PROCEDIMENTOS, FUNÇÕES E MÉTODOS

Disciplina de Lógica de Programação
Curso Técnico em Informática para Internet

Motivação

• A complexidade dos algoritmos está intimamente ligada à da aplicação a que se destinam. Em geral, problemas complicados exigem algoritmos extensos e de lógica mais complexa para sua solução.

• Sempre é possível dividir um problema grande em problemas menores e de solução mais simples. Assim, podemos solucionar cada um destes problemas mais simples e combinar essas soluções de forma que o problema maior seja resolvido.

Exemplo

- Um carro é montado utilizando diversas partes (motor, carroceria, para-brisas, pneus, etc.).
- Cada parte é fabricada por uma empresa diferente.
- Durante a fabricação de uma parte, a empresa se foca em resolver o problema específico dela.
  - Um fabricante de pneus se preocupa em fazer pneus mais seguros e resistentes, mas não se preocupa com a potência do motor.
- Mas quando se juntam todas as partes, temos um carro completo, que faz mais do que um pneu e um motor.

Lógica de um algoritmo é complexa?

- Quando a lógica de um algoritmo é complexa, podemos dividi-la em partes mais simples, e resolver cada parte separadamente.
- Ao final, podemos juntá-las e obter o algoritmo completo, que resolve o problema maior.
- Podemos codificar a solução destes problemas simples utilizando os chamados sub-algoritmos ou sub-rotinas.

O que é Sub-algoritmo?

- Um sub-algoritmo é um nome dado a um pequeno algoritmo, que pode ser reutilizado como parte de um algoritmo mais complexo.
- Por exemplo, um sub-algoritmo que calcule o quadrado de um número pode ser utilizado:
  - em um algoritmo que calcule a área de um quadrado
  - em um algoritmo que calcule o valor de y em uma função de segundo grau x^2+5x-6 dado o valor de x
  - e/ou em um algoritmo que verifique se três números formam lados de um triângulo retângulo.

Importância dos Sub-algoritmos

- subdivisão de algoritmos complexos, facilitando o seu entendimento;
- estruturação de algoritmos, facilitando principalmente a detecção de erros e a documentação de sistemas; e
- modularização de sistemas, que facilita a manutenção de softwares e a reutilização de subalgoritmos já implementados.

Reutilização de Software

- A ideia de reutilização de software tem sido adotada há muito tempo, por muitos desenvolvedores de sistemas de computador, devido à economia de tempo e trabalho que proporcionam.
- Seu princípio é o seguinte: um conjunto de algoritmos destinado a solucionar uma série de tarefas bastante corriqueiras é desenvolvido e vai sendo aumentado com o passar do tempo, com o acréscimo de novos algoritmos.

Bibliotecas de Software

- A este conjunto de algoritmos dá-se o nome de biblioteca, formada por um conjunto de vários sub-algoritmos.
- Hoje em dia, no desenvolvimento de novos sistemas, procura-se ao máximo basear sua concepção em sub-algoritmos já existentes em uma biblioteca, de modo que a quantidade de software realmente novo que deve ser desenvolvido é minimizada.
- Muitas vezes os sub-algoritmos são utilizados para conter certas sequências de instruções que são repetidas em um algoritmo.
- Nestes casos, os sub-algoritmos proporcionam uma diminuição no tamanho dos algoritmos.

Definição de Sub-algoritmos em Pseudocódigo

```
Algoritmo nome do algoritmo

var
declaração de variáveis

definição de sub-algoritmos

início
     instruções
fim
```

Definição de Sub-algoritmo

• A definição de um sub-algoritmo é formada por duas partes:

•um cabeçalho, onde são definidos o nome do sub-algoritmo, as variáveis locais, os parâmetros utilizados e o tipo do sub-algoritmo;

•o corpo do sub-algoritmo, onde estarão as instruções para a resolução do subproblema.

