



Aula 1

Professora

Drucilla do Bem Oliveira

drucoliveira@hotmail.com

1. INTRODUÇÃO

LINGUAGEM DE PROGRAMAÇÃO

A programação é a arte de fazer com que o computador faça exatamente o que desejamos que ele faça.

Ao nível mais simples consistirá em enviar uma seqüência de comandos para um computador por forma a atingir um determinado objetivo.

Um programa de computador é simplesmente um conjunto de instruções para dizer ao computador como executar uma determinada tarefa em particular. Tal como uma receita culinária: que diz ao cozinheiro/a como se deve fazer tal prato em particular. E descreve os ingredientes (data) e os passos a seguir (o processo) para converter tais ingredientes numa sopa, ou outra coisa qualquer que se esteja a cozinhar. os programa seguem o mesmo tipo de conceito

As Linguagens de programação:

- São linguagens para a comunicação **HOMEM → MÁQUINA**.
- Softwares que permitem o desenvolvimento de programas

Uma linguagem de Programação é um sistema para comunicação com o computar em uma forma que ambos, computador e programador, possam entender.

Os três tipo de linguagem de programação são:

- a) Linguagem de Máquina (Absoluta): È a linguagem que antende ao computador, por satisfazer o seu projeto lógico.

As instruções têm um formato especial representadas por códigos binários que quando interpretados pelos circuitos do computador enviam ordens de execução das operações

Ex: 0001	0000	0000	1010	0011	0101
1	0	0	A	3	3
Cód. de operação	Campo de operando				

10 → é o código da operação de soma

0A35 → endereço de um dos operando a ser utilizado na soma

- b) Linguagem Simbólica (de baixo nível, Mnemônico ou de montagem)

Essa linguagem surgiu a fim de simplificar a difícil programação da linguagem de máquina. Os códigos binários aqui são substituídos por siglas que empregam abreviações de nomes sugestivos que lembram a função da instrução.

Ex: ADD X

ADD ordena que seja somado o valor da variável X ao acumulador.

A tradução de um programa feito em linguagem simbólica para linguagem de máquina é feita por outro programa chamado montador simbólico.

c) Linguagem Automático ou de Alto nível

São geralmente semelhantes as linguagens usadas para descrever o problema que se deseja resolver. Ex: Algol, Pascal..

A tradução de um programa feito em linguagem de alto nível (programa fonte) para linguagem de máquina é feita por outro programa chamado compilador.

UM POUCO DE HISTÓRIA

Da mesma forma que falas com um amigo numa determinada língua, é a mesma forma como falas com um computador, usando a língua do computador. A única linguagem que um computador entende é a *linguagem binária*. O código binário é, infelizmente, muito difícil para os humanos lerem e escreverem, por isso somos "obrigados" a usar uma linguagem intermédia que mais tarde será traduzida para linguagem binária.

De uma forma muito inovadora, decidiu-se chamar o tradutor da linguagem intermediária para linguagem binária de interprete. Da mesma forma que normalmente se necessita de um interprete para traduzir de Russo para Inglês e de outro para traduzir de Chinês para Russo (por exemplo), também precisamos de interpretes diferentes para traduzir o Pascal para linguagem binária e outro para traduzir de Basic para linguagem binária.

Os primeiros programadores tinham que introduzir linguagem binária diretamente, e isso é muito difícil. Então o passo seguinte foi criar um tradutor que simplesmente traduziria uma língua comum (digamos o inglês) para a linguagem binária. Isso fez com que as coisas ficassem muito mais simples, então em vez de estarmos a massacrar a memória tentando lembrar que o código 001273 05 05 significa adicionar 5 a 4, os programadores por e simplesmente escreveriam `add 5 4`. Este simples avanço facilitou a vida a toda a gente. Estes códigos de sistema foram na realidade, as primeiras linguagens de programação, um para cada tipo de computador. Esta linguagem passou a ser conhecida como programação por *assembler*, hoje em dia ainda é utilizada em algumas tarefas muito específicas.

Embora tudo isto pareça um pouco primitivo, porque trata-se de dizer ao computador o que esta a acontecer ao nível do *hardware* - mover estes bytes desta localização da memória para uma outra localização etc. Era de qualquer das formas muito difícil e exigia muita programação e esforço para conseguir cumprir tarefas aparentemente simples.

Gradualmente os cientistas computacionais desenvolveram outras linguagens de mais alto nível para facilitar as suas vidas. Tudo isto faz com que a programação se torne numa coisa muito interessante, mas também faz que você como programador, tenha que compreender não só os conceitos da programação, mas também as vantagens e desvantagens em fazê-la numa determinada linguagem.

Evolução das linguagens

1957 - FORTRAN - problemas numéricos

1977 - FORTRAN 77 - conceitos de programação estruturada

1958 - ALGOL - processamento científico

1959 - COBOL - aplicações comerciais

1965 - BASIC - fácil de ser aprendida

1970 - PASCAL - programação estruturada feita por Niklaus Wirth outras linguagens:

1972 - C

1983 - Linguagem C++ e OO, uma extensão da linguagem C.

1995 - Linguagem Java

Outras linguagens de programação: Visual Basic, Delphi, C++ Builder, Visual C++...

2. ALGORITMOS

Algoritmo é uma seqüência de instruções ordenadas de forma lógica para a resolução de uma determinada tarefa ou problema. O algoritmo não é a solução de **um** problema, pois, se assim fosse, cada problema teria **um** único algoritmo. Sendo assim, algoritmo é um caminho para a solução de um problema, e em geral, os caminhos que levam a uma solução são muitas.

O aprendizado de algoritmos não se consegue a não ser através de muitos exercícios.

Algoritmos não se aprende:

- Copiando Algoritmos
- Estudando Algoritmos

Algoritmos só se aprendem:

- Construindo Algoritmos
- Testando Algoritmos

FASES DE UM ALGORITMO

Quando temos um problema e vamos utilizar um computador para resolvê-lo inevitavelmente temos que passar pelas seguintes etapas:

- a) Definir o problema.
- b) Realizar um estudo da situação atual e verificar quais a(s) forma(s) de resolver o problema.
- c) Terminada a fase de estudo, utilizar uma linguagem de programação para escrever o programa que deverá a princípio, resolver o problema.
- d) Analisar junto aos usuários se o problema foi resolvido. Se a solução não foi encontrada, deverá ser retornado para a fase de estudo para descobrir onde está a falha.

Como mencionado anteriormente programar um computador consiste em elaborar um conjunto finito de instruções, reconhecidas pela máquina, de forma que o computador execute estas instruções. Estas instruções possuem regras e uma sintaxe própria, como uma linguagem tipo português ou inglês, sendo isto chamadas de linguagem de computador.

No mundo computacional existe uma grande variedade de linguagens Pascal, C, C++, Cobol, Fortran, etc..... Nós iremos focar uma delas, o Pascal.

Para construir um algoritmo temos que estruturar o problema seguindo uma seqüência lógica e orientada de operações. Vamos ver o exemplo das operações da construção de uma casa: Vamos supor que as operações disponíveis neste caso são:

- Instalar portas e janelas
- Levantar paredes
- Construir telhado

Neste caso uma seqüência possível seria

1. Levantar paredes
2. Construir telhado
3. Instalar portas e janelas

Observe que o algoritmo não é único

1. Levantar paredes
2. Instalar portas e janelas
3. Construir telhado

A opção construir telhado não precisa ser a primeira, isto é lógico.

