Lógica de Programação e Algoritmos

Profa. Eliane Oliveira Santiago

Modularização

É a divisão do algoritmo em módulos ou sub-rotinas, para que o problema dividido em subproblemas possa ser facilmente interpretado e desenvolvido.

Temos dois tipos de módulos:

procedimentos e **funções.**

Necessidades de modularização

Os principais benefícios para o uso da modularização são:

- a divisão do algoritmo em módulos, para que ele seja melhor interpretado e desenvolvido; e
- a reutilização de código.

Os módulos acabam sendo mais simples e menos complexos.

A divisão em módulos facilita o entendimento parcial do problema e entendendo todos os problemas parciais, no final, teremos entendido o problema maior.

A maioria dos módulos pode ser vista como um minialgoritmo, ou seja, com entrada de dados, processamento desses dados e saída de resultados.

Modularização
dividir o problema em subproblemas.

Para construir os módulos, primeiro precisamos analisar o problema e dividi-lo em partes principais, que são os módulos; depois precisamos analisar os módulos obtidos para verificar se a divisão está coerente. Se algum módulo ainda estiver complexo, devemos dividi-lo também em outros submódulos. Por fim, precisamos analisar todos os módulos e o problema geral para garantir que o algoritmo mais os módulos resolvam o problema de forma simples.

Esse processo é chamado de **Método de Refinamento Sucessivo**, pois o problema complexo é dividido em problemas menores e, a partir do resultado dos problemas menores, teremos o resultado do problema complexo.

Procedimentos

Os procedimentos são aqueles módulos que executam um conjunto de comandos sem retorno para o módulo que o chamou.

```
procedimento <nome do módulo>(<lista de parametros)
início
      <comandos>;
fim_procedimento
```

A lista de parâmetros poderá ser vazia.

Exemplo: procedimento menu()

procedimento imprimir_dados(n1, n2 : inteiro)

Procedimentos

```
procedimento imprimir_dados()
início
  disc, fac : Cadeia
  disc ← "Lógica de Programação"
  fac ← "UNIP"
      escreva("nome da disciplina => " , disc);
      escreva("nome da instituição de ensino =>" , fac);
fim_procedimento
```

Neste exemplo, **imprimir_dados** é o nome do procedimento. O módulo parece um mini-algoritmo com declaração de variáveis, entrada de dados para essas variáveis e a saída de resultados. O que difere um algoritmo de um módulo é justamente sua simplicidade.

Procedimentos

```
procedimento imprimir_dados() //void imprimir_dados(){..}
início
  disc, fac : Cadeia //o tipo Cadeia é correspondente ao String, mas não existe o pseudocódigo
  disc ← "Lógica de Programação"
  fac ← "UNIP"
      escreva("nome da disciplina => " , disc);
      escreva("nome da instituição de ensino =>" , fac);
fim_procedimento
```

Neste exemplo, **imprimir_dados** é o nome do procedimento. O módulo parece um mini-algoritmo com declaração de variáveis, entrada de dados para essas variáveis e a saída de resultados. O que difere um algoritmo de um módulo é justamente sua simplicidade.

Funções

As funções são aqueles módulos que executam um conjunto de comando e retornam algum dado para o módulo que o chamou.

```
<tipo de retorno> <nome_da_função>( )
início
      <comandos>;
      retornar <valor de retorno>;
fim_função
```

Funções

```
funcao imprimir_dados( ) : Cadeia
início
      disc, fac, mens : Cadeia
      disc ← "Lógica de Programação"
      fac ← "Unip"
      mens ← "nome da disciplina => " + disc
             + "nome da instituição de ensino => "
             + fac
       retorne mens;
fim_função
```

Note que o tipo de retorno do módulo função **imprimir_dados** é **alfanumérico** e o retorno do módulo é o conteúdo da variável **mens** do tipo **alfanumérico**, ou seja, do mesmo tipo de dados, isso é obrigatório.

Variável global

Uma **variável global** é aquela declarada no início do algoritmo e pode ser utilizada por qualquer parte desse algoritmo, seja nos comandos do próprio algoritmo, bem como, dentro de qualquer módulo que pertença ao algoritmo.

Nesse caso, **sua declaração é feita apenas uma única vez,** não sendo permitido que o mesmo nome de variável seja declarado dentro de qualquer outra parte do algoritmo, por exemplo, dentro de qualquer outro módulo.

Variável local

Uma **variável local** é aquela declarada dentro de algum bloco, por exemplo, dentro de um módulo.

Nesse caso, **essa variável é válida e reconhecida somente dentro do bloco em que foi declarada**. Assim, o mesmo nome de variável pode ser declarado dentro de diferentes módulos (procedimentos ou funções), pois serão reconhecidas como uma nova variável.