Algumas regras dos Sub-algoritmos

- O nome dado a um sub-algoritmo segue as mesmas regras impostas às variáveis e deve ser único.
- As variáveis locais são variáveis definidas e utilizadas dentro do sub-algoritmo.
- Os parâmetros são dados recebidos e/ou enviados ao algoritmo/sub-algoritmo que utiliza a sub-rotina definida.

Tipos de sub-algoritmos

- Basicamente existem dois tipos de sub-algoritmos, diferenciados pelo número de valores retornados ao algoritmo principal:
  - as funções são sub-algoritmos que retornam um ou mais valores ao algoritmo/sub-algoritmo que o utiliza;
  - os procedimentos são sub-algoritmos que retornam zero ou mais valores ao algoritmo/subalgoritmo que o utiliza.

Funções

- O conceito de função em algoritmos é fortemente baseado no conceito de função matemática (como seno, cosseno, tangente, etc).
- Uma função serve para realizar determinado processamento e retornar um dado valor como resultado ao algoritmo/sub-algoritmo que a utiliza.

Sintaxe de uma função

```
função nome da função(lista de parâmetros): tipo de retorno
    declaração de variáveis locais
início
    conjunto de instruções
fim
```

Exemplo de função 1

```
Algoritmo "Quadrado"
var
      N : real

      função Quad(num : real) : real
      var
            resultado : real
      início
            resultado ← num * num
            retorne resultado
      fim

início
      leia N
      escreva Quad(N)
fim
```

Parâmetro recebido pela função

Tipo de retorno
Da função

Serve para especificar o valor
de retorno da função

Utiliza a função Quad(N)

Funcionamento de função

- O valor retornado pode ser especificado como um valor, uma variável, constante ou expressão do mesmo tipo do tipo de retorno especificado.
- Para utilizar uma função, basta escrever seu nome seguido por uma listagem de valores separados por vírgulas, entre parênteses. Esses valores serão associados aos parâmetros durante a execução da função.
- Quando o fluxo de execução de um algoritmo chega a uma expressão ou comando que utiliza uma função, o fluxo de execução é temporariamente redirecionado para o início do conjunto de instruções da função.

Funcionamento de função

- Essas instruções são executadas, até que se chegue a um comando retorne. Terminada a execução da função, o fluxo retorna ao ponto onde havia parado antes da execução da função e o valor retornado é utilizado em lugar do nome da função.
- Uma observação importante a ser mencionada é que quando uma instrução retorne é executada, a execução da função termina, independente se existe ou não instruções dentro da função após o retorne.

Exemplo de função 2

```
Algoritmo "Exemplo Retorne"
var
      numero : inteiro

      função FuncA(N : inteiro) : literal
      var
            str : literal
      inicio
            str ← "Teste"
            retorne str + N
            escreva "Mensagem Oculta"
      fim

inicio
      escreva "Informe um número inteiro: "
      leia numero
      escreva FuncA(numero)
fim
```

Procedimentos

- Um procedimento diferencia-se da função por não se especificar um tipo de retorno.

Sintaxe de Procedimentos

```
procedimento nome do procedimento ( lista de parâmetros )
    declaração de variáveis locais
inicio
    conjunto de instruções
fim
```

Exemplo de procedimento 1

```
Algoritmo "Histograma"
var
      n1, n2, n3, n4, n5 : inteiro
      procedimento MostraColuna(N : inteiro)
      var
            cont : inteiro
      inicio
            para cont de 1 até N faça
                  escreva "*"
            fim para
      fim

inicio
      leia n1, n2, n3, n4, n5
      MostraColuna(n1)
      MostraColuna(n2)
      MostraColuna(n3)
      MostraColuna(n4)
      MostraColuna(n5)
fim
```

Explicando procedimentos

- O algoritmo utiliza um procedimento para desenhar cada linha, a partir de um número inteiro.
- O procedimento é utilizado para mostrar a linha correspondente a cada número informado.
- Logo, ele é executado cinco vezes, como podemos ver no exemplo.
- Para executarmos um procedimento basta escrevermos seu nome e a listagem de parâmetros, se houver, entre parênteses.
- Como os procedimentos não retornam valores, não podem ser usados em expressões, como as funções.

