CURSO DE INFORMÁTICA

Algoritmos

Ricardo José Cabeça de Souza

Parte 5

Sumário

UNIDADE II – ESTRUTURAS DE CONTROLE

UNIDADE II – ESTRUTURAS DE CONTROLE

OBJETIVOS DE APRENDIZAGEM

Apresentar as estruturas utilizadas para realização de testes condicionais, seleção de elementos a serem utilizados nos algoritmos e estruturas de repetição de comandos.

2.1 INTRODUÇÃO

Na resolução de algoritmos, algumas vezes nos deparamos com situações onde nem todas as entradas de dados são válidas para execução da tarefa. Por exemplo, para calcular e exibir a raiz quadrada de um número. De acordo com o estabelecido na Unidade I, inicialmente analisamos a definição do problema. Se não temos dúvida sobre o que se deseja, passamos a analisar as entradas necessárias para se resolver o problema. Para se calcular a raiz quadrada de um número, precisamos logicamente de um número. Vamos representar esse número pela letra **N**. Somente esse dado é necessário para se calcular a raiz de um número.

Baseado nessa informação, passamos para o próximo passo que é como executar a tarefa para calcular a raiz quadrada de um número. Considerando que temos uma função para executar essa tarefa, a função **RAIZ( )**, podemos gerar a seguinte fórmula para executar a operação:

RESPOSTA ← RAIZ (N);

Por último, com base na análise acima, podemos informar o resultado ao usuário (**RESPOSTA**).

Depois de analisado o problema, podemos elaborar o algoritmo com a seqüência de operações a serem executadas para o cálculo da raiz quadrada de um número.

```
inicio
  real: N, RESPOSTA;
  imprimir ("Digite um número:");
  ler ( N );
  RESPOSTA ← RAIZ ( N );
  imprimir ("Raiz =",RESPOSTA);
fim.
```

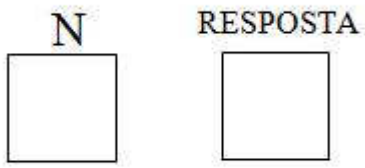
Aparentemente, a solução apresentada parece correta. Executando o teste de mesa, para determinados valores, o resultado esperado é apresentado, ou seja, a raiz quadrada do número solicitado.

Exemplo com um resultado correto:

INÍCIO

real: N, RESPOSTA;

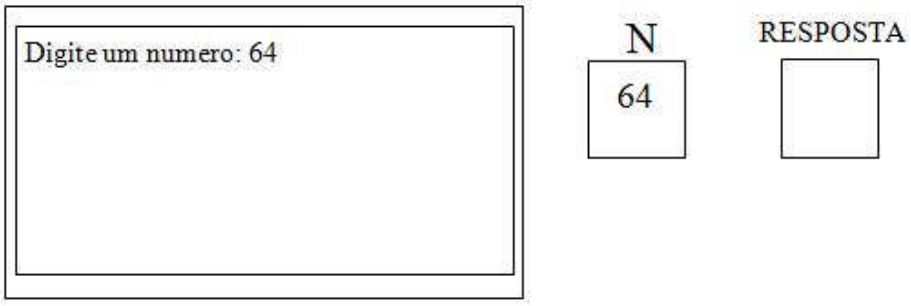
Com a execução dessas linhas, as variáveis são criadas na memória principal do computador, conforme mostrado abaixo:



imprimir ("Digite um numero:");

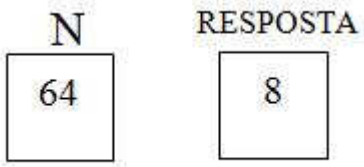
ler (N);

A execução desse trecho do algoritmo mostra o texto entre aspas no monitor do usuário e aguarda que seja feita a entrada de dados de um número. Digamos que o usuário digite o número 64 e tecle <enter>. Ao pressionar a tecla <enter> o valor digitado é lido e armazenado na variável N.



