1 Introdução

ESQUEMA DO CAPÍTULO

1.5 Modularização

1.5.1. sub-rotinas

Ex. 1:

Fazer um algoritmo que leia três notas distintas, coloque-as em ordem crescente e imprima o resultado.

Solução:

Uma forma possível encontra-se desenvolvida a seguir.

```
algoritmo
    defina os tipos das variáveis
        {leia notas}
    leia nota1, nota2, nota3
    ordene notas
        {escreva resultados}
    escreva nota1, nota2, nota3
fim algoritmo
```

1.5 Modularização

1.5.1. sub-rotinas (cont.)

Solução (cont.):

```
ref.: ordene notas
        se (nota1>nota2) ou (nota1>nota3) então {nota1 não é a menor}
                se nota2<nota3 então      {nota2 é a menor}
                                {trocar nota1 e nota2 de posição}
                        aux ← nota1
                        nota1 ← nota2
                        nota2 ← aux
                senão                   {nota3 é a menor}
                                {trocar nota1 e nota3 de posição}
                        aux ← nota1
                        nota1 ← nota3
                        nota3 ← aux
                fim se
        fim se
        se (nota2>nota3) então {nota2 não é a menor}
                        {trocar nota2 e nota3 de posição}
                aux ← nota2
                nota2 ← nota3
                nota3 ← aux
        fim se
fim ref.
```

1.5 Modularização

1.5.1. sub-rotinas (cont.)

Solução (cont.):

```
ref.: defina tipo das variáveis
      declare nota1, nota2, nota3 numérico
fim ref.
```

Note-se, entretanto, que o grupo de comandos abaixo difere entre si apenas pelas variáveis envolvidas, mas a funcionalidade é a mesma (isto é, a função é trocar as variáveis de lugar).

```
aux ← nota1
nota1 ← nota2
nota2 ← aux
```

```
aux ← nota1
nota1 ← nota3
nota3 ← aux
```

```
aux ← nota2
nota1 ← nota3
nota3 ← aux
```

1.5 Modularização

1.5.1. sub-rotinas (cont.)

Solução (cont.):

Utilizando-se sub-rotinas, tem-se o algoritmo ao lado, mais simples e fácil de compreender.

```
algoritmo
        {declare a sub-rotina TROCA}
    subrotina TROCA(a,b)
        declare a, b, aux numérico
        aux ← a
        a ← b
        b ← aux
    fim subrotina
        {defina os tipos das variáveis}
    declare nota1, nota2, nota3 numérico
        {leia notas}
    leia nota1, nota2, nota3
        {ordene notas}
    se (nota1>nota2) ou (nota1>nota3) então {nota1 não
é a menor}
        se nota2<nota3 então {nota2 é a menor}
                {trocar nota1 e nota2 de posição}
            TROCA(nota1, nota2)
        senão               {nota3 é a menor}
                {trocar nota1 e nota3 de posição}
            TROCA(nota1, nota3)
        fim se
    fim se
    se (nota2>nota3) então {nota2 não é a menor}
            {trocar nota2 e nota3 de posição}
        TROCA(nota2, nota3)
    fim se
        {escreva resultados}
    escreva nota1, nota2, nota3
fim algoritmo
```

1.5 Modularização

1.5.2. funções

As funções permitem uma analogia com o conceito de função em matemática e retornam um valor através do seu nome.

A declaração de funções pode ser feita como se segue:

```
função tipo NOME(lista-de-parâmetros)
      declaração das variáveis locais
      comandos da função
fim função
```

Ex. 2:

Fazer um algoritmo que leia as medidas dos três lados de um paralelepípedo e escreva o valor de sua diagonal. Utilize o conceito de função.

1.5 Modularização

1.5.2. funções (cont.)

Solução:

Um algoritmos possível é apresentado a seguir.

```
algoritmo
        {declare a função HIPOTENUSA}
    função numérico HIPOTENUSA(a,b)
        declare a, b numérico
        HIPOTENUSA ← RAIZ(a*a+b*b)
    fim função
        {defina os tipos das variáveis}
    declare a, b, c, d, diag numérico
    leia a, b, c
        {calcule diagonal, método direto, em uma etapa}
    diag <- HIPOTENUSA(HIPOTENUSA(a,b), c)
        {escreva resultado}
    escreva "A diagonal é:", diag
        {calcule diagonal, método longo, em duas etapas}
    d <- HIPOTENUSA(a,b)
    diag <- HIPOTENUSA(d,c)
        {escreva resultado}
    escreva "A diagonal é:", diag
fim algoritmo
```