UNIVERSIDADE FEDERAL DO PARÁ
CAMPUS UNIVERSITÁRIO DE SANTARÉM
FACULDADE DE SISTEMAS DE INFORMAÇÃO

PROGRAMAÇÃO BÁSICA
ALGORITMOS

Prof: Cássio D. B. Pinheiro

Santarém-PA
Março/2008

1 - HISTÓRICO DO COMPUTADOR

Embora o nosso objetivo seja o estudo de um sistema eletrônico de processamento de dados, ou seja, um computador eletrônico, temos que retornar ao passado e explicar não somente o momento em que foi criado esse sistema, mais também a época em que o homem teve a necessidade de fazer cálculos e os meios que foram sendo desenvolvidos para resolver o problema e para acelerar a obtenção do resultado.

O primeiro sistema que o homem usou para representar quantidades, foram os seus dedos, e ao mesmo tempo criou o problema de cálculo: representação do resultado. Tal problema foi resolvido, representando-se a quantidade obtida por pedras, gravetos, nós em barbantes, etc. Todavia, essa representação não era suficiente para os comerciantes e mercadores, e para tanto, eles passaram a utilizar placas de argila para seus cálculos. Os egípcios estabeleceram um grande progresso com o uso do papiro, já com os romanos nasceram os primeiros sistemas bancários e orçamentários, assim como a contabilidade.

Com o desenvolvimento do intelecto humano através dos séculos, esses elementos rudimentares tornaram-se fundamentais nas operações de cálculo sobre dados numéricos. Surgiram então, figuras ou símbolos para representá-los. Os árabes foram mais felizes, pois desenvolveram figuras gráficas representativas que consagraram o sistema numérico decimal. Os algarismos arábicos, a partir de então, passaram a ser utilizados por toda a humanidade.

As técnicas desenvolveram-se e, com os algarismos, as operações de cálculo sobre dados numéricos tornaram-se rápidas e mais precisas. O que se busca até os dias de hoje é a rapidez e a precisão, o que possibilita cada vez mais, o avanço do conhecimento humano.

A consolidação do conhecimento matemático, permitiu o desenvolvimento de todas as demais ciências exatas e estas, abriram caminho e deram base ao aperfeiçoamento das atividades tecnológicas, trazendo ao mundo o computador como ferramenta de utilização geral na sociedade.

2 - CONCEITOS BÁSICOS E TERMINOLOGIAS

2.1 - COMPUTADOR

O computador é uma máquina que transforma dados de entrada, em informações de saída. A idéia de entrada / processamento / saída, é a definição mais elementar que podemos dar ao computador.

2.2 - TIPOS DE COMPUTADORES

- **Microcomputadores**: são os menores computadores, largamente utilizados em trabalhos individuais - PCs.

- **Minicomputadores**: de tamanho moderado. São utilizados quando um microcomputador não pode suprir as necessidades do trabalho.

- **Mainframes**: Computadores de grande porte que trabalham com grande velocidade de processamento e podem acessar muitos dados. Muito utilizado na área comercial.

- **Supercomputadores**: Utilizados em aplicações de grande complexidade na área científica.

2.3 - O QUE É O PROCESSAMENTO DE DADOS

Processamento de dados consiste, em uma série de atividades ordenadamente realizadas, com o objetivo de produzir um arranjo determinado de informações a partir de outras obtidas inicialmente. Quando o processamento é feito através de uma máquina, geralmente composta de circuitos eletrônicos, o mesmo é dito Processamento Eletrônico de Dados.

Dentre as mais diversas vantagens do processamento eletrônico, podemos citar: Alta velocidade; Confiabilidade (imune ao erro humano); Precisão.

2.4 - SISTEMA DE PROCESSAMENTO DE DADOS

É a reunião de hardware e software usados para processamento de informações.

- Hardware - A parte física do computador. Como exemplo podemos citar: C.P.U, cabos, periféricos, componentes mecânicos, etc.

- Software - A parte lógica que compõe o computador. Como exemplo podemos citar: Sistemas aplicativos, instruções de controle, programas utilitários, etc.

3 - O COMPUTADOR

3.1 - FUNCIONAMENTO BÁSICO - FASES DE UM PROCESSO COMPUTACIONAL

- **Entrada de dados** - É o envio de qualquer tipo de dados ou informações para uma área de processamento.

- **Processamento** - É a execução de determinadas tarefas que envolvem operações como: soma de dois números, movimentação de informações e registro de informações, isto é, qualquer ação ou operação feita com um ou mais dados ou informações.

