Universidade Federal de Viçosa
Centro de Ciências Agrárias
Departamento de Engenharia Agrícola

DESENVOLVIMENTO DE ALGORITMOS

Elaborada por
Daniela de Carvalho Lopes
Evandro de Castro Melo

Viçosa, abril de 2002

ÍNDICE

1. INTRODUÇÃO

A utilização do computador na solução de problemas relacionados às mais diversas áreas está facilitando e agilizando o trabalho de muitas pessoas. Atualmente o conhecimento de técnicas de programação é um requisito importante, principalmente na formação de profissionais relacionados às ciências exatas, como a engenharia por exemplo.

Informalmente, um ALGORITMO é qualquer procedimento computacional bem definido que tenha algum valor ou conjunto de valores como ENTRADA e produza algum valor ou conjunto de valores como SAÍDA. Portanto, um algoritmo é um conjunto de passos computacionais que transformam uma entrada de dados (problema) em saída de dados (solução).

A implementação da solução de um problema em um computador consiste basicamente em dois passos:

- Montagem de uma seqüência de operações que, quando executadas, produzem um resultado, que é a solução do problema (ALGORITMO);
- Transformação da seqüência acima em uma linguagem executável por um computador (LINGUAGEM DE PROGRAMAÇÃO).

O algoritmo é um conjunto de passos para executar uma tarefa. (Receita). A formulação de um algoritmo pode ser descrita como um texto contendo instruções que serão executadas em uma ordem determinada.

Na computação um algoritmo é uma sequência precisa que pode ser utilizada por um computador para a solução de um problema. O algoritmo é composto por um conjunto finito de passos, cada um requerendo uma ou mais instruções. Cada uma destas instruções, também chamadas de operações, deve ser perfeitamente definida e clara.

Exemplo: Como somar dois números?

```
Algoritmo SomarDoisNumeros
    Início
        Ler (Num1)
        Ler (Num2)
        Soma = Num1 + Num2
        Escrever (Soma)
```

O ato de criar um algoritmo é uma arte que nunca poderá ser totalmente automatizada. Cada pessoa cria uma sequência diferente para solucionar um mesmo problema.

Para armazenar um algoritmo na memória do computador e para que o computador possa entender e comandar as operações a serem executadas, é necessário que o algoritmo seja transcrito para uma linguagem de programação (Delphi, Turbo Pascal, C, C++, Matlab, LabView, Visual Basic, etc). Seja qual for a linguagem de programação, a solução correta do problema apresentada pelo computador dependerá da qualidade do algoritmo criado para descrevê-lo. Um algoritmo é dito correto se, para cada instância de entrada, ele fornece a saída correta. Dizemos que um algoritmo correto resolve o problema computacional dado.

Outro aspecto importante é a validação do algoritmo gerado. A validação é um processo no qual se verifica se o algoritmo fornece os resultados corretos considerando todas as entradas de dados possíveis. Esta verificação é realizada independentemente do código de programação utilizado na implementação. Depois de implementado o programa também deverá ser validado verificando-se se está executando corretamente o que foi determinado no algoritmo.

O processo completo que resulta em um programa computacional capaz de solucionar determinado tipo de problema pode ser dividido em cinco áreas descritas a seguir:

- **Estabelecimento do problema:** estabelecer os objetivos do trabalho. O problema deve ser bem definido para que o trabalho de elaboração do algoritmo seja facilitado.
- **Desenvolvimento de um modelo:** influencia diretamente na solução do problema. Modelos mais complicados gastam mais tempo para serem resolvidos e estão mais sujeitos a erros.
- **Criação do algoritmo:** elaborar o algoritmo propriamente dito.
- **Avaliação do algoritmo:** testar o algoritmo. Avaliar as possíveis entradas e verificar se as soluções geradas são corretas. Avaliar, também, os casos de exceção.
- **Implementação:** traduzir o algoritmo para uma linguagem de programação específica.

2. DESENVOLVIMENTO DE ALGORITMOS

Recomenda-se que os algoritmos sejam estruturados partindo-se de uma descrição geral e que, gradativamente, sejam acrescentadas particularidades e detalhes à solução do problema. Esta técnica é chamada "refinamentos sucessivos", "construção hierárquica" ou "desenvolvimento *top-down* (de cima para baixo)".

Exemplo: Efetuar a divisão de dois números.

- Primeira Etapa: Descrição geral
  Algoritmo Divisão
  Fornecer o denominador
  Fornecer o divisor
  Divisão = Denominador / Divisor

- Segunda Etapa: Controle de erro
  Algoritmo Divisão
  Fornecer o denominador
  Fornecer o divisor
  Se o divisor for diferente de zero : Divisão = Denominador / Divisor
  Se o divisor for igual a zero: Divisão não existe

O refinamento sucessivo dos algoritmos permite abordar o problema de maneira mais objetiva diminuindo-se a probabilidade de erros e facilitando a sua reparação quando ocorrerem.

Existem diversas técnicas que ajudam no desenvolvimento de algoritmos facilitando seu entendimento para as pessoas.As mais utilizadas são o portugol e o fluxograma.

2.1. PORTUGOL

O portugol é uma forma de escrita estruturada, cuja finalidade é descrever, em uma sequência lógica, os passos para a resolução de um problema. Nesta técnica os algoritmos são escritos em uma linguagem simples que não possui muitas regras para sua escrita.

Os algoritmos escritos em portugol devem ser escritos de maneira que todas as linhas contenham uma única instrução

Exemplo: Ler dois números e determinar qual dele é o maior.

```
Algoritmo Maior
Inicio
    Escrever ('Digite o primeiro número')
    Ler (num1)
    Escrever ('Digite o segundo número')
    Ler (num2)
    Se num1>num2 então
      Escrever ('O maior número é ', num1)
    Se num2>num1 então
      Escrever ('O maior número é ', num2)
```