Outro exemplo trocar uma lâmpada queimada. Vamos supor que as operações disponíveis são:

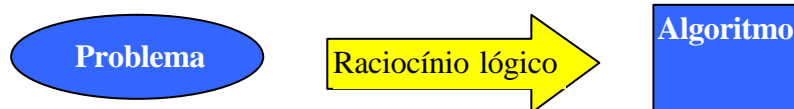
- buscar uma escada.
- subir na escada
- descer da escada
- posicionar a escada
- buscar uma lâmpada
- retirar a lâmpada velha
- colocar a lâmpada nova

O que pode ser um algoritmo possível.

1. buscar uma escada.
2. posicionar a escada
3. buscar uma lâmpada nova
4. subir na escada
5. retirar a lâmpada velha
6. colocar a lâmpada nova
7. descer da escada

Observe que a operação "subir na escada" não pode vir ante da operação "buscar uma escada"
Existe outro algoritmo possível?

Portanto para construir um algoritmo é preciso usar o raciocínio lógico.



ESTRUTURA DE ALGORITMOS

Antes de utilizarmos uma linguagem de computador, é necessário organizar as ações a serem tomadas pela máquina de forma organizada e lógica, sem nos atermos as regras rígidas da Sintaxe de uma linguagem. Para isto utilizaremos uma forma de escrever tais ações, conhecida como algoritmo ou pseudocódigo.

Os algoritmos terão a seguinte estrutura:

```
ALGORITMO <Nome do algoritmo>
<definições>
INÍCIO
    <Comandos>
FIM
```

3. VARIÁVEIS

Os valores no texto estático de um algoritmo podem ser representados na forma de constante ou na forma de variável

Constante → O valor é representado diretamente

Ex: 3, -1, verdadeiro

Variável → representado no texto do algoritmo por um nome que corresponde a uma posição da memória que contém seu valor.

Uma variável é formado por uma letra ou então por uma letra seguida de letras ou dígitos, em qualquer número. Não é permitido o uso de espaços em branco ou de qualquer outro caractere, que não seja letra ou dígito, na formação de um identificador.

Na formação do nome da variável de um nome significativo, para que se possa ter idéia do seu conteúdo sem abri-lá. Se utilizar palavras para compor o nome da variável utilize o “_” underline para separar as palavras.

Ex: Variável nome_aluno → guarda o nome do aluno

DECLARAÇÃO DA VARIÁVEL

Sintaxe: Tipo variável : nome_variável;

Exemplo: **int** : cep;

Em Pascal:

Sintaxe: nome_variável : tipo

TIPOS BÁSICOS

Todo valor (constante ou variável) de um programa tem um tipo de dado associado que determina um conjunto de objetos e um conjunto de operações

Existem quatro tipos básicos:

- inteiro ou int - Domínio Z

Operações: +, -, *, /, **, div, mod, >, ≥, <, ≤, = e ≠

Ex: 11 div 3 → 3

11 mod 3 → 2

- real → Domínio R

Operações: +, -, *, /, **, div, mod, >, ≥, <, ≤, = e ≠

- Carácter ou char - Domínio conjunto de caracteres alfa numéricos (A, B, ...Z, a, b, ...z, 0,1,...9, +,-, *etc)

Operações: >, ≥, <, ≤, = e ≠

- Logico ou log - falso e verdadeiro

4. COMANDOS BÁSICOS

COMANDO DE ATRIBUIÇÃO

O comando de atribuição é usado para atribuir o resultado de uma expressão à variável ou um valor qualquer que deseje que a variável receba. A sintaxe do comando é: variável ← expressão; (algoritmo). Em pascal usamos: variável := expressão. Por exemplo:

soma ← num1 + num2 (Pascal: soma := num1 + num2).

Neste exemplo estamos atribuindo a variável soma o valor resultante da soma da variável num1 com a variável num2.

É importante observar que o resultado da expressão do lado direito de um comando de atribuição deve ser coerente como tipo declarado para a variável do lado esquerdo. Por exemplo, o comando de atribuição: $x \leftarrow a < b$ só tem sentido se x é do tipo lógico.

COMANDOS DE ENTRADA E SAÍDA

Os comandos que iremos ver são os comandos LEIA e IMPRIMA, respectivamente, comando de entrada e de Saída (em Pascal escreve-se READ e WRITE).

Sintaxe: leia (var₁, var₂, ...var_N);
 Imprima ("texto", var₁, var₂,... var_N);

Por exemplo:

leia (A, X, LETRA) → serão lidos os valores das variáveis A,X,LETRA nesta ordem

obs.: As variáveis listadas no comando e os dados fornecidos pela unidade de entrada têm que concordar em ordem , número e tipo.

Exemplo comando de saída:

```
imprima ("Valor lido para N= ", N, "Fatorial de N = ", Fat);
```

Apresenta em uma unidade de saída a mensagem: VALOR LIDO PARA N =, o valor da variável N, seguido da mensagem, FATORIAL DE N = , e o valor da variável Fat.

Exemplo 1: Escrever um algoritmo para ler 2 valores numéricos do teclado e atribuí-lo a uma variável do tipo numérica.

```
ALGORITMO Leia  
VARIÁVEIS  
    int: num1, num2;  
INICIO
```

```
imprima ("Digite o primeiro número")
leia(num1);
imprima ("Digite o segundo número")
leia(num2);
imprima(num1, num2);
```

FIM.

Obs.: Em Pascal, quando usamos os comandos READ e WRITE, o cursor continuará posicionado na mesma linha. Para avançar o cursor para a próxima linha acrescentamos as letras LN no final dos comandos READ e WRITE. Desta forma os comandos terão a seguinte grafia: READLN e WRITELN.

5. OPERADORES

OPERADORES ARITMÉTICOS

Os operadores aritméticos são os seguintes: Normalmente utilizado em operações entre valores numéricos inteiros ou reais.

Adição +; Subtração -; Multiplicação *; Divisão /; Divisão inteira: DIV (QUOCIENTE); Resto da divisão: MOD (RESTO).

OPERADORES RELACIONAIS:

Podem ser usados com todos os tipos, normalmente, são usados em expressões condicionais, retornando sempre um resultado lógico TRUE ou FALSE.

Igual =; Diferença <>; Maior que >; Menor que <; Menor Igual a <=; Maior Igual a >=;

OPERADORES LÓGICOS

Os operadores lógicos realizam as operações da álgebra booleana. Os operadores são os seguintes:

AND (E) - somente resulta em verdadeiro se ambas operações forem verdadeiras

OR (OU) - é verdadeiro quando pelo menos um dos operandos for verdadeiro

NOT (NÃO) - inverte o resultado de uma expressão lógica

XOR (NÃO OU) - exclusivo or. só é verdadeiro se apenas um dos operandos for verdadeiro

Exemplo:

a) Operador E (AND):

```
TRUE AND TRUE  => TRUE
TRUE AND FALSE => FALSE
FALSE AND TRUE => FALSE
FALSE AND FALSE => FALSE
```

b) Operador OU (OR):

```
TRUE OR TRUE   => TRUE
TRUE OR FALSE  => TRUE
FALSE OR TRUE  => TRUE
FALSE OR FALSE => FALSE
```

c) Operador NÃO (NOT):

NOT TRUE => FALSE
 NOT FALSE => TRUE

d) Operador NÃO OU (XOR):

TRUE XOR TRUE => FALSE
 TRUE XOR FALSE => TRUE
 FALSE XOR TRUE => TRUE
 FALSE XOR FALSE => FALSE

PRECEDÊNCIA DE OPERADORES

1º) expressões dentro de parênteses

2º) Expressões aritméticas

- 1º) Funções,
- 2º) **,
- 3º) * e / e 4º) + e -.

