



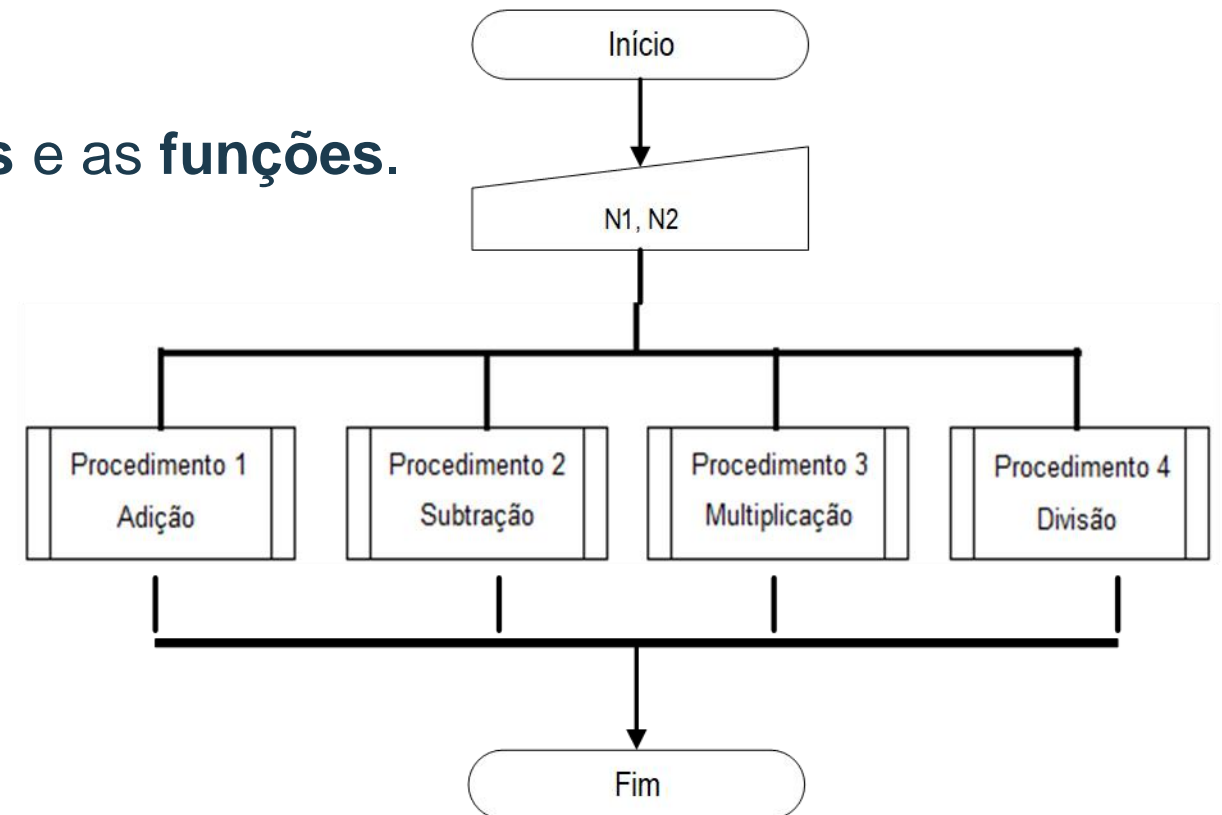
UNIDADE III

Lógica de Programação e Algoritmos

Profa. Ma. Eliane Santiago

Modularização

- É a organização do algoritmo em módulos identificados com nomes significativos, que indicam seu propósito.
- Cada módulo resolve um subproblema e é um algoritmo com entrada(s), processamento(s) e saída(s).
- Há dois tipos de módulos: os **procedimentos** e as **funções**.



Formas de Modularizar o Algoritmo

1. Procedimentos sem lista de parâmetros.
2. Procedimentos com lista de parâmetros.
3. Funções sem lista de parâmetros.
4. Funções com lista de parâmetros.

procedimento escrevaMenu()

procedimento comando(n : real)

funcao questao() : inteiro

funcao fatorial(n : inteiro) : real

Procedimentos

- Procedimentos são módulos projetados para executar um conjunto de comandos e não retornar nenhum valor.

