fez, mas de um outro. Por essa razão, teremos que preservar, no mínimo, a ordem dos comandos, para tornar possível um fácil retorno ao comando que produziu o erro. Estamos partindo da suposição de que as rotinas da biblioteca numérica não possuem erros. Além disso, as rotinas correspondentes aos tipos abstratos de dados são compiladas em separado, para testar prováveis erros. Portanto quando um erro acontecer, sobretudo durante a execução, deverá se retornar ao comando que fazia uso dessas rotinas .

Para conseguir este objetivo será usado em cada lugar onde existe uma marca "#", uma chamada a rotina . No caso de uma expressão, por exemplo :

```
DECODIFICAEXPRESS(Linha,aux,auxmatriz,auxcomplex,auxsegment) ;
```

Esta rotina tem por finalidade :

- 1.- Identificar os operandos e os operadores .
- 2.- Definir a ordem na qual serão desenvolvidas as operações .
- 3.- Analisar a validade das operações propostas .

4.- Definir o tipo de resultado da expressão .

Para fazer isso, será percorrida a cadeia que contém a expressão, descobrindo-se os identificadores. Com eles pode-se associar o tipo de dado correspondente. Se for de Tipo numérico, a biblioteca desses tipos será ativada, caso contrário será ativada a biblioteca dos TADs, e nelas testamos a validade das operações e a ordem na qual deverão ser executadas, assim como o tipo de resultado gerado por ela. Finalmente, com toda essa informação será montada uma rotina para o desenvolvimento da operação, a chamada dessa rotina está contida na rotina corrente . Pode-se perceber que dentro destas expressões poderiam aparecer constantes ou variáveis numéricas do tipo REAL ou INTEGER, por exemplo, mas essas estão contidas como operandos válidos dentro de nossa tabela de operadores.

Para o caso de leituras torna-se simples, pois foram definidas rotinas de leitura para cada um dos tipos numéricos correspondentes. Conseqüentemente a única coisa que deverá ser feita é um chamado às rotinas de leitura da seguinte maneira :

LEIAMATRIZ    |    LEIACOMPLEX    |    LEIASEGMENT    |    LEIAABSTRACTO

Os argumentos destas rotinas são uma string contendo a lista de variáveis e um arranjo de Matrizes, complexos, segments ou tipos abstratos, os quais armazenarão os valores lidos .

No uso das funções especiais, como é o caso das trigonométricas ou transcendentais, por exemplo, a transcrição é

linear pois, as funções e rotinas foram montadas para tal fim, conforme foi descrito no capítulo III, e para serem usadas bastará fazer uma referência elas. No entanto, já que o usuário pode tratar todas elas como se fossem funções, sendo que nem todas o são, a única tarefa a realizar será a adequação da forma de chamada.

O programa assim desenvolvido será compilado, sendo que os COMANDOS DATA TYPE, passam a ser simples comentários. A única coisa que foi mantida neles foi a declaração de tipo. Além disso, os comandos contendo "cerquinha", no início da linha, foram substituídos por uma chamada de rotinas . Então, o programa original manteve a ordem de seus comandos. Logo, ao ser compilado e a seguir executado, tendo que fazer menção a um erro, será invocado da linha correspondente . Convém precisar que no caso de erro nas rotinas que substituem os comandos com "cerquinha", a única informação recebida pelo usuário refere-se ao uso ilegal de uma rotina .

Para concluir queremos acrescentar que, com a finalidade de facilitar a correção dos erros por parte do usuário, no tempo da compilação, o usuário receberá uma listagem de seu programa juntamente com uma listagem anexa dos erros por ele cometidos e a linha de comandos correspondente .

No caso de erros durante a compilação, o gerenciador gera um ambiente que permite ao usuário revisar seu programa e sua

listagem em forma parcial ou total. Para a revisão daqueles é gerado o seguinte menu de opções :

```
-----  
(C)ontinuar   (P)agina Anterior   (L)inhas Anteriores   (F)im  
-----
```

Abaixo desse menu aparecerá o programa e seus erros desde sua página inicial, ficando detido no fim de cada página, onde o usuário apertará uma das letras associadas ao menu de opções para poder continuar .