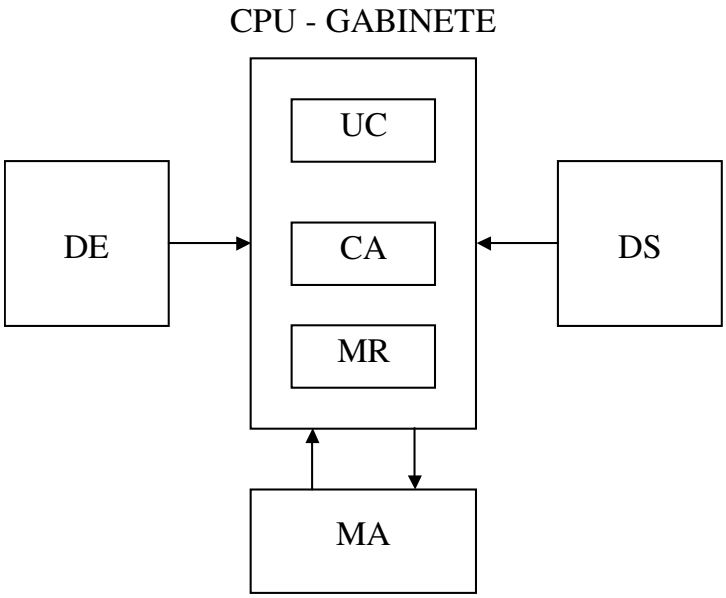
- **Saída de dados** - É qualquer resultado obtido após ter sido efetuado o processamento.

A execução de uma tarefa pelo computador é controlada por PROGRAMAS, cujas INSTRUÇÕES determinam as operações a serem executadas.

- **Instrução**: é um comando que, além de definir a operação a ser realizada pelo computador, identifica os dados necessários à sua execução.

- **Programa**: é um conjunto de instruções ordenadas logicamente, visando a execução de uma determinada tarefa.

3.2 - ESTRUTURA DE UM COMPUTADOR E SEUS COMPONENTES



- C.P.U. (Central Processing Unit) - Unidade Central de Processamento.
  - ⇒ C.A. - Circuitos de Apoio.
  - ⇒ U.C. - Unidade de Controle - Microprocessador ("Cérebro").
  - ⇒ M.R. - Memória Residente (Principal)

        - R.O.M. (Read Only Memory) - Memória Apenas de Leitura.
        - R.A.M. (Random Access Memory) - Memória de Gravação e Leitura.
- D.E. - Dispositivos de Entrada.
- D.S. - Dispositivos de Saída
- M.A. - Memória Auxiliar (Secundária).

4 - CONCEITOS BÁSICOS DE PROGRAMAÇÃO

Antes que possamos entrar no processo de construção de um programa, é necessário que tenhamos em mente alguns conceitos básicos, os quais, irão dar um embasamento para uma boa aprendizagem da programação. Estes conceitos são:

- PROGRAMA - Seqüência de comandos e / ou instruções em uma linguagem qualquer, as quais fazem o computador realizar uma tarefa determinada por um programador.

- COMANDO - Qualquer ordem em linguagem de computador, ou seja, sinais que controlam operações, em geral, de entrada e saída de dados.

- INSTRUÇÃO - Uma única ordem para o computador realizar determinada operação. Uma coleção de instruções formam um programa.

- DADOS - informação, elemento conhecido que serve de base à solução de problemas, ou utilizando um termo mais técnico, qualquer representação a qual pode ser atribuído um significado.

4.1 - CLASSES DE DADOS

4.1.1 - CONSTANTES

Constantes são classes de dados imutáveis utilizadas durante a execução de um programa escrito em uma linguagem de programação. Existem dois tipos de constantes: numéricas e alfanuméricas.

CONSTANTES NUMÉRICAS - Os números são representados basicamente de três formas: números inteiros, números reais de ponto fixo e números reais em ponto flutuante.

Exemplos: 17, 13.45, $1.2453x10^{-3}$

CONSTANTES ALFANUMÉRICAS OU STRING - É uma seqüência de caracteres alfanuméricos, normalmente colocados entre aspas.

Exemplos: "NOME DO EMPREGADO", "SALÁRIO HORA", "123"

4.1.2 - VARIÁVEIS

Variáveis são classes de dados utilizados para representar os valores que podem ser alterados ou modificados durante a execução de programas escritos em uma linguagem de programação. O valor de uma variável pode ser fornecido pelo operador do programa ou advir de cálculos efetuados dentro do programa. As variáveis representam os valores numéricos (variáveis numéricas) e os alfanuméricos (strings).

Exemplos: A = 5.31

B = 14
C = X + 8 - 3
D = "TÉCNICAS DE PROGRAMAÇÃO"

Em um conceito mais aprofundado, podemos dizer que VARIÁVEL é uma denominação e indicação à um local na memória de um computador.

VARIÁVEIS NUMÉRICAS - As variáveis numéricas são aquelas que recebem dados ou constantes numéricas para seu conteúdo.

Exemplos: A = 8
B = 14.53
C = X - 1 + 39

VARIÁVEIS ALFANUMÉRICAS - As variáveis alfanuméricas são aquelas que recebem dados ou constantes alfanuméricas para seu conteúdo.

