APONTAMENTOS DE PASCAL

PROGRAMAÇÃO DE COMPUTADORES

1º ANO
DA LICENCIATURA EM ENGENHARIA MECÂNICA

2000

ÍNDICE:

INTRODUÇÃO

Neste trabalho introduzimos alguns conceitos fundamentais para podermos comunicar com um computador de modo a que este execute uma dada tarefa. Para isso temos de especificar uma sequência de passos através de uma linguagem compreendida pelo computador (linguagem de programação), de modo a que ele possa efectuar essa tarefa, ou seja temos de escrever um programa. A actividade de escrever programas chama-se programação. Desejamos assim, com este trabalho, efectuar uma introdução à programação disciplinada, fornecendo os passos necessários ao desenvolvimento de um programa, de forma a obter programas bem estruturados, eficientes e sem erros.

A elaboração de um programa contém várias fases. Em primeiro lugar é necessário definir o objectivo ou seja definir exactamente o que se pretende. Em seguida é importante analisar as várias alternativas para obter a solução e escolher a melhor, construindo assim um algoritmo eficiente detalhando todos os passos necessários.

Uma vez definido o algoritmo é necessário codificá-lo numa linguagem de programação adequada, sendo essencial uma boa linguagem, com boas ferramentas de apoio, de modo que seja fácil expressar correctamente as ideias do programador e a estrutura dos algoritmos já desenvolvidos. Não nos podemos esquecer, que a nossa tarefa não termina com a escrita do programa, porque é ainda necessário efectuar testes para detectar e corrigir os erros.

A linguagem de programação escolhida é a linguagem Pascal desenvolvida no início da década de 70. Embora esta linguagem não seja utilizada em muitas aplicações reais, consideramos interessante o seu estudo, pois ela foi desenvolvida com a finalidade de servir para ensinar a programação, uma vez que dá origem a programas bem estruturados e organizados.

Dado que os alunos durante o seu curso de Engenharia terão de utilizar outras linguagens de programação, nomeadamente o Fortran 90 e o Visual Basic, ao longo de todo o trabalho, os temas comuns a um grande leque de linguagens de programação serão abordados de forma mais detalhada para a linguagem Pascal, não deixando de se referir de uma forma sucinta a “passagem” para a linguagem Fortran 90.

1.1 A estrutura do algoritmo e do respectivo programa

Um programa é um conjunto de instruções bem definidas, que especificam exactamente o que tem de ser feito, escrito numa linguagem de programação. Um algoritmo é mais abstracto que um programa pois pode ser expresso numa linguagem informal. Podemos dizer que um programa é a expressão de um algoritmo numa linguagem de programação.

Um algoritmo está sempre associado à solução de um problema ou seja a um objectivo, e não é mais do que a descrição da sequência de passos a seguir para atingir esse objectivo; de uma forma simplificada é a " receita clara e precisa ", que permite atingir o objectivo. Assim um algoritmo deve satisfazer um conjunto de características:

- Ser rigoroso ou seja cada instrução deve ser clara, não ambígua e só ter uma interpretação; por exemplo a instrução "somar x ao quadrado de um número " não especifica a que número se soma a variável x nem onde se guarda o resultado.

- Ser finito ou seja deve conduzir a uma situação em que o objectivo seja atingido e não existam mais instruções para serem executadas. Por vezes um algoritmo mal executado conduz a um programa que nunca termina.

- Deve permitir a entrada de dados e deve escrever resultados que traduzam o objectivo que se pretende atingir.

- Deve ser eficiente no tempo e no espaço. De uma forma simplificada podemos dizer que um algoritmo é eficiente no tempo quando realiza apenas as instruções necessárias para atingir o objectivo, e que é eficiente no espaço quando utiliza apenas as variáveis necessárias.

É aconselhável utilizar-se uma linguagem auxiliar abstracta e simples de compreender que exprima sem detalhes excessivos a estrutura do algoritmo; posteriormente a conversão do algoritmo para uma qualquer linguagem de programação já não oferece grandes dificuldades.

Um algoritmo é constituído por duas partes o cabeçalho e o corpo do algoritmo; no cabeçalho define-se o objectivo do algoritmo e declaram-se as variáveis(objectos):

Objectivo: resumo do que se pretende com o algoritmo

variáveis dados: <variável> , tipo , significado

variáveis resultado: <variável> , tipo , significado

variáveis auxiliares: <variável> , tipo , significado

O corpo do algoritmo é constituído pelas instruções que traduzem a sequência de passos necessários para se atingir o objectivo do algoritmo. As instruções (separadas por ";") podem ser representadas da seguinte forma:

1) De leitura : leia (<lista de variáveis>)
2) De escrita : escreva (<lista de expressões ou texto>)
3) Atribuição: <variável> ← <expressão>
4) Condicional: se <expressão lógica> então { <instrução 1> }
   ou: se <expressão lógica> então { <instrução 2> }
   senão {< instrução 1> }
5) Enquanto: enquanto <expressão lógica> faça {< instruções> }
6) Repita: repita < instruções> até <expressão lógica>
7) Para:
   para <variável> ← <exp1> até >exp2> faça {< instruções> }
   ou
   para <variável> ← <exp1> descendo até >exp2> faça {< instruções> }

As designações {<instruções>}, {<instrução 1>} e {<instrução 2>} significam uma qualquer sequência de instruções separadas entre si por ";". As chavetas podem ser dispensadas se < instruções> for apenas uma instrução. A designação <expressão> (ou <exp1>, <exp2>) significa uma expressão de qualquer tipo numérica ou lógica (ver parágrafo 2.9).

Consideremos por exemplo que queríamos construir um algoritmo para calcular a média de um conjunto de números inteiros positivos:

Objectivo: cálculo da média de um conjunto de números positivos

variáveis dados: n, inteiro, quantidade de números lidos;
numero, inteiro, número lido;
variáveis auxiliares: soma, inteiro, soma dos números lidos;

```
soma ← 0 ;  n ← 0;
escreva (“escreva um número positivo” ) ;
escreva ( “ um  número negativo  termina o  programa” );
leia (numero);
enquanto numero > = 0 faça
        soma ← soma + numero;
        n ← n + 1;
        escreva (“escreva um número positivo” ) ;
        escreva ( “ um  número negativo  termina o  programa” );
        leia (numero);
se   n = 0 então
    escreva (“impossível calcular a média”)
senão
    escreva (“a média é : “,soma / n);
```

Analisando este algoritmo vemos que:

- O objectivo do algoritmo explica o que se pretende.

- A declaração do tipo de variáveis(objectos) usadas, feita no cabeçalho do algoritmo, é seguida por um comentário onde se explica o significado de cada uma das variáveis.

- No início do corpo do algoritmo os valores das variáveis soma e n são postos a zero e em seguida é lido um número. Então começa um ciclo repetitivo: enquanto o número lido for positivo é repetida a sequência de instruções delimitadas pela chaveta, actualizando as variáveis soma e n e lendo um novo número. Depois de ler um número negativo se o total de números lidos for zero, existe uma instrução para escrever que não é possível calcular a média, e em caso contrário para escrever a média dos números lidos.

Apresentamos o programa correspondente, em linguagem Pascal, com a finalidade de o compararmos com o respectivo algoritmo:

```
Program media (input, output) ;

{este programa calcula a média de um conjunto de números positivos}

var numero , soma , n : integer;
{numero – número lido
 soma – soma dos números lidos
 n – quantidade de números lidos}

begin
    soma : = 0 ;   n : = 0;
    writeln ( 'escreva um número positivo' ) ;
    writeln ( 'um número negativo termina o programa' );
    read (numero);
    while numero > = 0 do
        begin
            soma : = soma + numero;
            n : = n + 1;
            writeln ( 'escreva um número positivo' ) ;
            writeln ( 'um número negativo termina o programa' );
            read (numero);
        end;
    if   n = 0 then
        writeln ('impossível calcular a média')
    else
        writeln ('a média é : ',soma / n)
end.    { Program media }
```

Analisando agora o programa podemos concluir que a sua estrutura é muito semelhante à do algoritmo que o originou.

- O nome do programa é media. A seguir ao nome do programa existe um comentário que explica o que o programa faz.

- A declaração do tipo de variáveis usadas, feita na linha que começa por "var", é seguida por um comentário onde se explica o significado de cada uma das variáveis.

- A palavra begin assinala o início da zona de instruções. Os valores das variáveis soma e n são postos a zero e é lido um número. Então começa um ciclo repetitivo: enquanto o número lido é positivo o computador repete a sequência de instruções delimitadas pelo par begin/end actualizando as variáveis soma e n e lendo um novo número. Depois de ler um número negativo o computador testa se o total de números lidos é zero, escrevendo neste caso que não é possível

calcular a média, e em caso contrário escreve a média dos números lidos.
As diferenças entre o algoritmo e o programa não são muito significativas:

- O programa inicia-se com uma linha do tipo:

    program <nome>(<ficheiros usados>);

onde os ficheiros usados são “input” (o teclado do terminal) e “output” (a saída no monitor do terminal). Segue-se um comentário que explica o objectivo do programa.

- A declaração de todas as variáveis (objectos) é feita com a palavra var .

- A zona de instruções que manipula os objectos encontra-se entre um **begin** e um **end**, seguido de um ponto final, que indica o fim do programa.

- As instruções são semelhantes às da linguagem de representação de algoritmos adoptada, com a correspondência de palavras:

    escreva ↔ writeln
    leia ↔ read
    se ↔ if
    senão ↔ else
    então ↔ then
    enquanto ↔ while

- Em vez de chavetas usa-se o par begin-end.

- Nas atribuições usa-se o sinal “:=” em vez de “←”.

- Os textos para serem escritos limitam-se por plicas em vez de aspas.

Antes de continuarmos a falar de programas em Pascal é necessário definir alguns conceitos que serão usados ao longo deste trabalho.
Um identificador é um nome que é usado para designar um programa ou entidades num programa.

Em Pascal um identificador tem que começar por uma letra (maiúscula ou minúscula), a qual pode ser seguida por qualquer combinação de letras ou de dígitos.

O nome de um identificador deve sugerir a utilização ou o significado da entidade representada.

Entre os identificadores legais em Pascal existem dois conjuntos. O conjunto dos *nomes reservados* constituído por todos os nomes (identificadores) que são utilizados em Pascal para um fim específico e que não podem ser utilizados a não ser no contexto em que são definidos para a linguagem; são exemplos desses nomes "begin", "end", "for", "div", "type", etc". O conjunto dos *nomes normalizados* constituído por todos os nomes que têm um significado previamente definido em Pascal, podendo no entanto ser alterado pelo programador; são exemplos os nomes de funções intrínsecas como "abs", "arctan", "cos", "exp", etc, e de procedimentos como "write", "read", "readln", etc.

Analisando o exemplo anterior podemos ver que tal como um algoritmo, um programa também é composto por duas partes distintas: um cabeçalho de programa que define o nome do programa e a maneira como ele comunica com o exterior, e o bloco que contém a descrição dos objectos utilizados pelo programa e o conjunto de instruções que manipulam esses objectos.

O cabeçalho é sintacticamente definido por:

<cabeçalho de programa> ::= program <identificador> (<identificadores de ficheiros>)

O bloco de um programa é sintacticamente definido por:

<bloco> : : = <zona de definição de constantes>
<zona de definição de tipos>
<zona de declaração de variáveis>
<zona de declaração de procedimentos ou funções>
<zona de instruções>

As definições sintácticas da zona de definição de constantes e da zona de declaração de variáveis serão tratadas no capítulo 2, sendo a zona de definição de tipos tratada no capítulo 4.
A definição da zona de declaração de procedimentos ou funções será dada no capítulo 5.

A zona de instruções é constituída pelas instruções correspondentes ao corpo do algoritmo do programa que manipulam os objectos usados pelo programa, sendo delimitada pelas palavras “begin” no início e “end” no fim.

Para facilitar a leitura de um programa devemos introduzir comentários, isto é frases em linguagem natural que explicam o significado e os objectivos de zonas do programa. Os comentários em Pascal são delimitados por símbolos “{” e “}” e são ignorados pelo computador assim como os espaços ou linhas em branco.

Dentro de um bloco de um programa é possível definir e associar acções com um nome; sempre que essa sequência de acções precise de ser executada só é necessário referir o seu nome.

É importante este conceito de podermos dividir um programa numa série de blocos ou subprogramas, cada um com tarefas bem definidas e usar esses subprogramas várias vezes dentro de um programa, sempre que necessário. Este processo corresponde ao conceito de abstracção procedimental realizado em programação através de subprogramas que serão analisados em pormenor no capítulo 5.

Consideremos novamente o exemplo que calcula a média de um conjunto de n números inteiros positivos, usando agora um procedimento para a leitura de cada número:

```
Program media (input, output) ;

{este programa calcula a média de um conjunto de números positivos}

var numero , soma , n : integer;
{numero – número lido
 soma – soma dos números lidos
 n – quantidade de números lidos}

procedure leitura (var numero : integer) ;
    begin
        writeln ( 'escreva um número positivo' ) ;
        writeln ( 'um número negativo termina o programa' ) ;
        read (número);
    end;

begin
    soma : = 0 ;   n : = 0;
    leitura (numero);
    while numero > = 0 do
        begin
            soma : = soma + numero;
            n : = n + 1;
            leitura (numero);
        end;
    if   n = 0 then
        writeln ('impossível calcular a média')
    else
        writeln ('a média é : ', soma / n)
end.    { Program media }
```

Analisando este programa vemos que:

- O nome do programa é media. A seguir ao nome do programa existe um comentário que explica o que o programa faz.

- A declaração do tipo de variáveis usadas, feita na linha que começa por "var", é seguida por um comentário onde se explica o significado de cada uma das variáveis.

- Em seguida é declarado um subprograma (um procedimento) que lê cada número escrito pelo utilizador.

- A palavra begin assinala o início da zona de instruções. Os valores das variáveis soma e n são postos a zero e em seguida é chamado o subprograma para ler um número. Então começa um ciclo repetitivo: enquanto o número lido é positivo o computador repete a sequência de

instruções delimitadas pelo par begin/end actualizando as variáveis soma e n e chamando o subprograma para ler um novo número. Depois de ler um número negativo o computador testa se o total de números lidos é zero, escrevendo neste caso que não é possível calcular a média, e em caso contrário escreve a média dos números lidos.
A diferença entre este programa e o apresentado anteriormente é a existência da declaração do procedimento de leitura de um número e a sua utilização dentro da zona de instruções sempre que é necessário ler um número.

Neste capítulo mostramos o aspecto de um programa em Pascal, embora não se tenha tentado ensinar como escrever um programa, uma vez que esta tarefa será feita nos restantes capítulos.

2. TIPOS DE INFORMAÇÃO ELEMENTARES

A utilização de tipos para caracterizar constantes, variáveis, expressões e funções é muito importante em informática.

Um tipo determina o conjunto de valores aos quais uma constante pertence, que podem ser assumidos por uma variável ou expressão, ou que podem ser gerados por uma função. Além disso cada operador ou função tem argumentos de um dado tipo e produz resultados de um certo tipo como veremos.

As linguagens de programação fornecem certos tipos elementares e a capacidade de definir certos tipos estruturados, permitindo ainda que o programador defina novos tipos de informação (elementares ou estruturados). Os tipos elementares são caracterizados pelo facto de as suas constantes serem tomadas como indecomponíveis, ao passo que os tipos estruturados são caracterizados pelo facto de as suas constantes serem constituídas por um agregado de valores.

Vamos considerar alguns tipos de informação elementar que existem pré-definidos como os tipos lógico, carácter, inteiro e real. Como tipos estruturados existem os tipos tabela, ficha, conjunto e ficheiro. Neste trabalho não falaremos de todos estes tipos de informação, nomeadamente dos tipos estruturados conjuntos e ficheiros.

2.1 Tipo lógico

Em programação é possível decidir se uma dada expressão é verdadeira ou falsa recorrendo ao tipo lógico (“boolean” em Pascal).

O tipo lógico só pode assumir dois valores “true” (verdadeiro) e “false” (falso).

As operações que podemos efectuar sobre valores lógicos, produzindo valores lógicos são de dois tipos, as operações unárias e as operações binárias. As operações unárias produzem um valor lógico a partir de um valor lógico. Em Pascal existem predefinidas três operações unárias (tabela 2.1):

- A operação “not” (“não” em linguagem natural) muda o valor lógico.

- A operação “succ” (“sucessor”) produz o valor lógico que aparece (na ordem definida entre as constantes lógicas) depois do seu argumento.

- A operação “pred” (“antecessor”) produz o valor lógico que aparece (na ordem definida entre as constantes lógicas) antes do seu argumento.

A	**not (A)**	**succ (A)**	**pred (A)**
false	true	true	-
true	false	-	false

Tabela2.1: Operações unárias sobre valores lógicos

As operações binárias aceitam dois argumentos do tipo lógico e produzem um valor do tipo lógico. Entre estas operações existem as operações lógicas de conjunção (representadas pelo operador “and”) e disjunção (representadas pelo operador “or”).

A	**B**	**A and B**	**A or B**
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Tabela2.2: Operações de conjunção e disjunção

As outras operações binárias, chamadas operações relacionais, comparam a posição relativa dos seus argumentos, na ordem definida para as constantes lógicas (succ(false)=true). As operações relacionais existentes em Pascal estão representadas na tabela 2.3.