Sintaxe:

```
procedimento <nomeDoModulo>([lista de parâmetros])  
var  
    // declaração de variáveis locais  
início  
    // bloco de comandos  
fimprocedimento
```

- A lista de parâmetros pode ser vazia

```
procedimento nomeProcedimento()
```

- mas quando não é vazia, cada parâmetro é uma variável de entrada para o algoritmo do módulo.


```
procedimento nomeProcedimento(var1, var2: <tipo>)  
    // bloco de comandos  
fimprocedimento
```

Procedimentos sem Parâmetros

Sintaxe:

```
procedimento nomeProcedimento()  
var  
    // declaração de variáveis locais  
Início  
    // bloco de comandos  
fimprocedimento
```

```
1. Algoritmo "Exemplo"  
2. Var  
3.     //declaração de variaveis  
4. Início  
5.     nomeProcedimento()  
6. Fimalgoritmo
```



```
Algoritmo "Exemplo"  
  
procedimento nomeProcedimento()  
inicio  
    escreval("          Operações          ")  
    escreval(" [+] Adição                      ")  
    escreval(" [-] Subtração                     ")  
    escreva(" Qual operação deseja efetuar? ")  
Fimprocedimento
```

Chamada de Procedimento sem Parâmetros

Algoritmo `"*Exemplo Chamada de Procedimento*"`

Var

procedimento `escrevaMenu()`

inicio

escreval (`"*****"`)

escreval (`"* Operações *"`)

escreval (`"* [+] Adição *"`)

escreval (`"* [-] Subtração *"`)

escreval (`"* [*] Multiplicação *"`)

escreval (`"* [/] Divisão *"`)

escreval (`"*****"`)

escreval (`" Digite a operação desejada: "`)

fimprocedimento

`// bloco principal`

Inicio

`escrevaMenu()`

Fimalgoritmo

Chamada de Procedimento sem Parâmetros

```
1. Algoritmo "Procedimento Menu e Operações"  
2. Var //declaração de variaveis globais  
3.   opcao : caractere  
4.   x, y : inteiro  
5.  
6. procedimento escrevaMenu()  
7. inicio  
8.   escreval("          Operações          ")  
9.   escreval(" [+] Adição          ")  
10.  escreval(" [-] Subtração          ")  
11.  escreva(" Qual operação deseja efetuar? ")  
12. fimprocedimento
```

```
13. Inicio  
14. escrevaMenu()  
15. leia(opcao)  
16. escreva("X = ")  
17. leia(x)  
18. escreva("Y = ")  
19. leia(y)  
20. //processamento e saída  
21. escolha(opcao)  
22.   caso "+"  
23.     escreval(x, "+", y, "=", x+y)  
24.   caso "-"  
25.     escreval(x, "-", y, "=", x-y)  
26.   outrocaso  
27.     escreval("Opcao invalida!")  
28.   fimescolha  
29. Fimalgoritmo
```

escrevaMenu()

Isto é uma ordem!

Procedimentos com variáveis locais

- Variáveis locais são declaradas dentro do módulo e são reconhecidas apenas dentro do módulo. Quando o módulo é encerrado, as variáveis são destruídas.

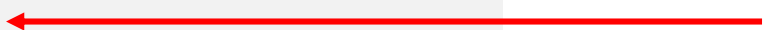
```
1.  procedimento calcularFatorial()  
2.  var //variáveis locais  
3.      n, fatorial, cont, aux: inteiro  
4.  inicio  
5.      //entrada  
6.      escreva("Digite um número inteiro: ")  
7.      leia(n)  
8.      fatorial ← 1  
9.      aux ← n  
10.  
11.     para cont de 1 até aux faca  
12.         fatorial <- fatorial * n  
13.         n ← n-1  
14.     fimpara  
15.     escreva("Fatorial de ", aux, " é ", fatorial)  
16. fimprocedimento
```


