1 Introdução à Programação

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
```

1.5 Modularização

1.5.3. exercício

Utilizando conceitos de modularização, escreva um algoritmo que leia um vetor de tamanho arbitrário com as notas de *Introdução à Estatística Computacional*, assumindo que a nota -1 é o indicativo de final de dados, e um valor inteiro entre 0 e 100 (por exemplo, 75). O algoritmo deve então calcular e escrever o percentil representado por este valor (por exemplo, 75 representa o terceiro quartil).

Supor disponíveis recursos tais como a sub-rotina ORDENA(A,B), que recebe o vetor A e o retorna ordenado em B, a função ARREDONDA(b), que retorna o inteiro proveniente do arredondamento do real b e outras que você julgar necessárias.

1.5 Modularização

1.5.3. exercício (cont.)

Sugestão:

Parta da seguinte versão inicial do algoritmo, deixando por último a definição da função PERCENTIL(DADOS, PERC).

```
algoritmo
    defina a função PERCENTIL(DADOS,PERC)
    defina os tipos das variáveis
    leia notas
    leia o percentil desejado
    calcule o percentil
    escreva o percentil
fim algoritmo
```

1.5 Modularização

1.5.3. exercício (cont.)

Solução:

A seguir, apresentamos uma solução possível para o problema, partindo da versão inicial e refinando-se primeiramente os comandos ***mais simples***.

```
ref.: leia notas
      i ← 1
      leia notas[i]
      enquanto notas[i]<>-1 faça
            i ← i+1
            leia notas[i]
      fim enquanto
fim ref.
```

```
ref.: leia o percentil desejado
      leia percent
fim ref.
```

1.5 Modularização

1.5.3. exercício (cont.)

Solução (cont.):

```
ref.: calcule o percentil
      valorPercentil ← PERCENTIL(notas,percent)
fim ref.
```

```
ref.: escreva o percentil
      escreva valorPercentil
fim ref.
```

```
ref.: defina os tipos das variáveis
      declare i, notas[1:10000],
            percent, valorPercent numérico
fim ref.
```

1.5 Modularização

1.5.3. exercício (cont.)

Solução (cont.):

Finalmente, refinando-se a função PERCENTIL, conforme apresentado ao lado.

```
ref.: defina a função PERCENTIL(DADOS,PERC)
        funcao numérico PERCENTIL(DADOS, PERC)
                {defina tipos das variáveis}
            declare DADOS[:], DADOSORD[:] numerico
            declare PERC, tam, pos, posInt, posSup, PosInf numérico
                {ache tamanho do vetor de dados; -1 indica final do vetor}
            tam<-0
            enquanto DADOS[tam+1]<>-1 faça
                tam ← tam+1
            fim enquanto
                {ordene dados}
            ORDENA(DADOS, DADOSORD)
                {ache posição do percentil}
            pos ← (tam+1)*( PERC /100)
                {ache percentil}
            posInt ← ARREDONDA(pos)
            se pos=posInt então {a posição já é inteira}
                PERCENTIL ← DADOSORD[posInt]
            senão {se posição não é inteira, faça uma interpolação}
                posSup ← ARREDONDA(pos+0,5)
                posInf ← ARREDONDA(pos-0,5)
                PERCENTIL <- DADOSORD[posInf] +
                        (DADOSORD[posSup]- DADOSORD[posInf])*(pos-posInf)
            fim se
        fim funcao
fim ref.
```

1.5 Modularização

1.5.3. exercício (cont.)

Solução (final):

Inserindo-se os refinamentos no algoritmo, tem-se ao lado o algoritmo completo.

```
algoritmo
        {defina a função PERCENTIL(DADOS,PERC)}
    funcao numérico PERCENTIL(DADOS, PERC)
            {defina tipos das variáveis}
        declare DADOS[:], DADOSORD[:] numerico
        declare PERC, tam, pos, posInt, posSup, PosInf numérico
            {ache tamanho do vetor de dados; -1 indica final do vetor}
        tam<-0
        enquanto DADOS[tam+1]<>-1 faça
            tam <- tam+1
        fim enquanto
            {ordene dados}
        ORDENA(DADOS, DADOSORD)
            {ache posição do percentil}
        pos <- (tam+1)*( PERC /100)
            {ache percentil}
        posInt <- ARREDONDA(pos)
        se pos=posInt então {a posição já é inteira}
            PERCENTIL <- DADOSORD[posInt]
        senão {se posição não é inteira, faça uma interpolação}
            posSup <- ARREDONDA(pos+0,5)
            posInf <- ARREDONDA(pos-0,5)
            PERCENTIL <- DADOSORD[posInf] +
              (DADOSORD[posSup]- DADOSORD[posInf])*(pos-posInf)
        fim se
    fim funcao
            {defina os tipos das variáveis}
    declare i, notas[1:10000], percent, valorPercent numérico
            {leia notas}
    i<-1
    leia notas[i]
    enquanto notas[i]<>-1 faça
            i <- i+1
            leia notas[i]
    fim enquanto
            {leia o percentil desejado}
    leia percent
            {calcule o percentil}
    valorPercentil <- PERCENTIL(notas, percent)
            {escreva o percentil}
    escreva valorPercentil
fim algoritmo
```

1.5 Modularização

1.5.4. exercício

a) Escreva um algoritmo que leia notas finais e frequência (S ou I) de alunos, escreva a nota final, a frequência e o conceito correspondente (A, B, C, D, E ou F, de acordo com os valores utilizados na UFMG). Após o último aluno, que terá nota -1, escrever uma tabela com o número de alunos que tiraram cada conceito.

Obs.: O cálculo do conceito ***deve ser implementado por meio de uma sub-rotina***, que receba a nota e a frequência e retorne o conceito correspondente.

b) Que modificação(ções) teria(m) que ser feita(s) nesta sub-rotina, para que a mesma fosse transformada em uma função que recebesse a nota e a frequência e retornasse o conceito?

1.5 Modularização

1.5.4. exercício (cont.)

Solução:

A seguir, apresentamos uma solução possível para a parte a) do problema, partindo da seguinte versão inicial.