A	B	A = B	A < > B	A < = B	A > = B	A < B	A > B
true	true	true	false	true	true	false	false
true	false	false	true	false	true	false	true
false	true	false	true	true	false	true	false
false	false	true	false	true	true	false	false

Tabela2.3: Operações relacionais sobre valores lógicos

2.2 O tipo carácter

A informação do tipo carácter (“char” em Pascal) é composta pelo conjunto (ordenado) de todos os caracteres que podem ser representados num dado computador.

As constantes do tipo carácter (ou alfanuméricas) são representadas dentro de um programa, entre plicas:

A : = ‘a’;
B : = ‘b’;

Esta representação surge da necessidade de distinguir uma constante do tipo carácter de uma variável. Existe uma relação de ordem total no conjunto de caracteres ou seja uma ordem definida sobre as constantes do tipo carácter que depende do código utilizado para representar os caracteres. Para o código ASCII a ordem entre os caracteres está representada no anexo A.

As operações que se podem efectuar sobre caracteres, produzindo caracteres são apenas as operações unárias:

- A operação “succ” produz o carácter que aparece (na ordem definida entre caracteres) imediatamente depois do seu argumento, ou seja succ (‘c’) tem o valor ‘d’ (para o código ASCII).

- A operação “pred” produz o carácter que aparece (na ordem definida entre caracteres) imediatamente antes do argumento, ou seja pred (‘d’) tem o valor ‘c’.

Note que apesar de estas operações terem o mesmo nome que operações semelhantes existentes para os valores lógicos, elas representam operações diferentes.

2.3 Cadeias de caracteres

Uma cadeia de caracteres ("string" em Pascal") é um tipo estruturado de informação, que é definido como uma sequência de caracteres. O comprimento da cadeia é o número de caracteres que a constitui.

As constantes de cadeias de caracteres, tal como as constantes alfanuméricas, são representadas em Pascal delimitadas por plicas:

nome: = 'João Pedro Silva'

1.4 O tipo inteiro

O tipo inteiro ('integer' em Pascal) é constituído por números sem parte decimal, podendo ser positivos, negativos ou zero. Sobre os elementos do tipo inteiro podemos efectuar as operações representadas na tabela 2.4 as quais transformam inteiros em inteiros.

Operador	**Definição**	**Uso**	**Significado**
-	unária	- a	simétrico de **a**
succ	unária	succ (32)	sucessor de 32 tem o valor 33
pred	unária	pred (2)	antecessor de 2 tem valor 1
abs	unária	abs (- 32)	valor absoluto de -32
sqr	unária	sqr (5)	o quadrado de 5
+	adição (binária)	a + b	**a** mais **b**
-	subtracção (binária)	a - b	**a** menos **b**
*	multiplicação (binária)	a * b	**a** multiplicado por **b**
div	divisão inteira (binária)	a div b	**a** dividido por **b**
mod	resto de divisão inteira (binária)	a mod b	resto da divisão inteira de **a** por **b**

Tabela 2.4: Operações para o tipo inteiro

2.5 Tipo real

Os números reais (em Pascal “real”) são números com parte decimal. Na maioria das linguagens de programação existem dois métodos para a representação de constantes do tipo real:

- A notação decimal, em que se representa o número, com ou sem sinal, por uma parte inteira, um ponto correspondente à virgula e uma parte decimal. São exemplos desta representação os números 7.0 , -43.00 e 0.856.

- A notação científica em que se representa o número com ou sem sinal, através de uma mantissa (inteira ou real) e de uma potência inteira de dez (expoente) que multiplicada pela mantissa produz o número. São exemplos desta representação os números 7E0, -4.3E1 e 8.56E-1, representados em notação decimal no parágrafo anterior.

Sobre os elementos do tipo “real” podem efectuar-se as operações representadas na tabela 2.5, as quais transformam reais em reais.

Note que int (a) é um número real igual à parte inteira de a ; por exemplo:

Int (2.56) = 2.

Como os computadores têm uma memória limitada existe sempre um limite para a gama de inteiros e reais representáveis. Este limite depende do computador e da versão do Pascal utilizada.

Operador	Definição	Uso	Significado
-	unária	- a	simétrica de **a**
abs	unária	abs (a)	valor absoluto de **a**
sqr	unária	sqr (a)	quadrado de **a**
sqrt	unária	sqrt (a)	raiz quadrada de **a** ($a \geq 0$)
sin	unária	sin (a)	seno de **a** (a radianos)
cos	unária	cos (a)	Cosseno de **a** (a radianos)
arctan	unária	arctan (a)	arco-tangente de **a** ($y=\arctan(a)$; $-\pi/2 < y < \pi/2$)
ln	unária	ln (a)	logaritmo natural de **a** ($a > 0$)
exp	unária	exp (a)	exponencial de **a**
int	unária	int (a)	parte inteira de **a**
frac	unária	frac (a)	parte fraccionária de **a**
+	adição (binária)	a + b	**a** mais **b**
-	subtracção (binária)	a - b	**a** menos **b**
*	multiplicação (binária)	a * b	**a** multiplicado por **b**
/	divisão (binária)	a / b	**a** dividido por **b**

Tabela 2.5: Operações para o tipo real

2.6 Transformações entre tipos

Embora as constantes do tipo inteiro, real, carácter e lógico tenham sido definidas como conjuntos disjuntos existem operações que transformam os elementos de um dos tipos de informação elementar num elemento de outro tipo de informação elementar.

2.6.1 Transformações entre inteiro e real

A transformação de um real num inteiro pode ser feita de dois modos distintos:

1) O real é arredondado para o inteiro mais próximo:

round (4.3) = 4
round (4.6) = 5
round (4.5) = 5

2) O real é transformado no inteiro que corresponde à sua parte inteira:

trunc (4.3) = 4
trunc (4.6) = 4

2.6.2 Transformações entre inteiro e carácter

As transformações entre o tipo inteiro e o tipo carácter são definidas com base na ordem definida sobre as constantes do tipo carácter.

A operação “chr” transforma um inteiro não negativo no carácter que aparece, na ordem definida para os caracteres, na posição correspondente ao seu argumento (o primeiro carácter aparece na posição zero).

A operação “ord” transforma um carácter num inteiro que representa a sua posição na ordem definida para os caracteres. Como já referimos no parágrafo 2.2 a ordem definida sobre as constantes do tipo carácter depende do código utilizado. Para o código ASCII (anexo A) teremos:

ord (‘L’) = 76
chr (114) = r
ord (chr (8)) = 8
chr (ord (‘a’)) = a

Estas operações só são definidas para inteiros entre zero e o máximo número de caracteres representáveis no computador, menos um.

2.6.3 Transformações entre tipo lógico e outros tipos

Existe uma operação que transforma lógico em inteiro produzindo a posição ocupada na ordenação entre valores lógicos, pelo seu argumento; a operação “ord” é definida por:

ord (false) = 0
ord (true) = 1

A operação “odd” (do Inglês ìmpar) transforma um inteiro em lógico, tendo o valor “true” se o argumento é ímpar e o valor “false” em caso contrário.

Existem ainda as operações relacionais, que transformam pares de valores de um mesmo tipo (inteiro, real ou carácter) num valor do tipo lógico; estas operações estão representadas na tabela 2.6.

Operador	Definição	Uso	Significado
=	igual	a = b	**a** igual a **b** ?
<>	diferente	a <>b	**a** diferente de **b** ?
<	menor que	a < b	**a** menor que **b** ?
< =	menor ou igual a	a < = b	**a** menor ou igual a **b** ?
>	maior que	a > b	**a** maior que **b** ?
> =	maior ou igual a	a > = b	**a** maior ou igual a **b** ?

Tabela 2.6: Operações relacionais - transformações entre tipo lógico e outros tipos

Por exemplo, 8 > 5 tem valor "true" e 'd' > 't' tem valor "false" porque o carácter 'd' não aparece depois do carácter ´t´ (considerando o código ASCII)

2.7 Constantes

Designa-se por constante toda a grandeza que não varia durante a execução de um programa.

Em Pascal as constantes são definidas na zona de definição de constantes que é composta pela palavra "const" seguida de uma série de definições de constantes, ou seja a definição de uma constante é igual a:

const <identificador> = <constante>

Como exemplos de definição de constantes podemos ter:

```
const  pi = 3.14159;
       taxa = 0.15;
       ano = 1998;

const  media = 15 ; idade = 19;
```

A utilização de nomes simbólicos para designar constantes representa duas vantagens em relação à referência directa ao seu valor. A

utilização de um nome simbólico que sugira o significado ou a finalidade da constante facilita a leitura e a compreensão do programa. Além disso se for necessário alterar o valor da constante no programa, só é preciso modificar a parte do programa em que a constante está definida.

2.8 Variáveis

Variável é uma entidade cuja grandeza pode variar durante a execução do programa.

Uma variável tem um nome, corresponde a uma posição de memória no computador e pode ter ou não um valor.

Uma variável é caracterizada pelo tipo que especifica a classe de valores que podem ser associados à variável, e a classe de operações que podem ser utilizadas para criar, aceder e modificar a variável.

Uma variável também é caracterizada pelo valor que é guardado na zona da memória correspondente à variável.

2.8.1 Declaração de uma variável

Uma variável tem que ser declarada; a declaração de uma variável é feita uma única vez no início do programa e serve para informar o computador da existência de uma nova variável, do seu tipo e da necessidade de associar ao nome da variável uma zona na memória.

A sintaxe da declaração de uma variável é igual a:

var <identificador> : <tipo>

Na tabela 2.8 apresentamos exemplos de declaração de variáveis para alguns tipos de informação elementar.

Tipo	Instrução
integer	var a,i : integer;
real	var b : real;
boolean	var log : boolean;
char	var letra : char;
string	var nome : string;

Tabela 2.8: Declaração de variáveis

Depois da declaração de uma variável existe uma associação entre o nome da variável e uma zona de memória, mas não existe nenhum valor associado à variável.

A definição de uma variável é a operação através da qual se cria ou modifica o valor da variável (neste caso o valor anterior da variável é perdido).

A referência ao valor da variável é a operação através da qual se tem acesso ao valor da variável, não alterando o seu valor. Consideremos dois exemplos de definição da variável A:

```
A : = x + y;
A : = A + 3;
```

No primeiro exemplo de definição da variável A é feita a referência ao valor das variáveis x e y não alterando os seus valores.

No segundo exemplo de definição da variável A é feita a referência ao seu valor para o somar ao número 3, passando o valor da variável A a ser igual ao seu valor anterior mais três unidades.

2.9 Expressões

Uma expressão é uma fórmula ou regra de computação que produz um valor ou resultado.

Uma expressão é composta por operadores e operandos. Os operandos podem ser constantes, variáveis, expressões ou valores gerados por funções. Os operadores podem ser unários (se apenas têm um operando) ou binários (se têm dois operandos, como por exemplo os operadores “+” ou “/”).

As expressões numéricas têm operadores numéricos e as expressões lógicas têm operadores lógicos e relacionais. Sendo X , Y e Z variáveis do tipo inteiro ou real, são exemplos de expressões numéricas:

```
X + 2 * Y
Sqr (X) / abs (Y)
```

e exemplos de expressões lógicas:

```
X <=Y
(X / Y <0) and (Y <> Z)
```

Para evitar ambiguidade em relação à ordem de aplicação dos operadores numa expressão, as linguagens de programação estabelecem duas regras que especificam a ordem de aplicação dos operadores:

1) Os operadores com maior prioridade são aplicados antes dos operadores com menor prioridade.

2) Se existem dois operadores com a mesma prioridade eles são aplicados da esquerda para a direita.

A utilização de parêntesis permite alterar a ordem de aplicação dos operadores.

Na tabela 2.7 apresentamos a lista de prioridades em Pascal.

Prioridade	**Operador**
máxima	aplicação de funções not, - (simétrico) *, /, div, mod, and +, -, or <, < =, =, < >, > =, >, in

Tabela 2.7: Prioridades em Pascal

Consideremos as duas expressões numéricas:

X : = 2 * 5 div 3 ;

Y : = 2 * (5 div 3) ;

Os operandos de multiplicação (*) e de divisão inteira(/) têm a mesma prioridade. Na primeira expressão eles são aplicados da esquerda para a direita efectuando-se assim em primeiro lugar o produto (2*5 = 10) e só depois a divisão inteira (10 div 3 = 3); o valor da expressão numérica é atribuído à variável X.

Na segunda expressão a divisão inteira (5 div 3 = 1) é feita em primeiro lugar e só depois se efectua o produto (2*1 = 2); o valor da expressão numérica atribuído à variável Y é igual a 2 unidades diferente do valor obtido para a variável X (= 3).

3. ESTRUTURAS DE CONTROLO

Neste capítulo vamos apresentar algumas instruções elementares, como a instrução de atribuição e as instruções de leitura e de escrita; também apresentaremos as estruturas de controlo, mecanismos através dos quais se pode especificar a ordem de execução das instruções de um programa.

Existem vários tipos de estruturas de controlo: a sequenciação, a selecção, a repetição e o salto que deliberadamente não apresentaremos pelo facto de gerar programas difíceis de ler e de modificar.

3.1 Estruturas de controlo I: SEQUENCIAÇÃO

É a estrutura de controlo mais simples e consiste em especificar que as instruções são executadas sequencialmente pela ordem que aparecem no programa.

Consideremos duas instruções inst1 e inst2; a especificação de que a instrução inst2 é executada imediatamente após a execução da instrução inst1 é obtida escrevendo em Pascal

```
inst1 ; inst2 ;
```

O símbolo ";" representa o operador de sequenciação sendo ao mesmo tempo o separador de instruções. Em linguagens em que cada instrução é escrita numa linha o fim da linha representa implicitamente o operador de sequenciação.

Em Pascal é possível especificar que um grupo de instruções deve ser considerado como um todo, sendo as instruções que o constituem executadas sequencialmente. Esta estrutura é chamada *instrução composta*, a sua sintaxe é dada por:

<instrução composta> ::= begin <instruções> end

O par begin/end funciona em Pascal como um par de parêntesis em matemática e como um parêntesis na linguagem algorítmica.

3.2 Instruções elementares

Neste parágrafo apresentaremos três instruções que embora muito simples são fundamentais em linguagens de programação: as instruções de atribuição, de leitura e de escrita de dados. A instrução de atribuição permite atribuir um valor a uma variável; as instruções de leitura e de escrita de dados permitem respectivamente a transferência de informação do exterior para o computador e reciprocamente.

3.2.1 Instruções de atribuição

Utilizando esta instrução podemos atribuir um valor a uma variável. A sintaxe da instrução de atribuição é dada por:

<variável> : = <expressão>

A variável é um identificador correspondente a uma variável e a expressão pode ser uma constante, uma variável, o valor de uma função ou uma combinação destas entidades através de operadores.

Na instrução de atribuição o valor da expressão é calculado e atribuído à variável, sendo o valor anterior da variável perdido.

São exemplos de operações de atribuição:

```
Nota : = 17;
contador : = contador + 1;
x : = y * 2 – z;
B : = chr (70);
```

No primeiro exemplo é atribuído o valor 17 à variável Nota. Se Nota não for uma variável do tipo inteiro ou real a instrução está sintacticamente incorrecta.

No segundo exemplo é atribuído à variável contador (que tem de ser do tipo inteiro ou real) o valor anterior da variável contador mais 1.

No terceiro exemplo o valor da expressão y * 2 – z é calculado e atribuído à variável x. Todas as variáveis x, y, z têm de ser do tipo inteiro ou real.

No último exemplo o carácter que aparece, na ordem definida para caracteres, na posição correspondente ao número 70, é atribuído à variável B que tem de ser do tipo carácter. Para o código ASCII (anexo A) temos B = 'F'.

3.2.2 Instruções de leitura de dados

Para a execução de um programa o computador necessita normalmente de obter valores do exterior; a obtenção de valores do exterior é feita através das instruções de leitura de dados. Esses dados podem ser lidos através do teclado do terminal, podem estar armazenados num ficheiro de dados, podem ser lidos através de sensores, etc.

Para ler dados através do teclado usamos:

read (< lista de variáveis >)

readln (< lista de variáveis >)

↳ esta instrução posiciona o indicador de leitura na linha seguinte, após a leitura

Exemplo de leitura de dois números x e y:

```
read (x, y);
```

3.2.3 Instruções de escrita de resultados

Depois de efectuar a manipulação da informação é importante que o computador possa apresentar os resultados a que chegou. Isto é feito através das instruções de escrita.

Os resultados podem ser visualizados no terminal, ser imprimidos, ser guardados num ficheiro, etc.

Para escrever resultados no monitor usamos:

write (< lista de expressões ou texto >)

writeln (< lista de expressões ou texto >)

↳ esta instrução posiciona o indicador de escrita no início da linha seguinte

Exemplo de escrita de dois números um real e um inteiro:

```
n = -12.14 ; k = 4;

writeln ('Teste de ensaio ');
writeln (n, n:11, n:6, k:4);
writeln ('   Teste de ensaio ');
writeln (n:9:2 , n:3:1, k:5);
```

resultado:

```
Teste de ensaio
-1.2140000000E+01 –1.214E+01– 1.2 E+01   4
  Teste de ensaio
   - 12.14–12.1    4
```

3.3 Estruturas de controlo II: SELECÇÃO

Os programas que podemos desenvolver até agora são muito simples e com capacidades extremamente limitadas: eles podem ler valores, efectuar cálculos e escrever resultados sendo as suas instruções executadas sequencialmente.