### 5.5 Limitações

Conforme vimos nas duas seções anteriores, nosso processo de Compilação tem uma parte adicional, na qual adaptamos o Pascal-M ao Pascal standard. Isso representa um gasto de tempo adicional no processo . Além disso, o Compilador de Pascal da Microsoft é um pouco lento. Por essa razão um dos principais problemas que teremos será de velocidade.

Por outro lado seguindo as características do Pascal, não tem sido possível a criação dinâmica de índices para os arranjos, isso tem-nos levado a definir o tipo MATRIZ usando como topo para o número de linhas e colunas o valor máximo para as linhas e colunas respectivamente . Isso pode nos levar a um estouro de memória. Caso o usuário queira usar usar matrizes de tamanho diferente, poderá fazê-lo por via dos DATA TYPE .

Outra das limitações de nosso trabalho refere-se ao Editor. Embora tendo-se colocado um conjunto de opções dentro dele, hoje em dia existem no mercado editores muito mais poderosos que ele, porém, estes objetivos fogem da meta deste trabalho .

Finalmente, poderíamos acrescentar que, por problemas de espaço no disquete, dado que o compilador da Microsoft consome muito, não será possível incluir o Sistema operacional junto ao compilador .

#### **5.6 O Pascal-M e o Pascal-SC**

No início do trabalho foi possível observar o fato que existe uma versão do Pascal, chamada Pascal-SC [ 25 ] , a qual pretende também servir para aplicações Técnicas e Científicas, sendo que nosso trabalho pretende atingir as mesma metas, e se faz necessária uma comparação entre um e outro .

Em primeiro lugar, ambos os trabalhos incorporam como novos tipos numéricos os Complexos, Intervalos e Matrizes, sendo que em Pascal-SC incorpora-se junto com isso a possibilidade de acrescentar a precisão dos reais. Em compensação o Pascal-M incorpora o tratamento amplo de Tipos Abstratos de Dados . Por outro lado, tanto em Pascal-SC, como em Pascal-M a metodologia adotada é a criação de Bibliotecas por tipo, sendo que Pascal-SC usa inclusão da Biblioteca e Pascal-M utiliza a ligação. Esta última técnica permite que sejam anexadas ao "EXE" do programa, somente as rotinas que este vai utilizar, ao contrário do que

acontece no \$INCLUDE, que incorpora todas as existentes .

Outra das diferenças do Pascal-M com respeito ao Pascal-SC refere-se ao ambiente de trabalho. Para o Pascal-M, existe um gerenciador, o qual governa todas as atividades inerentes ao trabalho de "rodar" um programa e atividades correlatas. Esse gerenciador gera um ambiente altamente autoinstrutivo, o qual, além de estar totalmente em Português, possui manuais de auxílio ao usuário. Além do mais, o editor do Pascal-M, possui uma maior quantidade de recursos de edição do que seu similar de Pascal-SC, sendo a única vantagem deste a análise de sintaxe desde o editor, a qual pode tornar a edição do programa muito demorada.



## C O N C L U S Õ E S

---

Ao finalizarmos este trabalho é necessário que façamos uma avaliação sobre a performance de nossa proposta. Esta avaliação será feita nos termos das metas que tínhamos trazado ao início dele.

### - Sobre as facilidades ao Usuário

Nosso trabalho produziu um gerenciador auto instrutivo que vai permitir ao usuário, por mais inexperiente que seja, trabalhar com facilidade dentro dele. Para isso, fornece recursos de fácil acesso para todas as tarefas que ele precisa realizar para "rodar" um programa num microcomputador ou apanhar informação necessária para isso, como consultas ao Diretório, por exemplo. Possui um editor razoável, cujo manejo não exigirá memorizar que funções tem cada tecla, pois sempre haverá um Menu para auxiliá-lo. Uma futura extensão de nosso gerenciador deverá visar a fornecer ao Editor de maiores recursos, como por exemplo copiadores de blocos. Além disso, após experimentar o uso dos manuais de ajuda ao usuário, poderia-se avaliar as necessidades de ampliar ou encurtá-los .