(3º) Comparações (<, ≤, >, ≥, =, ≠)

4º) não

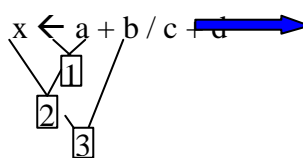
5º) e

6º) ou

Por exemplo: $x = \frac{a+b}{c+d}$

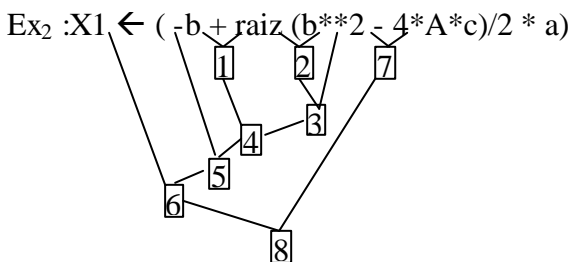
↓

$x = (a+b) / (c+d)$
(expressão correta)



$\xrightarrow{\text{a}}$

$a + \frac{b}{c} + d$
(expressão errada)



Exercício

1) Faça um algoritmo para ler a base e a altura de um triângulo. Em seguida, escreva a área do mesmo.

Obs.: Área = (Base * Altura) / 2

2) Faça um algoritmo que calcule a média aritmética de 4 valores inteiros.

DETALHANDO AS FASES PARA A CONSTRUÇÃO DO ALGORITMO

- 1) Leia cuidadosamente a especificação do problema até o final
- 2) Levantar e analisar todas as saídas exigidas na especificação do problema
- 3) Levantar e analisar todas as entradas citadas na especificação do problema
- 4) Levantar se é necessário gerar valores internamente ao algoritmo e levantar as variáveis possíveis e os valores iniciais de cada uma;
- 5) Levantar e analisar todas as transformações necessárias para, dadas as entradas e valores gerados internamente, produzir as saídas especificadas;
- 6) Testar cada passo do algoritmo
- 7) Construção do algoritmo.

Vamos construir o algoritmo do Exercício 1 acima, utilizando a metodologia indicada.

- 1) Primeira leitura
- 2) Segunda leitura: Anotações
Objetivo: Calcular a área do triângulo
- 3) Saída: área do triângulo
- 4) Entrada: Altura e base do triângulo
- 5) Variáveis necessárias:
inteiro: base;
inteiro: altura;
inteiro: area;
- 6) $area \leftarrow (base * altura)/2;$
- 7) Supondo que leia o valor 5, 6, respectivamente para a base e altura, teremos:
 $Area \leftarrow (5 * 6)/2;$
- 8) Construir o algoritmo.
Algoritmo triangulo
int: altura, base, area;
inicio
imprima ("Digite o valor da altura do triângulo");
leia (altura);
imprima ("Digite o valor da base do triângulo");
leia (base);
 $area \leftarrow (base * altura)/2;$
imprima ("A área do triângulo é: ", area);
fim.

6. ESTRUTURA DE CONTROLE

ESTRUTURA CONDICIONAL

ESTRUTURA SE... ENTÃO.. SENÃO (IF... THEN... ELSE).

Ocorre quando uma ação a ser executada depende do teste de uma condição

- *Simples:*

Sintaxe: se <condição>
então

```
C;  
fim-se;
```

Semântica: Quando o controle entra na estrutura é feito um teste na condição. Se a condição for verdadeira, então a ação C será executada e o controle sai da estrutura. Se a condição fornecer resultado falso o controle sai da estrutura sem executar a ação C.

```
Ex: Se a < 5  
    então  
    b ← a + 5;  
    fim-se;
```

```
Em Pascal: if a <5  
            then  
            b := a + 5;
```

- **Composta**

Sintaxe: se <condição>
 então
 C₁;
 else
 C₂;
 fim-se;

Semântica: Quando o controle entra na estrutura é feito um teste na condição. Se a condição for verdadeira, então a ação C₁ será executada e o controle sai da estrutura. Se a condição fornecer resultado falso o controle vai para o senão e executa a ação C₂ e sai da estrutura.

```
Ex: se A + B ≤ y ** 2  
    então  
    x ← y **2;  
    imprima (x, y);  
    senão  
    A ← b;  
    fim-se;
```

```
Em Pascal: if A + B ≤ y ** 2  
            then  
            Begin  
            x:= y **2;  
            writeln(x, y);  
            end  
            else  
            a:=b;
```

Exemplo: Dado dois valores A e B quaisquer, faça um algoritmo que imprima se A > B, ou A < B, ou A = B

Algoritmo maior
inteiro: a, b;

inicio

imprima ('digite os valores a e b');

leia (a,b);

se $a > b$

então

imprima ('a é maior que b');

senão

se $a < b$

então

imprima ('a é menor que b');

senão

imprima('a é igual a b');

fim-se;

fim-se;

fim.

ESTRUTURA ESCOLHA (CASE)

Sintaxe: escolha <valor>
 <opções> : <comandos>
 .
 .
 .
 <opções> : <comandos>
[else
 bloco;] opcional
fim_escolha

Em pascal:
case <valor> of
 <opções> : <comandos>;
 .
 .
 .
 <opções> : <comandos>;
[else
 <comandos>];
end;

Semântica: O comando escolha é um seletor de opções, executando apenas a opção que for igual à expressão. O comando else é opcional, quando colocado só é executado se nenhuma das opções forem válidas.

Ex₁: escolha cor

```
v": imprima ("vermelho");  
"a": imprima ("amarelo");  
"b": imprima ("branco");  
fim-escolha;
```

Em Pascal: CASE cor OF

```
"v": WRITELN ('vermelho');  
"a": WRITELN ('amarelo');  
"b": WRITELN ('branco');  
END;
```

Ex₂: escolha cor

```
"v": imprima ("vermelho");  
"a": imprima ("amarelo");  
"b": imprima ("branco");
```

SENAO

```
IMPRIMA("Nenhuma cor foi selecionada");  
fim-escolha;
```

Em Pascal: CASE cor OF

```
"v": WRITELN ('vermelho');  
"a": WRITELN ('amarelo');  
"b": WRITELN ('branco');  
ELSE  
WRITELN ('Nenhuma cor foi  
selecionada');  
END;
```

Obs.: O comando Escolha não aceita valores do tipo REAL e STRING.

7. ESTRUTURA DE REPETIÇÃO

ESTRUTURA DE REPETIÇÃO COM TESTE NO INÍCIO: ENQUANTO - FAÇA (WHILE-DO)

Sintaxe: enquanto <condição> faça
 C;
 fim_enquanto;

Semântica: Quando o controle entra na estrutura é feito um teste na condição. Se a condição for verdadeira o controle executa a ação C até encontrar o fim-enquanto. O controle então retorna ao início da estrutura e passa a repetir a seguinte seqüência.

1º) Testa se a condição é verdadeira

2º) Executa a ação até encontrar o fim-enquanto

3º) Retorna para o início da estrutura

Quando a condição fornecer resultado falso o controle sai da estrutura passando para o comando seguinte ao fim-enquanto.