Em programas mais complexos surge frequentemente a necessidade de decidir se uma instrução ou conjunto de instruções deve ou não ser executada, de acordo com o valor de uma expressão.

As duas instruções em Pascal que nos permitem seleccionar as instruções que devem ser executadas são: a instrução "if" que permite a escolha entre duas alternativas e a instrução "case" que permite a escolha entre alternativas múltiplas.

3.3.1 A instrução "if"

A instrução "if" permite a selecção entre duas alternativas; dependendo do valor de uma expressão do tipo lógico, esta instrução permite-nos seleccionar uma de duas instruções (que podem ser compostas) para ser executada.

A sintaxe da instrução "if" é definida pelo diagrama:

if <expressão lógica> then <instrução 1> else <instrução 2>

ou

if <expressão lógica> then <instrução 1>

Consideremos por exemplo que desejávamos somar 5 unidades ao menor de dois números x e y. Utilizando a instrução "if" teremos:

```
If x < = y then
   x : = x + 5
else
   y : = y + 5;
```

Na instrução "if" anterior se x for menor ou igual a y o valor de x é alterado passando a ser igual ao valor anterior mais cinco unidades; caso contrário as cinco unidades são adicionadas à variável y.

Consideremos que desejávamos calcular o valor absoluto dos n^{os} V1 que são divisíveis por V2:

```
if (V2 < > 0) and (V1 mod V2 = 0) then  V1 = abs (V1);
```

Como último exemplo, suponhamos que queríamos escrever o máximo de dois números inteiros a e b, realçando o caso de serem iguais; apresentamos o algoritmo correspondente e o respectivo programa em linguagem Pascal:

```
objectivo: escrever o máximo de dois números inteiros a e b,
           realçando o caso de serem iguais;
variáveis dados: a, b ,inteiros, números dados;

    escreva ("escreva dois número a e b ");
    leia (a, b);
    se  a > = b  então
        se  a = b  então
              escreva ("os números são iguais")
        senão
              escreva ("o maior é ", a)
    senão
          escreva ("o maior é ", b);
```

No algoritmo anterior depois da leitura dos números inteiros a e b são comparados os seus valores. Se a for igual a b é escrito que os

números são iguais; se a for maior que b o programa escreve que a é maior que b, e em caso contrário que o maior é o b.

No algoritmo as chavetas definem um conjunto de instruções que devem ser executadas sequencialmente. No programa correspondente as chavetas são substituídas pelo par begin/end.

```
Program maior;
var a, b : integer;

begin
     writeln ( ' escreva dois números a e b ');
     readln (a,b);
     if  a > = b  then
        begin
            if  a = b  then
                writeln ('os números são iguais')
            else
                writeln ('o maior é ', a);
        end
     else
         writeln ('o maior é ', b);
end .   { Program  maior }
```

A utilização de instruções "if" cujas instruções são, por sua vez, instruções "if" é muito comum em programação e costuma chamar-se instruções "if" encadeadas. Em instruções de "if" encadeadas convenciona-se que, salvo indicação em contrário um "else" pertence ao "if" mais próximo, não sendo necessário neste caso a introdução do par begin/end para realçar a que "if" pertence cada "else"; o programa anterior poderia então ser escrito com a forma:

```
Program maior;
var a, b : integer;

begin
     writeln ( ' escreva dois números a e b ');
     readln (a,b);
     if  a > = b then
        if  a = b then
           writeln ('os números são iguais')
        else
            writeln ('o maior é o', a)
     else
         writeln ('o maior é o', b);
end.   { Program maior }
```

3.3.2 A instrução “case”

A escolha entre múltiplas alternativas pode ser feita recorrendo a uma estrutura de “if” encadeados. Porém um encadeamento com muitos níveis pode originar instruções difíceis de compreender. Em alternativa ao encadeamento de instruções “if” podemos utilizar uma instrução de selecção múltipla – a instrução “case” em Pascal. No entanto esta instrução não é apenas uma variante de uma estrutura de “if” encadeados uma vez que a instrução “case” é apropriada em selecção baseada em variáveis que têm valores discretos, e também para pequenas gamas de valores; este aspecto é mostrado no exemplo apresentado.

A sintaxe da estrutura “case” é definida:

<instrução “case”> : : = case < expressão > of < corpo do “case” > end

A expressão, chamada o selector, é uma qualquer expressão cujo valor corresponde a um tipo enumerável (inteiro, lógico ou carácter).

O corpo do “case” é constituído pelos “elementos do case” cuja sintaxe é dada por:

<elemento do “case”> : : = <rótulos do “case”> : <instrução>

O rótulo do “case” é uma constante do mesmo tipo que a expressão. Ao encontrar a instrução “case” o computador calcula o valor da expressão e executa a instrução cujos rótulos do “case” contêm o valor da expressão; consequentemente numa instrução “case” não podem existir dois valores iguais de rótulo do case; se o valor da expressão não for igual a nenhuma das constantes em rótulos do “case” teremos ou não um erro de sintaxe conforme a versão do Pascal utilizada.

Consideremos que desejávamos converter uma nota quantitativa em nota qualitativa supondo que a correspondência seria:
18 – 20 = A, 14 – 17 = B , 10 – 13 = C , 5 – 9 = D e 0 – 4 = E
Utilizando a instrução “case” teríamos:

```
Program Conver;
var nota: integer;

begin
    write (‘ escreva a nota’);
     read (nota);
     case nota of
          18, 19, 20       : writeln (‘A’);
          14, 15, 16, 17 : writeln (‘B’);
          10, 11, 12, 13 : writeln (‘C’);
          5, 6, 7, 8, 9     : writeln (‘D’);
          0, 1, 2, 3, 4     : writeln (‘E’);
    end   { case }
end.   { Program conver }
```

Efectuada a leitura de uma nota, o computador executa a instrução cujos rótulos contenham o valor igual à nota; por exemplo para uma nota igual a 15 valores o computador escreve a letra B.

3.4 Estruturas de controlo III: REPETIÇÃO

Em programação, uma sequência de instruções executada repetitivamente é chamada ciclo.

Um ciclo é constituído por uma sequência de instruções, o corpo do ciclo, e por uma estrutura que controla a execução dessas instruções, especificando quantas vezes o corpo do ciclo deve ser executado.
Apresentamos três instruções que permitem a especificação de ciclos em Pascal, as instruções “while” , “repeat” e “for”.

3.4.1 A instrução while

A instrução “while” especifica a execução repetida de uma instrução enquanto uma determinada expressão do tipo lógico tiver um valor verdadeiro. A sintaxe da instrução “while” é definida por:

<instrução “while” > : : = while <expressão> do <instrução>;

A expressão é uma expressão lógica e a instrução pode ser uma instrução composta.

Suponhamos que desejávamos efectuar a soma de uma sequência de números pares cuja quantidade é desconhecida, assinalando o fim da sequência com a introdução de um número impar. No corpo do algoritmo

apresentado, em primeiro lugar, a variável soma é inicializada a zero e lê-se o primeiro número; depois inicia-se um ciclo correspondente a uma instrução “enquanto”, em que, enquanto o número lido for par são efectuadas as instruções de actualização da variável soma e de leitura de um novo número. Quando se lê um número impar as instruções do corpo do ciclo já não são mais executadas e a variável soma é escrita; também apresentamos o programa correspondente.

Objectivo: calculo da média de um conjunto de números pares;
variáveis dados: x , inteiro, número lido;
variáveis auxiliares: soma, inteiro, soma dos números lidos;

```
escreva (“escreva um número par” ) ;
escreva ( “ para terminar escreva um  número impar” );
soma ← 0 ;
leia (x);
enquanto numero mod 2 = 0 faça
    { soma ← soma + x;
      escreva (“escreva um número par” ) ;
      escreva ( “ para terminar escreva um  número impar” );
      leia (x);
escreva (“a soma é : “soma);
```

```
PROGRAM soma1;
{ soma de uma sequência de números pares }

var x, soma  : integer;

begin
    soma : = 0;
    writeln ( ‘ escreva um número par ‘);
    writeln ( ‘ para terminar escreva um número impar ‘);
    readln (x);
    while x mod 2 = 0 do
        begin
            soma : =  soma + x ;
            writeln ( ‘ escreva um número par ‘);
            writeln ( ‘ para terminar escreva um número impar ‘);
            readln (x) ;
        end;
     writeln (‘ A soma é ‘,soma);
end.     { Program soma1 }
```

3.4.2 A instrução "repeat"

A instrução "repeat" permite especificar a execução repetitiva de uma sequência de instruções até que certa expressão do tipo lógico tenha o valor verdadeiro.

A sintaxe da instrução "repeat" é definida por:

<instrução "repeat" > : : = repeat <Instruções> until <expressão>

A expressão é uma expressão lógica. Consideremos que queríamos somar uma sequência de números inteiros positivos até que o valor da soma seja maior que 1000 e ao mesmo tempo contar a quantidade de números lidos; apresentamos o algoritmo correspondente e o respectivo programa:

Objectivo: soma de uma sequência de números positivos até a soma ser maior que 1000;
variáveis dados: x, inteiros, números lidos;
variáveis resultado: soma, inteiro , soma dos números lidos;
n, inteiro , quantidade de números lidos;

```
soma ← 0;  n ← 0;
repita
    escreva ( " escreva um número positivo ");
    leia (x);  n ← n + 1;
    soma ← soma + x ;
até soma > 1000 ;
escreva("A soma de ",  n ," números é igual a ", soma);
```

```
PROGRAM soma2;
{ soma de uma sequência de números positivos até a soma ser maior que 1000}
var soma, x ,n: integer;

begin
    soma : = 0;   n : = 0;
    repeat
        writeln ( ' escreva um número positivo ');
        readln (x);   n : = n + 1;
        soma : = soma + x ;
    until  soma > 1000 ;
    writeln (' A soma de ', n,' números é igual a ', soma);
end.       { Program soma2 }
```

No algoritmo apresentado, após a inicialização a zero das variáveis soma e n, inicia-se um ciclo, correspondente à instrução “repita”, onde os números são sucessivamente lidos e somados e a variável n é actualizada; quando a variável soma for maior que 1000; então o ciclo termina e as variáveis n e soma são escritas.

Ao contrário da instrução “while”, o ciclo da instrução “repeat” é sempre executado pelo menos uma vez, porque a expressão que controla a execução do ciclo só é executada no fim do ciclo. No exemplo apresentado se o primeiro número lido for maior que 1000 o ciclo só é executado uma vez.

3.4.3 A instrução “for”

A instrução “for” ao contrário das instruções “while” e “repeat” dá origem a um ciclo contado, cuja execução é controlada por uma variável de controlo, que tem de ser de um tipo enumerável[1]. O ciclo é executado para uma sequência finita de valores desta variável. Antes da primeira passagem pelo ciclo esta variável é inicializada com o primeiro valor de sequência; depois de cada passagem pelo ciclo, a variável de controlo é actualizada com o valor seguinte da sequência. O ciclo termina depois de uma passagem com a variável de controlo igual ao último valor da sequência. Num ciclo de controlo sabemos sempre quantas vezes ele vai ser executado o que contrasta com os ciclos “while” e “repeat”.

A sintaxe da instrução “for” é igual a:

For <index> : = <exp 1> { to / down } <exp 2> do <instrução>;

index – variável inteira (ou carácter ou lógica)

exp1, exp2 – expressões do tipo inteiro (ou carácter ou lógico)

A instrução pode ser uma instrução composta. Os valores das expressões exp1 e exp2 representam respectivamente o valor inicial e final da sequência. Se o valor de exp1 for menor que o valor de exp2, a variável de controlo é acrescida de uma unidade em cada passagem

[1] Tipo enumerável é caracterizado pelo facto de ser possível enumerar todos os seus valores (por exemplo tipo lógico e tipo inteiro); um exemplo de tipo não enumerável é o tipo real.

pelo ciclo; caso contrário é diminuída de uma unidade em cada passagem pelo ciclo. Suponhamos que queríamos calcular a média das alturas dos alunos de uma turma construindo o algoritmo e o programa apresentados:

```
Objectivo: média das alturas de n alunos;
 variáveis dados: n, inteiro, número de alunos;
                  alt, real ,altura de cada aluno;
variáveis  resultado: media, real , média das alturas dos alunos;
variáveis  auxiliares: i,  inteiro  , contador;

    escreva ("escreva o número de alunos ");
    leia (n) ;  media ← 0.0;
    para i ← 1 até n faça
       escreva ("escreva a altura do aluno número", i );
       leia (alt);
       media ← media + alt;
    media ← media / n;
    escreva ("A média das alturas é igual a", media);
```

```
PROGRAM medias;   { média das alturas de n alunos}
var  i, n : integer;
      alt, media : real;

begin
    writeln ( ' escreva o número de alunos ');
    read (n) ;   media : = 0.0;
    for i : = 1 to n do
        begin
            writeln ( ' escreva a altura do aluno número ', i );
            read (alt);
            media : = media + alt;
        end;
    media : = media / n;
    writeln ('A média das alturas dos alunos é igual a', media)
end.   { Program medias }
```

No início do algoritmo, depois da declaração das variáveis, é lido o número de alunos da turma e a variável media é inicializada a zero. A instrução "para" , em cada passagem pelo ciclo, actualiza o valor do contador somando uma unidade ao seu valor anterior e efectua as instruções de leitura da altura de um aluno e de actualização da variável media; a instrução "para" permite ainda testar se o valor do contador já atingiu o último valor da sequência n, e neste caso, depois

de efectuar as instruções do ciclo mais uma vez , este termina. Uma vez terminado o ciclo é calculada a média das alturas dos alunos da turma. Suponhamos que queríamos construir um programa para somar os números pares menores ou iguais a n; o programa poderia ser:

```
PROGRAM somar ;
{ soma dos números pares menores ou iguais a n }
var  i, n, soma : integer;

begin
     writeln (‘ escreva o número n ‘);
     read (n) ;
     soma : = 0 ;
     for i : = 1 to  n div 2  do
          soma : = soma + 2 * i ;
     writeln (‘A soma dos números pares menores ou iguais a n é igual a ’, soma)
end.  { Program somar }
```

Ou então poderíamos escrever o programa:

```
PROGRAM somar ;
{ soma dos números pares menores ou iguais a n }
var  i, n, soma : integer;

begin
     writeln ( ‘ escreva o número n ‘);
     read (n) ;
     soma : = 0 ;
     for i : = 2 to  n  step 2 do
          soma : = soma + i ;
     writeln (‘A soma dos números pares menores ou iguais a n  é igual a ’, soma)
end.   {Program somar}
```

Comparando os dois programas anteriores vemos que apenas diferem na instrução de ciclo “for”.

No primeiro programa o valor inicial da variável de controlo é igual a 1 e o valor final é igual a metade de n. A variável soma é actualizada adicionando-se o dobro da variável de controlo, ou seja adicionando os números pares menores ou iguais a n. No segundo programa o valor inicial da variável de controlo é igual a 2 e o valor final igual a n, sendo esta variável actualizada pela soma de duas unidades; uma vez que a variável soma é actualizada pela adição da variável de controlo, estamos, como no caso anterior, a somar números pares menores ou iguais a n, sendo o número de operações efectuadas pelos dois programas igual. A utilização desta instrução “for” apenas é permitida em algumas versões de Pascal.

Ao terminar a execução de uma instrução “for” a variável de controlo é indefinida, ou seja, não deve ser utilizada sem que um novo valor lhe seja atribuído.

3.4.4 A escolha das instruções de repetição

A escolha da instrução de repetição deve ser feita tendo vários aspectos em atenção:

- Se as instruções que constituem o corpo do ciclo tiverem que ser repetidas enquanto uma dada condição se verifica devemos utilizar a instrução “while”

- Se as instruções que constituem o corpo do ciclo tiverem que ser repetidas até que uma dada condição se verifique devemos utilizar a instrução “repeat”

- Se sabemos de antemão quantas vezes é que o corpo do ciclo vai ser repetido a instrução “for” deve ser utilizada.

Não devemos esquecer que na maioria dos casos a instrução “for” é mais eficiente que as outras instruções de ciclo e por isso deve ser escolhida sempre que possível, isto é sempre que sabemos quantas vezes vai ser repetido o corpo do ciclo.

Pode acontecer não sabermos se o corpo do ciclo vai ser executado alguma vez; utilizando a instrução “while” que especifica que o valor da expressão que controla a execução do ciclo é calculada antes do início da sua execução, se o valor inicial desta expressão for falso o corpo do ciclo nunca será executado. Esta é uma das diferenças principais da instrução “while” relativamente à instrução “repeat”.

4. TIPOS ESTRUTURADOS

Um tipo estruturado de informação permite agregar um conjunto de valores, sob um único identificador ou seja com o mesmo nome.

Neste capítulo apresentaremos dois tipos estruturados de informação, as tabelas e as fichas.

Se a informação associada a uma variável é de um só tipo o tipo de estrutura a utilizar deve ser uma tabela. Pelo contrário as fichas permitem a agregação de informação de tipos diferentes.

4.1 Tipos estruturados I: TABELAS

Uma variável que esteja associada a um conjunto de valores é um tipo estruturado de informação; uma das estruturas de informação mais utilizadas em programação é a tabela.

Uma tabela é um agregado de elementos todos do mesmo tipo e com o mesmo nome. Os elementos das tabelas são acedidos fornecendo a posição que o elemento ocupa dentro da tabela; as entidades utilizadas para especificar a posição de um elemento de uma tabela são os índices.