Função ou Procedimento?

- Usar função quando se tem interesse num valor resultante do processamento.
  - Por exemplo: dado um número, verificar se ele é primo. Neste caso, o mais adequado seria o emprego de uma função cujo retorno é do tipo lógico, pois se deseja uma resposta entre “sim” (V) e “não” (F).
- Usar procedimento se o sub-algoritmo não precisa produzir um valor como resultado.
  - Por exemplo: mostrar um texto na tela.

Variáveis Globais e Locais

• Variáveis globais são aquelas declaradas no início de um algoritmo. Estas variáveis são visíveis (isto é, podem ser usadas) no corpo do algoritmo e por todos os sub-algoritmos definidos dentro dele.

• Variáveis locais são aquelas definidas dentro de um sub-algoritmo e, portanto, somente visíveis (utilizáveis) dentro do mesmo. Outros sub-algoritmos ou mesmo instruções do corpo do algoritmo não podem utilizá-las.

Exemplo de Escopo de Variáveis

```
Algoritmo "Exemplo Variáveis"
var
      X : real
      I : inteiro

      função Func() : real
      var
            X, Y : literal
      inicio
            ...
      fim

      procedimento Proc()
      var
            Y : literal;
      inicio
            ...
            X ← 4 * X
            I ← I + 1
            ...
      Fim

inicio
      ...
      X ← 3.5
      ...
fim
```

Exemplo de Escopo de Variáveis

- As variáveis X e I são globais e visíveis a todos os sub-algoritmos, com exceção da função Func, que redefine a variável X localmente;
- As variáveis X e Y, locais à função Func, não são visíveis no corpo do algoritmo ou ao procedimento Proc. A redefinição local no nome simbólico X como uma variável do tipo literal sobrepõe (somente dentro da função Func) a definição global de X como uma variável do tipo real;
- A variável Y dentro do procedimento Proc, que é diferente daquela definida dentro da função Func, é invisível fora deste procedimento;
- A instrução X ← 3.5 no corpo do algoritmo, bem como as instruções X ← 4 * X e I ← I + 1 dentro do procedimento Proc, atuam sobre as variáveis globais X e I.

Parâmetros

- Parâmetros são canais pelos quais se estabelece uma comunicação bidirecional entre um subalgoritmo e o algoritmo chamador (o algoritmo principal ou outro sub-algoritmo que o utiliza).
- Dados são passados pelo algoritmo chamador ao sub-algoritmo, ou retornados por este ao primeiro por meio de parâmetros.
- Parâmetros formais são os nomes simbólicos introduzidos no cabeçalho de sub-algoritmos, usados na definição dos parâmetros do mesmo.
- Dentro de um sub-algoritmo trabalha-se com estes nomes da mesma forma como se trabalha com variáveis locais ou globais.

Exemplo de parâmetros 1

```
Algoritmo "Media Com Função"
var
      Z : real

      função Media(X, Y : real) : real
      inicio
            retorne ( X + Y ) / 2
      fim

inicio
      Z ← Media( 8, 7 )
      escreva Z
fim
```

Parâmetros Formais

Parâmetros Reais

Parâmetros de tipos diferentes

- Caso o sub-algoritmo possua parâmetros com tipos diferentes, estes devem ser separados por ponto-e-vírgula.
- Por exemplo, se a função Media fosse definida com um parâmetro inteiro e um real, o seu cabeçalho seria:

```
função Media( X : inteiro; Y : real) : real
```

Parâmetros Formais e Reais

- Os parâmetros formais são utilizados somente na definição (formalização) do subalgoritmo.
- Os parâmetros reais substituem os formais a cada invocação do sub-algoritmo.
- Note que os parâmetros reais podem ser diferentes a cada invocação de um sub-algoritmo.
- Um detalhe a ser observado: o parâmetro real a ser utilizado no lugar de um parâmetro formal, durante a invocação de um sub-algoritmo, deve ser de um tipo equivalente.
- Por exemplo, é um erro passar um dado do tipo literal, como “ABC”, no lugar do parâmetro formal X da função Media, pois o dado e o parâmetro possuem tipos que não são equivalentes.