Obs: 1º) Na ação C tem que existir pelo menos um comando que altere o estado de pelo menos uma das variáveis da condição para que ela se torne falsa num determinado momento das repetições e o controle possa sair da estrutura.

2º) Se 1º teste da condição for falso o controle sai da estrutura sem executar nenhuma vez a ação C.

Ex: Soma \leftarrow 0; i \leftarrow 1;

Enquanto i \leq 20 faça

 Leia (A);

 Soma \leftarrow soma + A;

 I \leftarrow i + 1;

Fim-enquanto;

While i \leq 20 do

 Begin

 Readln(a);

 Soma := soma + a;

 I := i + 1;

 End;

Em pascal: soma:=0; i:=1;

ESTRUTURA DE REPETIÇÃO COM TESTE NO FINAL REPITA-ATÉ (REPEAT UNTIL)

Sintaxe: Repita
<Comandos>;
Ate <condição>

Em Pascal: REPEAT
<Comandos>
UNTIL <Condição>

Semântica: A ação será re-executada até que a condição se torne verdadeira quando, então, o controle sai da estrutura.

Obs. 1ª) A ação C será executada pelo menos uma vez pois o teste da condição é feito no final da estrutura

2ª) Pelo menos um dos comandos a serem repetidos deve alterar o estado de pelo menos uma das variáveis da condição para que ela se torne verdadeira em um determinado momento da execução e o controle possa sair da estrutura

3ª) Essa estrutura equivale a

<comando>;
enquanto não <condição> faça
<comandos>;
fim_enquanto;

Exemplo: Desenvolver um programa para ler 500 valores inteiros e positivos, calcular e imprimir todos os valores pares lidos e a soma dos valores ímpares.

Algoritmo n500

int: num, soma,i;

inicio

c ← 0;

imprima('Digite um número inteiro e positivo');

repita

leia(num);

se n mod 2 = 0

então

imprima('O número é par', num);

else

soma ← soma_num;

fim_se;

c ← c+1;

ate c = 500;

imprima('soma',soma);

fim.

Exemplo: Fazer um algoritmo que calcule e escreva o valor de S onde:

$$S = \frac{1}{1} + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + \dots + \frac{99}{50}$$

Exemplo: Faça um algoritmo que calcule a hipotenusa de 10 triângulos.

$$\text{hipotenusa}^2 = \text{cateto}^2 + \text{cateto}^2$$

ESTRUTURA DE REPETIÇÃO COM VARIÁVEL DE CONTROLE PARA-FAÇA (FOR ... TO... DO)

Sintaxe:

para V de I até L passo P faça

<comandos>

fim_para;

Em pascal:

a) for V: = I to L do

<comando>;

b) for <variável > := <início> dowto
<fim> do
 <comando>;

Onde V é uma variável inteira chamada variável de controle e <fim> e <n> são constantes, variáveis ou expressões inteiras, chamadas parâmetro da estrutura, sendo <início> o valor inicial da <variável>, <fim> o valor limite da <variável> e <n> o acréscimo sofrido pela <variável> ao término de cada execução da ação <comando>

Semântica: Quando o controle entra na estrutura <variável>< é inicializada com <início> e em seguida serão repetidos os seguintes passos:

1º) < variável> é comparada com <fim>, e se <variável> for menor ou igual a <<fim> continua a repetição

2º) Executa a ação <comando> até atingir o fim_para.

3º) O controle retorna ao início da estrutura e <variável é acrescida de <n>

Quando <variável> assumir um valor maior do que <fim> o controle sai da estrutura para o comando seguinte ao fim_para.

Exemplo: Somar todos os valores ímpar de 0 a 20

Algoritmo imparsoma;

inteiro : impar;

real: soma;

Início

 Soma ← 0;

 Para impar de 1 ate 20 passo 2 faça

 Soma ← soma + impar;

 Fim_para;

 Imprima(soma);

Fim.

Obs.:

- O para faça equivale a

$V \leftarrow I$

 Enquanto $V \leq L$ faça

 C;

$V \leftarrow V+P$;

 Fim_enquanto;

Isto é: o para_faz compacta a inicialização da variável de controle, o teste para verificar se ela ainda não atingiu um determinado limite e o incremento de um valor constante sobre essa variável.

-Se $P=1$ o passo pode ser omitido

Ex: Para I de 1 até 100 faça

 Imprima(I);

 Fim-para

- Se o passo $P < 0$ a condição a ser testada será $V \geq L$, isto é , v vai decrescer de I até no máximo L.

Todos os estruturas de repetição podem ser "aninhadas", isto é, ser colocadas uma dentro da outra.

Ex: Para x de 1 ate 2 faça

1 5 9

 Para y de 5 ate 7 faça

1 5 8

 Para z de 10 ate 8 faça

1 6 10

 Imprima(x,y,z);

1 6 9

1 6 8

Saída: 1 5 10

1 7 10

1 7 9
1 7 8
2 5 10
2 5 9
2 5 8
2 6 10

2 6 9
2 6 8
2 7 10
2 7 9
2 7 8

Exercício : Desenvolver um algoritmo para ler um valor inteiro $n \geq 1$, calcular e imprimir o valor de S dado por

$$s = \frac{1}{N} + \frac{2}{N-1} + \frac{3}{N-2} + \dots + \frac{N-1}{2} + N$$

Exemplo: Apresentar o resultado da tabuada de multiplicação de um número qualquer.

8. TIPOS ESTRUTURADOS

7.1 VARIÁVEIS UNI DIMENSIONAIS (VETORES)

Nem sempre os tipos básicos (inteiro, real, caracter e lógico) são suficientes para exprimir estruturas de dados em algoritmos. Por exemplo, considere um algoritmo para ler a nota de cada aluno de uma turma, calcular e imprimir a média da turma, considere a turma com 5 alunos. O algoritmo seria o seguinte.

Algoritmo nota

inteiro: n1, n2, n3, n4;

real média;

Início

Leia(n1);

Leia(n2);

Leia(n3);

Leia(n4);

Media $\leftarrow ((n1+n2+n3+n4)/5)$;

Imprima('Nota = ', n1, 'média', media);

Imprima('Nota = ', n2, 'média', media);

Imprima('Nota = ', n3, 'média', media);

Imprima('Nota = ', n4, 'média', media);

Fim.

Imagine agora que o número de alunos da turma seja 80. Só a declaração destas variáveis tornaria impraticável a redação do algoritmo. Neste caso deve-se usar uma estrutura para representar os dados no algoritmo, isto é, no lugar de variáveis simples declaradas diretamente como sendo de um tipo básico, usar uma estrutura de dados que deve ser primeiramente definida e em seguida declarar as variáveis estruturadas conforme esta definição.

A definição de novos tipos (ou estruturas) pode ser feita a partir dos tipos primitivos e possui o seguinte formato geral:

Tipo *nome* = definição;

O primeiro tipo que veremos será o vetor:

O vetor é uma estrutura homogênea e estática, isto é, todas as componentes são de um mesmo tipo e seu tamanho permanece o mesmo durante toda a execução do programa.

Sintaxe:

Tipo $\langle nome \rangle = \text{vet} [li : ls] \langle \text{tipo } t \rangle$

Onde li é limite inferior e ls é o valor superior. $\langle \text{tipo } t \rangle$ é um dos tipos básico. li e ls só podem ser valores inteiros.

