Universidade Federal do Vale do São Francisco
Curso de Engenharia de Computação

Introdução a Algoritmos – Parte 05

Prof. Jorge Cavalcanti
jorge.cavalcanti@univasf.edu.br
www.univasf.edu.br/~jorge.cavalcanti
www.twitter.com/jorgecav

Introdução a Algoritmos

- **Modularização**
  - Com o avanço do estudo sobre algoritmos os problemas a serem solucionado aumentam em complexidade.
  - Um método de resolução de problemas complexos é visualizá-los como compostos por problemas menores e tratar um a um os sub-problemas identificados.
  - Podemos desta forma construir um algoritmo composto por subalgoritmos denominados módulos.
  - Esta metodologia é denominada top-down (de cima para baixo), ou melhor, do genérico para o específico.

Introdução a Algoritmos

- **Modularização**
  - Desta forma ao nos depararmos com um problema complexo devemos buscar visualizá-lo como um conjunto de problemas mais simples.
  - Trataremos agora de como construir um algoritmo composto por módulos.
  - Assim como um algoritmo, que em geral possui um conjunto de entradas, efetua um processamento e gera um conjunto de saídas, um módulo também funciona da mesma forma.

Introdução a Algoritmos

- Da mesma forma com que o pseudocódigo define uma estrutura para construção de algoritmos, também é especificada uma estrutura para construção de módulos.
- Trabalharemos com dois tipos de módulos:
  - A função;
  - O procedimento.
- **Conceito de função.**
  - Uma função é um módulo que possui ou não um conjunto de entradas, efetua a execução de um conjunto de instruções e sempre gera um valor como **saída**, também denominado **retorno**.

Introdução a Algoritmos

- A estrutura de uma função é a seguinte:

funcao *<nome_da_função>* ([*<seqüência-de-declarações-de-parâmetros>*]) : *<tipo_de_dado>*
  // Seção de Declarações Internas
inicio
  // Seção de Comandos
fimfuncao

Obs.: A declaração de uma função deve estar entre o final da declaração de variáveis e a linha contendo **inicio** no algoritmo principal.

Introdução a Algoritmos

- O **tipo_de_dado** é o tipo do valor que a função vai retornar.
- A **seqüência-de-declarações-de-parâmetros** é uma lista com a seguinte forma geral:

  identificador1: tipo_de_dado; identificador2: tipo_de_dado; ...; identificadorN: tipo_de_dado

- Repare que o tipo_de_dado deve ser especificado para cada uma das N variáveis definidas como parâmetros, independente de mais de uma variáveis ser do mesmo tipo_de_dado.
- É na declaração de parâmetros que informamos quais serão as entradas da função (assim como informamos a saída no tipo_de_dado associado à função).

Introdução a Algoritmos

- Em um módulo, assim como em um algoritmo, podemos declarar variáveis que serão utilizadas nas manipulações efetuadas. Para esta finalidade existe a Seção de **Declarações Internas**.
- As regras para construção do *nome_da_função* são as mesmas aplicadas na definição dos identificadores de variáveis.
- Por fim, é na **Seção de Comandos** também denominado corpo da função que as entradas são processadas, saídas são geradas e/ou outras operações são feitas.

Introdução a Algoritmos

- Comando **retorne**

Forma geral:

retorne valor_de_retorno

- Quando se executa um comando retorne a função é encerrada imediatamente e o valor de retorno é retornado pela função.
- É importante lembrar que o valor de retorno fornecido tem que ser compatível com o tipo de retorno declarado para a função.

Introdução a Algoritmos

Exemplo de função:

```
algoritmo "exemplo1 função"
var       num:inteiro
          funcao quadrado (a: inteiro): inteiro
          inicio
                    retorne (a*a)
  fimfuncao
inicio
  escreva ("Entre com um número inteiro: ")
  leia (num)
  num <- quadrado(num)
  escreva ("O seu quadrado vale: ",num)
fimalgoritmo
```

Introdução a Algoritmos

Exemplo de função:

```
algoritmo "exemplo2 função"
var       num:inteiro
          funcao quadrado (a: inteiro): inteiro
          inicio
                    retorne (a*a)
   fimfuncao
inicio
   escreva ("Entre com um número: ")
   leia (num)
   escreva ("O quadrado de ",num)
   escreva (" é ", quadrado(num))
fimalgoritmo
```

Introdução a Algoritmos

Exercício

Construa uma função capaz de receber um número inteiro como parâmetro e retornar se este é ou não um número impar.