Chamada do Procedimentos calcularFatorial()

```
1.  procedimento calcularFatorial()  
2.  var //variáveis locais  
3.    n, fatorial, cont, aux: inteiro  
4.  inicio  
5.    //entrada  
6.    escreva("Digite um número inteiro: ")  
7.    leia(n)  
8.    fatorial ← 1  
9.    aux ← n  
10.  
11.    para cont de 1 até aux faca  
12.      fatorial <- fatorial * n  
13.      n ← n-1  
14.    fimpara  
15.    escreva("Fatorial de ", aux, " é ", fatorial)  
16.  fimprocedimento
```

- No bloco principal não há necessidades de ler os dados de entrada calcular o fatorial porque o procedimento já está fazendo isso.

```
17.  Inicio  
18.  Var  
19.    //variáveis globais  
20.  Inicio  
21.    calcularFatorial() ←  
22.  Fimalgoritmo
```



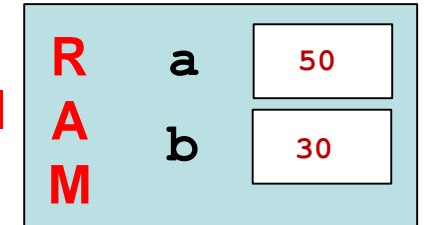
Sequência de execução dos procedimentos

- A alocação de memória para as variáveis locais é diferente da alocação para as variáveis globais. Portanto, é permitido identificar variáveis globais e locais com o mesmo nome.

```
1. Algoritmo "Variáveis G e L"  
2. Var //variáveis Globais  
3.   a, b: inteiro  
4.  
5. procedimento somarNumeros()  
6. var //variáveis Locais  
7.   a, b : inteiro  
8. inicio  
9.   escreva("A: ")  
10.  leia(a)  
11.  escreva("B: ")  
12.  leia(b)
```

```
13.   escreval(a, "+", b, " = ", a+b)  
14. Fimprocedimento  
15.  
16. Inicio  
17.   a <- 50  
18.   b <- 30  
19.   somarNumeros()  
20.   escreva("Fim do algoritmo")  
21. Fimalgoritmo
```

Global



```
A: 15  
B: 35  
15 + 35 = 50  
Fim do algoritmo
```

Interatividade

Analise os procedimentos procUm() e procDois() e escolha as alternativas corretas.

```
1. Algoritmo "Alg 01"  
2.  
3. procedimento procUm()  
4. var  
5.     i, n : inteiro  
6. inicio  
7.     escreva("Digite um Número: ")  
8.     leia(n)  
9.  
10.    para i de 1 ate 10 passo 1 faca  
11.        escreval(n, " * ", i, " = ", n*i)  
        fimpara  
procedimento
```

```
1. Algoritmo "Alg 02"  
2.  
3. Var  
4.     nome : caractere  
5.     salario, valor_bonus : real  
6.     tempo_servico : inteiro  
7.     bonus : inteiro  
8.  
9. procedimento procDois()  
10. inicio  
11.     escreval("Nome : ", nome)  
12.     escreval("tempo_servico : ", tempo_servico)  
13.     escreval("Salario : ", salario)  
14. fimprocedimento  
15. Fimalgoritmo
```

Interatividade

- I. No algoritmo Alg 01, as variáveis i e n são **variáveis locais** e o procedimento escreverá a tabuada de um número.
- II. No algoritmo procUm() as variáveis i e n são **variáveis globais** e o procedimento escreverá a tabuada de um número.
- III. No procedimento procDois() as variáveis nome, salario e tempo_servico são **variáveis locais** e o procedimento imprimirá os dados.
- IV. No procedimento procDois() as variáveis nome, salario e tempo_servico são **variáveis globais**, portanto acessíveis por todos os módulos.
- V. Em ambos os procedimentos as variáveis são locais.

Estão corretas apenas as afirmativas:

- a) Afirmativas I e IV, apenas.
- b) Afirmativas I e III, apenas.
- c) Afirmativas II e V, apenas.
- d) Afirmativas I, II e IV, apenas.
- e) Afirmativas IV e V, apenas.