RESPOSTA ← RAIZ (N);

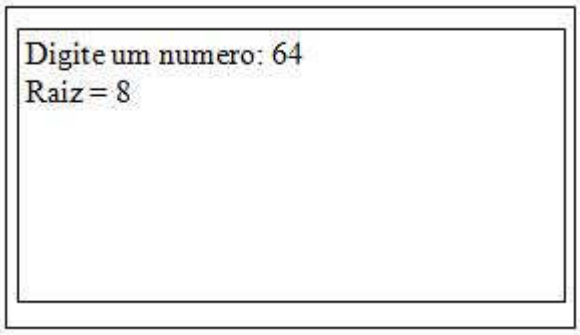
Com a execução da linha acima, é realizado o cálculo da raiz quadrada do número armazenado na variável N e o resultado armazenado na variável RESPOSTA.



imprimir (“Raiz =”,RESPOSTA);

FIM.

Finalmente, com a execução do último comando, é exibido o texto entre aspas seguido do valor armazenado na variável RESPOSTA.



Verifica-se desta forma que o algoritmo aparentemente está correto, executando as operações solicitadas e exibindo corretamente o resultado esperado. Contudo, um **erro** serio se encontra por trás desta solução. O algoritmo deve realizar o cálculo para qualquer número real. Porém, da forma que está escrito, o algoritmo não atende a certos critérios relacionados a **regras matemáticas** importantes para a solução do problema, antes não consideradas.

Matematicamente, a **raiz quadrada** de um número real **não negativo *x*** é o número real **não negativo** que, quando multiplicado por si próprio, iguala-se a ***x***. A raiz quadrada de ***x*** é simbolizada por $\sqrt{x}$. Por exemplo:

$$\sqrt{16} = 4 \text{ porque } 4 \times 4 = 16.$$

Baseado nesse conceito matemático, importante para a solução do problema, não podemos deixar de considerar que somente números reais maiores ou iguais a zero são **VÁLIDOS** para execução das operações requeridas.

Com o objetivo de tornar o algoritmo correto, é necessário ajuste ao programa inicial como a seguir:

- Antes de calcular a raiz, verificamos o número lido se encontra dentro dos valores requeridos. Caso positivo, executamos o cálculo e exibimos o resultado. Caso contrário, informamos ao usuário da impossibilidade de executar tal tarefa.

```
inicio
  real: N, RESPOSTA;
  imprimir ("Digite um número:");
  ler ( N );
  Se ( N > = 0 )
    então
       RESPOSTA ← RAIZ ( N );
       imprimir ("Raiz =",RESPOSTA);
    senão
       imprimir ("Não é possível realizar a operação");
  fim se;
fim.
```

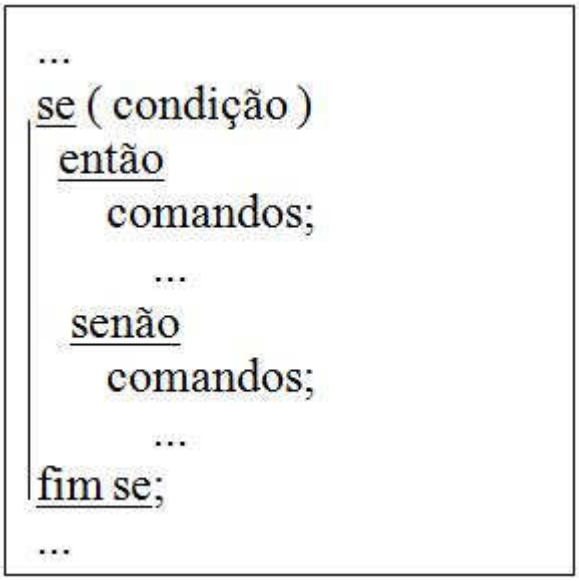
A estrutura utilizada chama-se **seleção**. Nesta unidade veremos algumas estruturas de controle utilizadas em algoritmos com o objetivo de ajustar a saída aos objetivos desejados.