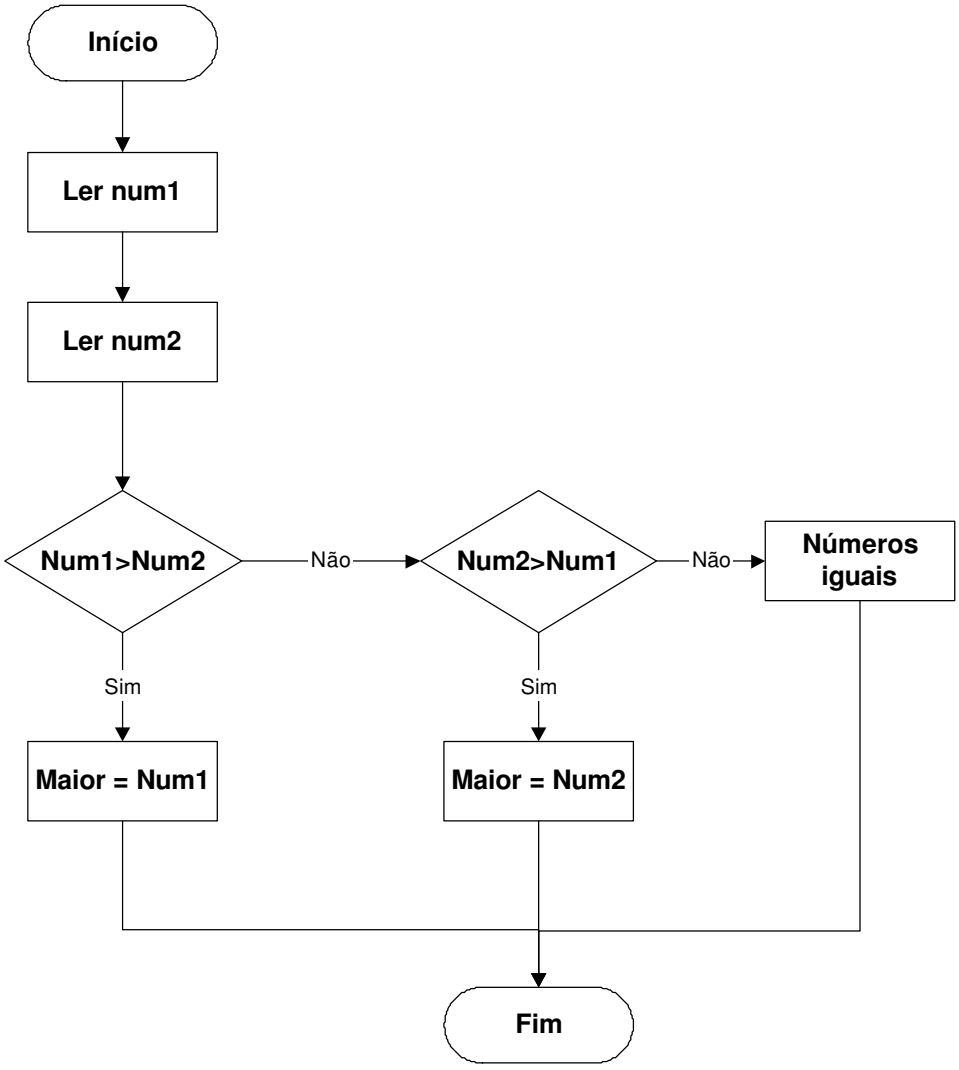
```
    Se num1=num2 então
      Escrever ('Os números são iguais')
FimMaior
```

2.2. FLUXOGRAMA

O fluxograma utiliza símbolos que indicam o tipo de operação que deverá ser realizada.

Exemplo: Ler dois números e determinar qual dele é o maior.

Algoritmo Maior



2.4. EXERCÍCIOS

Escreva algoritmos abaixo utilizando o portugol e o fluxograma:

- Somar dois números
- Calcular a média entre três números
- Ler um número e verificar se é maior que 10

3. CONCEITOS IMPORTANTES

3.1. COMPILADOR

Para que os computadores entendam as instruções que estão escritas em uma linguagem de programação estas instruções precisam ser “traduzidas” em um código especial chamado “linguagem de máquina”. Esta linguagem apresenta vários incovenientes pois todas as operações são escritas como conjuntos de uns e zeros para que possam ativar diretamente os dispositivos eletrônicos do computador.

Mas, a pessoa que estiver programando não precisa se preocupar com a tradução do código escrito em uma linguagem de programação para a linguagem de máquina. Os ambientes em que escrevemos nossos códigos (Pascal, Delphi, C e outros) possuem um programa que é executado quando pedimos para executar o código que escrevemos e é capaz de transformá-lo em um código escrito em linguagem de máquina. Este programa é chamado COMPILADOR.

3.2. SINTAXE E SEMÂNTICA DE UM PROGRAMA

Estes dois termos são muito utilizados pelos programadores quando o programa ou o algoritmo está sendo validado.

A SINTAXE de um programa está relacionada à forma como ele foi escrito. Ou seja, se as palavras e comandos existentes no código do programa estão escritos corretamente. Se existir erro de sintaxe significa que algum comando ou instrução existente no programa foram escritos de maneira incorreta e o programa não poderá ser executado.

A SEMÂNTICA está relacionada ao conteúdo da instrução, ou seja, o que a instrução significa. Quando existem erros de semântica significa que alguma instrução não está fazendo o que queríamos que fizesse e os resultados apresentados pelo programa não são corretos.

3.3. OPERADORES E EXPRESSÕES

As expressões são instruções formadas por operadores e cujo resultado dependerá de valores atribuídos a estes operadores.

Exemplo: A equação $\mathbf{x^2 + \sqrt{x}}$ é um exemplo de expressão aritmética. Como não existe o símbolo relacionado à raiz no teclado de um computador (e isto ocorre com outros símbolos) esta expressão deverá ser escrita em um algoritmo da seguinte maneira: **x^2+x^(1/2).** Os operadores que formam esta expressão são **^**, **+**, **( )** e /.

A equação $\mathbf{x^3y^3 + 4xy - \frac{x}{2}}$ deverá ser escrita em um algoritmo da seguinte forma: **x^3*y^3+4*x*y-x/2**. Os operadores que formam esta expressão são **^**, *, **-**, / e **+**.

Os operadores podem ser aritméticos, lógicos ou relacionais. Os operadores aritméticos geralmente fazem parte de expressões que envolvem cálculos. Os operadores lógicos fazem parte de expressões que envolvem resultados caracterizados como "verdadeiro" ou "falso". Finalmente, os operadores relacionais indicam uma comparação a ser realizada entre os termos de uma relação. A tabela mostrada a seguir apresenta os principais operadores, seus tipos e símbolos.