Resposta:

```
funcao eh_impar (a: inteiro): logico
inicio
    retorne (a%2<>0)
fimfuncao
```

Introdução a Algoritmos

Exercício:

Resposta alternativa:

```
funcao eh_impar (a: inteiro): caractere
inicio
    se (a%2<>0) entao
      retorne ("O número é impar.")
   senao
      retorne ("O número é par.")
   fimse
fimfuncao
```

Introdução a Algoritmos

- Caso o problema anterior estivesse da seguinte forma: Construa uma função que receba um número inteiro e retorne se este é ou não um número impar. Algo mudaria com relação à interpretação?
- Sim. Neste caso não estaria especificado se a função receberia o número através de um parâmetro ou se este seria lido através da entrada padrão. Logo, a seguinte solução também seria correta:

```
funcao eh_impar (): logico
var n:inteiro
inicio
    escreva ("Entre com um valor inteiro: ")
    leia (n)
    retorne (n%2<>0)
fimfuncao  //a chamada da função eh_impar deve ser
           //feita da seguinte forma: eh_impar()
```

Introdução a Algoritmos

Exercício

Utilizando-se do conceito de Modularização construa um algoritmo que resolva o problema de obter as raízes reais de uma equação do segundo grau, caso existam raízes reais.

Introdução a Algoritmos

```
algoritmo "Calcular Raízes"
var
   a, b, c, d: real
   funcao calcular_delta(a:real; b:real; c:real):real
   inicio
      retorne (b^2-4*a*c)
   fimfuncao
inicio
   escreva("Algoritmo que calcula as raízes reais ")
   escreval ("de uma equação do tipo: ax^2+bx+c")
   repita
      escreva ("Entre com o valor de a: ")
      leia (a)
   ate (a<>0)
   escreva ("Entre com o valor de b: ")
   leia (b)
   escreva ("Entre com o valor de c: ")
   leia (c)
   d<-calcular_delta(a,b,c)
   se (d<0) entao
      escreva ("A equação não possui raízes reais.")
   senao
      escreval ("x1 =",(-b+d^0.5)/(2*a))
      escreval ("x2 =",(-b-d^0.5)/(2*a))
   fimse
fimalgoritmo
```

Introdução a Algoritmos

Exercício

Construa um algoritmo que receba, como parâmetro, um número inteiro positivo, o qual representa a posição de um determinado termo na série de Fibonacci, a função deve retornar o valor do termo correspondente à posição recebida.

Introdução a Algoritmos

Atividade para a próxima aula.

Construa um algoritmo que tenha a capacidade de efetuar uma multiplicação entre valores naturais quaisquer e a capacidade de calcular o fatorial de um número natural qualquer. Tanto no cálculo da multiplicação quanto no cálculo do fatorial os únicos operadores aritméticos que podem ser utilizados são os de soma e subtração.

O algoritmo em questão deve possibilitar ao usuário fazer a seleção de qual operação será realizada. As entradas devem ser validadas e o conceito de modularização deve ser aplicado.

Introdução a Algoritmos

- **Procedimento**

  Um procedimento é um módulo que possui ou não um conjunto de entradas, efetua a execução de um conjunto de instruções e não gera um valor como saída, ou seja, não apresenta retorno.

Introdução a Algoritmos

- A estrutura de um procedimento é a seguinte:

  procedimento<nome_do_procedimento>
  ([<seqüência-de-declarações-de-parâmetros>])
  // Seção de Declarações Internas
  inicio
  // Seção de Comandos
  fimprocedimento

- Obs.:A chamada de um procedimento sem parâmetro segue a mesma regra aplicada à chamada de uma função sem parâmetro.

Introdução a Algoritmos

- A <seqüência-de-declarações-de-parâmetros> é uma lista com a seguinte forma geral:

  identificador1: tipo_de_dado; identificador2: tipo_de_dado; ...; identificadorN: tipo_de_dado
- Assim como na função o tipo_de_dado deve ser especificado para cada uma das N variáveis definidas como parâmetros. É na declaração de parâmetros que informamos quais serão as entradas do procedimento.
- Como um procedimento é um módulo, também podemos declarar, na Seção de Declarações Internas, variáveis que serão utilizadas nas manipulações efetuadas.