2.3 ESTRUTURAS DE CONTROLE

2.3.1 Seleção

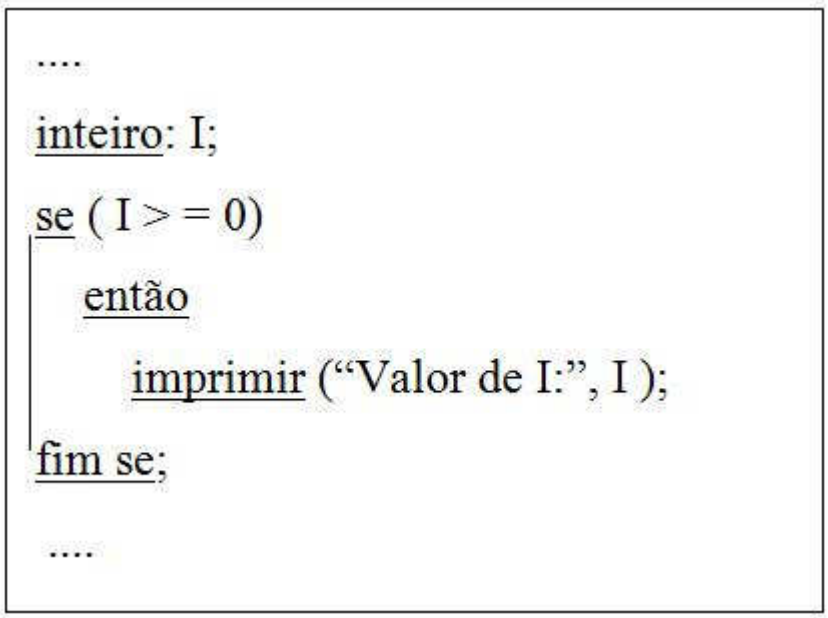
A **seleção** permite a seleção entre duas alternativas válidas para execução da tarefa requerida.

Formato geral da estrutura de seleção:



A estrutura executa todas as operações definidas abaixo da estrutura **então** caso o resultado do teste (condição) seja verdadeiro. Se o teste realizado for falso, todas as operações abaixo da estrutura **senão** são executadas.

A inclusão da opção ***senão*** da estrutura não é obrigatória, dependendo do caso, pode não haver necessidade da estrutura ***senão***, sendo utilizada somente a estrutura ***então***, como no exemplo a seguir:



Existe a possibilidade de utilizar a estrutura de seleção realizando testes com mais de uma condição, utilizando conectores lógicos para interligar as sentenças.

Exemplo de seleção com mais de uma condição:

```
...
se (A > B) e (A <> 0 )
   então
     imprimir (" A =",A);
   senão
     imprimir ("B =",B);
fim se;
...
```

Podemos também encadear (juntar) várias estruturas de seleção de forma hierárquica, com a finalidade de realizar testes de várias opções **dependentes** entre si, como mostrado no exemplo a seguir:

```
...
se (A > 0)
   então
     imprimir ("Valor Positivo");
   senão
     se ( A = 0 )
        então
           imprimir ("Valor Nulo");
        senão
           imprimir ("Valor Negativo");
     fim se;
fim se;
...
```

Observe que as condições apresentadas são **DEPENDENTES** umas das outras, sendo **obrigatória** a escrita de forma hierárquica, caso contrário, as condições não serão satisfeitas.

2.3.2 Seleção Múltipla

Quando o número de alternativas para escolha ultrapassa três possibilidades, podemos utilizar a estrutura de seleção múltipla, chamada de **escolha**, facilitando a escrita e evitando o uso de muitas estruturas **se_então_senão** encadeadas de forma hierárquica.

A instrução **escolha** seleciona uma opção a ser executada com base no valor de uma expressão ou variável.