As dimensões de uma tabela representam, por definição o número de valores (índices) que temos de especificar para determinar a posição de um elemento dentro da tabela.

- 1	5	10	0	7

Figura 4.1 – Tabela unidimensional

1	4	5	- 1
0	2	10	1
- 3	5	7	2

Figura 4.2 – Tabela bidimensional

Para referenciar um elemento da tabela da figura 4.1 temos apenas de especificar um valor que indica a posição deste elemento na linha, trata-se assim de uma tabela unidimensional. Na figura 4.2 representa-se uma tabela bidimensional, uma vez que, para referenciar um elemento temos de especificar dois valores, a linha e a coluna em que o elemento se encontra.

As tabelas em informática correspondem às noções matemáticas de vector e matriz.

4.1.1 Definição de tabela em Pascal

A definição de uma tabela exige dois passos sequenciais:

1. Definição de um tipo particular de tabelas especificando as suas dimensões, quais os limites de cada um dos seus índices e quais os tipos dos elementos da tabela.

2. Declaração de uma variável como sendo desse tipo.

A definição sintáctica de um tipo tabela é dada por:

<tipo tabela> ::= array [<tipos de índice>] of <tipo do componente>

Como exemplo apresentamos a definição de algumas tabelas, um vector com 21 componentes do tipo inteiro (variável indexada), uma matriz quadrada com 10 linhas com os coeficientes do tipo real (variável bi-indexada) e um vector com 30 componentes do tipo carácter:

```
type vector = array [0..20] of integer;
type matriz = array [1..10, 1..10] of real;
type AA = array [1..30] of char;

var A, B : vector;
      C : matriz;
      nome : AA ;
```

Os nomes das variáveis do tipo tabela obedecem às mesmas regras das variáveis de tipos de informação elementar. No mesmo programa não se pode usar uma variável do tipo tabela e uma variável de outro tipo com o mesmo nome. Consideremos como exemplo o algoritmo e o programa em Pascal que transpõe uma matriz de coeficientes inteiros com 4 linhas e 5 colunas:

Objectivo: transpor uma matriz (4 x 5);

Variáveis dados: A bi-indexada de inteiros, matriz dada;
Variáveis resultado: AT , bi-indexada de inteiros, matriz obtida;
Variáveis auxiliares: i, j , inteiros ,contadores;

```
escreva ("coeficientes da matriz");
para i ← 1 até 4 faça
    para j ← 1 até 5 faça
        escreva ("coeficiente  A[", i,j,"]  da matriz ");
        leia (A [i, j]);

para i ← 1 até 4 faça
    para j ← 1 até 5 faça
        AT [j, i] ← A [i, j];

escreva ("a matriz transposta")
para i ← 1 até 5 faça
    para j ← 1 até 4 faça
        escreva (AT [ i, j]);
```

No algoritmo anterior após a leitura da matriz A de 4 linhas e 5 colunas é criada a matriz AT transposta da matriz A. Finalmente é escrita a nova matriz AT. O programa correspondente é apresntado na página seguinte.

```
PROGRAM  mat;
{transpor uma matriz (4 x 5) }
type matriz = array [1..5, 1..5] of integer;
var A , AT : matriz;
      i, j : integer;

begin
     writeln (“coeficientes de matriz”);
     for i : = 1 to 4 do
          begin
               readln;
               for j : = 1 to 5 do
                    begin
                         writeln (‘escreva o coeficiente  A[‘, i,j,’]  da matriz’);
                         read (A [i, j]);
                    end;
          end;

     for i : = 1 to 4 do
          for j : = 1 to 5 do
               AT [j, i] = A [i, j] ;

     writeln (‘a matriz transposta’)
     for i : = 1 to 5 do
          begin
               writeln;
               for j : = 1 to 4 do
                    write (AT [ i, j]);
          end;
end.   { program mat }
```

Suponhamos que queríamos escrever um programa para calcular a média das classificações, na cadeira de Álgebra Linear e Geometria Analítica (ALGA), dos alunos do curso de Mecânica e do curso de Gestão Industrial, e também determinar se o aluno com classificação mais elevada é de Mecânica ou de Gestão. Para o efeito construíamos o algoritmo:

Objectivo: Determinação do melhor aluno e da média dos alunos de Mecânica e Gestão;

Var. dados: n, inteiro, número total de alunos,
vc, indexada de texto, vector que indica se o aluno é de Mecânica ou Gestão;
vn, indexada de inteiros, notas dos alunos;

Var. aux: i, inteiro, contador
cm, cg, inteiro, contador dos alunos de Mecânica e

Gestão;
sm, sg, inteiro, soma das notas dos alunos de Mecânica e Gestão;
max, inteiro, nota mais alta dos alunos de Mecânica ou Gestão;
imax, inteiro, indicador da coordenada do vector correspondente à nota mais alta;

Var. resultados: mediam, mediag, real, média das notas dos alunos de Mecânica e Gestão;

leia (n);

para i ← 1 até n faça { leia (vc [i]);

para i ← 1 até m faça { leia (vn [i]);

cm ← 0 ; cg ← 0 ; sm ← 0 ; sg ← 0 ; max ← 0;

para i ← 1 até n faça

se vc [i] = 'M' então { cm ← cm + 1;
sm ← sm + vn [i]

senão { cg ← cg + 1;
sg ← sg + vn [i];

se vn [i] > max então { max ← vn [i];
imax ← i;

mediam ← sm / cm;
mediag ← sg / cg;

escreva ("a média (a ALGA) dos alunos de Mecânica é " , mediam);
escreva (" a média (a ALGA) dos alunos de Gestão é " , mediag);

se vc [imax] = 'M' então escreva ("O melhor aluno (a ALGA) é de Mecânica ")

senão escreva ("O melhor aluno (a ALGA) é de Gestão ");

No algoritmo anterior pede-se ao utilizador para introduzir o número de alunos que se inscrevem nos dois cursos de Mecânica e Gestão. Em seguida iniciam-se dois ciclos (instrução “para”), para efectuar a leitura dos vectores vc, vector indicador do curso de cada aluno, e vn, vector que contem a nota de ALGA de cada aluno.
Inicializam-se a zero as variáveis cm, cg, sm, sg e max cujo significado está descrito no cabeçalho do algoritmo. Em seguida começa um ciclo repetitivo:

- Para cada aluno, se ele for do curso de Mecânica, actualizam-se as variáveis cm e sm, contador do número de alunos do curso de Mecânica e somatório das respectivas notas; caso contrário são actualizadas as variáveis cg e sg relativas ao curso de Gestão. Ao mesmo tempo determina-se a nota mais elevada e o curso do respectivo aluno.

No fim do algoritmo é escrita a média das notas de ALGA dos alunos de Mecânica mediam, e dos alunos do curso de Gestão mediag; também se indica o curso do aluno que obteve a melhor nota na referida cadeira.

Uma vez construído o algoritmo podemos escrever o programa correspondente:

```
Program Melhor;
Type vector = array [1..200] of integer ;
Type curso = array [1..200] of char ;
var vc : curso ;
      vn : vector ;
      n , i , cm , cg , sm , sg , max , imax : integer ;
      mediam , mediag : real;

begin
     writeln (‘escreva o número de alunos ’);
     readln ( n);
     for i : = 1 to n do
         begin
             writeln (‘escreva M ou G conforme o aluno ‘,i,’ for de Mecânica ou Gestão’);
              readln ( vc (i) ) ;
          end;
     for i : = 1 to n do
         begin
              writeln (‘escreva a nota do aluno ‘,i );
              readln ( vn (i) ) ;
         end;

     cm : = 0 ;  cg : = 0 ;  sm : = 0 ;  sg : = 0 ;  max : = 0 ;
     for i : = 1 to n do
         begin
              if  vc [i] = ‘M’ then
                 begin
                       cm : = cm + 1 ;
                       sm : = sm + vn [i] ;
                 end
              else
                 begin
                       cg : = cg + 1 ;
                       sg : = sg + vn [i] ;
                 end;
              if  vn [i] > max then
                 begin
                       max : = vn [i] ;
                       imax : = i ;
                  end ;
          end ;

     mediam : = sm / cm;
     mediag : = sg/ cg;

     writeln (‘ a média (a ALGA) dos alunos de Mecânica é ‘ , mediam) ;
     writeln (‘ a média (a ALGA) dos alunos de Gestão é ‘ , mediag) ;
     if  vc [imax] = ‘M’ then
        writeln (‘ O melhor aluno (a ALGA) é de Mecânica’)
     else
        writeln (‘ O melhor aluno (a ALGA) é de Gestão’)
end.          { Program Melhor }
```

Suponhamos que queremos um programa para escrever um texto com m linhas de n caracteres cada uma, em letras maiúsculas, ou seja o programa terá que passar a maiúsculas todas as letras minúsculas do texto. Para o código ASCII os inteiros que ocupam as posições correspondentes aos caracteres que representam as letras minúsculas estão compreendidos entre 97 e 122; além disso o inteiro que ocupa a posição correspondente a uma letra maiúscula é 32 unidades menor que o que ocupa a posição correspondente a uma letra minúscula (anexo A). Um exemplo do programa desejado é apresentado:

```
Program converte (input, output);

var linha : string ;
    li, m , n , x, i : integer ;

begin
    writeln ('escreva o nº de linhas do texto e o nº de caracteres por linha');
    readln ( m,n);

    for li : = 1 to m do

        begin
            for i : =1 to n do
                begin
                    writeln (' escreva o carácter ',i,' da linha ', li,' do texto') ;
                    read (linha [i]) ;
                end;

            for i : =1 to n do
                begin
                    x : = ord (linha [i]) ;
                    if ( x > =  97 ) and ( x < = 122)  then
                    linha [i] : = chr (x - 32);
                end;

            writeln;
            for i : = 1 to n do
            write (linha [i]) ;
        end

end.   { Program converte }
```

O bloco de instruções do programa inicia-se com a leitura do número de linhas m e de caracteres de cada linha n. Para cada linha o programa utiliza uma instrução "for" sendo o corpo deste ciclo constituído por uma instrução composta:

- uma instrução "for" para efectuar a leitura dos n caracteres de cada linha.

- uma instrução “for” onde o programa identifica os caracteres correspondentes a letras minúsculas utilizando a função intrínseca ord, que transforma um carácter no inteiro que corresponde à sua posição para a ordem definida para caracteres; se esse número x estiver compreendido entre 97 e 122 então o carácter correspondente é uma letra minúscula que é transformada na maiúscula correspondente através da função intrinseca chr(x-32).

- Uma instrução “for” que escreve a linha agora já com todas as letras maiúsculas.

4.2 Tabelas empacotadas

Por uma questão de eficiência de armazenamento de tabelas de caracteres em memória, é habitual representar as cadeias de caracteres como tabelas empacotadas “packed arrays”. Uma tabela empacotada armazena os seus constituintes em memória do modo mais compacto possível mas em contrapartida precisa de um maior tempo de processamento.

Propriedades particulares:

1. A uma variável do tipo cadeia pode ser atribuída, com uma única operação de atribuição, uma constante correspondente a uma cadeia de caracteres. Porém a constante a ser atribuída à variável tem que ter o mesmo número de caracteres que os definidos para os seus elementos(27 caracteres no exemplo seguinte):

```
Program letras;
Type nomes : = packed array [1 .. 27] of char;
var cadeira,regente: nomes;

    ...

    cadeira : = ‘Programação de computadores’

    ...

    regente : = ‘  Luísa Abreu Costa Sousa  ’
    ...

end.    { program letras }
```

2. As variáveis do tipo cadeia podem ser escritas como um todo utilizando as operações “write” e “writeln”. Por exemplo para as instruções:

```
writeln (‘A cadeira chama-se ’, cadeira, ‘.’);
writeln;
writeln (‘A regente da cadeira chama-se’, regente, ‘.’);
```

o programa escreve:

A cadeira chama-se Programação de Computadores.

A regente da cadeira chama-se Luísa Abreu Costa Sousa .

4.3 Tipos estruturados II – FICHAS

Uma ficha – “Record” – é um agregado de elementos que podem ser de tipos diferentes.

Cada elemento de uma ficha está associado a um nome, chamado o nome do campo do elemento, e o acesso é feito referindo o nome do campo. Além do nome cada elemento tem um tipo que identifica a classe de elementos que armazena.

Tal como as tabelas, as fichas são tipos estruturados de informação, diferindo das tabelas uma vez que permitem agrupar informação de tipos diferentes e os seus constituintes não são acedidos com base na posição que ocupam dentro da ficha, mas sim pela referência ao nome do campo.

Para definir um tipo como ficha temos de especificar o nome de cada um dos seus campos (elementos), e o respectivo tipo. A definição completa do tipo ficha é feita sintacticamente por:

<tipo ficha> : : = record <fichas> end

<fichas> : : = <identificador> : tipo

Consideremos o exemplo relativo ao processamento de informação sobre os alunos que frequentam a Faculdade de Engenharia.

```
Program Fichas ;
Type Palavra = packed array [1 .. 25] of char;
       Frase = packed array [1..40] of char;
Infoaluno = record
                      Numero : integer;
                       Nome : Palavra;
                       Morada : Frase;
                       Telefone : integer;
                       Media : real;
                       Ramo : Palavra;
                end ;    {Infoaluno}
var aluno : Infoaluno;

begin

    ...

    aluno . Numero : = 125;
    aluno . Nome : = 'José Afonso Pinto Marques';
    aluno . Morada : = 'R. de Cedofeita, 5, 4000 Porto';
    aluno . Telefone : = 200415;
    aluno . Media : = 16.5;
    aluno . Ramo : = 'Engenharia Mecânica';
    ...

end.  { program Fichas }
```

Utilizando a variável aluno como sendo do tipo Infoaluno, a informação do aluno nº 125 está atribuída à ficha representada pelo nome aluno. A referência a um dos elementos da ficha, por exemplo, aluno.Telefone, corresponde a uma variável do tipo declarado para o campo respectivo, neste caso um inteiro.

Para o processamento da informação sobre os alunos da Faculdade de Engenharia não é suficiente uma só ficha , tornando-se necessário uma colecção de fichas. Uma tabela de fichas é a estrutura adequada para representar colecções de fichas. Podemos então definir um tipo ConjuAlunos, como uma tabela, cujos elementos sejam fichas do tipo Infoaluno, e depois declarar a variável Alunos como sendo do tipo ConjuAlunos. Cada elemento da tabela Alunos é uma ficha que guarda a informação de cada aluno. O programa apresentado anteriormente pode então ser completado:

```
Program CFichas;
Type Palavra = packed array [1 .. 25] of char;
       Frase = packed array [1..40] of char;
Infoaluno = record
                      Numero : integer;
                       Nome : Palavra;
                       Morada : Frase;
                       Telefone : integer;
                       Media : real;
                       Ramo : Palavra;
                end ;    {Infoaluno}
ConjuAlunos = array [1..500] of Infoaluno;
Var Alunos : ConjuAlunos;
      Mora : frase;
...

begin
     ...
    for i : = 1 to n do
        ...

       Alunos [i] . Numero : = 125;
       Alunos [i] . Nome : = 'José Afonso Pinto Marques';
       for i : = 1 to 40 do
            read (Mora [i]);
       Alunos [i] . Morada : = Mora;
       Alunos [i] . Telefone : = 200415;
       Alunos [i] . Media : = 16.5;
       Alunos [i] . Ramo : = 'Engenharia Mecânica';

    ...

end.   { program CFichas }
```

Alunos [10]. Numero representa o número do aluno armazenado na décima posição da tabela Alunos e é uma variável do tipo inteiro. Alunos [10]. Nome representa o nome do aluno cuja informação está armazenada na décima posição da tabela Alunos e é uma variável do tipo palavra ou seja uma tabela de caracteres. Se a informação do aluno José Afonso Pinto Marques for armazenada na décima posição da tabela Alunos (i =10) então podemos obter informação deste aluno utilizando a componente 10 da tabela Alunos.

5. SUBPROGRAMAS E UNITS

Os subprogramas permitem a divisão de um programa em vários módulos, cada um dos quais pode ser desenvolvido independentemente dos outros.

Um subprograma pode executar as mesmas acções que um programa, receber valores, manipulá-los e produzir novos valores. Cada vez que um programa utiliza um subprograma diz-se que este foi chamado: o subprograma recebe informação do programa e fornece-lhe a informação produzida. Quando isto acontece as instruções do programa deixam temporariamente de ser executadas e o computador executa as instruções do subprograma até estas acabarem, continuando depois a execução do programa.

As linguagens de alto nível geralmente fornecem dois processos distintos de agrupar instruções, os procedimentos e as funções.

5.1 Procedimentos

A criação de um procedimento consiste em associar um nome a um conjunto de instruções correspondentes ao procedimento. Em Pascal um procedimento é criado através de uma declaração de procedimento que tem duas partes: um cabeçalho e um bloco.

No cabeçalho damos um nome ao procedimento e definimos como ele comunica com o programa (o que é feito através da especificação dos parâmetros formais); no bloco definimos as entidades usadas pelo procedimento e a sequência de acções a ser executada pelo procedimento. Suponhamos que desejávamos escrever um procedimento para somar e subtrair dois números:

```
procedure soma (x, y : real; var z, w : real);
{soma e subtracção de dois números x e y}

 begin

      z : = x + y ;   w : = x – y ;
 end ;   {soma}
```

Depois de definido um procedimento, as instruções que constituem o corpo desse procedimento são executadas quando o seu nome seguido pelo número apropriado de argumentos (parâmetros concretos) é encontrado num programa, ou seja quando o procedimento é chamado.