OPERAÇÃO	TIPO	SÍMBOLO
Adição	aritmético	+
Subtração	aritmético	-
Multiplicação	Aritmético	*
Divisão	Aritmético	/
Potência	Aritmético	^
Conjunção	Lógico	AND (E)
Disjunção	Lógico	OR (OU)
Negação	Lógico	NOT (NÃO)
Maior	Relacional	>
Maior ou igual	Relacional	>=
Menor	Relacional	<
Menor ou igual	Relacional	<=
Igual	Relacional	=
Diferente	Relacional	<>
Prioridade	Relacional	( )

Exemplo: A expressão **(2+3 >2) AND (3+1<2)** é analisada da seguinte maneira: o que está entre parêntesis é executado e a expressão é simplificada para **(5>2) AND (4<2)**. Como 5 é maior que 2, a primeira comparação resulta em **verdadeiro** e, como 4 é maior que 2, a segunda comparação resulta em **falso**. Um valor verdadeiro e um valor falso conectados por AND resulta em falso, que é o resultado final da expressão.

Os operadores lógicos se relacionam da seguinte forma:

- Verdadeiro AND Verdadeiro = Verdadeiro

- Verdadeiro AND Falso = Falso
- Falso AND Verdadeiro = Falso
- Falso AND Falso = Falso
- Verdadeiro OR Verdadeiro = Verdadeiro
- Verdadeiro OR Falso = Verdadeiro
- Falso OR Verdadeiro = Verdadeiro
- Falso OR Falso = Falso
- NOT Verdadeiro = Falso
- NOT Falso = Verdadeiro

Quando uma expressão é analisada o computador executa as operações de acordo com uma tabela de prioridades em que operadores com prioridades mais altas são analisados primeiro. A ordem de prioridade dos operadores é a seguinte:

- Parêntesis : ( )
- Potência : ^
- Multiplicação e divisão : * e /
- Adição e subtração : + e –
- Relacionais
- NOT (NÃO)
- AND (E)
- OR (OU)

As linguagens de programação possuem algumas funções já implementadas e que podem ser utilizadas nas expressões para facilitar o trabalho do desenvolvedor do programa. Exemplos de algumas destas funções são: truncar, arredondar, raiz quadrada, logaritmo e outras.

3.4. COMENTÁRIOS

Os comentários são utilizados para facilitar o entendimento do algoritmo. Pode ser um texto ou simplesmente uma frase que aparece delimitado por chaves (**{ }**) ou precedido por duas barras (//).

Os comentários explicam o que acontecerá quando determinada instrução for executada e servem, também, para ajudar quem escreveu o algoritmo, caso haja a necessidade de analisá-lo algum tempo depois de sua criação. São estruturas que existem na maioria das linguagens de programação.

Exemplo: Subtrair dois números

```
Algoritmo Subtração
        Início
        {Ler dois números}
        Ler (num1)
        Ler (num2)
        {Subtrair os dois números}
        Resultado = num1-num2
        Fim {fim do algoritmo subtração}
```

3.5. EXERCÍCIOS

1) Escreva as expressões abaixo no formato utilizado em algoritmos:

a) $\frac{2x^2 - \sqrt{y}}{3 + x}$

b) $\frac{1}{4y} + \left(\frac{3x}{2}\right)^3$

c) $\sqrt[3]{x^2} + 2$

2) Sabendo-se que A=10, B=3, X=2 e Y=1, quais os resultados fornecidos pelas expressões abaixo:

a) X+Y-A + (A –B^2+Y) – 4*X
b) B^(2+X) – A/3 +1
c) (A –B^3 >X) OR (X – 3 =1)
d) NOT (X*Y = B)

3) Faça um algoritmo que leia os coeficientes a, b e c referentes a equação do segundo grau (ax^2+bx+c) e exiba como resultado as raízes da equação.

4) Faça um algoritmo que leia a base e a altura de um triângulo e calcule sua área.

5) A potência de um motor pode ser expressa em cv ou kW . Sabe-se que 1kW = 0,736 cv. Faça um algoritmo que leia um valor de potência em cv e informe o valor de potência equivalente em kW.

6) Faça um algorimo para transformar um valor de temperatura em ºF para ºC, sabendo-se que: $C = \frac{(F - 32) \times 5}{9}$

4. VARIÁVEIS E CONSTANTES

4.1. VARIÁVEIS

Sabe-se que na matemática uma variável é a representação simbólica dos elementos de certo conjunto.

Nos algoritmos, destinados a resolver um problema no computador, as variáveis são armazenadas na memória do computador e os valores associados a elas podem ser modificados ao longo da execução do programa. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um único valor a cada instante.

Toda variável é identificada por um nome e será capaz de armazenar valores de um mesmo tipo. São cinco os tipos básicos de dados que podem se utilizados:

- **Inteiro**: qualquer número inteiro negativo, nulo ou positivo (2, 4 ,0 ,-1). As principais operações válidas para este tipo de variáveis são: soma, subtração, multiplicação, DIV (divisão que não calcula a parte fracionária, por exemplo 5DIV2=2) e MOD (calcula o resto da divisão, por exemplo 5MOD2=1);
- **Real**: qualquer número real negativo, nulo ou positivo (2,5; 3,44; 12,3). As principais operações válidas para este tipo de variáveis são: adição, subtração, multiplicação e divisão.
- **Caracter**: quaisquer caracteres alfa-numéricos (‘a’,’C’, ‘2’, ‘L’, ‘d’).
- **Cadeia de caracteres (String)**: qualquer conjunto de caracteres alfa-numéricos (‘Ana’, ‘hoje’, ‘LOUCO’, ‘Tudo bem?’). As principais operações válidas para este tipo de variáveis são: concatenação e tamanho.

Concatenar duas strings é fazer a união delas gerando uma única cadeia de caracteres. Por exemplo, a concatenação das strings *‘Nome:’* e *‘Ariosvaldo’* resultará na string *’Nome: Ariosvaldo’*. Em potugol a concatenação é escrita da seguinte maneira: *‘Nome:’+ ‘Ariosvaldo’=’Nome: Ariosvaldo’*. Avaliar o tamanho de uma string é verificar quantos caracteres ela possui. Por exemplo o tamanho da string *‘terra’* é *5*. Em portugol escreve-se: *Length (‘Terra) = 5*.

- **Lógico**: variáveis que poderão assumir valores Verdadeiro ou Falso. As principais operações válidas para este tipo de variáveis são: AND, OR e NOT.

Toda variável deve ser declarada. Declarar uma variável significa criar um local na memória que será rotulado com o nome da variável. A variável deve ser declarada antes do início da execução do programa identificando-se seu nome e tipo.