Resposta

Analise os procedimentos procUm() e procDois() e escolha as alternativas corretas.

```
1. Algoritmo "Alg 01"  
2.  
3. procedimento procUm()  
4. var  
5.     i, n : inteiro  
6. inicio  
7.     escreva("Digite um Número: ")  
8.     leia(n)  
9.  
10.    para i de 1 ate 10 passo 1 faca  
11.        escreval(n, " * ", i, " = ", n*i)  
        fimpara  
procedimento
```

```
1. Algoritmo "Alg 02"  
2.  
3. Var  
4.     nome : caractere  
5.     salario, valor_bonus : real  
6.     tempo_servico : inteiro  
7.     bonus : inteiro  
8.  
9. procedimento procDois()  
10. inicio  
11.     escreval("Nome : ", nome)  
12.     escreval("tempo_servico : ", tempo_servico)  
13.     escreval("Salario : ", salario)  
14. fimprocedimento  
15. Fimalgoritmo
```

I. No algoritmo Alg 01, as variáveis i e n são **variáveis locais** e o procedimento escreverá a tabuada de um número.

Resposta

- I. No algoritmo Alg 01, as variáveis i e n são **variáveis locais** e o procedimento escreverá a tabuada de um número.
- II. No algoritmo procUm() as variáveis i e n são **variáveis globais** e o procedimento escreverá a tabuada de um número.
- III. No procedimento procDois() as variáveis nome, salario e tempo_servico são **variáveis locais** e o procedimento imprimirá os dados.
- IV. No procedimento procDois() as variáveis nome, salario e tempo_servico são **variáveis globais**, portanto acessíveis por todos os módulos.
- V. Em ambos os procedimentos as variáveis são locais.

Estão corretas apenas as afirmativas:

- a) **Afirmativas I e IV, apenas.**
- b) Afirmativas I e III, apenas.
- c) Afirmativas II e V, apenas.
- d) Afirmativas I, II e IV, apenas.
- e) Afirmativas IV e V, apenas.

Procedimentos com Parâmetros

- A lista de parâmetros representa o conjunto de dados de entrada para o algoritmo.

```
procedimento nomeProcedimento (var1, var2 : <tipo1>, var3 : <tipo2>, var5 : <tipo3>)  
var  
    // declaração de variáveis locais  
Início  
    // bloco de comandos  
fimprocedimento
```

```
procedimento escreverDados (nome, sobrenome: caractere,  
                             idade : inteiro, salario: real)  
inicio  
    escreval ("Nome.....: ", nome, " ", sobrenome)  
    escreval ("Idade.....: ", idade, " anos.")  
    escreval ("Salario...: ", salario, " reais."))  
fimprocedimento
```

Procedimento com parâmetros

- Variáveis locais são declaradas dentro do módulo e são reconhecidas apenas dentro do módulo. Quando o módulo é encerrado, as variáveis são destruídas.

```
1. procedimento somar(a, b: inteiro)  
2. inicio  
3.     escreval(a, " + ", b, " = ", a+b)  
4. fimprocedimento
```

```
7. procedimento fatorial(a: inteiro)  
8. inicio  
9.     escreval(a, " - ", b, " = ", a-b)  
10 fimprocedimento
```

```
11. procedimento fatorial(n : inteiro)  
12. var  
13.     n, fatorial, cont, aux: inteiro  
14. inicio  
15.     fatorial <- 1  
16.     aux <- n  
17.  
18.     para cont de 1 ate aux faca  
19.         fatorial <- fatorial * n  
20.         n <- n-1  
21.     fimpara  
22.     escreva(aux, "! = ", fatorial)  
23. fimprocedimento
```


Chamada de Procedimento com parâmetros

- A lista de parâmetros deve estar na mesma sequência em que foi declarada.

```
1.  //bloco principal
2.  Var
3.      x : inteiro
4.      y : real
5.  Inicio
6.      leia(x)
7.      Leia(y)
8.      somar(x, y) // Erro!
9.
10.
11.
12.      fatorial(x)
13.      fatorial(5.0)
14.
15.
16.      subtrair(60, 20)
17.
18.
19.  Fimalgoritmo
20.
21.
```

Ordem e os tipos dos argumentos

- Atenção à sequência dos parâmetros é importante.

```
1. procedimento imprimirDados(nome: caractere, idade : inteiro)
2. inicio
3.     //bloco principal
4. fimprocedimento
5.
```