Utilizando uma interpretação simplificada da instrução de chamada de um procedimento podemos dizer que ao encontrar esta instrução o computador a substitui pelas instruções que constituem o corpo do procedimento, substituindo os parâmetros formais pelos parâmetros concretos. Consideremos que queríamos escrever um programa para somar e subtrair dois números se pelo menos um deles for negativo; apresentamos o algoritmo correspondente e o respectivo programa que utiliza o procedimento soma:

Objectivo: soma e subtracção de dois números x e y se pelo menos um for negativo;
variáveis dados: a, b , reais , números dados;
variáveis resultados: c, d ,reais , soma e produto;

```
escreva ("escreva dois números a e b");
leia (a, b);
se  (a < 0) ou  (b < 0) então
    { c ← a + b;
      d ← a – b;
      escreva ("a soma de a com b é " , c, " e a diferença é " , d )
senão
      escreva (" os nos são ambos positivos ");
```

No programa da página seguinte, correspondente ao algoritmo anterior, se um dos dois números lidos for negativo o procedimento soma é chamado para calcular a soma e a subtracção dos dois números; Os parâmetros formais x e y recebem os valores dos parâmetros concretos a e b e as operações são efectuadas, sendo os resultados transmitidos ao programa pelos parâmetros formais z e w.

```
Program seleccao;
{este programa soma e subtrai dois números dados se algum deles for negativo}
 var a, b, c, d : real;

 procedure soma (x, y : real ; var z, w : real);
{soma e subtracção de dois números x e y}

 begin

      z : = x + y ;   w : = x – y ;
 end ;   {soma}

 begin
       writeln ('escreva dois números a e b ');
       readln (a, b);
       if  (a < 0) or  (b < 0) then
           begin
               soma (a, b, c, d);
               writeln (' a soma de a com b é ' , c, ' e a diferença é ' , d );
           end
       else
           writeln (' os nos são ambos positivos ')
 end. { program selecção }
```

Neste programa os parâmetros formais z e w são antecedidos pela palavra "var". Como primeira aproximação podemos considerar que eles representam os valores que o procedimento vai modificar no programa que o chamou; as variáveis correspondentes aos parâmetros concretos c e d, vão receber os valores que os parâmetros formais z e w tinham dentro do procedimento quando termina a execução do corpo do procedimento.

Os parâmetros formais x e y não são antecedidos pela palavra "var", e consequentemente não podem modificar os parâmetros concretos correspondentes a e b, sendo a única ligação entre eles uma associação unidireccional de valores.

5.2 Funções

As linguagens de alto nível fornecem aos seus programadores um conjunto de funções predefinidas (funções intrínsecas) que podem ser utilizadas em qualquer programa. No entanto estas linguagens permitem que o programador crie as suas próprias funções e as utilize no seu programa. A utilização de funções definidas pelo programador compreende duas fases distintas: a definição da função que é feita fornecendo um processo de cálculo para valores da função e a

referência ao valor da função para um valor, ou valores dos seus argumentos.

Em Pascal uma função é definida através de uma declaração de função que consiste num cabeçalho seguido por um bloco; o cabeçalho da função define o nome da função, o número e o tipo dos seus argumentos (domínio da função) e o tipo dos seus valores (ou seja o contradomínio da função). No bloco definem-se as constantes, declaram-se as variáveis utilizadas pela função e apresentam-se as instruções que calculam o valor da função. O bloco tem necessariamente de conter pelo menos uma instrução de atribuição da forma

<nome> : = <expressão>

em que o símbolo nome é o identificador que aparece no cabeçalho da função imediatamente a seguir à palavra function ou seja é o nome da função, e a expressão é do tipo definido para a função.

Consideramos a função que calcula o factorial de um número n:

```
function factorial ( n : integer) : integer ;

{esta função calcula o factorial de n}
var f, i : integer;

begin
     f : = 1;
     if n > 1 then
        for i : = 1 to n do
              f : = f * i;
     factorial : = f ;
end ;  {factorial}
```

Na função factorial a variável f é do tipo definido para a função, uma variável inteira.
Num programa em que uma função foi definida, o seu valor é calculado sempre que numa expressão aparece o nome de função seguido do número apropriado de argumentos (os parâmetros concretos). Deste modo a função é chamada e tal como no caso dos procedimentos, os parâmetros concretos são associados aos parâmetros formais, e as instruções do corpo da função são executadas; o valor atribuído ao identificador que representa o nome da função é considerado o valor da função e é retornado para o programa que continua. Consideremos um

algoritmo para calcular o factorial de n números e o respectivo programa que utiliza a função factorial:

Objectivo: calcular o factorial de n números;

Variáveis dados: n, inteiro, quantidade de números;
numero, inteiro , números dados;
Variáveis resultados: xfact , inteiro, factorial;
Variáveis auxiliares: i, j ,inteiros, contadores;

```
escreva (“  escreva n”);
leia ( n) ;
    para i ← 1 até n faça
        escreva (“escreva um número”);
        leia( numero) ;
        se numero < 0 então
              escreva(“erro - o número é negativo “)
         senão
            xfact ← 1;
            if numero > 1 then
                para i ← 1 até numero faça
                    xfact ← xfact * i;
            escreva ( “O factorial do número “, numero,” é igual a”,
                        xfact );
```

No algoritmo anterior calcula-se o factorial de cada número lido que não seja negativo.

No programa correspondente ao algoritmo anterior, que se apresenta na página seguinte, em cada passagem pelo ciclo se o número lido não for negativo é chamada a função factorial para calcular o factorial do número lido; o valor da função é atribuído à variável xfact que é do tipo definido para a função ou seja do tipo inteiro. Como vemos a chamada de uma função dentro de um programa (ou subprograma) é feita utilizando uma instrução de atribuição, e não da mesma forma como se efectua a chamada de um procedimento.

```
Program Fact;
{Este programa calcula o factorial de n números}
var n, i, numero : integer ;
    xfact : integer ;

function factorial ( k : integer) : integer ;
{esta função calcula o factorial de k}
var f, i : integer;

begin
    f : = 1;
    if k > 1 then
        for i : = 1 to k do
            f : = f * i;
    factorial : = f ;
end ;  {factorial}

begin
    writeln (‘  escreva n ’);
    readln ( n) ;
    for i : = 1 to n do
        begin
            writeln (‘  escreva um número para calcular o factorial’);
            readln( numero) ;
            if numero < 0 then
                writeln (‘ erro - o número é negativo ‘)
            else
                begin
                    xfact : = factorial ( numero) ;
                    writeln( ‘ O factorial do número ‘ , numero, ‘ é igual a ‘ ,xfact ) ;
                end;
        end ;
end.   {program Fact }
```

5.3 Métodos de passagem dos parâmetros

Existem vários métodos de passagem de parâmetros, quando um programa (ou subprograma) chama um subprograma, que as linguagens de programação podem utilizar. Em Pascal é possível especificar três métodos de passagem de parâmetros: a passagem por valor, a passagem por referência e a passagem de argumentos funcionais. Apenas vamos referir os dois primeiros.

5.3.1 Passagem por valor

Quando um parâmetro é passado por valor, o valor do parâmetro concreto é atribuído ao parâmetro formal correspondente e a única ligação entre os parâmetros concretos e os parâmetros formais é uma

associação unidireccional de valores; os parâmetros formais não podem modificar os parâmetros concretos e consequentemente este processo pode não ser útil na chamada de procedimentos. Em Pascal a passagem por valor é utilizada em todos os parâmetros existentes num grupo de parâmetros que não seja antecedido pela palavra “var”; é o exemplo dos parâmetros concretos a e b no programa selecção apresentado no parágrafo 5.1.

Neste programa quando se encontra a chamada ao procedimento soma é feita uma associação unidireccional entre os valores dos parâmetros concretos a e b e os parâmetros formais x e y, ou seja a única ligação entre o ponto de chamada e o subprograma é feita através do valor que é passado dos parâmetros concretos para os parâmetros formais, não podendo estes modificar os parâmetros concretos. A passagem por valor deve ser utilizada sempre que um parâmetro concreto não precise de ser modificado por um subprograma.

5.3.2 Passagem por referência

Quando um parâmetro é passado por referência, o que o parâmetro formal recebe não é o valor do parâmetro concreto correspondente, mas a sua localização na memória do computador; na passagem por referência os parâmetros formais e os parâmetros concretos partilham o mesmo espaço de memória, e deste modo, qualquer modificação feita aos parâmetros formais é transmitida aos parâmetros concretos.

Em Pascal para especificar que um ou vários parâmetros devem ser passados por referência, o grupo de parâmetros em que são declarados deve ser antecedido pela palavra “var”.

Como exemplo de utilização de passagem por referência temos os parâmetros concretos c e d do programa selecção apresentado no parágrafo 5.1:

Neste programa quando se encontra a chamada ao procedimento soma, a comunicação entre o ponto de chamada e o procedimento é feita através da referência às posições de memória onde estão armazenados os valores das variáveis c e d, passando os valores dos parâmetros concretos para os parâmetros formais. Depois da chamada do procedimento c e z são dois nomes para a mesma variável (o mesmo acontece entre d e w). Como os valores de z e w são alterados dentro

do procedimento o mesmo acontece para as variáveis c e d. Vemos assim que um parâmetro formal em que seja utilizada a passagem por referência corresponde à mesma variável que o parâmetro concreto correspondente, apenas com outro nome. A passagem por referência deve então ser utilizada sempre que um parâmetro concreto deva ser modificado por um subprograma

5.4 A estrutura em blocos de um programa em Pascal

Um bloco é um conjunto de declarações e de instruções associadas a um programa ou subprograma.

Um programa é constituído por um cabeçalho de programa seguido por um bloco. No cabeçalho dá-se o nome ao programa. Da mesma maneira um procedimento ou uma função é constituída por um cabeçalho que dá o nome ao procedimento ou à função, e define o seu modo de comunicação com o resto do programa, seguido por um bloco. O desenvolvimento de um programa é a construção de uma série de blocos em que a informação (constantes, variáveis, tipos estruturados, procedimentos, funções, etc) que é declarada dentro de um bloco apenas é utilizável dentro desse bloco. Consideramos o programa ExemploBlocos apresentado:

```
program ExemploBlocos (input, output) ;
var P1, P2, P3 : integer ;

procedure Bloco1 (var Pb1 : integer) ;
    var B11, B12 : real ;
        B13 : integer ;
    begin
        <instruções do bloco1>
    end ; {Bloco1}


procedure Bloco2 (var Pb2 : integer) ;
    var B21, B22 : integer ;

procedure Bloco3 (Pb31 : integer ;
                  var Pb32 : integer) ;
    var B31, B32 : integer ;
    begin
        <instruções do bloco3>
    end; {Bloco3}

    begin
       <instruções do bloco2>
    end; {Bloco2}

begin
    <instruções do ExemploBlocos>
end. {ExemploBlocos}
```

Neste programa as variáveis P1, P2, P3 declaradas no programa exterior são utilizáveis por todas as instruções do programa. Mas as variáveis Pb1, B11, B12 e B13 declaradas no bloco do procedimento Bloco 1 são apenas utilizáveis pelas instruções deste bloco.

As variáveis Pb2, B21 e B22 declaradas no bloco do procedimento Bloco2 são apenas utilizáveis no Bloco2 e no Bloco3 uma vez que o procedimento Bloco3 é declarado dentro do procedimento Bloco2.

Os procedimentos Bloco1 e Bloco2 são declarados no bloco exterior e consequentemente podem ser utilizados pelo programa ExemploBlocos.

A estrutura de blocos de um programa em Pascal consiste em considerar que estes se encontram hierarquicamente organizados estando o bloco exterior hierarquicamente acima dos blocos definidos dentro dele.

A utilização de subprogramas torna os programas mais simples de escrever e de ler; ao ler um programa que contenha vários comentários

encontramos referências aos resultados de cada subprograma facilitando a compreensão do programa; além disso os programas ficam mais curtos uma vez que se o programa utiliza várias vezes um subprograma essa utilização é feita chamando o subprograma em vez de listar todo o conjunto de instruções que fazem parte desse subprograma. As alterações também são mais fáceis uma vez que ao desejar modificar parte de um programa sabe-se mais facilmente onde fazer essa alteração.

Considere o algoritmo seguinte que efectua o produto de uma matriz por um vector e calcula o traço da matriz se ela for quadrada:

Objectivo: produto de uma matriz por um vector e cálculo do traço se for quadrada;

Variáveis dados: m, n, inteiros, dimensões da matriz e dos vectores;
A, bi-indexada de reais , matriz dada;
B, indexada de reais, vector dado;
Variáveis auxiliares: i, j, inteiros, contadores;
Variáveis resuitado: C, indexada de reais, vector calculado;
traco, real, traço da matriz;

```
escreva(‘ introduza a dimensão da matriz (mxn) ‘);
leia(m,n);

para i ← 1 até n faça

    { escreva(‘ introduza a componente ‘, i ,  ‘ do vector ’);
      leia (B[i]);

para i ← 1 até m faça
      para j ← 1 até n faça

        { escreva (‘ introduza acomponente ‘, i , j ,  ‘ da matriz ’);
          leia (A[i,j]);

para i ← 1 até m faça

      { soma ← 0.0;
        para j ← 1 até n faça
            soma ← soma +A [i,j ] * B [j];
        C [i] ← soma;
```

```
se m = n então
        traco ← 0.0;
        para i ← 1 até m faça
              traco ← traco + A[i,i];
        escreva (" o traço da matriz é igual a ", traco);


escreva (‘ vector resultado’ );
para i ← 1 até m faça
       escreva ( C[i]);
```

Apresenta-se o programa em linguagem Pascal (não estruturada), correspondente ao algoritmo anterior e em seguida o programa em linguagem Pascal estruturada.

```
Program produto ( input,output );
{ Produto de uma matriz por um vector e cálculo do traço da matriz }

type matriz = array [1..20,1..20] of real;
type vector = array [1..20] of real;
var m, n, i, j: integer;   traco :real;
     A: matriz;
     B, C: vector;

begin
    writeln(‘ introduza a dimensão da matriz (mxn) ‘);
    readln(m,n);
    for i := 1 to n do
          begin
               writeln (‘ introduza acomponente ‘, i ,  ‘ do vector ’);
               readln (B[i]);
          end;

    for i := 1 to m do
          for j := 1 to n do
               begin
                    writeln (‘ introduza acomponente ‘, i , j ,  ‘ da matriz ’);
                    readln (A[i,j]);
               end;

    for i := 1 to m do
          begin
               soma := 0.0;
               for j := 1 to n do
                    soma := soma +A [i,j ] * B [j];
               C [i] := soma;
          end;
```

```
    if m = n then
        begin
            traco := 0.0;
            for I := 1 to n do
                traco := traco+ A[i,i];
        writeln ( ' o traço da matriz é igual a ', traco);
        end;

    writeln ('vector resultado' );
    for i := 1 to m do
        write ( C[i]);

end.
```

```
Program produto (input,output);
{Produto de uma matriz por um vector e cálculo do traço da matriz}

type matriz = array [1..20,1..20] of real;
type vector = array [1..20] of real;
var m, n: integer;  traco: real;
     A: matriz;
     B, C: vector;


procedure lervec ( var X: vector; n : integer);
var i: integer;

begin
     for i := 1 to n do
          begin
               writeln (' introduza acomponente ', i ,  ' do vector ');
                    readln (X[i]);
               end;
end;   { lervec}


procedure lermat ( var M: matriz; m, n : integer);
var i,j: integer;

begin
      for i := 1 to m do
           for j := 1 to n do
                begin
                    writeln (' introduza acomponente ', i , j ,  ' da matriz ');
                    readln (M[i,j]);
                end;
end;   { lermat}
```

```
Procedure prod ( M: matriz; var Y: vector; X: vector; m, n: integer);
var i,j :integer;
     soma:real;

begin
    for i := 1 to m do
          begin
              soma := 0.0;
              for j := 1 to n do
                    soma := soma +M [i,j ] * X [j];
              Y [i] := soma;
    end;
end;   { prod}


procedure escvec (  X: vector; n : integer);
var i: integer;

begin
     writeln ( ‘ vector resultado’ );
     for i := 1 to n do
          write ( X[i]);
end;   { escvec}

function soma (A: matriz; m: integer): real;
var i : integer;
      traco: real;

begin
     traco := 0.0;
     for i := 1 to m do
          traco := traco+ A[i,i];
     soma := traco
end;   { soma}


begin
    writeln(‘ introduza a dimensão da matriz (mxn) ‘);
    readln(m,n);
    lervec (B, n);  lermat (A, m, n);
    prod (A, C, B, m, n);
    if m = n then
        begin
            traco := soma (A, m);
            writeln ( ' o traço da matriz é igual a ', traco)
        end;
    writeln (‘vector resultado’ );
    escvec (C, m);
end.
```

Neste último programa, após a definição dos tipos de vector e matriz e da declaração das variáveis utilizadas na zona de instruções, existe a declaração dos procedimentos e funções utilizadas no programa: os

procedimentos de leitura e de escrita de um vector, de leitura de uma matriz e do produto de uma matriz por um vector e ainda uma função para calcular o traço de uma matriz quadrada.