Essa especificação apenas indica um modelo. Para efetivar esta estrutura dentro de um algoritmo é necessário declará-la dando um nome à variável que será criada segundo o modelo especificado:

Ex₁: Tipo $v = \text{vet} [1 : 80]$ real

V: Notas;

A variável notas é uma variável do tipo v , ou seja, um vetor de 80 componentes reais numeradas de 1 a 80

O número de elementos de um vetor será dado por $ls - li + 1$. Isto significa que as posições do vetor são identificadas a partir de li , com incrementos unitários, até ls .

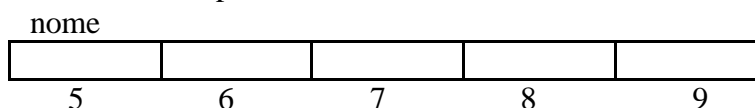
Ex₂: Qual é o número de elementos

Tipo $v = \text{vet} [5 : 9]$ caracter;

V: Nome;

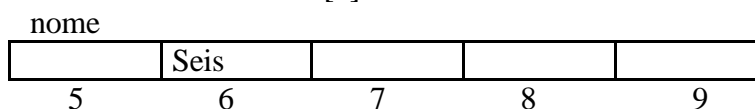
Solução:

O vetor tem $9 - 5 + 1 = 5$ elementos e pode ser visto como



Para referência a um elemento do vetor utiliza-se o nome do vetor e a identificação do elemento (índice) entre colchetes. Por exemplo, se desejamos atribuir o valor "seis" ao elemento identificado pelo índice 6 do vetor anterior, temos:

Nome [6] \leftarrow "seis"



Vamos resolver o algoritmo proposta no início deste capítulo para calcular e imprimir a nota é a média da turma 80 alunos.

Ex₃: Algoritmo

Tipo $v = \text{vet} [1:80]$ inteiros;

Int: i , soma;

Real: media;

v : nota;

Início

Soma \leftarrow 0;

Para i de 1 até 80 faça

Imprima('Digite as notas dos alunos');

Leia (nota[i]);

Fim-para;

Para i de 1 até 80 faça

soma \leftarrow soma + nota[i];

Fim-para;

Media \leftarrow soma / 80;

Para i de 1 até 80 faça

Imprima ('Nota = ', nota[i], ' media = ', media);

Fim-para;

Fim.

Ex₄: Faça um algoritmo que leia, via teclado, 20 valores do tipo inteiro e determine qual o menor valor existente no vetor e imprima valor e seu índice no vetor.

Algoritmo par

Tipo v = vet [1:20] inteiros;

Int: i, menor, índice;

v: valores;

Início

Para i de 1 ate 20 faça

Imprima('digite um valor');

Leia (valores[i]);

Fim-para;

menor ← valores [1];

índice ← 1;

Para i de 2 ate 20 faça

se valores [i] < menor

então

menor ← valores [i];

índice ← i;

fim-se;

Fim-para;

Imprima ('Menor valor é ', menor, 'seu índice é ', índice);

Fim.

7.2 VARIÁVEIS BIDIMENSIONAIS (VETORES)

È também uma estrutura homogênea e estática diferindo do vetor por possuir duas ou mais dimensões.

A =

a11	a12
a21	a22
a31	a33

Matriz A 3x2 (Linha x Coluna)

Sintaxe: Tipo m = mat [li₁:ls₁, li₂:ls₂] <tipo básico>;

Da mesma forma, esta especificação corresponde apenas à criação do modelo, e, para efetivar a estrutura de dados dentro do algoritmo, é necessário uma declaração dando um nome à variável que segue o modelo; por exemplo vamos criar a matriz acima:

Tipo m = matriz [1:3,1:2] real;

m:A;

A variável A é uma matriz de 3 linhas e 2 colunas do tipo real, ou seja, contendo valores reais.

Ex₅: Definir uma variável para armazenar os dados de uma matriz 4 por 4 de números do tipo REAL.

algoritmo Atribui

Tipo m: matriz [1:4,1:4] real

int: i,j;

m: valores;

início

para i de 1 ate 4 faça

para j de 1 ate 4 faça

```
    imprima('Digite valores');
    leia (valores [ i , j] )
  fim para
fim para
fim.
```

Ex₆: Dado uma matriz de ordem 3x3 faça um algoritmo que:

- Calcule a soma dos elementos da primeira coluna;
- Calcule o produto dos elementos da primeira linha;

Algoritmo Atribui

Tipo m: matriz [1:3,1:3] real

int: i,j;

real: soma, produto;

m: valores;

inicio

 soma ← 0;

 produto ← 1;

 para i de 1 ate 4 faça

 para j de 1 ate 4 faça

 imprima('Digite valores');

 leia (valores [i , j])

 fim para

 fim para

 para i de 1 ate 4 faça

 para j de 1 ate 4 faça

 se j = 1

 entao

 soma ← soma + valores[i,j];

 fim-se;

 se i = 1

 entao

 produto ← produto*valores[i,j];

 fim-se;

 fim para

 fim para

 imprima(' A soma dos valores da primeira coluna é:', soma);

 imprima(' O produto dos valores da primeira linha é:', produto);

 fim.

8. ESTRUTURA DE DADOS HETEROGÊNEAS - REGISTRO

O tipo registro é diferente das demais formas de definir variáveis, porque permite que uma variável armazene valores de diversos tipos diferentes.

Exemplo: Imagine que fosse desejado armazenar informações de uma pessoa, tais como: Nome, Idade, Altura, Sexo, Número de Dependentes, Profissão.

Na forma tradicional, seria necessário definir uma variável para cada tipo de informação, ou seja:

Cadeia : Nome;
Inteiro : Idade;
Real : Altura;
Character : Sexo;
Inteiro : NumDep;
Cadeia : Profissão;

Utilizando o tipo registro, a definição seria a seguinte:

Tipo

```
reg =registro
  Cadeia      : Nome;
  Inteiro     : Idade;
  Real        : Altura;
  Character   : Sexo;
  Inteiro     : NumDep;
  Cadeia      : Profissão;
Fim_registro;
```

Reg: pessoa;

Ao definir uma variável como sendo do tipo REGISTRO, devemos definir, também quais serão as partes componentes desta variável (Nome, Idade, Altura, Sexo, NumDep e Profissão), junto com o seu tipo. Quando estamos trabalhando com REGISTRO, as partes componentes do mesmo recebem um Nome próprio, o qual é conhecido como “campo”. Desta forma, uma variável REGISTRO pode ter campos de qualquer tipo válido do Pascal, sendo permitido inclusive que um REGISTRO seja definido dentro do outro, ou como parte de um vetor ou matriz.

Continuando o Exemplo, caso desejarmos atribuir um valor a variável Pessoa, devemos fazê-lo da seguinte forma:

```
Pessoa.idade : = 45
```

O uso do “.” indica que esta variável possui campos, e que “Idade” é um deles. É importante lembrar que as operações realizadas sobre uma variável REGISTRO são as mesmas de uma variável “comum”, a única diferença que devemos indicar o Nome da variável, seguido de um ponto(.), seguido do Nome do campo.

É possível atribuir o conteúdo de uma variável REGISTRO para outra variável, de mesmo tipo, da mesma forma que é feito como as outras variáveis.