Correto!

```
1. imprimirDados("Fulano", 18)
2.
3. imprimirDados(var_nome, 21)
4.
5.
```

Errado!

```
1. imprimirDados(18, "Fulano")
2.
3. imprimirDados(var_nome, 21.69)
4.
5.
```

Chamada de Procedimento com Parâmetros

```
1. Algoritmo "Procedimento Operações"
2. Var //declaração de variaveis globais
3.   x, y : inteiro
4.
5. procedimento somar(a, b: inteiro)
6. inicio
7.   escreval(a, " + ", b, " = ", a+b)
8. fimprocedimento
9.
10. procedimento subtrair(a, b: real)
11. inicio
12.   escreval(a, " - ", b, " = ", a-b)
13. fimprocedimento
14.
15. //bloco principal
16. Inicio
17.   x <- 30
18.   y <- 210
19.   somar(x, y)
20.   subtrair(60,20)
21.   Subtrair(y, x)
Fimalgoritmo
```

Chamada de Procedimento com Parâmetros

```
1. Algoritmo "Procedimento ImprimirDados"
2. Var
3. // variaveis globais
4.     nome : caractere
5.     idade : inteiro
6.
7.
8. procedimento imprimirDados(texto: caractere; numero: inteiro)
9. inicio
10.     escreval("Nome : ", texto)
11.     escreval("Idade : ", numero)
12. fimprocedimento
13.
14. //modulo principal
15. Início
16.     nome ← "Maria"
17.     idade ← 20
18.     imprimirDados(nome, idade)
19.
20. Fimalgoritmo
```

Interatividade

O procedimento abaixo recebe dois inteiros como parâmetros de entrada e verifica se o primeiro é divisível pelo segundo. Assinale a alternativa que apresenta a chamada de procedimento correta.

```
procedimento testarDivisibilidade(dividendo, divisor : inteiro)  
var  
    resultado : logico  
inicio  
se ((dividendo mod divisor) = VERDADEIRO) entao  
    escreval(dividendo, " é divisível por ", divisor)  
    senao  
        escreval(dividendo, " é divisível por ", divisor)  
    fimse  
fimprocedimento
```

- a) testarDivisibilidade(21,4)
- b) testarDivisibilidade("15", 2)
- c) testarDivisibilidade(15.90, 4)
- d) a <- testarDivisibilidade(200 ,"5")
- e) testarDivisibilidade(18, 3, 0)

Resposta

O procedimento abaixo recebe dois inteiros como parâmetros de entrada e verifica se o primeiro é divisível pelo segundo. Assinale a alternativa que apresenta a chamada de procedimento correta.

```
procedimento testarDivisibilidade(dividendo, divisor : inteiro)  
var  
    resultado : logico  
inicio  
se ((dividendo mod divisor) = VERDADEIRO) entao  
    escreval(dividendo, " é divisível por ", divisor)  
    senao  
        escreval(dividendo, " é divisível por ", divisor)  
    fimse  
fimprocedimento
```

- a) testarDivisibilidade(21,4)
- b) testarDivisibilidade("15", 2)
- c) testarDivisibilidade(15.90, 4)
- d) a <- testarDivisibilidade(200 ,"5")
- e) testarDivisibilidade(18, 3, 0)

Funções

- Função é um bloco de programa identificado por um nome, por meio do qual será referenciada em qualquer parte do programa principal. Ela tem diversas similaridades com um procedimento.
- Uma função é um módulo que tem por finalidade executar um bloco de códigos separado da estrutura principal e retornar um valor.

```
função nomeFuncao([lista_parâmetros]) : <tipo_de_retorno>  
inicio  
    //algoritmo  
retorne valor  
fimfuncao
```

Funções sem parâmetros

- Retornam uma constante, como o número do PI.
- Não requerem valores de entrada para processar o algoritmo.