Introdução a Algoritmos

- As regras para construção do nome_do_procedimento são as mesmas aplicadas na definição dos identificadores de variáveis.
- Por fim, é na Seção de Comandos, também denominado corpo do procedimento, onde as entradas são processadas e demais operações são feitas.
- **Exemplo de procedimento:**

```
algoritmo "exemplo procedimento"
  procedimento frase()
  inicio
     escreval ("Ola! Eu estou vivo.")
  fimprocedimento
inicio
  frase()
  escreval ("Diga de novo:")
  frase()
fimalgoritmo
```

Introdução a Algoritmos

Exercício

Construa um procedimento que escreve no monitor a mesma frase 10 vezes. O alinhamento que deve ser obedecido para a escrita das frases é apresentado a seguir:

Sou um procedimento!

Sou um procedimento!

Sou um procedimento!

...

Introdução a Algoritmos

- **Escopo de variáveis**
- Com a introdução do conceito de módulo passou a ser possível, além de declarar variáveis no algoritmo principal, declarar variáveis nos módulos, seja como parâmetros ou dentro dos mesmos.
- Desta forma tornou-se necessário o estabelecimento de regras que normatizassem a visibilidade das variáveis, estabelecendo-se o conceito de **escopo de variáveis**.
- O escopo de variáveis é o conjunto de regras que determinam a validade de variáveis nas diversas partes do algoritmo, ou seja, onde cada uma das variáveis pertencentes ao algoritmo existe estando disponível para manipulação.

Introdução a Algoritmos

- **Variáveis locais**
  - Variáveis locais são aquelas que só têm validade dentro do módulo no qual são declaradas.

```
algoritmo "exemplo"
  funcao func1 (...):inteiro
  var
    abc,x,z:inteiro
  inicio
    ...
  fimfuncao
  funcao func2 (...):logico
  var
    z:inteiro
  inicio
    ...
  fimfuncao
inicio
  ...
fimalgoritmo
```

Introdução a Algoritmos

- **Variáveis globais**

- Variáveis globais são declaradas na seção de declarações de variáveis do algoritmo.
- Elas são conhecidas e podem ser alteradas por todas os módulos que constituem o algoritmo.
- Quando um módulo tem uma variável local ou um parâmetro com o mesmo nome de uma variável global o módulo dará preferência à variável local ou o parâmetro.

Introdução a Algoritmos

- Variáveis globais

Exemplo:

```
algoritmo "exemplo"
var
   j:inteiro
   procedimento f1(i: inteiro)
   inicio
         j<-18
         i<-17
         escreval (i)
   fimprocedimento
inicio
   j<-3
   f1(j)
   escreva (j)
fimalgoritmo
```

Introdução a Algoritmos

- Variáveis globais

Exemplo:

```
algoritmo "exemplo"
var
   i:inteiro
   procedimento f1(i: inteiro)
   inicio
         i<-18
         escreval (i)
   fimprocedimento
inicio
   i<-3
   f1(i)
   escreva (i)
fimalgoritmo
```

Introdução a Algoritmos

- **Parâmetros**
- Os parâmetros são declarados como sendo as entradas de um módulo.
  - Um parâmetro é uma variável local do módulo que é inicializada com um valor externo ao módulo.
- Existem dois tipos de passagem de parâmetro para um módulo:
  - A passagem de parâmetro por valor;
  - Passagem de parâmetro por referência.
- Em todos os módulos usados até o momento utilizamos da passagem de parâmetro por valor.
- Ou seja, em todas as aplicações que utilizamos o conceito de módulo sempre ao chamarmos o módulo que pretendíamos executar passávamos para o mesmo variáveis ou constantes que, respectivamente, continham ou representavam o valor que desejávamos inicialmente para cada um dos parâmetros do módulo.

Introdução a Algoritmos

- **Parâmetros (passagem por valor)**
- Ao se alterar o valor de um parâmetro, passado por valor, esta alteração não terá efeito na variável que foi passada ao módulo.
- Isto ocorre pois quando se passa parâmetros por valor para um módulo, são copiados os valores das variáveis ou constantes para os parâmetros.
  - Isto é, os parâmetros passados por valor existem independentemente das variáveis que foram passadas para o módulo, estes apenas são inicializados com uma cópia dos valores passados para o módulo.
- É esta característica, da passagem de parâmetro por valor, que possibilita a passagem de uma constante como parâmetro para um módulo.