Construindo Métodos em Java

Um método em Java é um conjunto de instruções que, assim como os procedimentos e funções, pode receber dados externos (os parâmetros) e devolver resultados (como as funções).

Métodos Estáticos

- Estes métodos são utilizados diretamente a partir das classes em que foram definidos e não necessitam de instâncias para serem executados.
- Todos os programas que escrevemos até agora utilizavam uma **abordagem chamada monolítica**.
- Nesta abordagem, todo o código do programa é escrito em um único conjunto de instruções.
- A partir de agora, escreveremos programas que utilizam a **abordagem modular**.
- Iremos escrever programas que dividem seu código em diversos procedimentos e funções, utilizando o conceito de métodos.

Definindo Métodos

- Assim como nos algoritmos, para criarmos um método em Java precisamos defini-lo dentro de uma classe.
- Como todos os programas em Java são classes, podemos definir métodos dentro das classes de nossos programas.

Formato geral de uma definição de método estático em Java

```
static tipo de retorno nome do método ( parâmetros )
{
    instruções
}
```

Formato geral de uma definição de método estático em Java

- A palavra static é usada para especificar que o método definido é estático.
- Por enquanto, em todos os métodos que escrevermos, utilizaremos essa palavra.
- O restante da definição do método é semelhante à definição de uma função na forma:

```
função nome do método ( parâmetros ) : tipo de retorno
início
      instruções
fim
```

Observações sobre métodos em Java

• Como todas as definições de métodos exigem um valor de retorno, todos os métodos em Java se comportam como funções.

• Mas sendo assim, como implementar procedimentos em Java?

• Para contornar esse problema, existe um tipo especial em Java chamado **void**.

• O tipo void, quando usado como tipo de retorno de um método, especifica que o método não irá retornar valores.

• Logo, se desejamos implementar um procedimento em Java, basta criarmos um método cujo tipo de retorno é void.

Parâmetros de metódos em Java

- A lista de parâmetros na definição de métodos em Java é uma lista separada por vírgulas, na qual o método declara o nome e o tipo de cada parâmetro.
- Diferentemente dos algoritmos, para cada parâmetro deve ser especificado seu tipo, mesmo que existam vários parâmetros consecutivos com o mesmo tipo.
- Por exemplo, um método chamado media que calcula a média de dois números reais, poderia ser definido da seguinte forma:

```
static double media ( double a, double b )
{
      ...
}
```

Exemplo de métodos em Java

```
1     public class Quadrado2 {
2
3           static double quad( double num ) {
4                 double resultado;
5                 resultado = num * num;
6                 return resultado;
7           } // fim do metodo quad
8
9           public static void main(String[] args) {
10                double N;
11
12                System.out.print("Numero: ");
13                N = Double.parseDouble(System.console().readLine());
14
15                System.out.printf("O quadrado do número é %f\n",
16                                                quad(N));
17
18          } // fim do metodo main
19
20    } // fim da classe Quadrado2
```

Variável local

Retorno do método

Chamada do método quad(N)

Observações sobre métodos

- Olhando para a definição de main, podemos perceber que ele também é um método estático que recebe um parâmetro e não retorna valor (pois o tipo de retorno é void).
- Todo programa Java é uma classe que define um método chamado main.
- Podemos definir classes sem main, mas essas classes não podem ser executadas como programas.
- Como main é um método, toda variável declarada dentro dele é uma variável local ao método. Ou seja, não podemos utilizar variáveis declaradas dentro de main em outros métodos da classe.

Observações sobre métodos

- Por exemplo, utilizar a variável N declarada no método main do Programa Quadrado2 seria um erro, pois ela é visível somente dentro do método main.
- Em outras palavras, variáveis declaradas dentro do método main não são globais.
- Note que na linha 16 (Programa Quadrado2) não escrevemos o nome da classe antes do nome do método, quando o invocamos.
- Quando um método é chamado dentro da própria classe em que é definido, isso não é necessário.