Exemplos: A = "NÚMERO DE FILHOS"
B = "1 - DÍVIDA / MÊS"

4.1.3 - REGRA PARA FORMAÇÃO DE NOMES DE VARIÁVEIS

Basicamente, nomes de variáveis podem ter qualquer tamanho, podendo aceitar letras e números e caracter de sublinhar, sendo que, o primeiro deve ser uma letra. Os nomes de variáveis devem ser criados de acordo com os dados que irão fornecer as características da mesma, ou seja, o dado armazenado na variável é que deve fornecer subsídios para a formação de seu nome.

Exemplos: A
B1
NOME
IMPOSTO_DE_RENDA
cData
nNr_itens

4.2 - OPERADORES

Em processamento de dados, normalmente trabalhamos com processos de cálculos numéricos, comparações, decisões e etc. Estes processos são chamados de operações, isto é, ações definidas por uma instrução de computador.

Uma operação é composta de operando (dado que será operado) e operador (indicação da ação a ser executada sobre operandos).

4.2.1 - TIPOS DE OPERADORES

OPERADORES MATEMÁTICOS

OPERADOR	SIGNIFICADO
^	exponenciação
-	Menos Unário (negação)
*	multiplicação
/	diviso
+	adiço
-	subtração

OPERADORES LÓGICOS

OPERADOR	SIGNIFICADO
AND	Condicional "E"
OR	Condicional "OU"
NOT	Condicional "NÃO"
XOR	Condicional "OU Exclusivo"

Tabela Verdade (V = Verdadeiro, F = Falso)

X	Y
V	V
V	F
F	V
F	F

AND	OR	XOR
V	V	F
F	V	V
F	V	V
F	F	F

OPERADORES RELACIONAIS

OPERADOR	SIGNIFICADO
=	Igualdade
<> ou ><	Desigualdade
<	Menor que
>	Maior que
<= ou =<	Maior ou igual

4.2.2 - PRECEDÊNCIA ENTRE OPERADORES

OPERADOR	SIGNIFICADO	PRECEDÊNCIA
-	Menos Unário (NEGAÇÃO)	9 (Maior)
^	Exponenciação	8
*	Multiplicação	7
/	Diviso	7

+	Adiço	6
-	Subtração	6
=	Igualdade	5
<> ou ><	Desigualdade	5
<	Menor que	5
>	Maior que	5
<= ou =<	Menor ou igual	5
>= ou =>	Maior ou igual	5
NOT	NÃO	4
AND	E	3
OR	OU	2
XOR	OU Exclusivo	1 (Menor)

Ob.: Quando a prioridade é a mesma, os operadores são avaliados da esquerda para direita.

5 - ALGORITMO

Algoritmo poderia ser definido como a organização sequencial dos passos necessários para se chegar à solução completa de um determinado problema.

O conceito central da programação e da ciência da computação é o do algoritmo. Programação é basicamente construir algoritmos. Normalmente a programação estruturada é apresentada como a arte ou técnica de construir e formular algoritmos de uma forma sistemática.

Num algoritmo devemos distinguir claramente dois aspectos complementares: um aspecto estático e um aspecto dinâmico.

A formulação de um algoritmo geralmente consiste em um texto contendo comandos que devem ser executados numa ordem prescrita. Esse texto é uma representação concreta do algoritmo e tem um caráter evidentemente estático, atemporal, expandido somente no espaço (da folha de papel).

Este texto somente nos interessa pelos efeitos que ele pode explicitar na sua execução em um determinado período de tempo. Dado um conjunto de "valores iniciais" cada execução de um algoritmo é um evento dinâmico, evoluindo no tempo.

5.1 - METODOLOGIA DE DESENVOLVIMENTO DE ALGORITMOS

1. Algoritmos devem ser feitos para serem lidos por seres humanos. Tenha em mente que seus algoritmos deverão ser lidos e entendidos por outras pessoas (e por você mesmo) de tal forma que possam ser corrigidos, receber manutenção e ser modificados.

2. Escreva comentários no momento em que estiver escrevendo o algoritmo. Um algoritmo não documentado é um dos piores erros que um programador pode cometer e é sinal de amadorismo (mesmo com muitos anos de experiência). Como o objetivo de se escrever comentários é facilitar o entendimento do algoritmo, eles devem ser tão bem concebidos quanto o próprio algoritmo. A melhor maneira de se conseguir isso é escrevê-lo nos momentos de maior intimidade com os detalhes, ou seja, durante a resolução dos problemas.

3. Os comentários deverão acrescentar alguma coisa e não apenas frasear os comandos. O conjunto de comandos nos diz o que está sendo feito; os comentários nos dizem porque.

4. Use comentários no prólogo. Todo algoritmo ou procedimento deverá ter comentários no seu prólogo para explicar o que ele faz e fornecer instruções para seu uso. Alguns destes comentários seriam:

- Uma descrição do que faz o algoritmo;
- Como utilizá-lo;
- Explicação do significado das variáveis mais importantes;
- Estruturas de dados utilizadas;

- Os nomes de quaisquer métodos especiais utilizados, juntamente com referências nas quais mais informações possam ser encontradas;
- Autor;
- Data de escrita e da última atualização.