Pontos de retorno da função

função nomeFuncao([lista_parâmetros]) : <tipo_de_retorno>

inicio

//algoritmo

retorne valor

fimfuncao

função numeroPI() : real

inicio

retorne 3,14159

fimfuncao

//bloco principal

Inicio

r <- numeroPI()

escreva("Número do PI = ", numeroPI())

se (var > numeroPI()) então

//faça alguma coisa

fimse

Fimalgoritmo

Funções com parâmetros

Algoritmo "Funcoes com parametros"

Var

 //declaracao das variaveis globais

 a, b, r: inteiro

 //funcao para calcular o delta

funcao delta(a, b, c: inteiro) : inteiro

inicio

retorne b*b-4*a*c

fimfuncao

 //procedimento para escrever as raízes

procedimento escrevaRaizes(a, b, c: inteiro)

var

 x1, x2 : real

inicio

se (delta(a,b,c)>=0) então

 x1 <- (-1*b+raizq(delta))/(2*a)

 x2 <- (-1*b-raizq(delta))/(2*a)

escreva("raiz 1 é ", x1)

escreva("raiz 2 é ", x2)

senão

escreva("Não existem raízes reais")

fimse

fimfuncao

 // bloco principal

 Inicio

escreva("Informe o valor de A: ")

leia(a)

escreva("Informe o valor de B: ")

leia(b)

escreva("Informe o valor de C: ")

leia(c)

escrevaRaizes(a, b, c)

Fimalgoritmo

O número é par?

```
funcao ehPar(numero : inteiro) : logico  
var  
    resposta : logico  
inicio  
    se (numero mod 2) = 0 então  
        resposta = VERDADEIRO  
    senão  
        resposta = VERDADEIRO  
    fimse  
fimfuncao
```

O número é ímpar?

```
funcao ehImpar(numero : inteiro) : logico  
var  
    resposta : logico  
inicio  
    se (numero mod 2) <> 0 então  
        resposta = VERDADEIRO  
    senão  
        resposta = VERDADEIRO  
    fimse  
fimfuncao
```

Qual é o maior?

```
funcao maior(x, y: inteiro) : inteiro  
var  
    m : inteiro  
inicio  
    se (x > y) então  
        m = x  
    senão  
        m = y  
    fimse  
    retorne m  
fimfuncao
```

Qual é o menor?

```
funcao menor(x, y: inteiro) : inteiro  
var  
    m : inteiro  
inicio  
    se (x < y) então  
        m = x  
    senão  
        m = y  
    fimse  
    retorne m  
fimfuncao
```

Interatividade

Quanto vale x ao término da função?

```
funcao S(n: inteiro) : inteiro  
var  
    x, i, j : inteiro  
inicio  
    x <- 0  
    para i de 1 até n passo 1 faca  
        para j de 1 ate 3 passo 1 faca  
            x <- x+1  
        fimpara  
    fimpara  
    retorne x  
fimfuncao
```

- a) 3.
- b) 5.
- c) 8.
- d) 12.
- e) 15.

```
Algoritmo  
Var  
    n : inteiro  
Inicio  
    escreva (S(5))  
Fimalgoritmo
```

Resposta

Quanto vale x ao término da função?

```
funcao S(n: inteiro) : inteiro  
var  
    x, i, j : inteiro  
inicio  
    x <- 0  
    para i de 1 até n passo 1 faca  
        para j de 1 ate 3 passo 1 faca  
            x <- x+1  
        fimpara  
    fimpara  
    retorne x  
fimfuncao
```

- a) 3.
- b) 5.
- c) 8.
- d) 12.
- e) 15.

```
Algoritmo  
Var  
    n : inteiro  
Inicio  
    escreva (S(5))  
Fimalgoritmo
```

Passagem de Parâmetros por Valor

`imprimirDados(nome, idade)`

Nome é uma variável do tipo caractere e idade, do tipo inteiro.

`imprimirDados("José", 73)`

Em vez de passar a variável como argumento do procedimento, é informado um valor. Nessa chamada de procedimento, José é um dado do tipo caractere e 73 é um dado do tipo inteiro.

`imprimirDados(nome, 50)`

O primeiro argumento é uma variável e o segundo é um número.

`imprimirDados("José", idade)`

O primeiro argumento é um dado do tipo caractere e o segundo é uma variável do tipo caractere.