Exemplo: Calcular o triplo de um número fornecido pelo usuário.

```
Algoritmo TRIPLO;
        Var {indica que variáveis serão declaradas}
        Numero, Resultado: Real
        Início
                Ler (Numero)
                Resultado := Numero * 3
                Escrever (Resultado)
        Fim
```

Quando um valor é fornecido a uma variável para que seja armazenado na memória do computador dizemos que o valor foi atribuído à variável. O comando de atribuição é representado pelo símbolo **:=** . Uma atribuição só será válida se o valor atribuído for do mesmo tipo da variável. Por exemplo se tentarmos atribuir o valor "AGRICULTURA" a uma variável do tipo real ocorrerá um erro, mas se atribuirmos o valor "AGRICULTURA" a uma variável do tipo *String* o programa será executado corretamente. A atribuição é diferente da comparação. Por exemplo, quando a expressão **x:=2** (atribuição) for executada o valor 2 será armazenado no espaço de memória reservado para a variável x que deverá ser do tipo inteiro. Quando a expressão **x=2** (comparação) for executada o computador verificará se o valor armazenado no espaço reservado para a variável x do tipo inteiro é igual a 2.

4.2. CONSTANTES

As constantes são armazenadas na memória do computador e os valores associados a elas não podem ser modificados ao longo da execução do programa. Uma constante pode ser um número, um valor lógico ou uma sequência de caracteres.

As constantes também devem ser declaradas, como mostra o exemplo a seguir:

Exemplo: Calcular o triplo de um número fornecido pelo usuário.

```
Algoritmo TRIPLO;
        Const {indica que constantes serão declaradas}
        T : Inteiro = 3
        Var {indica que variáveis serão declaradas}
        Numero, Resultado: Real
        Início
                Ler (Numero)
                Resultado := Numero * T
                Escrever (Resultado)
        Fim
```

Observe que, no exemplo anterior, o valor associado a **número** será diferente para cada execução do programa e dependerá do valor digitado pelo usuário. Mas, o valor associado a **T** será sempre 3.

4.3. REGRAS PARA NOMES DE VARIÁVEIS E CONSTANTES

Os nomes de variáveis e constantes devem ser formados por um ou mais caracteres, sendo que o primeiro deve ser uma letra e os seguintes letras ou dígitos. Não é permitido a utilização de símbolos especiais (-, +, *, & ou outros). Também não deve haver espaços entre os caracteres que definem o nome de uma variável ou constante.

A tabela a seguir apresenta alguns exemplos de nomes válidos e inválidos para variáveis.

Nomes INVÁLIDOS para variáveis/constantes	Nomes VÁLIDOS para variáveis/constantes
2NOTA	NOTA
MEDIA SIMPLES	MEDIASIMPLES
%	PORCENTO
VARIAÇÃO	VARIACAO

4.4. EXERCÍCIOS

1) Identificar quais as variáveis e constantes foram utilizadas nos exercícios 3,4 e 5 do capítulo anterior dizendo o seu tipo.

2) Assinale com um X as variáveis que apresentam nomes inválidos justificando sua resposta. Para todas as opções indique o tipo da variável.

( ) NOTA ( ) AH! ( ) km/h ( ) 5(8)
( ) a{b} ( ) SALA115 ( ) Xa3 ( ) “ABC”

3) Para cada linha de comando abaixo dizer quais os tipos das variáveis e avaliar o resultado de acordo com os valores de entrada determinados.

a) soma ← (num1 MOD 2) * (num1 DIV 2)
Considerar: num1 = 3, num1 = 16, num1 = 25

b) resultado ← (var1 OR var2) AND (var3)
Considerar: var1, var2 e var3 = False; var1 e var3 = True e var2 = False
var1 e var3 = False e var2 = True

c) completo ← nome + sobrenome
Considerar: nome = ‘Genésio’ e sobrenome = ‘Silva’

d) valor ← (x / y) - 4/(y^2)*x + x*(y-1)
Considerar: x = 2 e y = 4 ; x = 24 e y = 8; x = 6 e y = 4

5. ESTRUTURA CONDICIONAL (ANÁLISE DE CONDIÇÕES)

A estrutura condicional permite a escolha do grupo de ações e estruturas a ser executado quando determinadas condições, representadas por expressões lógicas são satisfeitas.

5.1. COMANDO SE

No comando **SE** quando a condição avaliada é verdadeira uma determinada sequência de comandos é executada e quando a condição avaliada é falsa outra sequência de comandos é executada. A estrutura condicional **SE** pode ser simples ou composta.

A sintaxe da estrutura condicional simples é :

Se (condição) então
(sequência de comandos)
FimSe

A sintaxe da estrutura condicional composta é:

Se (condição) então (sequência de comandos A)
Senão (sequência de comandos B)
Ou
Se (condição 1) então (sequência de comandos 1)
Senão se (condição 2) então (sequência de comandos 2)
...
Senão se (condição N) então (sequência de comandos N)
Senão (sequência de comandos N+1)

O comando **SE** pode ser traduzido da seguinte maneira: Se o resultado da condição for verdadeiro execute a sequência de comandos 1 senão execute a sequência de comandos 2.

Exemplos:

- Como entrar em casa?

```
Algoritmo EntrarNaCasa
  Pegar chave do portão no bolso
  SE (chave  está no bolso) OR (chave está na bolsa) ENTÃO Abrir o portão
    SENÃO Desesperar-se, você está para fora por tempo  indeterminado
```

- Comparar dois números verificando qual deles é o maior.

```
Algoritmo Comparacao
  var
  num1,num2: Real
  Inicio
  Ler (num1)
  Ler (num2)
  Se (num1 > num2) Então
    Escrever ('O primeiro número é o maior')
  Senão se (num1 < num2) Então
    Escrever ('O segundo número é o maior')
  Senão
    Escrever ('Os números são iguais')
FimComparacao
```

5.2. COMANDO CASO

O comando **CASO** permite que a condição avaliada resulte em valores diferentes de verdadeiro ou falso. Isto ocorre porque existe uma sequência de ações que são executadas de acordo com o resultado da expressão avaliada. A sintaxe do comando **CASO** é:

CASO (Expressão)
(Opção 1): (sequência de comandos 1)
(Opção 2): (sequência de comandos 2)
...
(Opção n): (sequência de comandos N)

O comando **CASO** pode ser traduzido da seguinte maneira: caso o resultado da expressão seja igual a opção 1 execute a sequência de comandos 1, caso o resultado da expressão seja igual a opção 2 execute a sequência 2.