5. Utilize espaços em branco para melhorar a legibilidade. Espaços em branco, inclusive linhas em branco, são muito valiosos para melhorar a aparência de um algoritmo. Normalmente, o programador deve:

- Deixar uma linha em branco entre as declarações e o corpo do algoritmo;
- Deixar uma linha em branco antes e outra depois de um comentário.

6. Uma das dificuldades naturais de um iniciante em programação é como começar a desenvolver um algoritmo. Os passos seguintes, se seguidos, podem auxiliar nesta tarefa:

- Cada algoritmo é identificado por um nome (início do algoritmo);
- Cada passo possui uma definição da ação a ser executada ou de uma decisão a ser tomada;
- Ocasionalmente há um comentário, que é incluído como uma explicação para auxiliar a compreensão;
- Utilize sempre formas para destacar linhas que serão ou não executadas, ou que serão repetidas (estilo na programação);
- Sempre tornar explícito o fim do algoritmo.

7. Algoritmos devem ser feitos observando o constante refinamento das operações. Isto significa que o programador deve, primeiramente, escrever os passos básicos e aumentar o nível de detalhamento de cada passo escrito, continuando o detalhamento até o ponto em que não haja mais a necessidade ou possibilidade de maiores detalhes.

8. Sempre que possível, utilizar os operadores lógicos para melhor visualização da lógica de programação utilizada para resolver o problema.

5.2 - LÓGICA DE PROGRAMAÇÃO

Quando o computador testa uma condição, ele só "decide algo" com base no resultado, que pode ser verdadeiro ou falso. Desta forma, podemos realizar qualquer tarefa contando apenas com estes dois resultados, bastando para isso manipular os testes de condições de forma a chegarem a resultado satisfatório.

Por exemplo, observe o seguinte problema:

O algoritmo abaixo permite que somente pessoas maiores de 10 anos e sócias entrem em um clube.

Algoritmo Clube Versão 1.0
A idade é maior que 10?

Se for, continue; caso contrário, proiba a entrada;
É sócio do clube?
Se for, permita entrada; caso contrário, proíba.
Fim algoritmo

Na emissão do resultado, foi necessário o teste de mais de uma condição, em linhas separadas, uma verificando a idade e outra verificando se é sócia ou não. Para permitir o teste de mais de uma condição em uma mesma linha, com resultado verdadeiro ou falso, existem os operadores lógicos. Tais operadores apresentam um resultado de acordo com a combinação das condições a serem testadas.

Utilizando operadores lógicos, o algoritmo descrito acima seria escrito da seguinte forma.

Algoritmo Clube Versão 1.1
A idade é maior que 10 e é sócio do clube?
Se for, permita entrada; caso contrário, proíba.
Fim Algoritmo

A utilização dos operadores lógicos, além de melhorar a compreensão do algoritmo, permite que o mesmo se torne mais compacto, melhorando assim a sua visualização, a qual tornará mais fácil os possíveis adendos e alterações.

Os operadores lógicos são a base das operação dos atuais computadores, regendo o funcionamento de todas as operações realizadas.

6 - CONTROLES DE FLUXO DE COMANDOS

6.1 - SEQUÊNCIA SIMPLES

Estrutura de Controle onde os comandos são executados sequencialmente, um após o outro, e não existe a possibilidade de um comando não ser executado.

Ex: Algoritmo onde é feita uma leitura de cinco valores e após esta leitura, é calculada e impressa a média aritmética dos valores lidos.

```
leia( VAL_A )
leia( VAL_B )
leia( VAL_C )
leia( VAL_D )
leia( VAL_E )
MEDIA <-- ( VAL_A + VAL_B + VAL_C + VAL_D + VAL_E ) / 5
imprima( MEDIA )
```

6.2 - CONDICIONAL SIMPLES

Estrutura de Controle onde, dependendo de uma condição, existirão comandos que serão ou não executados.

Ex: Algoritmo onde é feita uma leitura de um valor, e dependendo de uma condição, é feita ou não a impressão deste valor.

```
leia( VALOR )
se VALOR > 30
   imprima( "O valor lido é maior que 30" )
   imprima( VALOR )
fim se
```

6.3 - CONDICIONAL COMPLETA

Estrutura de Controle onde, dependendo de uma condição, haverá a escolha de um ou outro bloco de comandos à ser executado.

Ex: Algoritmo onde é feita a leitura de dois valores e após esta leitura será impresso o maior valor lido.

```
leia( VALOR_A )
leia( VALOR_B )
se VALOR_A > VALOR_B
   imprima( "O Valor A é igual a " + VALOR_A )
senão
   imprima( "O Valor B é igual a " + VALOR_B )
```

```
fim se
```

6.4 - LAÇO CONTROLADO POR VARIÁVEL

Estrutura de Controle onde ocorre uma repetição de um determinado bloco de comandos, e o número de vezes que o bloco de comandos será repetido é controlado pelo valor de uma variável qualquer.