Exemplo: Caso duas variáveis, digamos A e B sejam definidas como sendo REGISTROs, e caso seja desejado passar o conteúdo, isto é os valores existentes nos campos, a variável A para a variável B, bastará realizar a seguinte atribuição:

```
A: =B
```

Exemplo1:

Faça um algoritmo para ler do teclado os seguintes dados definido em um REGISTRO tendo os seguintes campos: Nome, Semestre, Sala, Curso, Notas (total de seis).

Algoritmo ler;

```
Tipo reg = registro
  Cadeia [50] : Nome;
  Inteiro     : Semestre;
  Inteiro     : Sala ;
```

```
Cadeia [30] : Curso;
Notas : vetor [1:6] inteiro;
Fim_registro;
Início
  reg: dados;
  inteiro: i;
  Imprima('Digite os seguintes dados: Nome - Semestre - Sala - Curso - Notas);
  Imprima( 'Nome');
  Leia(dados.nome);
  Imprima( 'Semestre');
  Leia(dados.semestre);
  Imprima( 'Sala');
  Leia(dados.sala);
  Imprima( 'Curso');
  Leia(dados.curso);
  Imprima( 'As 6 notas');
  Para i de 1 ate 6 faça
    Leia(dados.notas[i]);
  Fim-para;
Fim.
```

Observe o seguinte registro :

```
tipo f = registro
  cadeia[50]      : nome;
  Ender          : endereco;
  Cadeia[25]     : cidade;
  Cadeia[2]      : estado;
  Real           : salario;
Fim_registro;
f : func;
```

Há na definição o tipo *ender*, tipo de variável não básica, portanto precisamos defini-la.

```
tipo ender = registro
  cadeia[50]      : rua;
  inteiro        : NRO, CEP;
  fim_registro;
```

Isto significa que *ender* é o nome do modelo de um outro registro embutido no modelo *f*.

A atribuição de valores às variáveis que compõem o registro será feita da seguinte maneira:

```
Func.nome ← 'Fulano de tal';
Func.endereco.rua ← 'Rua das árvores',
Func.endereco.nro ← '1001';
Func.endereco.cep ← '30000';
Func.cidade ← 'Juiz de Fora';
Func.Estado ← 'MG';
Func.Salario ← '3.000';
```

Exemplo 2:

Vamos ampliar o exemplo anterior, acrescentando a definição de um outro campo (Endereço) que será também um REGISTRO o qual terá os seguintes campos: Rua, Bairro, Cidade, Estado, CEP. Faça um programa para ler as informações de um aluno, junto com o endereço descrito acima

```
Algoritmo ler;
  Tipo ende = registro
    Cadeia[30] : Rua;
    Cadeia[15] : Bairro, Cidade;
    Cadeia[2] : Estado;
    Inteiro : CEP;
  Fim-registro;
  Tipo reg = registro
    Cadeia [50] : Nome;
    Inteiro : Semestre;
    Inteiro : Sala;
    Cadeia [30] : Curso;
    Notas : vetor [1:6] inteiro;
    ende :Endereco;
  Fim_registro;
```

Início

```
reg: dados;
inteiro: i;
Imprima('Digite os seguintes dados: Nome - Endereço - Semestre - Sala - Curso - Notas);
Imprima( 'Nome');
Leia(dados.nome);
Imprima(Rua);
Leia(dados.endereco.rua);
Imprima(Bairro);
Leia(dados.endereco.bairro);
Imprima(Cidade);
Leia(dados.endereco.Cidade);
Imprima(Estado);
Leia(dados.endereco.estado);
Imprima(Cep);
Leia(dados.endereco.cep);
Imprima( 'Semestre');
Leia(dados.semestre);
Imprima( 'Sala');
Leia(dados.sala);
Imprima( 'Curso');
Leia(dados.curso);
Imprima( 'As 6 notas');
Para i de 1 ate 6 faça
  Leia(dados.notas[i]);
Fim-para;
```

Fim.

Outro exemplo do uso de registro

```
Tipo v = vetor[1:100] r;
v: cadastro;
tipo r = registro
  cadeia[50] : nome;
  inteiro : cpf;
  real : salario;
```

```

    caracter  : sexo;
    Inteiro   :
    Fim_registro;

```

f : func;

Este exemplo trata de um vetor onde cada elemento do vetor contém os dados de um funcionário de uma empresa de 100 funcionários.

Nome	Nome	Nome					Nome	Nome	Nome
CPF	CPF	CPF					CPF	CPF	CPF
Salario	Salario	Salario					Salario	Salario	Salario
Sexo	Sexo	Sexo					Sexo	Sexo	Sexo
1	2	3	4	97	98	99	100

A atribuição de dados ao funcionário 3 seria:

Cadastro[3].nome → 'Fulano de tal';

Cadastro[3].cpf → '123456789000';

Cadastro[3].salario → '3000' ;

Cadastro[3].sexo → 'f';

Exemplo3:

Escreva um algoritmo para ler as informações de um vetor de 30 alunos, onde os elementos do vetor e do tipo registro definido do exemplo anterior.

Algoritmo ler;

```

    Tipo ende = registro
    Cadeia[30] : Rua;
    Cadeia[15] : Bairro, Cidade;
    Cadeia[2] : Estado;
    Inteiro : CEP;

```

Fim-registro;

Tipo reg = registro

```

    Cadeia [50] : Nome;
    Inteiro : Semestre;
    Inteiro : Sala;
    Cadeia [30] : Curso;
    Notas : vetor [1:6] inteiro;
    ende :Endereco;

```

Fim_registro;

Tipo vet= vetor[1:30] reg;

Início

vet: vdados;

inteiro: i,j;

Imprima('Digite os seguintes dados: Nome - Endereço - Semestre - Sala - Curso - Notas);

Para i:=1 ate 30 faça

```

    Imprima( 'Nome');
    Leia(vdados[i].nome);
    Imprima(Rua);
    Leia(vdados[i].endereco.rua);

```

```
Imprima(Bairro);
Leia(vdados[i].endereco.bairro);
Imprima(Cidade);
Leia(vdados[i].endereco.Cidade);
Imprima(Estado);
Leia(vdados[i].endereco.estado);
Imprima(Cep);
Leia(vdados[i].endereco.cep);
Imprima( 'Semestre');
Leia(vdados[i].semestre);
Imprima( 'Sala');
Leia(vdados[i].sala);
Imprima( 'Curso');
Leia(vdados[i].curso);
Imprima( 'As 6 notas');
Para j de 1 ate 6 faça
    Leia(vdados[i].notas[j]);
Fim-para;
```

```
Fim-para;
```

```
Fim.
```

Exemplo 4: Escreva um algoritmo que dado o código da profissão, emita o nome da profissão.