Exemplo: Fazer um algoritmo que permita ao usuário escolher entre calcular o dobro ou o triplo de um número fornecido por ele.

```
Algoritmo DobroOuTriplo
    Var
    OP: Inteiro
    Num, Resultado: Real;
    Início
        Escrever (‘ Opções:’)
        Escrever (‘ 1 – Calcular o dobro do número’)
        Escrever (‘ 2 – Calcular o triplo do número’)
        Escrever (‘ Escolha uma opção: ’)
        Ler (OP)
        Escrever (‘ Digite o número:’)
        Ler (Num)
        Caso (OP)
        1: Resultado := Num*2
        2: Resultado := Num*3
        Escrever (Resultado);
    Fim
```

5.3. EXERCÍCIOS

1) Fazer um algoritmo para ler os três lados de um triângulo, dizer se ele é isósceles, escaleno ou equilátero.
OBS: Antes de verificar o tipo do triângulo deve-se verificar se os lados fornecidos formam triângulo (L1+L2 > L3; L2+L3 >L1 e L3+L1 > L2)

2) Fazer um algoritmo para ler três notas, imprimir a maior delas e a média destas notas .

3) Faça um algoritmo que permita ao usuário fornecer três números e escolher entre as opções: calcular a média, calcular a soma ou calcular a multiplicação dos números.

6. ESTRUTURA DE REPETIÇÃO (ITERAÇÕES)

A estrutura de repetição permite que uma sequência de comandos seja executada repetidamente até que uma determinada condição de interrupção seja satisfeita.

6.1. COMANDO ENQUANTO

O **ENQUANTO** é uma estrutura de repetição condicional, isto é, uma vez iniciada será executada até que a condição estipulada seja falsa. A sintaxe do comando **ENQUANTO** é mostrada a seguir.

```
ENQUANTO (condição) FAÇA
        (sequência de comandos)
FimEnquanto
```

No caso da condição ser um contador, este deve ser incializado antes do início da estrutura e deve ser incrementado dentro dela.

Exemplo: Fazer um algoritmo que imprima na tela os números inteiros de 1 a 100.

```
Algoritmo Imprime1a100
Var
Contador:Inteiro
Início
        Contador := 1 {Contador é iniciallizado}
        Enquanto (Contador<=100) Faça
                Escrever(Contador)
                Contador := Contador +1 {Contador é incrementado}
        FimEnquanto
FimAlgoritmo
```

Quando o comando **ENQUANTO** é utilizado, pode acontecer de nenhuma iteração ser realizada. Isto ocorre, quando, a condição já é falsa na primeira vez que o comando é executado.

Exemplo: Fazer um algoritmo que permita ao usuário definir quantas vezes o número 10 será impresso na tela.

```
Algoritmo ImprimeNumero
Var
Contador, Quantidade :Inteiro
Início
        Contador := -1
        Escrever ('Quantas vezes você deseja que o número 10 seja
                        impresso na tela?')
        Ler (Quantidade)
        Enquanto (Contador >= Quantidade) Faça
                Escrever('10')
                Contador := Contador +1
        FimEnquanto
```

FimAlgoritmo

Neste exemplo se a quantidade informada pelo usuário dor zero nenhuma iteração será realizada.

6.2. COMANDO PARA

O **PARA** é uma estrutura de repetição incondicional, isto é, uma vez iniciada será executada quantas vezes estiver estipulada no intervalo determinado. A sintaxe do comando **PARA** é apresentada a seguir:

PARA Contador := ValorInicial ATÉ ValorFinal FAÇA
Sequência de comandos
FimPara

O comando **PARA** sempre utiliza uma variável inteira para representar o contador que deverá ser declarada pelo desenvolvedor do programa. Quando o comando **PARA** é utilizado, todas as iterações estipuladas no intervalo são feitas.

Exemplo: Fazer um algoritmo que imprima na tela os números inteiros de 1 a 100.

```
Algoritmo Imprime1a100
Var
Contador:Inteiro
Início
        Para Contador := 1 até 100  Faça
                Escrever(Contador)
        FimPara
FimAlgoritmo
```

6.3. COMANDO REPITA

O **REPITA** é uma estrutura de repetição condicional, isto é, uma vez iniciada será executada até que a condição estipulada seja falsa. Assim como acontece com o comando ENQUANTO, no caso da condição ser um contador, este deve ser incializado antes do início da estrutura e deve ser incrementado dentro da estrutura. A sintaxe do comando **REPITA** é apresentada a seguir.

REPITA
Sequência de comandos
ATÉ (condição)

Quando o comando **REPITA** é utilizado, pelo menos uma iteração é realizada. Isto ocorre, porque a condição só será avaliada no final da estrutura do comando.

Exemplo: Fazer um algoritmo que imprima na tela os números inteiros de 1 a 100.

```
Algoritmo Imprime1a100
Var
Contador:Inteiro
Início
        Contador:=1
        Repita
                Escrever(Contador)
                Contador := Contador +1
        Até (Contador >=100)
FimAlgoritmo
```

6.4. EXERCÍCIOS

1) Fazer um algoritmo que calcule e imprima o valor de S na série abaixo. O valor de N será informado pelo usuário.

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{N}$$

2) Fazer um algoritmo que leia e imprima inteiros positivos. O programa pára quando é lido um valor negativo ou zero.

3) Fazer um algoritmo que, dados o valor inicial e a razão, calcule a PA ou a PG correspondente e imprima os resultados. O tipo de progressão calculado e o número de termos serão definidos pelo usuário.

7. EXERCÍCIOS DE REVISÃO

1) Qual o valor de L após a execução do algoritmo abaixo?

```
Algoritmo Revisao1
Var
X,Y,L: Real
A,B,C: Lógico
Inicio
    A:=Falso
    B:=Verdadeiro
    C:=Falso
    X:=1,5
    Y:=3,2
    X:=X+1
    SE C OR ((X+Y) > 5) OR (NOT A AND B)) ENTÃO
            L := 0
            SENÃO L := 1
    ESCREVER (L)
FimAlgoritmo
```

2) Fazer um algoritmo que calcule o fatorial de N, sendo que o valor inteiro N será fornecido pelo usuário.
OBS.: $\mathbf{N!} = \mathbf{1} \times \mathbf{2} \times \mathbf{3} \times \cdots \times \mathbf{(N-1)} \times \mathbf{N}$ e $\mathbf{0!} = \mathbf{1}$

3) Qual a diferença entre:

a) O comando PARA e o comando ENQUANTO

b) O comando ENQUANTO e o comando REPITA

c) O comando SE e o comando CASE

4) Fazer um algoritmo que leia o número do CPF, o número de dependentes e a renda anual de um contribuinte e calcule o valor do seu imposto de renda. Haverá um desconto de R$ 600,00 por dependente e os valores da alíquota para o cálculo do imposto são:

Renda líquida	Alíquota
Até R$ 2.000,00	Isento
De R$ 2.001,00 até R$ 5.000,00	5%
De R$ 5001,00 até 10.000,00	10%
Acima de R$ 10.000,00	15%

Como resultados deverão ser impressos todos os dados de entrada, o valor do desconto e o valor do imposto.