Ex: Algoritmo onde é feita uma leitura de cinco valores e após esta leitura, é calculada e impressa a média aritmética dos valores lidos.

```
CONTROLE <-- 0
MÉDIA <-- 0
faça enquanto CONTROLE < 5
   leia( VALOR_LIDO )
   MÉDIA <-- MÉDIA + VALOR_LIDO
   CONTROLE <-- CONTROLE + 1
fim faça
MÉDIA <-- MÉDIA / 5
imprima( MÉDIA )
```

6.5 - LAÇO CONTROLADO POR OPERADOR

Tipo de Fluxo onde ocorre uma repetição de um determinado bloco de comandos, e o número de vezes que o bloco de comandos será repetido é controlado pelo valor de uma variável lida através do teclado ou qualque outro dispositivo de entrada.

Ex: Algoritmo onde é feita uma leitura de N valores e após esta leitura, é calculada e impressa a média aritmética dos valores lidos.

```
NÚMERO_LEITURAS <-- 0
MÉDIA <-- 0
leia( VALOR_LIDO )
faça enquanto VALOR_LIDO <> 0
   MÉDIA <-- MÉDIA + VALOR_LIDO
   NÚMERO_LEITURAS <-- NÚMERO_LEITURAS + 1
   leia( VALOR_LIDO )
fim faça
MÉDIA <-- MÉDIA / NÚMERO_LEITURAS
imprima( MÉDIA )
```

6.6 - EXEMPLOS DE ALGORITMOS

Exemplo 01: Faça um algoritmo que leia 3 (Três) números e imprima o maior entre eles.

Solução: Comentários:

```
leia( NR_1, NR_2, NR_3 )      Leitura dos Valores dos Números
se NR_1 > NR_2                Comparação do 1º com o 2º
     se NR_1 > NR_3           Comparação do 1º com o 3º
          imprima( NR_1 )     Impressão se o maior for o 1º
     senão
          imprima( NR_3 )     Impressão se o maior for o 3º
     fim se
senão
     se NR_2 > NR_3           Comparação do 2º com o 3º
          imprima( NR_2 )     Impressão se o maior for o 2º
     senão
          imprima( NR_3 )     Impressão se o maior for o 3º
     fim se
fim se
```

Exemplo 02: Faça um algoritmo que leia N números maiores que 0 (zero) e imprima o maior e o menor entre eles.

```
Solução:                      Comentários:

leia( NÚMERO )                Primeira leitura do Número
MAIOR <-- NÚMERO              Variável MAIOR recebe o Nº lido
MENOR <-- NÚMERO              Variável MENOR recebe o Nº lido
faça enquanto NÚMERO <> 0     Controle do Nº de repetições
     se NÚMERO > MAIOR        Comparação do Nº lido com MAIOR
          MAIOR <-- NÚMERO    Atribuição do novo Maior Valor
     senão
          se NÚMERO < MENOR   Comparação do Nº lido com MENOR
               MENOR <-- NÚMERO
          fim se
     fim se
     leia( NÚMERO )           Próxima Leitura do Número
fim faça
imprima( "Maior é: "+MAIOR ) Impressão do Maior Valor
imprima( "Menor é: "+MENOR ) Impressão do Menor Valor
```

6.7 - EXERCÍCIOS

Escreva um algoritmo que, sendo informadas as coordenadas de dois pontos, seja mostrado no vídeo a coordenada do ponto médio entre eles.
Fórmulas: ***xp=(x1+x2)/2*** e ***yp=(y2+y1)/2***

Escreva um algoritmo que, sendo informados 3 valores (a, b, e c), o mesmo calcule a média aritmética (m) entre os valores e compare com o valor de b informando se a média é maior, menor ou igual a este valor.

Escreva um algoritmo que, sendo informados os valores de Juros (j), Capital (c) e Tempo (t), seja mostrado no vídeo o valor da Taxa (i).
Fórmula: ***i=(j*100)/(c*t)***

Escrever algoritmos que, sendo fornecidos os valores de a, b e c, mostre no vídeo o valor das seguintes expressões:
a) w=(15+a)/(2-b/c)
b) x=b+a/(raiz(6-c))
c) y=(raiz(9-c))/(b-a+c)
d) z=raiz(b/a)/raiz(3-c)

Escreva um algoritmo que, sendo informados três números para o computador, seja impresso o menor entre eles. Sendo que todos os números informados devem ser de valores diferentes.