Escopo de variáveis: local ou global

O escopo de uma variável é onde, dentro do algoritmo, uma variável é válida ou pode ser reconhecida.

Por exemplo, se declararmos uma variável **x** no início de um algoritmo, essa variável **x** pode ser usada e alterada em qualquer parte desse algoritmo e em nenhum momento declarada novamente, ou seja, ela é única no algoritmo inteiro. Mas é muito importante ter o controle dessas variáveis globais, justamente porque elas podem ser alteradas a qualquer momento.

No entanto, se declararmos, usarmos e alterarmos uma variável **y** dentro do módulo soma, poderemos também declará-la, utilizá-la e alterá-la dentro do módulo subtração, do módulo divisão e do módulo multiplicação, se assim o desejarmos. Nesse caso, não precisando tomar os cuidados necessários que uma variável global precisa.

Parâmetros

O uso de argumentos passados como parâmetros em módulos, sejam eles funções ou procedimentos é muito comum.

Os exemplos vistos anteriormente para procedimentos e funções estão sem o uso de parametrização de módulos, por isso, após o nome dos módulos, os parênteses estão vazios.

É dentro dos parênteses que os argumentos são passados como parâmetros aos módulos.

```
<tipo de retorno><nome do módulo> (<var1>,  <var2> : tipo1, <var3> :<tipo2>)
início_módulo
      <comandos>
      <retorne> valor
fim_módulo;
```

Parametrização em procedimentos
(valores de entrada pro módulo)

A quantidade de argumentos que pode ser passada como parâmetro não é determinada, pondendo ser um único argumento ou uma quantidade finita de argumentos.

```
procedimento imprimir_dados (disc, fac : Cadeia)
início
    escreva("nome da disciplina => " , disc)
    escreva("nome da instituição de ensino => " , fac)
fim_procedimento
```

Neste exemplo, não é mais necessária a declaração das variáveis **disc** e **fac** dentro do módulo **Dados**, pois esses dados estão declarados no cabeçalho do módulo como argumentos que receberão dados que serão passados como parâmetros na chamada desse módulo.

Parametrização em funções
(valores de entrada pro módulo)

A quantidade de argumentos que pode ser passada como parâmetro não é determinada, pondendo ser um único argumento ou uma quantidade finita de argumentos.

```
real divisao(x: real, y: inteiro)
Início
    resultado : real
    resultado ← x/y
    retorne resultado
fim_procedimento
```

Neste exemplo, não é mais necessária a declaração das variáveis **disc** e **fac** dentro do módulo **Dados**, pois esses dados estão declarados no cabeçalho do módulo como argumentos que receberão dados que serão passados como parâmetros na chamada desse módulo.

Chamada de procedimento
Exemplos

```
//chamada de procedimento                    ⇔ voz de comando
imprimir_dados()
imprimir_dados("LPA", "Unip")

//chamada de função      ⇔ //responde a uma pergunta
a ← divisao(10, 5)
leia(x,y)    //x e y variáveis inteiras
se(soma(x,y)>100) então
      //faça alguma coisa.
fimse
```

Resumo

Procedimentos

```
//sem parâmetros de entrada
procedimento nome()
:
Fimprocedimento
```

```
//com parâmetros de entrada
procedimento nome(x: inteiro)
:
fimprocedimento
```

Funções

```
//sem parâmetros de entrada
função nomeFuncao() : tipo
    var valor : tipo
      :
    retorne valor
Fimfuncao
```

```
//com parâmetros de entrada
função nomeFuncao(x: inteiro) : tipo
  var valor : tipo
      :
    retorne valor
fimfuncao
```

Bibliografias

BÁSICA

- GOMES, Ana Fernanda A. Campos, Edilene Aparecida V. Fundamentos da Programação de Computadores – Algoritmos, Pascal e C/C++. Prentice Hall, 2007.
- CARBONI, Irenice de Fátima. Lógica de Programação. Thomson.
- XAVIER, Gley Fabiano Cardoso. Lógica de Programação - Cd-rom. Senac São Paulo – 2007.

COMPLEMENTAR

- FORBELLONE, André Luiz Villar. Eberspache, Henri Frederico. Lógica de Programação – A construção de Algoritmos e Estrutura de Dados. Makron Books, 2005.
- LEITE, Mário - Técnicas de Programação – Brasport - 2006.
- PAIVA, Severino – Introdução à Programação – Ed. Ciência Moderna – 2008.
- PAULA, Everaldo Antonio de. SILVA, Camila Ceccatto da. Lógica de Programação –Viena – 2007.
- CARVALHO, Fábio Romeu, ABE, Jair Minoro.Tomadas de decisão com ferramentas da lógica paraconsistente anotada: Método Paraconsistente de Decisão (MPD), Editora Edgard Blucher Ltda. - 2012.