- Na chamada do procedimento, o parâmetro pode ser um valor ou uma variável.
- O procedimento (ou a função) recebe uma cópia do valor.

Escopo de Variáveis: local ou global

- Variáveis locais são declaradas dentro do módulo e são reconhecidas apenas dentro do módulo. Quando o módulo é encerrado, as variáveis são destruídas.

x e y são
variáveis globais

Reconhecidas por todos
os módulos do
algoritmo.

```
1.  Algoritmo "Procedimento Menu e Operações"
2.  Var //declaração de variaveis globais
3.    x, y : inteiro
4.
5.  procedimento soma()
6.    var
7.      a, b : inteiro
8.    inicio
9.      escreval("Numero 1 : ")
10.     leia(a)
11.     escreval("Numero 2 : ")
12.     leia(b)
13.     escreval(a, " + ", b, " = ", a+b)
14.    fimprocedimento
15.
16.  //bloco principal
17.  Inicio
18.    leia(x,y)
19.    soma()
20.  Fimalgoritmo
```

a e b são
variáveis locais

Reconhecidas
apenas dentro do
módulo

O que acontece quando você modifica uma variável dentro do procedimento?

Saída do algoritmo

- Antes do procedimento:
x= 100 e y= 200
- Dentro do procedimento:
x= 120 e y= 230
- Depois do procedimento:
x= 100 e y= 200

```
1.  Algoritmo "Procedimento modificarDados"
2.  Var
3.  // variaveis globais
4.    x, y : inteiro
5.
6.  procedimento modificarDadosDoParametro(x, y : inteiro)
7.  inicio
8.    x <- x + 20
9.    y <- y + 30
10.   escreval("Dentro do procedimento: x= ", x, " e y= ", y)
11.  fimprocedimento
12.
13.
14.  //modulo principal
15.  Início
16.    x <- 100
17.    y <- 200
18.   escreval("Antes do procedimento: x= ", x, " e y= ", y)
19.
20.   modificarDadosDoParametro(x, y)
21.
22.   escreval("Depois do procedimento: x= ", x, " e y= ", y)
23.  Fimalgoritmo
```

Escopo de Variáveis: local ou global

- Variáveis locais são declaradas dentro do módulo e são reconhecidas apenas dentro do módulo. Quando o módulo é encerrado, as variáveis são destruídas.

Saída:

Antes – a: 30 e b: 15

Dentro – a: 15 e b: 30

Depois - a: 30 e b: 15

```
1.  Algoritmo "Passagem por Valor"
2.  Var //declaração de variaveis globais
3.    a, b : inteiro
4.
5.  procedimento troca(a, b : inteiro)
6.    var
7.      aux : inteiro
8.    inicio
9.      aux <- a
10.     a <- b
11.     b <- aux
12.     escreval(" Dentro do modulo a: ", a, " e b: ", b)
13.  fimprocedimento
14.
15.  //bloco principal
16.  Inicio
17.    a <- 30
18.    b <- 15
19.    escreval("Antes - a: ", a, " e b: ", b)
20.    troca(a,b)
21.    escreval("Depois- a: ", a, " e b: ", b)
22.  Fimalgoritmo
```

Proposta de Algoritmo

- Projetar um algoritmo para testar a divisibilidade de um número por 2, 3 ou 6, seguindo as regras clássicas da Matemática.
- Escrever um procedimento para testar a divisibilidade de um número. O procedimento deve receber o dividendo e o divisor como parâmetros de entrada e, caso o divisor seja 2, usar a função `divisibilidade2()`, caso seja 3 chamar a função `divisibilidade3()`; caso seja 6 chamar a função `divisibilidade 6` e, se outra opção for selecionadas, escrever a mensagem *“Opção Inválida!”*.

Proposta de Algoritmo

Regras da Divisibilidade que devem ser aplicadas:

- Um número é divisível por 3 quando a soma dos seus algarismos for divisível por três.