```
algoritmo
        declare subrotina para cálculo do conceito
        defina os tipos das variáveis
        inicialize variáveis
        leia nota, freq
        enquanto nota <>-1 faça
                calcule conceito
                escreva nota,freq, conc
                contribuir para tabela de frequências
                leia nota, freq
        fim enquanto
        escreva tabela de frequências
fim algoritmo
```

1.5 Modularização

1.5.4. exercício (cont.)

Solução (cont.):

Temos a seguir os refinamentos dos comandos, iniciando-se pelos mais simples.

```
ref.: calcule conceito
      CONCEITO(nota, freq, conc)
fim ref.
```

```
ref.: escreva tabela de frequências
      para i de 1 até 6 faça
            escreva tab[i]
      fim para
fim ref.
```

```
ref.: declare subrotina para cálculo do conceito
      subrotina CONCEITO(valor, freq, result)
            {declare variáveis locais}
        declare valor numérico
        declare freq, result caracter
            {calcule conceito}
        se freq = "I" então
            result ← "F"
        senão se valor < 40 então
            result ← "F"
        senão se valor < 59 então
            result ← "E"
        senão se valor < 69 então
            result ← "D"
        senão se valor < 79 então
            result ← "C"
        senão se valor < 89 então
            result ← "B"
        senão
            result ← "A"
        fim se fim se fim se fim se fim se fim se
      fim subrotina
fim ref.
```

1.5 Modularização

1.5.4. exercício (cont.)

Solução (cont.):

```
ref.: inicialize variáveis
      para i de 1 até 6 faça
            tab[i] ← 0
      fim para
fim ref.
```

```
ref.: defina os tipos das variáveis
      declare nota, i, tab[1:6] numérico
      declare freq, conc       literal
fim ref.
```

```
ref.: contribuir para tabela de frequências
      se conc = "F" então
            tab[6] ← tab[6] + 1
      senão se conc = "E" então
            tab[5] ← tab[5] + 1
      senão se conc = "D" então
            tab[4] ← tab[4] + 1
      senão se conc = "C" então
            tab[3] ← tab[3] + 1
      senão se conc = "B" então
            tab[2] ← tab[2] + 1
      senão se conc = "A" então
            tab[1] ← tab[1] + 1
      fim se fim se fim se fim se fim se fim se
fim ref.
```

1.5 Modularização

1.5.4. exercício (cont.)

Solução (cont):

Finalmente, inserindo-se os refinamentos, temos o algoritmo completo ao lado.

```
algoritmo
                {declare subrotina para cálculo do conceito}
    subrotina CONCEITO(valor, freq, result)
                {declare variáveis locais}
        declare valor numérico
        declare freq, result literal
                {calcule conceito}
        se freq = "I" então
                result ← "F"
        senão se valor < 40 então
                result ← "F"
        senão se valor < 59 então
                result ← "E"
        senão se valor < 69 então
                result ← "D"
        senão se valor < 79 então
                result ← "C"
        senão se valor < 89 então
                result ← "B"
        senão
                result ← "A"
        fim se fim se fim se fim se fim se fim se
    fim subrotina
                {defina os tipos das variáveis}
    declare nota, i, tab[1:6]       numérico
    declare freq, conc   literal
                {inicialize variáveis}
    para i de 1 até 6 faça
                tab[i] <- 0
    fim para
                {processe}
    leia nota, freq
    enquanto nota<>-1 faça
                {calcule conceito}
        CONCEITO(nota, freq, conc)
        escreva nota, freq, conc
                {contribuir para tabela de frequências}
        se conc = "F" então
                tab[6] ← tab[6] + 1
        senão se conc = "E" então
                tab[5] ← tab[5] + 1
        senão se conc = "D" então
                tab[4] ← tab[4] + 1
        senão se conc = "C" então
                tab[3] ← tab[3] + 1
        senão se conc = "B" então
                tab[2] ← tab[2] + 1
        senão se conc = "A" então
                tab[1] ← tab[1] + 1
        fim se fim se fim se fim se fim se fim se
        leia nota, freq
    fim enquanto
                {escreva tabela de frequências}
    para i de 1 até 6 faça
                escreva tab[i]
    fim para
fim algoritmo
```

1.5 Modularização

1.5.4. exercício (cont.)

Solução (final):

b) Implementando-se o cálculo do conceito por uma ***função***, teríamos o refinamento ao lado, que teria uma chamada um pouco diferente da chamada à sub-rotina,

```
ref.: calcule conceito (via sub-rotina)
        CONCEITO(nota, freq, conc)
fim ref.
```

como se segue.

```
ref.: calcule conceito (via função)
        conc ← CONCEITO(nota, freq)
fim ref.
```

```
ref.: declare função para cálculo do conceito
        função caracter CONCEITO(valor, freq)
                {declare variáveis locais}
            declare valor numérico
            declare freq, result literal
                {calcule conceito}
            se freq = "I" então
                result ← "F"
            senão se valor < 40 então
                result ← "F"
            senão se valor < 59 então
                result ← "E"
            senão se valor < 69 então
                result ← "D"
            senão se valor < 79 então
                result ← "C"
            senão se valor < 89 então
                result ← "B"
            senão
                result ← "A"
            fim se fim se fim se fim se fim se fim se
                {retorne resultado}
            CONCEITO ← result
        fim subrotina
fim ref.
```