### - Sobre as Estruturas Numéricas

A linguagem Pascal-m proposta incorpora todas as estruturas numéricas (Intervalos, Complexos e Matrizes) que nós traçamos como meta ao início do trabalho. Para essas estruturas tem-se

considerado sua aritmética e funções standard . Além disso, o uso delas será semelhante à forma como são usadas as estruturas numéricas já existentes .

#### - Sobre os Tipos Abstratos de Dados

A linguagem fornece uma seção especial(DATA TYPE) para a sua criação, onde se permite a definição de tipo e operações associadas a ele e dentro do programa eles poderão ser usados, declarando-se previamente um "#", para a sua identificação como comando especial . Uma futura extensão de nosso trabalho deve visar à eliminação de tais "#" .

#### - Performance

Sobre a performance do compilador Pascal-M, quanto a cumprir as finalidades que deram origem a ele, é boa, porém, em termos de ser um compilador eficiente, ainda deixa a desejar. As razões disto são devido a estar usando um Pré-Compilador. Por isso, nossa recuperação de erros, não é simple . Por outro lado, o fato de usar como "Compilador Base" o dcompilador da Microsoft, o qual precisa de dois passos prévios para a geração do Objeto e, além disso, um processo "link" para gerar o programa executável, faz, de nosso processo de compilação um processo lento . É desejável, numa continuação deste trabalho, a construção integral deste compilador, medida que facilitaria a tarefa de recuperação de erros. Além disso, este compilador deverá visar fazer deste processo, um processo mais rápido .

**A N E X O I**  
-----**M A N U A L   D O   E D I T O R**

## Movimentos do Cursor :

Para movimentar o cursor você poderá usar as setas que se encontra no lado direito do seu teclado. Verifique que a tecla BLOCO/NUM esta desativada, pois nesse caso as setas ficam desativadas e você estará usando os números. A Tecla "Pag Seta Superior" levará o cursor ao início da Página corrente do Arquivo e a tecla "Pag Seta Inferior" levará o cursor ao fim da Página corrente do arquivo .

Nota : O arquivo apresentará 18 linhas por página .

## Uso das Funções do Menu

Para usar as opções descritas no MENU situado na parte superior da tela você vai utilizar as teclas programadas F1,F2,F3....e F8 estas teclas se encontram no lado esquerdo do teclado e funcionam assim .

F1 : Serve para gravar, mas não sai do editor .

F2 : Sai do editor, mas, antes confere se foi feita alguma modificação. Nesse caso, perguntará se deseja gravar ou não

F3 : Leva a linha ativa do cursor para o início do arquivo .

F4 : Apaga a linha corrente .

F5 : Muda o estado de inserção de ativo(S) para passivo(N) ou vice-versa. Para verificar em que estado se encontra a inserção, observe a mensagem que aparece junto ao [F5]. Inserir(S) indica estado ativo.

F6 : Serve para copiar o conteúdo do arquivo corrente num novo arquivo. Isto é muito útil para quando você deseja modificar um programa, que está funcionando, e preservá-lo das modificações .

F7 : Leva a linha ativa para o fim do arquivo.

F8 : Apaga a palavra corrente a partir da posição corrente . O texto que se encontrava `a direita sera "puxado" `a esquerda

Observa-se que não temos nenhuma opção para apagar caracter, Ocorre que isso pode ser feito usando-se a tecla DEL, esta tecla encontra-se na parte inferior direita .

A N E X O    I I

---

M A N U A L    D O    G E R E N C I A D O R

---

Regras Gerais :

- O gerenciador está previsto para trabalhar sempre no drive "A" . No caso que se esteja trabalhando com winchester, o gerenciador estará no drive C .
- No drive "B" voce colocará seu disco de trabalho, o qual deve estar previamente FORMATADO. Para formatar seu disquete use o utilitário FORMAT de seu Sistema Operacional .
- Quando um dos utilitários do gerenciador precisar de um nome de arquivo, voce deverá observar a seguinte sintaxe :

<drive>:<nome>.<ext>

onde <drive> : Indica o drive ativo, por DEFAULT será B:

opções : A | B | C

<nome> : Conjunto de 11 caracteres como máximo .