Formato geral da estrutura de seleção múltipla:

```
...
imprimir ("Escolha uma opção:");
ler ( OP );
escolha (OP)
   caso VALOR_1:
      comandos;
         ...
   caso VALOR_2:
      comandos;
         ...
         ...
   caso VALOR_n:
      comandos;
         ...
   senão
      comandos;
         ...
fim escolha;
```

Para exemplificar o uso da estrutura de seleção múltipla (escolha), vamos elaborar um algoritmo para ler dois números inteiros e em seguida o usuário define um número representando a operação desejada: **1** para **soma**, **2** para **subtração** e **3** para **multiplicação**. Para facilitar a tarefa, podemos apresentar ao usuário um **MENU** com as opções disponíveis e em seguida solicitar a entrada de dados da opção do usuário.

```
inicio
  //Declaração de variáveis
  inteiro: OP, A,B;
  //Entrada dos números
  imprimir ("Digite o 1 numero:");
  ler (A);
  imprimir ("Digite o 2 numero:");
  ler (B);
  //Exibição do menu
  imprimir (".....MENU.....");
  imprimir ("1 – Soma");
  imprimir ("2 – Subtração");
  imprimir ("3 – Multiplicação");
  imprimir ("Escolha uma opção:");
  ler (OP);
  //Verificação da opção escolhida
  Escolha (OP)
    caso 1:
      imprimir ("Soma=",A+B);
    caso 2:
      imprimir ("Subtração=",A-B);
    caso 3:
      imprimir ("Multiplicação=",A*B);
    senão
      imprimir ("Valor inválido!");
  fim escolha;
fim.
```

No exemplo mostrado acima, as variáveis **A** e **B** são utilizadas para armazenar os valores a serem manipulados. A variável **OP** será utilizada para receber o valor da opção do usuário, com base nas operações disponíveis. Após a escolha do usuário, apenas uma das operações disponíveis é executada. Caso a opção do usuário seja um valor diferente dos itens disponíveis (diferente de **1**, **2** ou **3**), a estrutura escolha encaminha para execução da opção **senão**, mostrando a mensagem de erro ao usuário.

2.3.3 Repetição – Enquanto_Faça

Nos itens anteriores, você obteve a possibilidade de escolher a opção desejada a ser executada, com base em uma condição, mas cada instrução continua a ser executada apenas uma vez.

Em determinadas situações, precisamos realizar certas operações repetidas vezes, gerando um ciclo contínuo. Um **ciclo** (ou **laço**) é constituído por uma seqüência de instruções e por uma estrutura que controla a execução dos ciclos, especificando quantas vezes (ou até determinada condição se tornar verdadeira) o ciclo de instruções deve ser executado.

A estrutura de repetição **enquanto_faça** permite especificar a execução repetitiva de um conjunto de instruções enquanto uma determinada expressão (condição) tiver valor verdadeiro.

Formato geral da estrutura de repetição enquanto_faça:

```
...
enquanto ( condição_verdadeira ) faça
  comandos;
      ...
fim enquanto;
...
```

Observe que os comandos inseridos dentro da estrutura enquanto somente são executados se a condição especificada nos parênteses for verdadeira.

Exemplo da estrutura de repetição enquanto_faça:

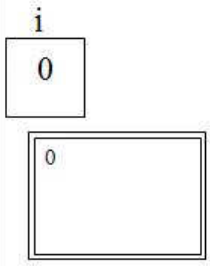
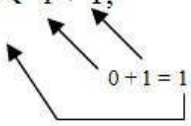
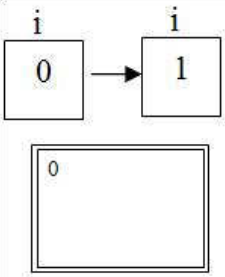
```
inicio
   inteiro: i;
   i← 0;
   enquanto ( i < 4 ) faça
       imprimir ( i );
       i ← i + 1;
   fim enquanto;
fim.
```

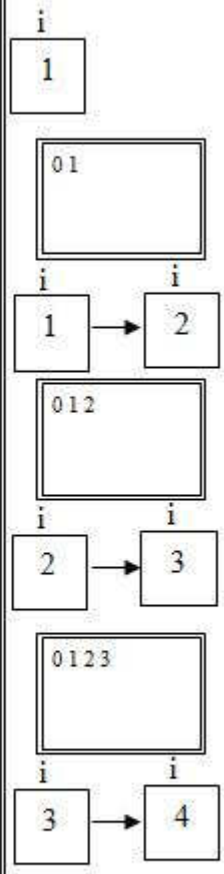
No exemplo mostrado acima, foi criada uma variável (i) que será utilizada para duas coisas: controlar a execução do laço e exibir informações para o usuário.