5) O que está errado no algoritmo abaixo?

```
Algoritmo Revisao6
Var
 Inteiro: N, PAR, X
Inicio
        Ler (N)
        X := MOD (N / 2)
        SE (X=0) ENTAO
                PAR := Verdadeiro
        SENÃO PAR := Falso
FimAlgoritmo
```

8. VETORES

O vetor é uma variável composta capaz de armazenar mais de um valor ao mesmo tempo. É uma estrutura de dados homogênea porque todos os dados armazenados dentro de um vetor devem ser do mesmo tipo.

Exemplo:

Vetor Notas

8	10	9,5	2	7,2

O vetor Notas armazena apenas valores do tipo Real. Cada posição do vetor é identificada por um índice. Assim, a primeira posição, que contém o valor 8, possui índice 1, a segunda posição, que contém valor 10, possui indice 2 e assim sucessivamente até a última posição, que contém o valor 7,2, e possui índice 5. O índice é utilizado sempre que deseja-se ler um valor armazenado ou atribuir um valor a determinada posição de um vetor. Os limites dos índices são determinados no momento da declaração do vetor. Assim, é possível existir vetores cujos índices variam, por exemplo, de 2 a 6, 0 a 4 ou 1 a 100. Para acessar, modificar ou fazer referência a um elemento de um vetor utiliza-se a seguinte notação: **Nome_do_Vetor [índice].**

Exemplos

- Armazenar a nota 8 na primeira posição do vetor Notas: **Notas [1] := 8**.
- Armazenar o conteúdo da segunda posição do vetor Notas na variável MaiorNota: **MaiorNota := Notas[2]**.
- Escrever na tela o conteúdo do vetor Notas:

```
...
Para  i:=1 até 5 faça
    Escrever (Notas [i])
FimPara
...
```

Um vetor é declarado da seguinte maneira:
NomeVetor: VETOR [LimiteInferior .. LimiteSuperior] DE TIPO

Exemplo: Um professor tem 80 alunos e deseja saber quais foram aprovados e quais foram reprovados, gerando um relatório.

```
Algoritmo Alunos
Var
 Aprovados, Reprovados: vetor [1 .. 80] de String
 NotaFinal: Real
Nome: String
 i : Inteiro
Inicio
        {Inicaliza os vetores}
        Para i :=1 até 80 faça
                Aprovados[i] := ' '
```

```
            Reprovados[i] :=' '
        FimPara
        Para i := 1 até 80 faça
            Ler (Nome)
            Ler ( NotaFinal)
            Se NotaFinal >= 6.0 então
                Aprovados [i] := Nome
                Senão Reprovados[i] := Nome
        FimPara
        Para i := 1 até 80 faça
            Se Aprovados[i] <> ' '  then
                Escrever ( 'O aluno ', Aprovados[i], ' foi aprovado.')
                Senão Escrever ( 'O aluno ', Reprovados[i], ' foi reprovado')
        FimPara
FimAlgoritmo
```

8.1. EXERCÍCIOS

1) Considere que o vetor apresentado a seguir está armazenado na memória do computador e se chama CRR.

!	U	O	T	R	E	C	A

O que será impresso quando a seguinte sequência de comandos for executada?

```
Para i:= 2 até 4 faça
    Aux := CRR[i]
    CRR[i]:= CRR[8 – i + 1]
    CRR [8 – i +1] := Aux
FimPara
Aux := CRR[1]
CRR[1] := CRR[8]
CRR[8] := Aux
```

2) Fazer um algoritmo que leia 35 notas, armazene em um vetor e imprima a maior nota e a posição do vetor onde ela está armazenada

9. MATRIZES

A matriz é uma variável composta bidimensional (um vetor com mais de uma linha) capaz de armazenar mais de um valor ao mesmo tempo. É uma estrutura de dados homogênea porque todos os dados armazenados dentro de uma matriz devem ser do mesmo tipo.

Exemplo:

Matriz Notas – Matriz 3 x 5 (Linha x Coluna)

Coluna 1	Coluna 2	Coluna 3	Coluna 4	Coluna 5	
8	10	9,5	2	7,2	Linha 1
9	3,6	5	4	10	Linha2
8	9,5	9,5	1	7,7	Linha3

A matriz Notas armazena apenas valores do tipo Real. Cada posição da matriz possui um índice que identifica a linha e a coluna onde o valor se encontra. Assim, a primeira posição, que contém o valor 8, possui índice [1,1], a segunda posição que contém valor 10 possui indice [1,2] e assim sucessivamente até a última posição que contém o valor 7,7 e possui índice [5,5]. Assim como acontece com os vetores, o índice é utilizado sempre que deseja-se ler um valor armazenado ou atribuir um valor a determinada posição de um vetor. Para acessar, modificar ou fazer referência a um elemento de uma matriz utiliza-se a seguinte notação: **Nome_da_Matriz [linha, coluna].**

Exemplos

- Armazenar a nota 8 na primeira linha e primeira coluna da matriz Notas: **Notas [1,1] := 8**.
- Armazenar o conteúdo da terceira linha e quarta coluna da matriz Notas na variável MenorNota: **MenorNota := Notas[3,4]**.
- Escrever na tela o conteúdo da matriz Notas:

```
...
Para  i:=1 até 3 faça
    Para j=1 até 5 faça
    Escrever (Notas [i,j])
    FimPara
    Ir para próxima linha
 FimPara
...
```

Neste exemplo o contador **i** faz referência às linhas da matriz e o contador **j** faz referência às colunas.