Introdução a Algoritmos

Parâmetros (passagem por valor)

Exemplo:

```
algoritmo "exemplo 1"
var
   j:inteiro
   procedimento f1(i: inteiro)
   inicio
         i<-18
         escreval (i)
   fimprocedimento
inicio
        j<-3
         f1(j)
         escreva (j)
fimalgoritmo
```

Introdução a Algoritmos

Parâmetros (passagem por valor)

Exemplo:

```
algoritmo "exemplo 2"
var
   i:inteiro
   procedimento f1(i: inteiro)
      inicio
         i<-18
         escreval (i)
   fimprocedimento
inicio
        i<-3
         f1(i)
         escreva (i)
fimalgoritmo
```

Introdução a Algoritmos

- **Parâmetros (passagem por referência)**

- Em algumas situações torna-se necessário que módulos manipulem posições de memória alocadas fora de seu escopo. Em outras palavras, torna-se necessário que módulos possam manipular variáveis com declarações externas.
- Uma situação onde este fato é facilmente visualizado é quando torna-se preciso que uma função retorne mais de um valor.
- Para atender esta necessidade existe passagem de parâmetro por referência.

Introdução a Algoritmos

- **Parâmetros (passagem por referência)**
- Na passagem de parâmetro por referência, ao invés de ocorrer a declaração de uma variável local ao módulo e esta ser inicializada com um valor externo, ocorre **apenas a passagem para o módulo da localização na memória de uma determinada variável**.
- Em outras palavras, especifica-se que uma variável externa será visível (manipulável) dentro do módulo.
- Obs.: Devido à natureza da passagem de parâmetro por referência torna-se impossível a passagem de constantes para módulos que se utilizem desta modalidade de passagem de parâmetro.

Introdução a Algoritmos

```
algoritmo "Exemplo de passagem de parâmetro por referência"
var
   a, b:inteiro
   procedimento troca (var x: inteiro; var y:inteiro)
   var aux:inteiro
   inicio
      aux <- x
      x <- y
      y <- aux
   fimprocedimento
inicio
   escreva ("entre com a: ")
   leia(a)
   escreva ("entre com b: ")
   leia(b)
   troca(a,b)
   escreval ("Valor de a: ",a)
   escreval ("Valor de b: ",b)
fimalgoritmo
```

Introdução a Algoritmos

Exemplo **procedimento sem parâmetros**

```
algoritmo "procedimento sem parâmetros“

var n, m, res: inteiro
      procedimento soma
      var aux: inteiro
      inicio
      // n, m e res são variáveis globais
      aux <- n + m // variável pode ser opcional
      res <- aux
      fimprocedimento
inicio
// Seção de Comandos
      n <- 4
      m <- 9
      soma
      escreva(res)
fimalgoritmo
```

Introdução a Algoritmos

Exemplo **procedimento com parâmetros**

```
algoritmo "procedimento com parâmetros"

var n, m, res: inteiro
    procedimento soma (x,y: inteiro)
    inicio
     // n, m e res são variáveis globais
     res <- x + y
    fimprocedimento
inicio
// Seção de Comandos
      n <- 4
      m <- 9
      soma (n,m)
      escreva(res)
      fimalgoritmo
```

Introdução a Algoritmos

Exercício:

Construa uma função que receba como parâmetros dois números inteiro A e B, respectivamente, e retorne o quociente e o resto da divisão de A por B. As operações aritméticas de multiplicação, divisão, quociente da divisão inteira e resto da divisão inteira **não** podem ser utilizadas na construção da função. Elabore um algoritmo que se utilize de forma coerente da função construída.

Introdução a Algoritmos

Exercício:

Analise o seguinte algoritmo e indique o que será impresso na saída padrão.

```
algoritmo "exercício variável global"
var num, first, sec: inteiro
      funcao func(first:inteiro; sec:inteiro):inteiro
      inicio
        first <- (first+sec)\2
        num <- num - first+1
        retorne (first)
      fimfuncao
inicio
   first <- 0
   sec <- 50
   num <- 10
   escreval ("num antes = ", num)
   escreval ("first antes = ", first)
   escreval ("sec antes = ", sec)
   num <- num + func(first, sec)
   escreval ("num depois = ", num)
   escreval ("first depois = ", first)
   escreval ("sec depois = ", sec)
fimalgoritmo
```