<ext> : Conjunto de 3 caracteres como máximo .

Dos três só o nome é obrigatório. A extensão será, por DEFAULT "PAS", no caso de arquivos de Programas Fonte e "EXE" no caso de Programas a serem executados . No caso de você não incluir <ext> o ponto não será necessário .

## Opções do Gerenciador :

- (E)ditar : Serve para escrever um arquivo novo ou modificar um feito anteriormente . Ao ingressar ao editor voce será informado se seu arquivo e' novo ou não.  
Maiores detalhes sobre o editor voce encontrará no Manual do editor
- (C)ompilar : Serve para Compilar, usando o compilador da IBM, programas feitos em Pascal ou PASCAL-M. Se a compilação for bem sucedida, ele gerará um programa do mesmo nome que o seu, porém do tipo ".EXE" , o qual será executável .
- (R)un : Serve para Executar todo programa executável(Tipo ".EXE" ou ".COM"), porém, no caso dos tipos ".COM" ao dar o nome de seu arquivo, deverá indicar junto com ele se <ext> .
- (I)mprimir : Serve para listar na impressora um arquivo qualquer Esta impressão poderá ser parcial ou total . Para isso voce será perguntado, uma vez reconhecida a existencia do Arquivo .
- (A)pagar : Serve para eliminar arquivos . Tenha cuidado com esta opção pois uma vez apagado o arquivo, este será irrecuperável .

(D)iretorio: Serve para revisar o conteúdo dos disquetes que você está usando . Por DEFAULT será fornecido o conteúdo total do disquete no drive "B" . Voce poderá usar todos os recursos que oferece o sistema operacional para a comando DIR .

(M)anual : Serve como um nível de ajuda para o usuário e permite revisar

- Manual do Gerenciador (E' quem governa o MENU principal)
- Manual do Editor

Para o primeiro caso pressione a tecla "G", para o segundo caso, use a tecla "E" . Para sair pressione a tecla "F" ou ENTER .

Uma vez escolhida, a opção aparecerá na tela, na parte superior:

(C)ontinua      (P)agina traz      (L)inhas traz      (F)im

e na parte inferior a primeira página do manual respectivo, para passar às páginas seguintes pressione a tecla "C"; no caso de que queira recuar uma página use a tecla "P"; no caso de que só precise recuar meia página use a tecla "L" . Para sair do manual use a tecla "F"

(T)erminar : Fim do Programa Gerenciador e retorno ao Sistema Operacional

Observação Final :

No caso de que, por erro, voce ingresse um opção não desejada, aperte ESC e retornará ao MENU principal

Boa Sorte !



B I B L I O G R A F I A

---

- [AHO 77] AHO, Alfred V. Ullman, Jeffrey D. . Principles of  
Compiler Design. Philadelphia, Addison Wesley,  
1977.
- [ALA 78] ALAGIC S. Arthur. The Design of well structured and  
Correct Programs. New York, Springer-Verlag, 1978.
- [BRE 85] BREZIN, J. P. . Some Aspects of the 801 Compiler  
Project. New York, Watson Research Center, 1985 .
- [CLA 84] CLAUDIO, Dalcidio M. . Noções de Aritmética  
Intervalar. Porto Alegre, UFRGS, 1984.
- [DIU 82] DIUDONNE, Jean . Fundamentos de Análisis Funcional.  
Madrid, Reverte, 1985.
- [FRI 77] FRIEDMAN, Frank and KOFFMAN, Elliot B. . Problem  
Solving and Structured Programming in FORTRAN.  
Philadelphia, Addison Wesley, 1977.
- [GAR 85] GARCIA LOPEZ, Javier . Noções de Matemática  
Intervalar. Porto Alegre, CPGCC-UFRGS, 1985
- [GHE 85] GHEZZI, Carlo e MEHDI, Jazayeri. Conceitos de  
Linguagens de Programação . Rio de Janeiro, Campus,  
1985.