Uma matriz é declarada da seguinte maneira:
NomeMatriz: VETOR [LInferiorLinha .. LSuperiorLinha, LInferiorColuna .. LSuperiorColuna] DE TIPO

Exemplo: Um professor tem 80 alunos, cada um com 4 notas e deseja saber qual a média de cada um, gerando um relatório.

```
Algoritmo Alunos
Var
 Notas: Vetor [1 .. 80, 1..4] de Real
 Media: Vetor [1..80] de Real
 Nota: Real
 I,j : Inteiro
Inicio
        Para i:=1 até 80 faça
                Media[i] := 0
        FimPara
        Para i := 1 até 80 faça
                Para j := 1 até 4 faça
                Escrever (‘Forneça a nota ‘, j, ‘ do aluno ‘, i)
                Ler ( Nota)
                Notas[i,j] := Nota
                Media[i]:=Media[i]+Nota;
                FimPara
        FimPara
        Para i := 1 até 80 faça
                Escrever (‘Aluno ‘, i)
                Escrever (‘Média: ‘,Media[i])
                Para j := 1 até 4 faça
                        Escrever (‘Nota ‘, j,’:’, Notas[i,j])
                FimPara
        FimPara
FimAlgoritmo
```

9.1. EXERCÍCIOS

1) Fazer um algoritmo que some duas matrizes N x N (N será fornecido pelo usuário) e imprima o resultado.

2) Fazer um algoritmo que leia uma matriz 20 x 20 e divida cada elemento de uma linha da matriz pelo elemento da diagonal principal desta linha. A matriz modificada deverá ser impressa.

3) Faça um algoritmo que leia uma matriz NxM (N e M deverão ser fornecidos pelo usuário) e imprima o maior e o menor número da matriz e a posição onde se encontram.

4) Dada uma matriz MAT 4 x 5 fazer um algoritmo para somar os elementos de cada linha gerando um vetor SOMALINHA com 4 posições. Em seguida somar os elementos do vetor gerando uma variável TOTAL impressa no final. O algoritmo deverá ser testado para o seguinte conjunto de dados:

MAT

8	-1	5	3	2
0	3	0	1	4
9	12	-11	5	6
2	1	2	1	0

5) Considere uma turma com 10 alunos, cada um com 4 notas. Estes dados são armazenados em uma matriz 10 x 5 , em que a primeira coluna armazena a matrícula do aluno e as 4 últimas armazenam as suas notas. Fazer um algoritmo que:
   a) Leia estes dados, armazenando-os ;
   b) Imprima a média de cada aluno;
   c) Imprima a maior média e a matrícula do aluno que a possui

10. PROCEDIMENTOS E FUNÇÕES

As funções e procedimentos são blocos de intruções que são executados quando necessário e realizam ações específicas como mostrar resultados ou calcular algo. As funções e procedimentos são também chamadas de sub-rotinas.

As variáveis declaradas dentro de um procedimento ou de uma função só existiram na memória do computador enquanto a sub-rotina estiver sendo executada e, por isso, são chamadas variáveis locais. As variáveis declaradas dentro do bloco principal são chamadas variáveis globais e existirão na memória do computador enquanto ele estiver sendo executado. Dizemos que as variáveis globais são visíveis no bloco principal e nos blocos escritos dentro dele e que as variáveis locais são visíveis apenas dentro do bloco onde foram declaradas.

Exemplo: Fazer um algoritmo para calcular a área de figuras geométricas. O usuário poderá escolher entre calcular a área de um retângulo ou quadrado.

```
Algoritmo Areas
Var {Variáveis globais}
Opcao:Integer
Area:Real

    Procedimento ExibeResultado
    Inicio
            Escrever (Area)
    FimProcedimento

    Função AreaRet : Real
    Var {variáveis locais}
    Base, Altura:Real
    Inicio
        Ler (Altura)
        Ler (Base)
        AreaRet := Base*ALtura
    FimFuncao

    Função AreaQua : Real
    Var {variáveis locais}
    Lado:Real
    Inicio
        Ler (Lado)
        AreaQua := Lado*Lado
    FimFuncao

Inicio {Esta parte do algoritmo é executada logo após a leitura das variáveis
    globais}
    Escrever ('1 – Calcular area de um retângulo')
    Escrever ('2 – Calcular area de um quadrado')
```

```
        Escrever ('Escolha a opção: )
        Ler (Opcao)
        Case (Opcao)
                1: Area := AreaRet
                2: Area:= AreaQua
        ExibeResultado
FimAlgoritmo
```

Como se pode observar no exemplo anterior, um procedimento é um bloco de instruções que realiza uma ação sem retornar valores ao programa principal (por exemplo, exibir resultados). Uma função é um bloco de instruções que realiza uma ação e retorna um valor ao programa principal (por exemplo, calcular a área de uma figura geométrica). Quando declaramos uma função devemos informar qual o seu tipo, ou seja, qual o tipo do valor que será retornado ao programa principal.

Podemos escrever procedimentos dentro de funções ou vice-versa, procedimentos dentro de procedimentos e funções dentro de funções. Supondo que escrevemos uma função A dentro de uma função B, as variáveis declaradas dentro da função A serão visíveis também na função B. Mas as variáveis locais à função B só serão visíveis dentro dela.

Exemplo: Fazer um algoritmo, utilizando funções e procedimentos, que leia dois valores e calcule a seguinte equação: **R** = **V1**+ **(V1 * V2)**

```
        Procedimento Calcula
        Var
        Valor1, Resultado:Real {variáveis visíveis no procedimento e na função}
        InicioProcedimento
                Função Continua: Real
                Var
                Valor2:Real {variável visível apenas na função}
                InicioFunção
                        Ler (Valor2)
                        Continua :=  Valor1*Valor2
                FimFunção
                Ler (Valor1) {Esta instrução é executada imediatamente após a
                        instrução InícioProcedimento}
                Resultado := Valor1 + Continua;
        FimProcedimento
```

No exemplo anterior a variável Valor1 pode ser utilizada dentro da função Continua pois esta função está declarada no mesmo bloco que a variável. Mas, Valor2 só pode ser utilizada dentro da função pois, como é local a ela, não é visível para o bloco maior.

10.1. PASSAGEM DE PARÂMETROS

Existem alguns casos em que um mesmo procedimento ou função precisam ser executados realizando uma mesma instrução para diversas variáveis. Neste caso utiliza-se uma variável genérica na instrução chamada

parâmetro. No exemplo a seguir V é um parâmetro, ou seja, quando a função Quadrado for chamada o valor da variável indicada entre parêntesis será copiado para V e a instrução dentro da função será executada corretamente. Os parâmetros sempre são declarados entre parêntesis logo depois do nome do bloco (procedimento ou função).