Escrever um algoritmo que, sendo fornecidos, o nome e o salário de um funcionário de uma empresa, o algoritmo calcula e imprime os dados acima e o valor do imposto de renda, utilizando as seguintes regras:
a) 5 % quando o salário for menor que R$ 800,00;
b) 10 % quando o salário for maior ou igual a R$ 800,00 e menor que R$ 1.500,00;
c) 15 % quando o salário for maior ou igual a R$ 1.500,00 e menor que R$ 2.500,00;
d) 20 % quando o salário for maior ou igual a R$ 2.500,00.

Escrever um algoritmo que imprime todos os números pares entre 50 e 80.

Escrever um algoritmo que calcule a somatória entre os cinco primeiros números múltiplos de 3, maiores que 100.

Escreva um algoritmo que imprima todos os números múltiplos de 7 situados entre 350 e 500, inclusive.

Escreva um algoritmo que imprima todos os números múltiplos de 4 situados entre 100 e 200, e ao fim desta impressão, o mesmo deverá imprimir a média aritmética dos 8 primeiros números encontrados, que obedeçam as condiç·es determinadas acima.

Ob.: Mensagens devem ser impressas, no caso da ocorrência de "divisão por zero" ou "raiz quadrada de número negativo"

7 - LISTAS OU VETORES

Uma lista ou vetor, em processamento de dados, é um grupo de informações de mesmo tipo. Cada informação é representada na memória do computador por uma variável. Por serem todas as variáveis do mesmo tipo, uma lista também é conhecida como "variável compostas homogêneas".

O nome de qualquer variável de uma lista é composto do nome da lista acrescido de um indicador de posição. Os números que indicam a posição de uma variável em uma lista são chamados de "índice".

Exemplo: Seja VET uma lista que possui doze variáveis do tipo inteiro. Os nomes das variáveis que formam esta lista são:

VET(1), VET(2), ..., VET(12)

7.1 - OPERAÇÕES MAIS IMPORTANTES SOBRE LISTAS

a) Acesso ao K-ésimo elemento da lista

Esta operação é relativamente fácil, temos somente que nos preocupar em saber se o elemento desejado pertence a lista.

```
- Lista VET
- Lista com N elementos
- Acesso ao K-ésimo elemento

Início
     Se K > 0 e K <= N então
          VAR <-- VET( K )
          Imprima VAR
     Fim se
Fim
```

b) Inserir um novo elemento na lista

Neste caso temos que saber em que posição desejamos inserir o novo elemento.

```
- Lista VET
- Tamanho da lista N
- Inserir elemento VALOR antes do K-ésimo elemento.

Início
     Se K > 0 e K <= N então
          Faça para I = N - 1 até K passo -1
               VET( I+1 ) <-- VET ( I )
          Fim para
          VET( K ) <-- VALOR
     Fim se
```

```
Fim
```

c) Remover o K-ésimo elemento da lista

Neste caso temos duas situações distintas:

1 - Queremos remover o elemento de índice igual a K, onde o valor de K é conhecido.

```
- Lista VET
- Tamanho da lista N
- Elemento a ser removido de índice K

Início
     Se K > 0  e  K < N então
          Faça para I = K até N - 1
               VET( I ) <-- VET( I + 1 )
          Fim para
          N <-- N - 1
     Senão
          Se K = N então
               N <-- N - 1
          Fim se
     Fim se
Fim
```

2 - Queremos remover o elemento de índice igual a K, onde K é um valor qualquer e só conhecemos o conteúdo da variável.

```
- Lista VET
- Tamanho da lista N
- Elemento a ser removido de valor VALOR

Início
     Leia VALOR
     I <-- 1
     Faça enquanto VET( I ) <> VALOR e I <= N
          I <-- I + 1
     Fim faça
     Se I <= N então
          Se I < N então
               Faça para J = I até N - 1
                    VET( J ) <-- VET( J + 1 )
               Fim para
          Fim se
          N <-- N - 1
     Senão
          Imprima "Valor não pertence a lista"
     Fim se
Fim
```

d) Leitura da lista

Nos exemplos anteriores consideramos as listas na memória. Neste caso desejamos ler a lista de um dispositivo externo para a memória.

```
- Lista VET
- Tamanho da lista N
- Dispositivo externo: Teclado

Início
     Faça para I = 1 até N
          Leia VALOR
          VET( I ) <-- VALOR
     Fim para
Fim
```

e) Gravação da lista

Neste caso desejamos gravar a lista que está na memória em um dispositivo externo.

```
- Lista VET
- Tamanho da lista N
- Dispositivo externo: Arquivo

Início
     Abrir arquivo
     Faça para I = 1 até N
          Criar registro
          VALOR <-- VET( I )
     Fim para
     fechar arquivo
Fim
```

f) Soma dos elementos de uma lista

```
- Lista VET
- Tamanho da lista N
- Acumulador SOMA

Início
     SOMA <-- 0
     Faça para I = 1 até N
          SOMA <-- SOMA + VET( I )
     Fim para
     Imprima SOMA
Fim
```