Os parâmetros concretos correspondentes aos parâmetros formais precedidos pala palavra ”var” são os que podem ser modificados pelo procedimento uma vez que partilham o mesmo espaço de memória; deste modo qualquer alteração feita a esses parâmetros formais pelo procedimento é transmitida aos parâmetros concretos correspondentes ou seja as variáveis correspondentes aos parâmetros concretos vão receber os valores que os respectivos parâmetros formais têm dentro do procedimento quando termina a execução do corpo do procedimento. Por exemplo nos procedimentos de leitura de um vector o parâmetro formal correspondente X tem que ser antecedido pela palavra “var” caso contrário, os valores lidos para as coordenadas do vector não são transmitidos para o parâmetro concreto correspondente (o vector B), quando o procedimento è chamado na zona de instruções do programa. No procedimento de escrita de um vector como o vector escrito não é alterado dentro do procedimento já não é necessário preceder o respectivo parâmetro formal pela palavra “var”.

A função que calcula o traço da matriz só é chamada se a matriz A for quadrada; a função é chamada usando uma instrução de atribuição e não da mesma forma que se chama um procedimento. Além disso ao chamar uma função só obtemos um valor ao contrário do que acontece com os procedimentos.

Na zona de instruções sempre que se chama um procedimento ou uma função tem de haver uma correspondência entre o número e o tipo de parâmetros formais e concretos.

5.5 Units em Pascal

Ao construir um programa, por vezes existe necessidade de se utilizar uma porção do código várias vezes como por exemplo a escrita e leitura de vectores e matrizes, o produto de matrizes, a inversão de matrizes, etc.

Já vimos que a melhor maneira de o conseguir é através da correcta utilização de procedimentos e funções.

No entanto se ao construir um programa pudéssemos utilizar constantes, tipos de variáveis definidos , variáveis, procedimentos e funções que já estivessem construídos, conseguíamos diminuir substancialmente o tempo de desenvolvimento de novos programas.

Uma técnica de programação utilizada em Pascal, que permite a utilização inter-programa de entidades previamente definidas, é a construção de Units.

5.5.1 Estrutura de uma Unit

Uma Unit é um conjunto de constantes, tipos de variáveis definidos, variáveis, procedimentos e funções. Uma Unit é como um programa que pode ser chamado do nosso programa, permitindo-lhe utilizar todas as constantes e tipos estruturados definidos na Unit e variáveis, procedimentos e funções declarados na Unit.

A estrutura de uma Unit não é igual à de um programa:

```
UNIT <identificador>;

INTERFACE
uses <lista de units>;          (opcional)
   <declarações visíveis>

IMPLEMENTATION
   <declarações privadas>
   <procedimentos e funções>

BEGIN
   <inicialização de código>
END.
```

O cabeçalho de uma Unit começa com a palavra Unit seguida do nome da Unit tal como acontece com o nome de um programa. Logo a seguir é a secção da interface que começa com a palavra “interface” e termina com a palavra “implementation”. Se uma Unit usa outras Units tem de as especificar no início da interface colocando a palavra “uses” antes do nome das referidas Units.

A interface determina o que é visível para qualquer programa (ou Unit) que a use. Na interface de uma Unit pode-se definir constantes, definir tipos de variáveis, declarar variáveis, procedimentos e funções.

Os procedimentos e funções visíveis para os outros programas têm o cabeçalho na interface, mas é na secção de implementação que aparece a declaração dos procedimentos e das funções.
A secção de implementação é a parte "privada" da Unit. Tudo que é declarado na interface é visível na implementação; para além disso a implementação pode ter declarações adicionais, embora não sejam visíveis para outros programas que usem a Unit. É nesta secção que aparece a declaração de todos os procedimentos e funções cujos cabeçalhos aparecem na interface.

A secção de implementação tem início na palavra "implementation" e geralmente termina na palavra "end". No entanto se colocarmos a palavra "begin" antes da palavra "end", com instruções entre elas, obtemos uma secção chamada inicialização. Nesta secção pode-se, por exemplo, definir variáveis, código que pode ser utilizado pela Unit ou pelos programas que a usem (através da interface) ou efectuar a abertura de ficheiros.

Uma Unit deve ser compilada e guardada num ficheiro especial (turbo TLP). Um programa que use constantes, tipos de variáveis, variáveis, procedimentos e funções declaradas numa Unit deve incluir no seu cabeçalho a palavra "uses" seguida do nome da Unit. Se o programa utiliza várias Units, a lista dos seus nomes deve aparecer depois da palavra "uses".

Como exemplo de uma Unit apresentamos a Unit VECTORES onde se define um tipo vector e é feita a declaração dos procedimentos de leitura e escrita de vectores :

```
UNIT VECTORES;

INTERFACE
const
        maxdim = 10;
type
       vector = array [1..maxdim] of integer ;
var x: vector;

procedure levector (var x : vector ; var n : integer) ;
procedure escvec (x : vector ; n : integer) ;
```

```
IMPLEMENTATION

procedure levector ;
   var i : integer ;
   begin
      writeln (‘ dimensão do vector ’) ;
      readln ( n ) ;
      for i : = 1 to n do
         begin
            write (‘Componente ’, i, ‘ ? ’) ;
            readln ( x[i] ) ;
         end ;
   end;

procedure escvec ;
      var i : integer ;
      begin
         for i : = 1 to n do
            writeln (‘Componente [ ’, i ,’ ] = ‘ ,  x [i] ) ;
      end;

END.   { UNIT VECTORES }
```

Suponhamos que desejávamos construir um programa para ler um vector e o transformar multiplicando por 2 todas as componentes que fossem números ímpares ; usando a Unit VECTORES o programa resulta mais simples, uma vez que não é necessário rescrever os procedimentos de leitura e escrita de vectores, sendo apenas necessário chamá-los uma vez que estão declarados na Unit VECTORES :

```
PROGRAM  utilvec;
uses VECTORES;
var n, i : integer ;

begin
    levector (x, n) ;
    for i : = 1 to n do
        if  x[i] mod 2  <> 0 then
           x [i] : = 2 * x [i] ;
    escvec (x, n) ;
end.   {program utilvec}
```

6. A LINGUAGEM FORTRAN 90

Neste capítulo apresentamos de uma forma sucinta os temas tratados nos capítulos anteriores.

6.1 Tipos de informação elementares em Fortran 90

Tipo	Sintaxe	Valor
integer (inteira)	0	0
	23	23
	+23	23
	-2590	-2590
	999999	999999
real (real)		
precisão simples	0.	0
	1000.	1000
	123.45	123.45
	0.056	0.056
	5.6e-2, 5.6E-2	0.056
	6e5	600000
Precisão dupla	0.d0	0
	5.6d-2	0.056
	6d5	600000
complex (complexa)	(2.3, 4.5)	2.3+4.5i
	(0.,3.)	3i
	4.,0.)	4+0i
character (alfanumérica)	“valor=”	valor=
	“IVA”	IVA
	“iva”	iva
	“25+30”	25+30
	“ana paula”	ana paula
logical (lógica)	.true.	true ou 1
	.false.	False ou 0

Tabela 6.1: Tipos de constantes e variáveis

Operador	Definição	Uso	Significado
.not.	negação	.not.a	complemento de **a**
.and.	conjunção	a.and.b	produto booleano de **a** e **b**
.or.	disjunção	a.or.b	soma booleana de **a** e **b**
.eqv.	equivalência	a.eqv.b	equivalência lógica de **a** e **b**
.neqv.	Não equivalência	a.neqv.b	não-equivalência lógica de **a** e **b**

Tabela 6.2: Operações sobre valores lógicos

A	B	.not.A	A .and.B	A.or. B	A .eqv.B	A.neqv. B
true	true	false	true	true	true	false
true	false	false	false	true	false	true
false	true	true	false	true	false	true
false	false	true	false	false	true	false

Tabela 6.3: Operações sobre valores lógicos

Operador	Definição	Uso	Significado
**	potenciação	a**b	**a** elevado a **b**
*	multiplicação	a*b	**a** multiplicado por **b**
/	divisão	a/b	**a** dividido por **b**
+	adição	a+b	**a** mais **b**
-	subtracção	a-b	**a** menos **b**
+	mais unário	+a	o mesmo que **a**
-	menos unário	-a	**a** com sinal trocado

Tabela 6.4: Operadores numéricos sobre operandos inteiros e reais

Operações com inteiros

- São efectuadas sobre operandos inteiros
- resultado da divisão é truncado, não arredondado

10/3 = 3	1/3 = 0
3 * 10/3 = 10	10/3*3 = 9
2**2 = 4	2** (-2) = 0

Operações com Reais ou com Reais e Inteiros

- São efectuadas sobre operandos reais e/ou inteiros

10./3. = 3.3333333 1./3. = 0.33333333

1./3 = 0.33333333 1/3. = 0.33333333

3/2/3. = (3/2)/3. = 0.33333

2.**(2) = 2**(2.) = 4.

2.**(-2) = 2**(-2) = 0.25

integer n	n = 10./3	n = 3
real x	x = 10/3	x = 3.
real y	y = 10/3.	y = 3.3333333

Funções de Conversão		
Função	**Exemplo**	**Significado**
int	n = int (a)	transforma a num inteiro
real	a = real (n)	transforma n num real
dble	a = dble (n)	transforma n para dupla precisão
Funções Numéricas		
Função	**Exemplo**	**Significado**
abs	abs (-2)	= 2 valor absoluto
mod	mod (3., 2.)	= 1. resto da divisão
max	max (1, -1, 2)	= 2 valor máximo
min	min (1, -1, 2)	= -1 valor mínimo
Funções Matemáticas		
Função	**Exemplo**	**Significado**
sqrt	y=sqrt (x)	raiz quadrada ($x \geq 0$)
exp	y=exp (x)	exponencial
Log	y=log (x)	logaritmo natural ($x>0$)
log10	y=log10 (x)	logaritmo na base 10 ($x>0$)
sin	y=sin (x)	seno (x radianos)
cos	y=cos (x)	cosseno (x radianos)
tan	y=tan (x)	tangente (x radianos)
asin	y=asin (x)	arco-seno ($\|x\| < 1$; y radianos ; $-\pi/2 \leq y \leq \pi/2$)
acos	y=acos (x)	arco-cosseno ($\|x\| < 1$; y radianos ; $0 \leq y \leq \pi$)
atan	y=atan (x)	arco-tangente (y radianos ; $-\pi/2 < y < \pi/2$)
sinh	y=sinh (x)	seno hiperbólico
cosh	y=cosh (x)	cosseno hiperbólico
tanh	y=tanh (x)	tangente hiperbólica

Tabela 6.5: Funções intrísecas (Nota: a, x são reais ; n é inteiro)

Operador	Definição	Uso	Significado
==	igual	a == b	**a** igual a **b** ?
/ =	diferente	a / =b	**a** diferente de **b** ?
<	menor que	a < b	**a** menor que **b** ?
< =	menor ou igual a	a < = b	**a** menor ou igual a **b** ?
>	maior que	a > b	**a** maior que **b** ?
> =	maior ou igual a	a > = b	**a** maior ou igual a **b** ?

Tabela 6.6: Operadors relacionais (Note: a e b são expressões numéricas)

6.2 Definição de constantes em Fortran 90

real, parameter : : nome = valor

Exemplos da definição de constantes:

real, parameter : : pi = 3.14159
integer, parameter :: ano =1998

6.3 Declaração de variáveis em Fortran 90

Tipo		Instrução
integer		integer : : int, k, xpto
real		
	simples	real : : alpha, b_i, j0
	dupla	double precision : : x, y
complex		
	simples	complex : : z1, z2
	dupla	complex, double precision : : a
character		character : : nome
		character (len = 3) : : sim
logical		logical : : yes, no

Tabela 6.7: Declaração de variáveis

Se nada for dito em contrário, o FORTRAN assume implicitamente:

- **inteiras** todas as variáveis que começam por i, j, k, l, m, n
- **reais** todas as outras a-h, o-z

Em **FORTRAN 90** recomenda-se o uso da instrução

```
implicit none
```

e a subsequente declaração de todas as variáveis usadas.

6.4 Instrução de atribuição em Fortran 90

Mantendo os exemplos do parágrafo 3.2.1 apresentamos as respectivas instruções de atribuição para a linguagem Fortran 90.

```
Nota = 17
contador = contador + 1
x = y * 2 – z
```

6.5 Instruções de leitura e escrita em Fortran 90

- Para ler dados a partir do teclado usamos

  ```
  read fmt, lista
  ```

  ou

  ```
  read (*, fmt) lista
  ```

  fmt = * formato livre

- Para escrever resultados no monitor usamos

  ```
  print fmt, lista
  ```

  ou

  ```
  write (*, fmt) lista
  ```

  fmt = * formato livre

Nas instruções de leitura a palavra “lista” refere-se a uma lista de variáveis.

Nas instruções de escrita a palavra "lista" refere-se a uma lista de variáveis, expressões ou texto. Alguns exemplos de leitura e escrita são apresentados, considerando novamente n=12.14 e K = 6:

```
Read * , x, y      ! Leitura de dois números

Print *, "  Teste de ensaio"
Write (*,  "( a) ")  "  Teste de ensaio "
print   " ( es 14.5,  f10.2, i5 ) " , n, n, k
write ( * , " ( es 14.5,  f10.2, i5 ) " )  n ,n, k
```

Na primeira instrução são lidos dois números e colocou-se um comentário que explica o que a instrução faz. As outras instruções são de escrita; nas duas primeiras é escrito um texto e nas duas últimas são escritos os números n e k obtendo-se os resultados:

```
 Teste de ensaio
 Teste de ensaio
   1.21400E+01     12.14    4
   1.21400E+01     12.14    4
```

6.6 A instrução "if" em Fortran 90

A estrutura "if" é uma estrutura de controlo condicional que apresenta várias sintaxes:

Sintaxe 1

```
If (expressão lógica)  then
    <bloco de instruções>
[else
    bloco de instruções]
end if
```

Sintaxe 2

```
If (expressão lógica)  instrução
```

Sintaxe 3

```
if (expressão lógica) then
    <bloco de instruções>
else if (expressão lógica) then
    <bloco de instruções>
else if (expressão lógica) then
    <bloco de instruções>
else if ...
    ...
else
    <bloco de instruções>
end if
```

Apresentamos os exemplos do parágrafo 3.3.1, utilizando a instrução "if" da linguagem Fortran 90:

- Primeiro exemplo

```
If  (x  <  = y ) then
    x : = x + 5
else
    y : = y + 5
end if
```

- Segundo exemplo

```
if (V2 / = 0 .and. mod (V1,V2) == 0) V1 = abs (V1);
```

- Terceiro exemplo

```
Program maior
!  escreve o maior de dois números
   Implicit none
   Integer :: a, b

   Print*," escreva dois números a e b"
   read*, a,b
   if  (a > = b) then
      if  (a == b) then
         print*, "os números são iguais"
      else
         print*, "o maior é o", a
      end if
   else
      print*, "o maior é o", b
   end if
end Program maior
```

6.7 A estrutura “case” em Fortran 90

A estrutura case é uma estrutura de controlo condicional com a seguinte sintaxe:

```
select case (expressão)
     case (caso1)
       <bloco de instruções>
     case (caso2)
       <bloco de instruções>
     ...
    [case default
       <bloco de instruções>]
end select
```

em que expressão pode ser uma variável inteira, alfanumérica ou lógica.

Como exemplo apresentamos o exemplo do parágrafo 3.3.2 , agora utilizando a linguagem Fortran 90:

```
PROGRAM conver
   Implicit nome
   Integer: : nota

   Print*,” escreva a nota”
   read*,nota
   select case (nota)
       case (18, 19, 20)
          print *, “A”
       case (14, 15, 16, 17)
          print *, “B”
       case (10, 11, 12, 13)
          print *, “C”
       case (5,6,7, 8, 9)
          print *, “D”
       case default
          print *, “E”
   end select

end program conver
```

6.8 A instrução “while” em Fortran 90

A sintaxe de instrução “while” é dada por

```
     do while <expressão lógica>
              <bloco de instruções>
     end do
```

O exemplo do parágrafo 3.4.1, agora em linguagem Fortran, é igual a:

```
PROGRAM soma1
! soma de uma sequência de números pares
   implicit none
   integer :: x, soma

    soma = 0
    print*," escreva um número par"
    print*," para terminar escreva um número impar"
    read*, x
    do while mod ( x, 2)  == 0
              soma =  soma + x
              print*," escreva um número par"
              print*," para terminar escreva um número impar"
              read*, x
        end do
     print*, soma
end Program soma1
```

6.9 A estrutura "do" em Fortran 90

Um ciclo com controlo de ciclo é controlado por um contador que toma valores sucessivos até um limite pré-estabelecido e tem a seguinte sintaxe:

```
do <index> = <expr1>, <expr2> [ , <expr3>]
     <bloco de instruções>
end do
```

em que

index – contador

expr1 – valor inicial do contador

expr2 – valor final do contador

expr3 – incremento do contador

- O contador deve obedecer às seguintes regras:
  - O contador é uma variável
  - As boas regras de programação sugerem que o contador seja um inteiro (embora o FORTRAN aceite contadores reais)
  - O valor do contador não pode ser alterado no bloco de instruções

- Ao terminar a execução do ciclo, o contador tem o valor de expr2

- O incremento, expr3, pode ser positivo ou negativo

- O exemplo do parágrafo 3.4.3 é apresentado, agora em linguagem Fortran:

```
PROGRAM medias
! média das alturas de n alunos
   Inplicit none
   Integer :: i, n
   real :: alt, media

   print*," escreva o número de alunos "
   read * , n
   media = 0.0
   do i  = 1 , n
       print*, "escreva a altura do aluno número ", i
       read *, alt
       media  = media + alt
   end do
   media  = media / n
   print *, "A média das alturas dos alunos é igual a" , media
end program media
```

6.10 Tabelas em Fortran 90

Declaração de variáveis indexadas;

```
integer, dimension (0 , 20) :: A, B
real, dimension (10, 10) : : C
```

Apresentamos o exemplo do parágrafo 4.1.1 que calcula a transposta de uma matriz de 4 linhas e 5 colunas:

```
Program mat
!  ler uma matriz (4, 5) e transpor
   implicit none
   integer, dimension (4, 5) : : A
   integer, dimension (5, 4) : : AT
   integer : : i, j

   print *, “coeficientes da matriz”
   do i = 1, 4  ! ler por linhas
       print*, “ escreva o coeficiente  A(”, i,j,”) da matriz”
       read *, A (i, j), j = 1, 5)
   end do

   do i = 1, 4      !    transpor
       do j = 1, 5
           AT (j, i) = A (i, j)
       end do
   end do

   print *, “ matriz A transposta ”
   print “(4i5)”, &
       (( AT (i, j), j = 1, 4), i = 1, 5))

end program mat
```

6.11 Subprogramas em Fortran 90

6.11.1 Subroutinas externas

- Subprogramas do tipo subroutine têm a seguinte sintaxe:

```
subroutine nome (argumentos)
     <bloco de instruções>
end subroutine nome
```

- São chamadas através da instrução

```
call nome (argumentos)
```

- As variáveis da lista de argumentos têm que ter o mesmo tipo no programa principal e na subroutina e são usadas na mesma ordem, mas não precisam de ter os mesmos nomes.