Algoritmo profissao;

Tipo reg =registro

```
    Inteiro    : codigo;
```

```
    Cadeia[50] : nome;
```

Fim_registro;

Tipo vet = vetor[1..10] reg;

Inicio

```
    Vet:Vprof;
```

```
    Inteiro: i, cod;
```

```
    Imprima('Preencha tabela');
```

```
    Imprima('Digite código e profissão');
```

```
    Para i de 1 to 10 faça
```

```
        Leia(vprof[i].codigo);
```

```
        Leia(vprof[i].nome);
```

```
    Fim-para;
```

```
    Imprima('Digite código procurado');
```

```
    Leia(cod);
```

```
    Para i de 1 ate 10 faça
```

```
        Se vprof[i].codigo = cod então
```

```
            Imprima(vprof[i].nome);
```

```
        Fim-se;
```

```
    Fim-par
```

```
Fim.
```

Uma empresa compra uma série de produtos de diversos fabricantes, e precisa que sejam emitidos os seguintes relatórios: Qual o produto que possui a maior quantidade em estoque, e qual o que tem a menor quantidade. Qual o produto mais caro e o mais barato Quais são os produtos pertencentes ao fabricante XYZ

Algoritmo empresa;

Tipo fab = vetor[1..10] cadeia[20];

Tipo reg = registro

```
Cadeia[20] : produto, fabricante;  
Inteiro    :quant;  
Real       :preco;
```

```
Fim_registro;
```

```
Tipo vet = vetor[1:100] reg;
```

```
Inicio
```

```
vet : vprod;
```

```
Inteiro : i, menorq, maiorq, cont;
```

```
Real: menorp, maiorp;
```

```
Cadeia[20] : nmenorq, nmaiorq , nmenorp, nmaiorp;
```

```
Fab:vfab;
```

```
Cont ← 0;
```

```
Imprima('Preencha tabela');
```

```
Imprima('Digite nome do produto, quantidade, preço e fabricante');
```

```
Para i de 1 ate 100 faça
```

```
    Leia(vprod[i].produto);
```

```
    Leia(vprod[i].quant);
```

```
    Leia(vprod[i].preco);
```

```
    Leia(vprod[i].fabricante);
```

```
Fim-para;
```

```
Maiorq ← vprod[1].quant;
```

```
Menorq ← vprod[1].quant;
```

```
Nmenorq ← vprod[1].quant;
```

```
Nmaiorq ← vprod[1].quant;
```

```
Maiorp ← vprod[1].preco;
```

```
Menorp ← vprod[1].preco;
```

```
Nmenorp ← vprod[1].preco;
```

```
Nmaiorp ← vprod[1].preco;
```

```
Para i de 2 ate 100 do
```

```
    Se vprod[i].quant > maiorq entao
```

```
        Maiorq ← vprod[i].quant;
```

```
        Nmaiorq ← vprod[i].produto;
```

```
    Fim-se;
```

```
    Se vprod[i].quant < menorq entao
```

```
        Menorq ← vprod[i].quant;
```

```
        Nmenorq ← vprod[i].produto;
```

```
    Fim-se;
```

```
    Se vprod[i].preco > maiorp entao
```

```
        Maiorp ← vprod[i].preco;
```

```
        Nmaiorp ← vprod[i].produto;
```

```
    Fim-se;
```

```
    Se vprod[i].preco < menorp entao
```

```
        Menorp ← vprod[i].preco;
```

```
        Nmenorp ← vprod[i].produto;
```

```
    Fim-se;
```

```
    Se vprod[i].fabricante = XYZ então
```

```
        Vfab[cont] ← vprod[i].produto;
```

```
        Cont ← cont+1;
```

```
    Fim-se;
```

```
Fim-para;
```

```
Imprima( 'Produto que possui maior quantidade em estoque ',nmaiorq);
```

Imprima('Produto que possui menor quantidade em estoque ',nmaiorq);

Imprima('Produto que possui maior preço ',nmaiorp);

Imprima('Produto que possui menor preço ',nmenorp);

Imprima('Produto que fabricados por XYZ')

Para i de 1 ate cont faça

 Imprima(vfab[cont]);

Fim-para;

Fim.

9 SUBROTINAS

Um matemático uma vez disse que um grande problema se resolve dividindo-o em pequenas partes e resolvendo tais partes em separado. Estes dizeres servem também para a construção de programas. Os profissionais de informática quando necessitam construir um grande sistema, o fazem, dividindo tal programa em partes, sendo então desenvolvido cada parte em separado, mais tarde, tais partes serão acopladas para formar o sistema. Podemos denominá-las de subrotinas

O QUE É UMA SUBROTINA E POR QUÊ?

Uma subrotina nada mais é do que um grupo de comandos que constitui um trecho de algoritmo com uma função bem definida. Cada subrotina, durante a execução do algoritmo, realiza uma tarefa específica da solução do problema e, para tal, pode contar com o auxílio de outras subrotinas do algoritmo. Desta forma, a execução de um algoritmo contendo várias subrotinas pode ser vista como um processo cooperativo. A construção de algoritmos compostos por subrotinas, ou seja, a construção de algoritmos através de modularização possui uma série de vantagens:

Torna o algoritmo mais fácil de escrever. O desenvolvedor pode focalizar pequenas partes de um problema complicado e escrever a solução para estas partes, uma de cada vez, ao invés de tentar resolver o problema como um todo de uma só vez. .

Torna o algoritmo mais fácil de ler. O fato do algoritmo estar dividido em subrotinas permite que alguém, que não seja o seu autor, possa entender o algoritmo mais rapidamente por tentar entender as suas subrotinas separadamente, pois como cada subrotina é menor e mais simples do que o algoritmo único

E possível entender o que um algoritmo faz por saber apenas o que as suas subrotinas fazem, sem que haja a necessidade de entender os detalhes internos às subrotinas. Além disso, se os detalhes nos interessam, sabemos exatamente onde examiná-los.

Economia de tempo, espaço e esforço. freqüentemente, necessitamos executar uma mesma tarefa em vários lugares de um mesmo algoritmo. Uma vez que uma subrotina foi escrito, ele pode, como veremos mais adiante, ser "chamado" quantas vezes quisermos e de onde quisermos no algoritmo, evitando que escrevamos a mesma tarefa mais de uma vez no algoritmo, o que nos poupará tempo, espaço e esforço.

COMPONENTES DE UMA SUBROTINA

As subrotinas que estudaremos daqui em diante possuem dois componentes: corpo e cabeçalho. O corpo de uma subrotina é o grupo de comandos que compõe o trecho de algoritmo correspondente à subrotina.

9.1 Criando Procedimentos

Um procedimento é um bloco precedido de um cabeçalho. Com isto será possível fazer referência ao bloco de qualquer ponto do algoritmo. A sintaxe da declaração será:

Procedimento <nome do procedimento>

Início

<declarações>

C₁;

C₂;

.

.

C_N;

Fim-procedimento;

Exemplo:

```
Procedimento Troca
  Início
    Inteiro: aux;
    Aux ← x;
    X ← Y;
    Y ← aux;
  Fim-Procedimento;
```

A declaração de um procedimento deve vir sempre no início do bloco em que estiver sendo declarado (antes de qualquer comando executável). Um procedimento só é executado sob chamada. A chamada no procedimento é feita por um comando que se resume no nome do procedimento. Observe o uso do procedimento troca no algoritmo abaixo:

Algoritmo leia;

Procedimento TROCA;

Início

```
  Inteiro: aux;
  aux ← X;
  X ← Y;
  Y ← aux;
```

Fim_procedimento;{troca}

Início

```
  Inteiro: X,Y, A, B, C, D;
  X ← A;
  Y ← B;
  TROCA;
  Imprima (A, B);
  A ← X;
  B ← Y;
  Imprima (A, B);
  C ← 4;
  D ← 9;
  Imprima (C, D);
  X ← C;
  Y ← D;
  TROCA;
  C ← X;
  D ← Y;
  Imprima (C, D);
```

Fim.