g) Soma de listas

```
- Listas: VET1, VET2 e VET3
- Tamanho das listas: N, M e N+M respectivamente
- Concatenar as listas VET1 e VET2 em VET3
```

```
Início
     Faça para I = 1 até N
          VET3( I ) <-- VET1( I )
     Fim para
     I <-- I - 1
     Faça para J = 1 até M
          VET3( J + I ) <-- VET2( J )
     Fim para
Fim
```

h) Determinação do maior e menor elemento

```
- Lista VET
- Tamanho da lista N

Início
     MAIOR <-- VET( 1 )
     MENOR <-- VET( 1 )
     Faça para J = 2 até N
          Se VET( I ) > MAIOR então
               MAIOR <-- VET( I )
          Fim se
          Se VET( I ) < MENOR então
               MENOR <-- VET( I )
          Fim se
     Fim para
     Imprima MAIOR, MENOR
Fim
```

8 - PROGRAMAS E LINGUAGENS DE PROGRAMAÇÃO

8.1 - PROGRAMAS E SUB-ROTINAS

Séries de instruções de computador são chamadas programas. Partes contidas do programa são chamadas sub-rotinas.

As sub-rotinas podem ser procedures, se realizarem apenas algum trabalho, ou funções, se também resultarem em algum valor. "Abra a porta" é semelhante a uma procedure, "Diga-me seu nome" é semelhante uma função. As sub-rotinas também são chamadas subprogramas e rotinas.

Muitas sub-rotinas usam parâmetros para especificarem exatamente que trabalho deve ser feito; por exemplo, uma sub-rotina que calcule uma raiz quadrada precisa de um parâmetro para especificar um número à ser usado. Muitas sub-rotinas indicarão se sua operação foi bem sucedida, por meio de um código de retorno.

8.2 - LINGUAGENS DE PROGRAMAÇÃO E TRADUTORES

Os computadores só podem executar programas que aparecem na forma detalhada conhecida como linguagem de máquina. Entretanto, para a conveniência das pessoas, os programas podem ser representados em outras formas. O conjunto de todas as instruções de máquina diferentes que um processador pode executar é chamado conjunto de instruções. Se os detalhes de um programa em linguagem de máquina forem substituídos por símbolos significativos (como os termos ADD ou MOVE), então a linguagem de programação é conhecida como linguagem assembly (também chamada assembler, assembler simbólico ou macro assembler).

Assembly é chamada uma linguagem de baixo nível, pois os programas em assembly são escritos numa forma próxima a linguagem de máquina. Outras formas de linguagem de programação são mais abstratas, e produzem muitas instruções de máquina para cada comando escrito pelo programador. Estas são chamadas linguagens de alto nível; exemplos são: BASIC, Pascal, FORTRAN, COBOL, PL/I, C e FORTH. Os programas que traduzem os programas em linguagem de alto nível para uma forma utilizável pelo computador são chamados tradutores; para linguagens de baixo nível, os tradutores são chamados assemblers.

Não existe uma diferença real entre um tradutor e um assembler - ambos produzem de uma linguagem de programação orientada para os humanos para uma forma em linguagem de máquina. Um tradutor que executa cada instrução da linguagem, uma por uma, enquanto a traduz, é chamado interpretador. Enquanto que, um tradutor que traduz todo o programa e somente após esta tradução que o mesmo é executado, normalmente é chamado de compilador.

Quando uma pessoa escreve uma programa de computador, a forma utilizada é chamada código-fonte, ou simplesmente fonte. Quando o código-fonte é traduzido (por um assembler ou compilador), o resultado é chamado código-objeto. O código-objeto está quase pronto para ser

usado, mas ele deve passar por uma pequena transformação, realizada por um linkeditor ou linker, para produzir um módulo de carga, que é um programa terminado, pronto para o uso.

Costuma-se dizer que um erro em um programa é um bug, e o processo de tentar encontrar erros ou tentar consertá-los, é chamado debugging ou depuração.

Em geral existem muitas formas de realizar um objetivo com um programa de computador. O esquema, fórmula ou método que um programa usa é o seu algoritmo. Para muitas tarefas - mesmo as mais simples como ordenar dados em ordem alfabética - existem diferenças incríveis na eficiência de diferentes algoritmos, e continua-se pesquisando métodos cada vez melhores.

9 - PROGRAMAÇÃO MODULAR

Programação Modular é um sistema para projetar programas como um conjunto de unidades individuais inter-relacionadas (chamadas módulos), que podem mais tarde ser agrupadas para formar um programa completo.

9.1 - VANTAGENS DA PROGRAMAÇÃO MODULAR

- Qualidade dos Programas
- Flexibilidade
- Padronização
- Planejamento
- Controle e Manutenção
- Utilização

A Programação Modular é uma técnica de programação que tem por objetivo simplificar o desenvolvimento de programas através de sua divisão em "partes". Cada uma dessas partes pode ser facilmente entendida, programada, testada e modificada, pois o problema que está sendo resolvido é menor.