- O subprograma termina quando encontra a instrução end subroutine e volta ao programa principal (para a instrução imediatamente a seguir)

```
program exemplo1
    implicit none
    integer : : caso, n1, n2
    real : : x1, x2
    ...

    call xpto (caso, n1, (x1), n2, x2)
    ...
end program exemplo1

subroutine xpto (iflag, i, x , j, y)
    implicit none
    integer : : iflag, i, j
    real : : x, y
    ...
end subroutine xpto
```

Nota: o terceiro argumento x1 é passado por valor e os restantes são passados por referência.

6.11.2 Funções externas

- Subprogramas do tipo function têm a seguinte sintaxe:

```
function nome (argumentos) result (var)
    <bloco de instruções>
end function nome
```

- São usadas de um modo semelhante às funções intrínsecas e calculam apenas um valor

  valor = nome (argumentos)

  - A variável var e a função nome têm que ter o mesmo tipo

  - As variáveis da lista de argumentos têm que ter o mesmo tipo no programa principal e na função e são usadas na mesma ordem, mas não precisam de ter os mesmos nomes

  - O subprograma termina quando encontra a instrução end function, voltando ao programa principal (para a instrução imediatamente a seguir)

```
program exemplo2
    implicit none
    integer : : caso, n1, n2
    real : : x1, x2, x
    real, external :  xpto
    ...
    x = xpto (caso, n1, x1, n2, x2)
    ...
end program exemplo2

function xpto (iflag, i, x, j, y)  result (f)
    implicit none
    integer : : iflag, i, j
    real : : x, y, f
    ...
    f =x * y / j
end function xpto
```

6.11.*3 Subprogramas Módulos*

- Subprogramas do tipo módulo têm a seguinte sintaxe:

```
module parâmetros
   implicite none
   real : : a, b
end module parâmetros

program derivados 2
use parâmetros
implicite none
...

end program derivados2
```

Regras

- Subprogramas do tipo módulo têm a seguinte sintaxe:

```
modulo nome
     <bloco de instruções>
end modulo nome
```

- São usados através da instrução

  use nome

  que deve ser a primeira instrução logo a seguir a program, subroutine ou function

- Servem para transferir informação entre o programa principal e os subprogramas, ou entre subprogramas.

6.11.4 Subprogramas internos

- Apresenta-se o exemplo de um programa que contém um subprograma interno do tipo subroutine::

```
program derivadas
    implicite none
    real (kind = 8) : : x, dx, d1, d2
    ...
    call deriv (x, dx, d1, d2)
    ...

Contains
    subroutine deriv (y, dy, dy1, dy2)
        implicit none
        real (kind = 8) : : y, dy, dy1, dy2
        ...
    end subroutine deriv

end program derivadas
```

Regras

- Subprogramas internos podem ser do tipo function e/ou subroutine e têm a mesma sintaxe que os subprogramas externos.

- Subprogramas internos aparecem dentro do programa principal a seguir à instrução contains e antes da instrução end.

- Partilham todas as variáveis com o programa principal, mas podem também ter argumentos e variáveis internas.

BIBLIOGRAFIA:

J. Pavão Martins, “Introdução à Programação usando o Pascal”, McGRAW-HILL, Portugal, 1994

John Konvalina, & Stanley Wileman, “Programing with Pascal”, McGRAW-HILL, USA, 1987

Byron S. Gottfried, “Programação em Pascal”, McGRAW-HILL, Portugal, 1994

Brian D. Hahn, “Fortran 90 for Scientists and Engineers”, Department of Applied Mathematics, University of Cape Town, University Press, Cambridge, 1995

Programmer’s Guide Microsoft Fortran, Power Station Version 4.0
Development System for Windows 95 and Windows NT Workstation
1995 Microsoft Corporation

ANEXO A :

Sub-tabela do código ASCII que estabelece a relação entre letras e números de ordem:

Número de Ordem	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84
Caracter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

Número de Ordem	85	86	87	88	89	90	97	98	99	100	101	102	103	104	105	106	107
Caracter	U	V	W	X	Y	Z	a	b	c	d	e	f	g	h	i	j	k

Número de Ordem	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122
Caracter	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

ANEXO B

Com a colaboração de:

André Teixeira Puga
Catarina Castro
Cristina Faria
Isilda Viana

EXERCÍCIOS PROPOSTOS

1) Escreva um algoritmo que calcule a média das idades dos alunos de uma turma.

.

2) Escreva um algoritmo que efectue a soma dos n primeiros números pares.

3) Escreva um algoritmo que efectue o produto dos n primeiros números ímpares.

4) Escreva um algoritmo que determine a idade do aluno mais velho que frequenta a Faculdade de Engenharia.

5) Escreva um algoritmo que determine o número de dias em que a temperatura máxima diária da cidade de Estocolmo é positiva, durante um mês de inverno.

6) Escreva um algoritmo que efectue a soma dos múltiplos de três maiores que 1000 e menores que 2000.

7) Escreva um algoritmo que determine o nome do aluno mais pesado de uma turma, e também o respectivo peso.

8) Escreva um algoritmo que calcule o factorial de um número.

9) Escreva um algoritmo para calcular a média de todos os elementos de uma matriz.

10) Escreva um algoritmo para somar os quadrados dos elementos de uma matriz não quadrada A.

11) Escreva um algoritmo para calcular a média de todos os elementos de uma matriz triangular inferior.

12) Escreva um algoritmo que some os elementos de uma matriz posicionados nas colunas de índice par.

13) Escreva um algoritmo para trocar duas linhas ou colunas de uma matriz.

14) a) Escreva um algoritmo para calcular a soma de duas matrizes não quadradas..

    b) Traduza em linguagem Pascal o algoritmo da alínea anterior.

15) Escreva um programa em linguagem Pascal que some os elementos de uma matriz (não quadrada) acima da diagonal principal.

16) a) Escreva um algoritmo para calcular o determinante de uma matriz triangular.

b) Traduza em linguagem Pascal o algoritmo da alínea anterior.

17) Escreva um programa em linguagem Pascal que determine a transposta de uma matriz não quadrada A sem criar outra.

18) Considere dois vectores A e B com a mesma dimensão n. Escreva um programa em Pascal que transforme os dois vectores trocando as componentes negativas do vector A pelas componentes do vector B com o mesmo índice.

Exemplo: Dados: A = (1, -1, 8, 0, -5) ; B = (2, 4, -2, 2 3) ;

Resultado: A = (1, 4, 8, 0, 3); B = (2, -1, -2, 2, -5) ;

19) Escreva um programa em linguagem Pascal que escreva as horas, minutos e segundos correspondentes a um número n lido em segundos.

Exemplo: dado n = 10000
resultado 2h 46m 40s

20) Considere o seguinte programa em linguagem Pascal

```
Program ensaio;
 var x, y, z, w : integer;

procedure soma (a ,b : integer; var c, d :integer);
var i : integer;
begin
i := 0 ;
repeat
     i := i+1;
     c := a+b;
     a :=i*a;
     d := b-a;
until ( i = 5 )  or  ( d < 0 );
end;

begin
soma (x, y, z, w);
writeln( x, y, z, w);

end.          {program ensaio}
```

Percorra o programa para x = 2 e y = 20 e comente os resultados obtidos.

21) Considere o seguinte Programa em linguagem Pascal:

```
Program EXAME;
var a,b,c,d,e,h : integer;
procedure teste (x : integer; var p,q,r,z : integer);
var y : integer;
begin
    p:= 0; q:= 0; r:= 0; z:= 0;
    while x < 0 do
      begin
            y:= x mod 3;
            if y = 2  then  p:= p+1;
            if y = 1 then  q:= q+1
            else r:= r+1;
            x:= x div 3;  z:= z + y;
      end;
end;
function prod (x,y): integer;
var z,w : integer;
begin
    z:= x * y;
    if z > 10 then w:= z * 2
              else w:= z / 2;
    prod := w;
end;
begin
        a:= 60; h:= 0;
        Teste (a,b,c,d,e,);
         if d = f then h: = prod (b+d,c+e);
         write (a,b,c,d,e,h);
 end.
```

Percorra o programa apresentando tabelas separadas para o programa principal e para os subprogramas.

22) Escreva um programa em linguagem Pascal que inverta a ordem das componentes de um vector dado.

Exemplo: vector dado A = (1,4,2,7,5)
vector resultado A = (5,7,2,4,1)

23) Escreva um programa em linguagem Pascal que ordene por ordem crescente as componentes de um vector dado.

24) Elabore um programa em linguagem Pascal tal que dada uma matriz troque as suas linhas de forma a que os elementos da 1ª coluna fiquem por ordem crescente.

25) Considere a série:

$$\sum_{i=1}^{\infty} u_i = x^2 + x^4/2! + x^6/3! + \ldots + x^{2n}/n! + \ldots$$

a) Escreva uma função em linguagem Pascal para calcular o termo $\mathbf{u_n}$ a partir do termo anterior $\mathbf{u_{n-1}}$

b) Escreva um programa em linguagem Pascal que calcule a soma dos n primeiros termos da série e utilize a função criada na alínea anterior.

26) Considere a série:

$$\cos x = 1 - x^2/2! + x^4/4! - x^6/6! + \ldots + x^{2n}/(2n)! * (-1)^{n-1} + \ldots$$

a) Escreva uma função em linguagem Pascal para calcular o termo $\mathbf{u^n}$ a partir do termo anterior $\mathbf{u_{n-1}}$

b) Escreva um programa em linguagem Pascal que calcule a soma dos n primeiros termos da série e utilize a função criada na alínea anterior.

27) a) Defina uma função em Pascal que calcule arcos(x), sabendo que as funções trignométricas intrínsicas que a linguagem Pascal dispõe são: cos, sen e arctan.

b) Elabore um programa em Pascal que use a função da alínea anterior para tabelar a função:

f(x) = arcos(x) /x;

para os valores de x = 0. 0.2, 0.4, 0.6 e 0.8

Ex:

x	f(x)
0.0	.....
0.2	.....
0.4	.....
0.6	.....
0.8	.....

28) Suponha que queria escrever um programa em linguagem Pascal para efectuar o produto de uma matriz por um vector utilizando um procedimento que calcula o referido produto mas que apresenta alguns erros.

```
Procedure produto (M :matriz; X,Y :vector; m,n :integer);
var i,j :integer;
    soma:real;

begin
soma:=0.0;
for i:= 1 to m do
        begin
        for j= 1 to n do
                soma:= soma +M [i,j ] * X [i,j ]
        Y [j] := soma;
        end;
end;
```

a) Corrija o procedimento apresentado.

b) Escreva o programa completo utilizando procedimentos de leitura de uma matriz e de um vector e de escrita de um vector.

29) Considere um vector de inteiros X e escreva um programa em linguagem Pascal, utilizando programação estruturada, para calcular:

i) O produto das suas componentes.
ii) O valor máximo e mínimo das suas componentes.
iii) A norma euclidiana do vector X.

30) Considere uma matriz A não quadrada, e escreva um programa em linguagem Pascal que crie um vector tal que cada componente é igual ao máximo da linha correspondente da matriz A; o programa deve ser construído utilizando um procedimento para ler uma matriz, um procedimento para escrever um vector e uma função que calcule o máximo de um vector.

31) Suponha que queria escrever um programa em linguagem Pascal para efectuar o produto de uma matriz por um vector utilizando uma função que calcula o produto interno de dois vectores

a) Escreva uma função para calcular o produto interno de dois vectores.

b) Escreva o programa completo utilizando procedimentos de leitura de uma matriz e de um vector e de escrita de um vector, e também a função da alínea anterior para efectuar o produto interno de cada linha da matriz pelo vector.

32) Escreva um programa em linguagem Pascal que crie e escreva uma matriz A com m linhas e n colunas em que todos os coeficientes da matriz são nulos excepto os da primeira e última linha e os da primeira e última coluna (note que a matriz não é lida, tem de ser criada).

$$A = \begin{bmatrix} 1 & 1 & 1 & ... & ... & 1 & 1 \\ 1 & 0 & 0 & ... & ... & 0 & 1 \\ ... & ... & ... & ... & ... & ... & ... \\ 1 & 0 & 0 & ... & ... & 0 & 1 \\ 1 & 1 & 1 & ... & ... & 1 & 1 \end{bmatrix}$$

33) Chama-se número de Amstrong a um número que é igual à soma dos cubos dos seus algarismos. Escreva um algoritmo para determinar todos os números de Amstrong menores que 10 000.

Exemplo: $153 = 1^3 + 5^3 + 3^3$

34) Diga justificando convenientemente qual a diferença entre tabelas e fichas.

35) a) Escreva um programa em linguagem Pascal, usando programação estruturada, que determine a solução de um sistema A x = b cuja matriz dos coeficientes é uma matriz triangular inferior(nxn), em que a componente x_i é dada por:

$$x_i = \frac{\left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j \right)}{a_{ii}} \quad \text{para i} = 1,..n.$$

b) Utilizando os procedimentos da alínea anterior escreva um programa em linguagem Pascal que calcule a inversa de uma matriz A triangular inferior.

Sugestão: terá que resolver n sistemas considerando sucessivamente para o vector b:

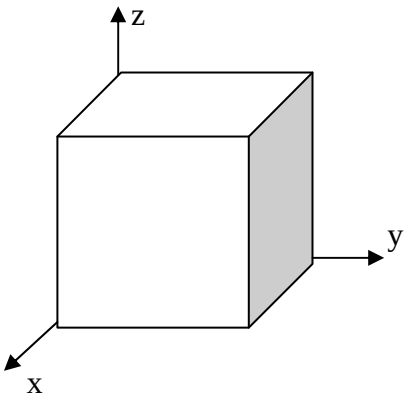
$$b_1 = \begin{bmatrix} 1 \\ 0 \\ . \\ . \\ . \\ 0 \\ 0 \end{bmatrix} \qquad b_2 = \begin{bmatrix} 0 \\ 1 \\ . \\ . \\ . \\ 0 \\ 0 \end{bmatrix} \qquad .... \qquad b_{n-1} = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 1 \\ 0 \end{bmatrix} \qquad b_n = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 0 \\ 1 \end{bmatrix}$$

As soluções obtidas são as colunas correspondentes da matriz inversa .

36) Escreva um programa em linguagem Pascal que anule todos os elementos positivos de uma matriz.

37) Dada uma matriz quadrada A (n x n) e um vector X com n componentes, escreva um programa em linguagem Pascal, usando programação estruturada, que calcule o produto X^T A X ; suponha que já existem os procedimentos de leitura de uma matriz e de leitura e escrita de um vector.

38) Mediu-se a temperatura nas faces de um cubo de alumínio instrumentado para o efeito. Sabendo que o comprimento da aresta do referido cubo é igual a 40 mm e que as medidas foram efectuadas em toda a superfície da peça de 10 em 10 mm definiu-se um tensor A(i,j,k) para i,j,k = 1, ... ,5 onde cada elemento representa a temperatura no ponto de coordenadas x, y e z. Escreva um programa em linguagem Pascal, utilizando programação estruturada, que calcule a temperatura máxima, mínima e média para a face assinalada na figura.



39) Implemente um programa em linguagem Pascal capaz de converter todas as letras minúsculas que figuram numa cadeia de caracteres, ou string, para maiúsculas.

Para tal deverá utilizar (não construir!): a função **"function Length(s:string): integer;"** que devolve a quantidade de caracteres presentes na string **s**; a função "**function ord(c:char):integer;"** que devolve o número de ordem de um caracter e a função "**function chr(o:integer):char**;" que devolve o caracter correspondente ao número de ordem dado. Por último, relembre que a sub-tabela de códigos ASCII que establece a relação entre letras e números de ordem é a seguinte:

Número de Ordem	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84
Caracter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

Número de Ordem	85	86	87	88	89	90	97	98	99	100	101	102	103	104	105	106	107
Caracter	U	V	W	X	Y	Z	a	b	c	d	e	f	g	h	i	j	k

Número de Ordem	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122
Caracter	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

40) Escreva o número 301,3 na base dois, num sistema de vírgula flutuante com 13 bits para a mantissa e 6 bits para o expoente.