- [GIL 77] GILL Arthur . **Applied Algebra for the Computer Sciences** . New Jersey, Prentice Hall, 1977.
- [GUT 78] GUTTANG J. V. and HOROWITS, E. . **The Design of Data Type Especifications**. Potomac, Computer Sciences Press, 1978 .
- [HAU 72] HAUSER, Arthur A. **Variáveis Complexas**. Rio de Janeiro, Livros Técnicos e Científicos, 1972.
- [HP 85] HP do Brazil . **HP-85, Manual de Referencia**. São Paulo, 1985
- [IBM 82] IBM . **FORTRAN Compiler Reference Manual** . Boca Raton, 1982 (P/N 6024012).
- [IBM 82a] IBM, **BASIC Reference Manual** . Boca Raton, 1982 . (P/N 6025013)
- [LUD 81] LUDOW-WIECHERS, Jorge A. . **Algebra Lineal**. Mexico, Limusa, 1981.
- [MIC 83] MICROSOFT Corporation . **Microsoft Pascal Reference Manual**. New York, 1983.
- [MUN 77] MUNAKA Toshiri. **Matrices And Linear Algebra**. New York, Holden Day, 1977
- [NEA 85] NEAGA, Michel. **Pascal- SC User's Guide** . Karlsruhe 1985.

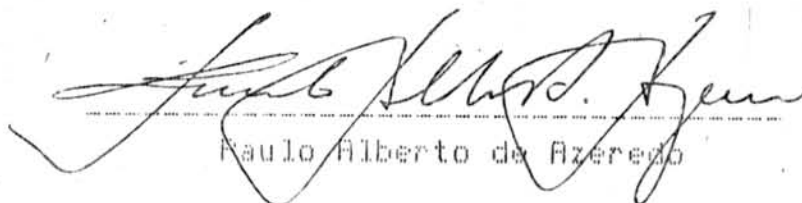
- [PIE 84] PIESSENS, Robert and HUYSMANS, Rudi . Algorithm 619, Automatic Numerical Inversion of Laplace Transform . **ACM Transactions on Mathematical Software**, New York **10(3)** : 348-53, Sep 1984.
- [RAM 86] RAMALHO, Jose Antonio A. . **Wordstar : Manual para Processamento de Textos** . São Paulo, Atlas, 1986.
- [REI 81] REID, J.K. . **The Relationship numerical Computations and Programming Language**. Amsterdam, North Holland 1981.
- [SCO 83] SCOPUS . **Turbo Pascal Language Manual** . 1983.
- [TAN 83] TANEBAUM, Aaron M. & AUGENSTEIN Moshe . **Estructura de Datos em Pascal** . Madrid, Prentice Hall International, 1983 .
- [VEL 86] VELOSO, Paulo A. S. **Tipos Abstratos de Dados : Programação, Especificação e Implementação** . Belo Horizonte, UFMG, 1986.
- [WEL 83] WELSH, Jim and ELDER, John . **Pascal : Curso Introductorio** . Madrid, Prentice-Hall International, 1983.
- [WOO 87] WOOD, Steve . **Turbo Pascal, Manual de Referencia** . Rio de Janeiro, Mc Graw Hill, 1987.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

POS-GRADUAÇÃO EM CIENCIA DA COMPUTAÇÃO

Implementação de um Compilador para  
Software Numérico

Dissertação apresentada aos Srs.

  
Paulo Alberto de Azeredo

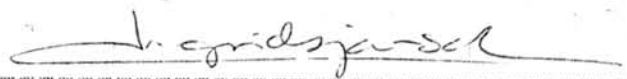
  
Dalcídio Moraes Cláudio

  
Magda Bercht

  
Julio César Ruiz Claeysen

Visto e permitida a impressão.

Porto Alegre 19/ 02/ 88



Ingrid E. S. Jansch Pôrto

Coordenador do Curso de Pós-Graduação em  
Ciência da Computação