O ponto básico para a Programação Modular é a abordagem de resolução de problemas através de sua decomposição, ou seja, no desenvolvimento de um "Sistema" podemos dividi-lo em vários "Programas", e os "Programas" podemos dividir em varias "Sub-rotinas" ou simplesmente "Rotinas".

Um problema deve ser subdividido até o ponto em que, se fizermos uma nova subdivisão, cada módulo tenha que ser descrito através de seus comandos elementares, e não mais por chamadas a outros módulos. Um Módulo é uma "parte" de um algoritmo, e pode ser um programa completo que faz parte de um sistema, ou um procedimento de auxílio a um programa ou até mesmo uma função.

Procedimento - Módulo que, a partir de informações de entrada, executa uma tarefa e gera informações de saída. A chamada (execução) a um Procedimento é feita através de seu nome.

Função - Módulo que, a partir de informações de entrada gera apenas uma informação de saída. A informação de saída é devolvida pela função na própria posição em que foi chamada.

9.2 - UM EXEMPLO DE ALGORITMO NÃO MODULAR

O algoritmo lê um vetor inicial de 10 elementos e, em seguida lê vários vetores de 10 elementos cada. Para cada vetor lido mostra cada elemento que possui conteúdo idêntico ao do elemento que está na mesma posição no primeiro vetor. O fim dos vetores é indicado por um vetor no qual a soma dos seus dois primeiros elementos é igual a 10.

```
ALGORITMO VETOR 1 – Versão não Modular
```

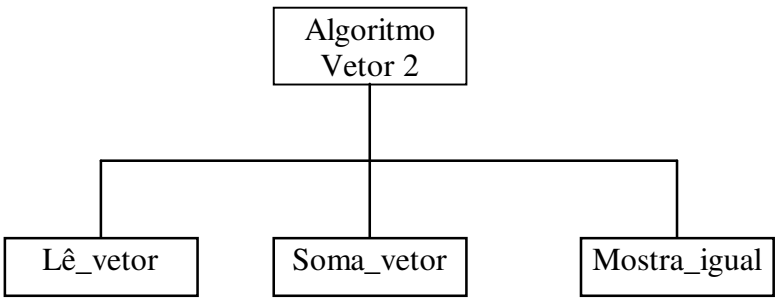
```
Inicio
     Dimensione PRIMEIRO[ 10 ], OUTRO[ 10 ]
     Para IND = 1 até 10 faça
          Leia PRIMEIRO[ IND ]
     Fim para
     Para IND = 1 até 10 faça
          Leia OUTRO[ IND ]
     Fim para
     SOMAVET <- OUTRO[ 1 ] + OUTRO[ 2 ]
     Enquanto SOMAVET <> 10 faça
          Para IND = 1 até 10 faça
               Se OUTRO[ IND ] = PRIMEIRO[ IND ] Então
                    Escreva ( OUTRO[ IND ] )
               Fim se
          Fim para
          Para IND = 1 até 10 faça
               Leia( OUTRO[ IND ] )
          Fim para
          SOMAVET <- OUTRO[ 1 ] + OUTRO[ 2 ]
     Fim enquanto
Fim
```

9.3 - UM EXEMPLO DE ALGORITMO MODULAR

Agora temos o algoritmo descrito anteriormente, sendo que, o mesmo será apresentado utilizando o recurso da programação modular.

Neste exemplo utilizamos a técnica de programação modular, na qual criamos os módulos: Lê_vetor, Soma_vetor e Mostra_Igual. Este módulos são inter-dependentes, pois cada um deles, pode ser analisado independentemente um do outro, mas o programa só será executados com a utilização de todos os módulos.

O organograma do algoritmo Vetor 2, nos mostra que o mesmo, através do seu módulo principal, gerência os três módulos. Isto é, os módulos secundários são executados (chamados) através do módulo principal.




```
ALGORITMO VETOR 2 - Versão Modular
Inicio
     Lê_Vetor[ PRIMEIRO ]
     Lê_Vetor[ OUTRO ]
     SOMAVET <- Soma_Vetor( OUTRO[ 1 ], OUTRO[ 2 ] )
```

```
     Enquanto SOMAVET <> 10 faça
          Mostra_Igual
          Lê_Vetor( OUTRO )
          SOMAVET <- Soma_Vetor( OUTRO[ 1 ], OUTRO[ 2 ] )
     Fim enquanto
Fim

PROCEDIMENTO Lê_Vetor( VETOR )
Inicio
     Para IND = 1 até 10 faça
          Leia VETOR[ IND ]
     Fim para
Fim

FUNÇÃO Soma_Vetor( NUM1, NUM2 )
Inicio
     SOMA <- NUM1 + NUM2
     Retorne SOMA
Fim

PROCEDIMENTO Mostra_Igual
Inicio
     Para IND = 1 até 10 faça
          Se OUTRO[ IND ] = PRIMEIRO[ IND ] Então
               Escreva OUTRO[ IND ]
          Fim se
     Fim para
Fim
```