Para melhor entendimento, vamos analisar cuidadosamente, passo a passo a execução das tarefas, através **do teste de me**sa.

TESTE DE MESA

ALGORITMO	SIMULAÇÃO	COMPUTADOR VIRTUAL
a) Declaração da variável **i** e atribuição do valor inicial;	Na memória principal do computador é criada uma variável com o nome **i**, representada pela figura ao lado e em seguida o valor **0** é atribuído a esta variável;	i 0
b) Executando o ciclo (laço);	Ao encontrar a estrutura enquanto, o valor da condição é verificado. Com o valor de i=0, ou seja, condição verdadeira, entra no laço para execução das instruções.	i 0

ALGORITMO	SIMULAÇÃO	COMPUTADOR VIRTUAL
c) Execução da 1ª instrução do ciclo (laço);	A primeira instrução é exibir o valor de i no monitor	
d) Execução da 2ª instrução do ciclo (laço);	A segunda instrução é atribuir à variável **i** valor igual a **i+1**: i ← i + 1; 	

ALGORITMO	SIMULAÇÃO	COMPUTADOR VIRTUAL
d) Após a execução da 2ª instrução do ciclo (laço), encontra o fim_enquanto, retornando ao teste inicial da estrutura;	enquanto (i < 4) faça imprimir (i); i ← i + 1; fim enquanto; O teste do valor de **i** é efetuado novamente. Considerando que **i** continua sendo menor que **4**, entra no laço para executar os comandos novamente. Novamente a instrução para exibir o valor de **i** é executada novamente e o valor de **i** é incrementado(aumentado) em **1**. Essas operações continuam sendo executadas até que o valor de **i** fique maior que **4**, **terminando** finalmente a execução do laço.	

Observando o exemplo mostrado acima, este algoritmo exibe na tela a seqüência numérica de **0 até 3**, um número por vez, até que o valor de **i** ultrapasse o valor definido na condição do laço, ou seja, **i** fique superior a **3**.

2.3.4 Repetição – Faça_Enquanto

Esta estrutura de repetição é muito semelhante a estrutura enquanto_faça, executando as operações de forma invertida, ou seja, inicia executando as operações e somente após a execução das instruções definidas no laço é que o teste da condição é verificado.

Formato geral da estrutura de repetição faça_enquanto:

```
...
faça
  comandos;
      ...
enquanto ( condição_verdadeira );
...
```

Exemplo da estrutura de repetição enquanto_faça:

```
inicio
   inteiro: i;
   i← 0;
   faça
      imprimir ( i );
      i ← i + 1;
   enquanto ( i < 4 );
fim.
```

Estruturalmente falando, enquanto_faça e faça_enquanto se destinam a um mesmo propósito: executar comandos repetidas vezes. A principal **diferença** entre a estrutura de repetição enquanto_faça e faça_enquanto esta na execução obrigatória de pelo menos uma vez as instruções definidas dentro do laço da estrutura **faça_enquanto**, enquanto que na estrutura **enquanto_faça** as instruções somente serão executadas se a condição inicial for verdadeira, caso contrário, nenhuma operação será realizada.

REFERÊNCIAS BÁSICAS

[1] FARRER, Harry. et. al. **Programação estruturada de Computadores. Algoritmos estruturados**. 2 Ed. Rio de Janeiro: LTC Livros Técnicos e Científicos Editora Ltda, 1989.

[2] MARTINS, J. Pavão. **Introdução à programação usando Pascal**. Lisboa: Editora McGraw-Hill de Portugal Lda, 1994.

[3] ARAÚJO, Everton Coimbra de. **Algoritmos: Fundamentos e Prática**. 2 Ed. ampl. e atual. Florianópolis: VisualBooks Editora, 2005.

[4] GUIMARÃES, Ângelo de Moura. LAGES, Newton Alberto de Castilho**. Algoritmos e estruturas de dados**. Rio de Janeiro: Campus, 1985.

[5] SCHILDT, Herbert. **C Completo e total**. São Paulo: MAKRON BOOKS, 1997.