41) a) Escreva um algoritmo para passar um número inteiro da base 10 para a base 2.

b) Percorra o algoritmo para o número x = 252.

42) Escreva o número 51.45 na base dois, num sistema de vírgula flutuante com 10 bits para a mantissa e 6 bits para o expoente.

ANEXO C

EXERCÍCIOS RESOLVIDOS

1) Considere o algoritmo apresentado que efectua as seguintes tarefas:

Objectivo:

1. Criação de um vector B cujas componentes são iguais à norma euclidiana de cada linha de uma matriz A;
2. Determinação do número de elementos da matriz A que são simultaneamente múltiplos de 3 e de 5;

Variáveis dados: m, n, inteiros, dimensões da matriz;
A, bi-indexada de inteiros , matriz dada;
Variáveis auxiliares: i, j, inteiros, contadores;
resto3, resto5, inteiros, restos da divisão inteira;
norma, inteiro, quadrado da norma de cada linha da matriz A;
Variáveis resultado: B, indexada de inteiros, vector calculado;
mult, inteiro, número de elementos da matriz que são simultaneamente múltiplos de 3 e de 5;

```
escreva(" introduza a dimensão da matriz (mxn) ");
leia( m,n);

para i ← 1 até m faça
        para j ← 1 até n faça

            { escreva (" introduza a componente ", i , j ,  " da matriz ");
              leia (A[i,j]);

para i ← 1 até m faça

        { norma ← 0;
          para j ← 1 até n faça
                norma ← norma +A [i,j] * A [i,j];
          B [i] ← sqrt (norma);

mult ← 0;
para i ← 1 até m faça
     para j ← 1 até n faça

          { resto3 ← A [i,j] – A [i,j] "/" 3 * 3;
            resto5 ← A [i,j] – A [i,j] "/" 5 * 5;
            se (resto3 = 0)  e (resto5 = 0)    então    mult ← mult + 1;
```

```
escreva (“ vector B criado”);


para i ← 1 até m faça
    escreva (B[i]);


escreva ( “ número de elementos da matriz que são simultaneamente múltiplos de 3 e
            de 5 é igual a “, mult) ;
```

a) Escreva um programa em linguagem Pascal (não estruturada);
b) Escreva um programa em linguagem Pascal estruturada utilizando procedimentos de leitura de uma matriz, de escrita de um vector, um procedimento para a tarefa 1 e uma função para a tarefa 2.

Program1a;

```
type matriz = array [1..20,1..20]of integer;
type vector = array [1..20] of real;
var m, n, i, j, resto3, resto5, mult, norma : integer;
    A : matriz;
    B : vector;

begin
   writeln (' introduza a dimensão da matriz (mxn) ');
    readln ( m,n);

    for i := 1 to m do
        for j := 1 to n do
            begin
                writeln (' introduza a componente ', i , j ,  ' da matriz ');
                 readln (A[i,j]);
            end;


    for i := 1 to m do
        begin
            norma := 0;
            for j := 1 to n do
                norma := norma +A [i,j] * A [i,j];
            B [i] := sqrt (norma);
        end;
```

```
    mult := 0;
    for i := 1 to m do
        for j := 1 to n do
            begin
                resto3 := A [i,j] – A [i,j] div 3 * 3;
                resto5 := A [i,j] – A [i,j] div 5 * 5;
                if (resto3 = 0)  and (resto5 = 0)    then    mult := mult + 1;
            end;

    writeln ('vector B criado');
    for i := 1 to m do
        writeln (B[i]);

    writeln ( ' número de elementos da matriz que são simultaneamente múltiplos de 3 e
                de 5 é igual a ', mult);

end.
```

Program1b;

```
type matriz = array [1..20,1..20] of integer;
type vector = array [1..20] of real;
var m, n, mult : integer;
     A : matriz;
     B : vector;

Procedure lermat (var M: matriz; m,n: integer);
var i, j: integer;
begin
    for i := 1 to m do
        for j := 1 to n do
            begin
                writeln (' introduza a componente ', i , j ,  ' da matriz ');
                readln (M[i,j]);
            end;
end;

Procedure normv (M: matriz; var X: vector; m,n: integer);
var i, j, norma; integer;
begin
    for i := 1 to m do
        begin
            norma := 0;
            for j := 1 to n do
                norma := norma +M [i,j] * M [i,j];
            X [i] := sqrt (norma);
        end;
end;
```

```
Function multiplo ( M: matriz; m,n: integer): integer;
var i, j, mult,resto3,resto5 :integer
    mult := 0;
    for i := 1 to m do
        for j := 1 to n do
            begin
                resto3 := M [i,j] – M [i,j] div 3 * 3;
                resto5 := M [i,j] – M [i,j] div 5 * 5;
                if (resto3 = 0)  and (resto5 = 0)    then    mult := mult + 1;
            end;
    multiplo :=mult;
end;

Procedure escvec (B: vector; m:integer);
Var i: integer;
begin
    writeln ('vector B criado');
    for i := 1 to m do
        writeln (B[i]);
end;

begin
    writeln (' introduza a dimensão da matriz (mxn) ');
    readln ( m,n);
    lermat (A,m,n);
    normv (A,B,m,n);
    escvec (B,m);
    mult := multiplo (A,m,n)
    writeln ( ' número de elementos da matriz que são simultaneamente múltiplos de 3 e
                de 5 é igual a ', mult);
end.
```

2) Considere o algoritmo apresentado que efectua as seguintes tarefas:

Objectivo:

1. Criação de um vector B cujas componentes são iguais à norma euclidiana de cada linha de uma matriz A;
2. Determinação do elemento máximo de uma matriz e da posição respectiva ou seja indicação da linha e coluna;
3. Troca do elemento máximo da matriz encontrada no ponto 2 com o elemento mínimo da coluna onde se encontra esse elemento máximo;
4. Determinação do traço da matriz A transformada se ela for quadrada;

Variáveis dados: m, n, inteiros, dimensões da matriz e do vector;
A, bi-indexada de reais , matriz dada;
Variáveis auxiliares: i, j, inteiros, contadores;
norma, real, norma de cada linha da matriz;
min, real ,elemento mínimo da matriz;
imin, inteiro, índice da linha do elemento mínimo da matriz;

Variáveis resultado: B, indexada de reais, vector calculado;
max, real ,elemento máximo da matriz;
imax, jmax, inteiros, índices do elemento máximo da matriz;
traco, real, traço da matriz;

```
escreva(" introduza a dimensão da matriz (mxn) ");
leia(m, n);

para i ← 1 até m faça
    para j ← 1 até n faça
        escreva (" introduza a componente ", i , j , " da matriz ");
        leia (A[i,j]);

para i ← 1 até m faça
    norma ← 0.0;
    para j ← 1 até n faça
        norma ← norma +A [i,j] * A [i,j];
    B [i] ← sqrt (norma);

max ←A [1,1];  imax ← 1;  jmax ← 1;
para i ← 1 até m faça
    para j ← 1 até n faça
        se A [i,j] > max então
            max ← A [i,j];
            imax ← i;
            jmax ← j;
```

```
min ← A [1,jmax]; imin ← 1;
para i ← 1 até m faça
      se A [i,jmax]< min então { min ← A [i,jmax];
                                 imin ← i;

A [imax, jmax] ← min;
A [imin, jmax] ← max;


se  m = n  então

      { traco ← 0.0;
        para i ← 1 até n faça
              traco ← traco +A [i,i];


escreva (" Vector resultado" );
para i ← 1 até m faça
       escreva ( B[i]);


escreva (" matriz A transformada");
para i ← 1 até m faça
     { para j ← 1 até n faça
            escreva (A[i,j]);


escreva ( " O elemento máximo da matriz é o elemento A[",imax,",",jmax,"]=",
            max);

se m = n  então   escreva ( " O traço da matriz é igual a ", traco);
```

a) Escreva um programa em linguagem Pascal (não estruturada);

b) Escreva um programa em linguagem Pascal estruturada utilizando procedimentos de leitura e escrita de uma matriz, escrita de um vector, um procedimento para cada uma das tarefas 1, 2, e 3 e uma função para o cálculo do traço da matriz (tarefa 4).

3) Considere o algoritmo apresentado que efectua as seguintes tarefas:

Objectivo:

1. Produto de uma matriz A por uma matriz B;
2. Determinação do número de elementos positivos e não positivos da matriz A;
3. Criação de um vector D cujas componentes são iguais à soma dos elementos positivos de cada linha da matriz A;
4. Determinação do produto dos elementos da diagonal principal da matriz B se ela for quadrada;

Variáveis dados: m, n, p , inteiros, dimensões das matrizes e do vector;
A, B , bi-indexada de reais , matrizes dadas;
Variáveis auxiliares: i, j, k, inteiros, contadores;
soma, real, somatório;

Variáveis resultado: C, bi-indexada de reais, matriz resultado;
D, indexada de reais, vector calculado;
positivo, inteiro ,número de elementos positivos da matriz A;
npos, inteiro ,número de elementos não positivos de matriz A;
prod, real, produto dos elementos da diagonal principal de B;

```
escreva(" introduza a dimensão da matriz A (mxn) ");
leia(m, n);
escreva(" introduza o nº de colunas da matriz B (nxp) ");
leia(p);

para i ← 1 até m faça
        para j ← 1 até n faça
                { escreva (" introduza a componente ", i , j ,  " da matriz A ");
                  leia (A[i,j]);

para i ← 1 até n faça
        para j ← 1 até p faça
                { escreva (" introduza a componente ", i , j ,  " da matriz B ");
                  leia (B[i,j]);

para i ← 1 até m faça
        para j ← 1 até p faça
                { soma ← 0.0;
                  para k ← 1 até n faça
                        soma ← soma + A [i,k] *B [k,j];
                  C [i,j] ← soma;
```

```
positivo ← 0; npos ← 0;
para i ← 1 até m faça
      para j ← 1 até n faça
            se A [i,j] > 0 então positivo ← positivo + 1;
npos ← m*n – positivo;

para i ← 1 até m faça

   { soma ← 0.0;
     para j ← 1 até n faça
           se A [i,j] > 0.0 então       soma ← soma + A [i,j];
     D [i] ← soma;

se  n = p  então

   { prod ← 1.0;
     para i ← 1 até n faça
           prod ← prod *B [i,i];

escreva (“ matriz  C = A x B ”);
para i ← 1 até m faça
      para j ← 1 até p faça
            escreva (C[i,j]);

escreva (“ Vector resultado” );
para i ← 1 até m faça
     escreva ( D[i]);

escreva ( “ O nº de elementos positivos da matriz é igual a ”,positivo, “ e o nº de
            elementos não positivos é “, npos);

se n = p então   escreva ( “ O produto dos elementos da diagonal principal  da matriz
                                B é igual  a ”, prod);
```

a) Escreva um programa em linguagem Pascal (não estruturada);

b) Escreva um programa em linguagem Pascal estruturada utilizando procedimentos de leitura e escrita de uma matriz, escrita de um vector, um procedimento para cada uma das tarefas 1, 2, e 3 e uma função para o cálculo do produto dos elementos da diagonal principal da matriz (tarefa 4).

4) Considere o algoritmo apresentado que efectua as seguintes tarefas:

Objectivo:

1. Determinação da transposta de uma matriz quadrada A de inteiros;
2. Determinação do número de elementos pares e de elementos negativos da matriz A;
3. Criação de uma matriz B (mxp) cujos elementos são iguais à soma dos respectivos índices i + j, excepto os elementos da diagonal principal que são iguais ao número de elementos negativos de A;
4. Determinação do produto dos elementos da diagonal não principal da matriz A ;

Variáveis dados: m, n, p, inteiros, dimensões da matrizes;
A, bi-indexada de inteiros , matriz dada;
Variáveis auxiliares: i, j, inteiros, contadores;
resto, inteiro, resto da divisão inteira;
aux, inteiro, variável auxiliar;
Variáveis resultado: C, bi-indexada de inteiros, matriz calculada;
pares, inteiro ,número de elementos pares da matriz;
negativo, inteiro ,número de elementos negativos da matriz;
prod, inteiro, produto dos elementos da diagonal não principal de A;

```
Escreva (" introduza a dimensão da matriz quadrada (nxn) ");
leia( n);

para i ← 1 até n faça
      para j ← 1 até n faça

        { escreva (" introduza a componente ", i , j ,  " da matriz A ");
          leia (A[i,j]);


para i ← 1 até n faça
     para j ← i+1  até n faça

       { aux ← A [i,j];
         A [i,j] ← A [j,i];
         A [j,i] ← aux;


pares ← 0; negativo ← 0;
para i ← 1 até n faça
     para j ← 1 até n faça

       { resto ← A [i,j] – A [i,j] "/" 2 * 2;
         se resto = 0 então  pares ← pares + 1;
         se A [i,j] < 0 então   negativo ← negativo +1;
```

```
escreva (“ introduza a dimensão da matriz B (mxp) “);
leia(m, p);
para i ← 1 até m faça
    para j ← 1 até p faça
        se i = j então B[i,j] ← negativo
                senão B [i,j] ← i + j;

prod ← 1;
para i ← 1 até n faça

        { j ← n-i+1;
          prod ← prod * A [i,j];

escreva (“ matriz A transposta”);
para i ← 1 até n faça
       para j ← 1 até n faça
            escreva (A[i,j]);


escreva (“ matriz B criada”);
para i ← 1 até m faça
       para j ← 1 até p faça
            escreva (B[i,j]);


escreva ( “ número de elementos pares da matriz é igual a “, pares, “e o número de
            elementos negativos é “,negativo);

escreva ( “ O produto dos elementos da diagonal não principal  da matriz A é
            igual a ”, prod );
```

a) Escreva um programa em linguagem Pascal (não estruturada);
b) Escreva um programa em linguagem Pascal estruturada utilizando procedimentos de leitura e escrita de uma matriz, um procedimento para cada uma das tarefas 1, 2, e 3 e uma função para o cálculo do produto dos elementos da diagonal não principal da matriz A (tarefa 4).

6) Considere o algoritmo apresentado que efectua as seguintes tarefas:

Objectivo:

1. Produto de uma matriz de inteiros A por uma matriz B;
2. Determinação do número de elementos da matriz A que são múltiplos de 3 e da linha e coluna ocupada pelo ultimo múltiplo de 3;
3. Troca da linha s com a linha q da matriz A;
4. Determinação do número de elementos nulos da matriz A ;

Variáveis dados: m, n, p , inteiros, dimensões das matrizes;
A, bi-indexada de inteiros , matrizes dadas;
B , bi-indexada de reais , matrizes dadas;
s, q, inteiros, números das linhas a trocar;
Variáveis auxiliares: i, j, k, inteiros, contadores;
soma, real, somatório;

Variáveis resultado: C, bi-indexada de reais, matriz resultado (A x B);
nmult, inteiro, número de múltiplos de 3;
imult, inteiro, linha da matriz A onde se encontra o último mult3;
jmult, inteiro, coluna da matriz A onde se encontra o último mult3;
nulos, inteiro ,número de elementos nulos da matriz A;

```
escreva(" introduza a dimensão da matriz A (mxn) ");  leia (m, n);
escreva(" introduza o nº de colunas da matriz B (nxp) "); leia (p);
escreva (" introduza o nº das linhas atrocar"); leia (s,q);

para i ← 1 até m faça
        para j ← 1 até n faça

                { escreva (" introduza a componente ", i , j ,  " da matriz ");
                  leia (A[i,j]);

para i ← 1 até n faça
        para j ← 1 até p faça

                { escreva (" introduza a componente ", i , j ,  " da matriz ");
                  leia (B[i,j]);

para i ← 1 até m faça
        para j ← 1 até p faça

                { soma ← 0.0;
                  para k ← 1 até n faça
                        soma ← soma + A [i,k] *B [k,j];
                  C [i,j] ← soma;
```

```
imult ← 0; jmult ← 0; nmult ← 0;
para i ← 1 até m faça
        para j ← 1 até n faça

                se A [i,j] – A[i,j] "/" 3*3 = 0 então   nmult ← nmult +1
                 imult ← i;
                 jmult ← j;


para j ← 1 até n faça

        aux ← A[s,j];
        A[s,j] ← A[q,j];
        A[q,j] ← aux;

nulos  ← 0;
para i ← 1 até m faça
       para j ← 1 até n faça
            se A [i,j] = 0  então   nulos ← nulos + 1;

escreva (" matriz  A transformada ");
para i ← 1 até m faça
        para j ← 1 até n faça
              escreva (A[i,j]);

escreva (" matriz  C = A x B ");
para i ← 1 até m faça
        para j ← 1 até p faça
              escreva (C[i,j]);

escreva ( " O nº de elementos da matriz que são múltiplos de 3 é ", nmult ," e
              o último aparece na linha",  imult," e na coluna ",jmult, ". O nº de
              elementos nulos é ", nulos);
```

a) Escreva um programa em linguagem Pascal (não estruturada);

b) Escreva um programa em linguagem Pascal estruturada utilizando procedimentos de leitura e escrita de uma matriz, um procedimento para cada uma das tarefas 1, 2, e 3 e uma função para o cálculo do número de elementos nulos da matriz A (tarefa 4).