Um PROCEDIMENTO, é uma tipo de subrotina que é ativada através da colocação de seu Nome em alguma parte do programa. Desta forma, assim que o Nome de um PROCEDIMENTO é encontrado, ocorre um desvio no programa, para que os comandos da subrotina sejam executados. Ao término da subrotina, a execução retornará ao ponto subsequente a chamada do PROCEDIMENTO.

Exemplo: Faça um procedimento para calcular a área de 4 quadrado de lado 4, 5 e 6 respectivamente e imprima o resultado.

Algoritmo ler

```
  Procedimento quadrado
```

```

Início
  Inteiro: result;
  Result ← l*l;
  Imprima(' A área do quadrado é ', Result);
Fim_procediemnto; {quadrado}
Início
  Inteiro: l;
  L ← 4;
  Quadrado;
  L ← 5;
  Quadrado;
  L ← 6;
  Quadrado;
Fim.

```

Exemplo: Construir uma subrotina para escrever na tela números de 1 ate N.

<pre> Algoritmo video Procedimento EscreveNoVideo; Início Inteiro:numero; Para número de 1 ate n faça imprima (número); Fim_procedimento; Início Inteiro: n; Imprima('Digite um valor para n'); leia(n); escrevenovideo; imprima ('fim'); Fim. </pre>	<pre> Program video; Var Número, n : integer;; PROCEDURE EscreveNoVideo; Begin For número := 1 to n do writeln (número); end; Begin Writeln('Digite um valor para n'); readln(n); escrevenovideo; writeln ('fim'); end. </pre>
---	--

Exemplo: Faça uma PROCEDURE para desenhar uma moldura no vídeo.

```

Program moldura;
Var
  i : integer;;
PROCEDURE molduravideo;
Begin
  For i := 2 to 80 do
    Begin
      Gotoxy(i,2)
      write ('=');
      Gotxy(i,24);
      Write('`');
    end;
  For i:= 2 to 78 do
    begin

```

```

    Gotoxy(2,i)
    write ('|');
    Gotoxy(78,i)
    write ('|');
  end;
end;
Begin
  molduravideo;
end.

```

9.2 Variáveis Globais e Locais

Damos o Nome de variáveis globais para aquelas variáveis que são definidas logo após o comando VAR do programa principal, sendo desta forma visíveis em qualquer parte do programa.

Exemplo:

<pre> Algoritmo video Procedimento EscreveNoVideo; Inicio Inteiro:numero, n ; Imprima('Digite um valor para n'); leia(n); Para número de 1 ate n faça imprima (número); Fim_procedimento; Inicio escrevenovideo; Fim. </pre>	<pre> PROGRAM Video; PROCEDURE EscreveNoVÍdeo; VAR Número, N : INTEGER; BEGIN Writeln('Digitar valor para n '); READ(N); FOR número := 1 TO N DO WRITE(Número); END; BEGIN EscreveNoVÍdeo; END. </pre>
--	--

Damos o Nome de variáveis locais às variáveis que são declaradas dentro de uma sub-Rotina, sendo que as mesmas só podem ser manipuladas dentro da sub-Rotina que as declarou, não sendo visíveis em nenhuma outra parte do programa.

Exemplo:

<pre> Algoritmo teste; Procedimento Setanome; Inicio Imprima('Digita nome'); leia(Nome); fim-procedimento; Inicio Cadeia[80] : Nome ; (variável global) Setanome; Imprima(Nome); fim </pre>	<pre> VAR Nome : STRING[80]; (variável global) PROCEDURE Setanome; BEGIN Writeln('Digita nome'); READ(Nome); END; BEGIN Setanome; WRITE(Nome); END </pre>
--	--

```

PROGRAM Teste;

```

Obs: É possível definir variáveis globais e locais com o mesmo Nome, sendo qualquer mudança no conteúdo da variável local não afetará o conteúdo da variável global.

Exemplo:

```
Algoritmo teste;
Procedimento Setanome;
Início
    Imprima('Digite nome');
    Leia(Nome);
fim-procedimento;
Procedimento mudança;
    Cadeia[80]: Nome;
Início
    imprima('Digite nome: ');
    leia(Nome);
Fim_procedimento;
Início
    Cadeia[80] : Nome ;
    Setanome;
    Imprima(Nome);
    Mudanca;
    Imprima(Nome);
fim
```

```
PROGRAM Teste;
VAR
    Nome : STRING[80];
PROCEDURE Setanome;
BEGIN
    Write('Digite nome: ');
    READ(Nome);
END;
PROCEDURE Mudança;
VAR
    Nome : STRING[80];
BEGIN
    Write('Digite nome: ');
    READ(Nome);
END;
BEGIN
    Setanome;
    WRITE(Nome);
    mudança;
    WRITE(Nome);
END;
```

No Exemplo acima, a variável global “Nome” e a variável local “Nome” representam posições de memória totalmente diferentes, logo, qualquer mudança no conteúdo da variável local, não afetará o conteúdo da variável global.

9.3 Passagem de Parâmetros

Até agora vimos que para ativar uma sub-Rotina bastaria colocar o seu Nome em alguma parte do programa. Mas isto nem sempre significa que o trabalho de escrever o programa irá diminuir. Com o que vimos até agora, dependendo da tarefa a ser realizada pela sub-Rotina, o trabalho de um programador pode até ser bem complicado. Por Exemplo, como faríamos para ler 5 vetores, todos com tamanhos diferentes? Poderíamos, por Exemplo, criar 5 sub-Rotinas, uma para cada vetor a ser lido. Isto sem dúvida resolveria esta situação, mas, e se fossem 100 vetores?, ou 1000? Seria realmente uma tarefa muito trabalhosa ter de escrever 100, ou 1000 sub-Rotinas, isto só para ler os vetores, imagine se tivéssemos também que ordená-los, ou realizar outro processo qualquer. Com toda esta dificuldade, o uso das sub-Rotinas deveria ser considerado. Como já foi dito, as sub-Rotinas foram criadas para serem genéricas o bastante para se adaptarem a qualquer situação, visando justamente a possibilidade de reutilização do código. Para realizar esta “mágica”, foi criado o conceito de passagem de parâmetros, ou seja, passar informações para serem tratadas dentro da Sub-Rotina.

Sintaxe:

```
PROCEDURE <Nome> (<Variável> : <Tipo>);  
    <Definições>;  
BEGIN  
    <comandos>;  
END;
```

Obs: Variável do mesmo tipo são separadas por vírgulas (.). Variáveis de tipos diferentes, são separadas por ponto e vírgula (;).

Exemplo:

```
PROGRAM Teste;  
VAR  
    Número      : INTEGER;  
    Funcionário  : STRING;  
  
PROCEDURE EscreveNome(N : INTEGER; Nome : STRING);  
VAR  
    I : INTEGER;  
BEGIN  
    FOR i : = 1 TO n DO  
        BEGIN  
            WRITE(Nome);  
        END;  
    END;  
END;  
  
BEGIN  
    WRITELN('Digita número e nome funcionário');  
    READ(Número, Funcionário);  
    EscreveNome(Número, Funcionário);
```

END.

Obs: Os números dados aos parâmetros não necessitam serem iguais as variáveis passadas para sub-Rotina. No Exemplo acima, o valor contida em “Número” será passado para o parâmetro “N”, da mesma forma que o valor ontido na variável “Funcionário” será passada para o parâmetro “Nome”. Note que os nomes são diferentes.