[6] LOUDON, Kyle. **Dominando algoritmos com C**. São Paulo: CIENCIA MODERNA COMPUTAÇÃO, 2000.

[7] LAUREANO, Marcos**. Programando em C para Linux, Unix e Windows**. Rio de Janeiro: Brasport Livros, 2005.

[8] MEDINA, Marco. FERTIG, Cristina. **Algoritmos e programação: teoria e prática**. São Paulo: NOVATEC INFORMATICA, 2005.

[9] FERREIRA, Aurélio Buarque de Holanda. **Novo Dicionário Aurélio Século XXI.** Rio de Janeiro: Nova Fronteira, 1999.

[10] INFORMAÇÃO. In: Wikipédia: a enciclopédia livre. Disponível em: <http://pt.wikipedia.org/wiki/Informa%C3%A7%C3%A3o>. Acesso em: 15/07/2009.

[11] PEREZ, Anderson Luiz Fernandes. **Linguagens de programação: sintaxe e semântica de linguagens de programação e conceitos de linguagens compiladas e interpretadas**. Disponível em <http://www.univasf.edu.br/~anderson.perez/ensino/intprog/> acesso em 16/07/2009.

[12] BARBOSA, Lisbete Madsen. **Ensino de algoritmos em cursos de computação.** São Paulo: EDUC, 2001.

[13] BIT. In: Wikipédia: a enciclopédia livre. Disponível em: <http://pt.wikipedia.org/wiki/Bit>. Acesso em: 15/07/2009.

[14] CÓDIGO-FONTE. In: Wikipédia: a enciclopédia livre. Disponível em: <http://pt.wikipedia.org/wiki/C%C3%B3digo-fonte>.

Acesso em: 15/07/2009.

[15] PI (Π). In: Wikipédia: a enciclopédia livre. Disponível em: <http://pt.wikipedia.org/wiki/Pi>. Acesso em: 17/07/2009.

[16] BUGS. In: Wikipédia: a enciclopédia livre. Disponível em: <http://pt.wikipedia.org/wiki/Bug> . Acesso em: 15/08/2009.

[17] FREEDMAN, Alan. **Dicionário de Informática**. São Paulo: Makron Books, 1995.

[18] RAIZ QUADRADA. Wikipédia a Enciclopédia Livre. Disponível em <http://pt.wikipedia.org/wiki/Raiz_quadrada> acesso em 30/07/2009.

REFERÊNCIAS COMPLEMENTARES

SCHILDT, Herbert. **C Completo e total**. São Paulo: MAKRON BOOKS, 1997.

DAMAS, Luís. **Linguagem C**. 10. ed. Rio de Janeiro: LTC, 2007.

LOUDON, Kyle. **Dominando algoritmos com C**. São Paulo: CIENCIA MODERNA COMPUTAÇÃO, 2000.

JAMSA, Kris. **Programando em C/C++: a bíblia**. São Paulo: Makron Books, 2000.

LOPES, Anita. **Introdução a programação: 500 algoritmos resolvidos**. Rio de Janeiro: Campus, 2002.

LAUREANO, Marcos. **Programando em C para Linux, Unix e Windows.** Rio de Janeiro: BRASPORT LIVROS, 2005.

DEITEL, Paul J.; DEITEL, Harvey M. **C++ como programar**. Porto Alegre: Bookman, 2001.

PINTO, Wilson Silva. **Introdução ao desenvolvimento de algoritmos e estrutura de dados**. São Paulo: Érica, 1990. KERNIGHAN, Brian W.; RITCHIE, Dennis M. **C: a linguagem de programação**. Rio de Janeiro: Campus, 2000. (005.133 K39c).

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo. **Estudo dirigido de algoritmos**. São Paulo: Érica, 1997. SALVETTI, Dirceu Douglas; BARBOSA, Lisbete Madsen. **Algoritmos**. São Paulo: Makron Books, 1998. (005.1. S183A).

MIZRAHI, Victorine Viviane. **Treinamento em linguagem C: curso completo**. Módulo I. São Paulo: McGraw-Hill, 1990.