Exemplos:

- 274 não é divisível por 3 porque: $2+7+4 = 13 \rightarrow 1+3 = 4$.
- 25848 é divisível por 3 porque: $2+5+8+4+8 = 27 \rightarrow 2+7 = 9$.
- Um número é divisível por 2 quando é par, ou seja, quando o último dígito é: 0, 2, 4, 6 ou 8.
- Um número é divisível por 6 quando for simultaneamente divisível por 2 e por 3.
- **Oportunidade de reusar as funções!!!!**

Algoritmo Modularizado

Algoritmo "Divisibilidade"

Var

x, y : inteiro

funcao divisibilidade2(num : inteiro) : logico

var

x : inteiro

inicio

x <- num%10

retorne ((x=0) ou (x=2) ou (x=4) ou (x=6) ou (x=8))

fimfuncao

funcao divisibilidade3(num : inteiro) : logico

var

x, soma : inteiro

inicio

enquanto (num > 0) faca

x <- num mod 10 // pega o último dígito do número

soma <- soma + x

num <- num div 10 // retira o último dígito do numemro

se ((num<1) e (soma > 10)) entao

num <- soma

soma <- 0

fimse

fimenquanto

retorne ((soma = 3) ou (soma = 6) ou (soma = 9))

fimfuncao

Algoritmo Modularizado

```
funcao divisibilidade6(num : inteiro) : logico  
inicio  
    se (divisibilidade3(num) e divisibilidade2(num)) entao  
        retorne VERDADEIRO  
    senao  
        retorne FALSO  
    fimse  
fimfuncao
```

//bloco principal

Inicio

x <- 25848

y <- 71

testarDivisibilidade(x, 2)

testarDivisibilidade(x, 3)

testarDivisibilidade(x, 6)

testarDivisibilidade(y, 2)

testarDivisibilidade(y, 3)

testarDivisibilidade(y, 6)

Fimalgoritmo



 Console simulando o modo texto do MS-DOS

```
25848 é divisível por 2  
25848 é divisível por 3  
25848 é divisível por 6
```

```
71 NÃO é divisível por 2  
71 NÃO é divisível por 3  
71 NÃO é divisível por 6
```

```
>>> Fim da execução do programa !
```

Resumo

Benefícios da modularização

- Melhor organização do algoritmo.
- Blocos menores e independentes.
- Reduzem a complexidade do algoritmo.
- Facilita a compreensão do código.
- Propicia o reuso.

Procedimentos

- Não retornam nada.
- São executados numa linha de comando.
- A execução produz um efeito.

Funções

- Retornam um valor.
- O valor de retorno da função deve ser do mesmo tipo da função.
- Podem ser invocadas:
 - ✓ numa atribuição;
 - ✓ na condição de uma estrutura de decisão ou repetição;
 - ✓ como argumento da função escreva().
- A execução responde a uma questão.

Interatividade

Qual será a saída após a execução do bloco principal do algoritmo abaixo?

a) $x = -10$
 $n = 50$
 $x = 10$

c) $x = 50$
 $n = 50$
 $x = 50$

e) $x = 10$
 $n = 50$
 $x = 10$

b) $x = 10$
 $n = 10$
 $x = 50$

d) $x = 10$
 $x = 50$
 $x = 10$

Algoritmo "Teste"

Var

x : inteiro

procedimento f(n : inteiro)

inicio

$n \leftarrow n * 5$

escreval("n = ", n)

fimdoprocedimento

Inicio

$x \leftarrow 10$

escreval("x = ", x)

f(x)

escreval("x = ", x)

Fimalgoritmo

Resposta

Qual será a saída após a execução do bloco principal do algoritmo abaixo?

a) $x = -10$
 $n = 50$
 $x = 10$

c) $x = 50$
 $n = 50$
 $x = 50$

e) $x = 10$
 $n = 50$
 $x = 10$

b) $x = 10$
 $n = 10$
 $x = 50$

d) $x = 10$
 $x = 50$
 $x = 10$

Algoritmo "Teste"

Var

$x : \underline{\text{inteiro}}$

procedimento f($n : \text{inteiro}$)

inicio

$n \leftarrow n * 5$

escreval("n = ", n)

fimdoprocedimento

Inicio

$x \leftarrow 10$

escreval("x = ", x)

f(x)

escreval("x = ", x)

Fimalgoritmo

ATÉ A PRÓXIMA!