Exemplo: Fazer um algoritmo que calcule o somatório: **S=1+4+9+ ... +10000**

```
Algoritmo CalculaResultado
Var
S,Valor:Real

Função Quadrado (V: Real):Real
Início
        Quadrado := V*V
FimFunção

Inicio
        S := 1
        Para Valor := 2 até 100 Faça
                S:=S+ Quadrado (Valor)
        FimPara
FimAlgoritmo
```

Existem dois tipos principais de passagem de parâmetros: passagem por valor e passagem por referência. O exemplo anterior apresentou a passagem por valor em que o valor da variável original é copiado para o parâmetro. Qualquer alteração que seja feita no parâmetro não alterará a variável original. Na grande maioria dos casos os parâmetros utilizados em funções executam passagens por valor.

Na passagem por referência o endereço de memória da variável é passado como parâmetro e qualquer alteração feita no parâmetro alterará a variável original. O que diferencia a passagem por valor da passagem por parâmetro no código do bloco é a forma de declarar o parâmetro. Na passagem por referência deve-se escrever a palavra var antes do nome do parâmetro. O exemplo a seguir apresenta dois algoritmos semelhantes que ilustram a diferença entre estes dois tipos de passagem de parâmetros.

Exemplo: Fazer um algoritmo que dados os valores de X e Y calcule a seguinte equação: $\mathbf{Z = X^2+Y^2}$

- Maneira 1 - Passagem de parâmetros por referência

```
Algoritmo CalculaZ1
var
Z,X,Y:Real
Início
        Procedimento SomaQuadrados (var N1,N2:Real)
        Inicio
                N1 :=N1*N1
```

```
            N2 :=N2*N2
            Z := N1+N2
        FimProcedimento
        Ler (X)
        Ler (Y)
        SomaQuadrados (X,Y)
FimAlgoritmo
```

- Maneira 2 – Passagem de parâmetros por valor

```
Algoritmo CalculaZ2
var
Z,X,Y:Real
Início
        Procedimento SomaQuadrados (N1,N2:Real)
        Inicio
            N1 :=N1*N1
            N2 :=N2*N2
            Z := N1+N2
        FimProcedimento
        Ler (X)
        Ler (Y)
        SomaQuadrados (X,Y)
FimAlgoritmo
```

O algoritmo CalculaZ1 utiliza a passagem de parâmetros por referência pois antes da declaração do parâmetro no procedimento SomaQuadrados existe a palavra var e o algoritmo CalculaZ2 utiliza a passagem de parâmetros por valor. Suponha que o usuário forneça o valor 2 para X e 3 para Y. Quando o algoritmo CalculaZ1 terminar de ser executado o valor de X será 4, o valor de Y será 9 e o valor de Z será 13. Quando o algoritmo CalculaZ2 terminar de ser executado o valor de X será 2, o valor de Y será 4 e o valor de Z será 13.

10.2. EXERCÍCIOS

1) Quando o algoritmo abaixo for executado o que será exibido na tela?

```
Algoritmo ExercicioCap11
Var
R1,R2,X,Y,A:Real
Início
        Procedimento Somatorio1 (P1:Real)
        Inicio
        P1:=2*P1
        R1:= P1+ 4
        FimSomatorio1

        Procedimento Somatorio2 ( var P2,P3:Real)
        Inicio
        P2 := 3*P2
```

```
        P3 := 2*P3
        R2 := P2 + P3
        FimSomatorio2

X := 1
Y := 3
A := 5
Somatorio1 (A)
Somatorio2 (Y, X)
Escrever (A, X, Y, R1, R2)
```

2) Refazer os exercícios abaixo utilizando procedimentos e funções.
   a) Capítulo 3 – Exercício 3: Utilizar três funções (CalcularDelta, CalcularX1 e CalcularX2) e um procedimento (ExibirResultados).
   b) Capítulo 3 – Exercício 6: Utilizar um procedimento para ler os dados de entrada e um para exibir os resultados.
   c) Capítulo 5 – Exercício 1: Utilizar um procedimento para ler os dados de entrada, um para exibir os resultados e um para verificar se os lados fornecidos formam realmente um triângulo.
   d) Capítulo 5 – Exercício 2: Utilizar uma função para retornar o maior número e um procedimento para exibir os resultados.
   e) Capítulo 5 – Exercício 3: Utilizar um procedimento para exibir um menu com as opções de cálculos, três funções (CalculaMedia, CalculaSoma, CalculaMultiplicacao) e um procedimento para exibir os resultados.
   f) Capítulo 6 – Exercício 3: Utilizar um procedimento para exibir um menu com informações sobre o programa e as opções PA ou PG, duas funções (CalculaPA, CalculaPG) e um procedimento para exibir os resultados.
   g) Capítulo 6 – Exercício 4: Utilizar dois procedimentos (LerDadosEntrada e ExibeREsultados).
   h) Capítulo 7 – Exercício 5: Utilizar 3 procedimentos (LerDadosEntrada, CalcularImposto e ExibeResultados).

11. LITERATURA

FARRER, H. **Programação estruturada de computadores: algoritmos estruturados**. Ed.Guanabara, 1985. 241 p.

LEISERSON, C.E.; STEIN,C.; RIVEST,R.L.; CORMEN, T.H. **Algoritmos**. Ed.Campus, 2002. 1ª edição, 936 p.

LOPES, A.; GARCIA, G. **Introdução à programação**. Ed.Campus, 2001. 1ª edição, 584 p.

MANZANO, J.A.; OLIVEIRA, J.F. **Algoritmos: lógica para o desenvolvimento de programação**. São Paulo, Erica, 1998. 13ª edição, 274p.

MANZANO, J.A.; OLIVEIRA, J.F. **Estudo dirigido de algoritmos**. São Paulo, Erica, 1998. 7ª edição, 240p.

NIKLAUS, W. **Algorithms and data structures**. London, Prentice-Hall International, 1986. 288p.

PINTO, W.S. **Introdução ao desenvolvimento de algoritmos e estruturas de dados**. São Paulo, 1990. 2ª edição.

PREISS, B.R. **Programação estruturada de computadores**. Ed.Campus, 2002. 1ª edição.

VENANCIO, C.F. **Desenvolvimento de algoritmos**. São Paulo, Erica, 1998. 131p.

ZIVIANI, N. **Projeto de algoritmos com implementações em PASCAL e C**. São Paulo, Pioneira, 1994. 267p.

ZIVIANI, N. **Algoritmos – lógica para desenvolvimento de programação de computadores**. São Paulo, Pioneira, 1994. 13ª edição, 267p.