UNIVERSIDADE FEDERAL DO PAMPA
CAMPUS BAGÉ

INTRODUÇÃO A ALGORITMOS E PROGRAMAÇÃO

FABRICIO FERRARI
fabricio@ferrari.pro.br

CRISTIAN CECHINEL
contato@cristiancechinel.pro.br

BAGÉ, ABRIL DE 2008, VERSÃO 2.0

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Parte I

Conceitos Preliminares

Capítulo 1

O Computador

Um computador é uma máquina que manipula dados a partir de uma lista de instruções.

Os computadores podem ser mecânicos (computador analógico) ou eletrônicos (computadores digitais).

1.1 Histórico dos Computadores

- **MECÂNICOS**
  - Ábaco 1000 A.C
  - Ossos de Napier 1612
  - Pascaline, Pascal 1642
  - Tear automático, Jacquard 1801
  - Máquina de diferenças, Babbage 1882
  - Tabulador eletromecânico, Hollerith 1890
- **1ª GERAÇÃO** – eletro-eletrônicos
  - Z1, Z2, Z3 (relés), Konrad Zuse 1935
  - ABC (válvulas), Atanosoff 1936
  - MARK-1, 1941, 120 m^2, 10 multiplicações em 3 segundos
  - ENIAC, 1946, 30 toneladas, 18000 válvulas, 5000 somas/s
- **2ª GERAÇÃO** – transistores 1947

  - TX-0, 1957
  - PDP-1, Digital, 1^o computador comercial

- ▷ **3^a GERAÇÃO** – circuitos integrados 1958
  - IBM 360, 1965
  - PDP-11, sucesso universitário

- ▷ **4^a GERAÇÃO** – microprocessadores 1970
  - Intel 4004, 1971, 4 bits
  - Intel 8008, 1972
  - Altair 8800, 1974, montado em kits
  - Apple, 1976, TV+Teclado, BASIC escrito por Bill Gates
  - IBM-PC, 1981, computador pessoal, (projeto aberto, processador 8088 Intel, 16 bits, 4.77 MHz, 16 kb RAM, US$ 4400.)

1.2 Arquitetura Básica

Internamente os computadores modernos podem ser caracterizados por três partes distintas, a unidade central de processamento (UCP), a memória (MEM) e os dispositivos de entrada e saída (E/S), conforme esquema na Figura 1.1.

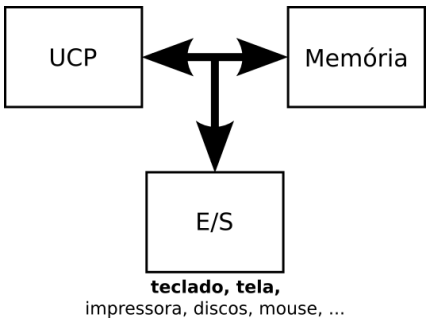

Figura 1.1: Arquitetura básica (Von Neumann)

1.2.1 Unidade Central de Processamento (UCP).

A UCP (ou CPU da sigla em inglês, *Central Processing Unit*) é um conjunto de dispositivos eletrônicos responsável pelas operações de processamento referentes aos cálculos lógicos e matemáticos. Para execução das operações de processamento citadas, a UCP realiza sempre as seguintes tarefas[1]:

1. busca de uma instrução na memória;
2. interpretação de uma instrução;
3. execução de uma operação representada na instrução;
4. gravação de eventuais resultados do processamento;
5. reinício de todo o processo (caso necessário)

Fazem parte da maioria das UCPs as seguintes unidades:

Unidade Aritmética e Lógica (UAL) responsável por realizar cálculos matemáticos mais complexos de maneira mais rápida.

Registradores Memória temporária para armazenar dados a serem processados

Unidade de Controle (UC) Controla o fluxo de dados na UCP: busca na memória, chamadas da UAL, controle geral das tarefas da UCP.

Relógio Gerador de pulsos que determinam um ciclo de tarefas da UCP. Em cada ciclo (ou pulso) a UCP realiza uma tarefa, assim quanto maior a frequência do relógio da UCP, mais tarefas esta pode realizar num mesmo intervalo de tempo.

1.2.2 Memória

A memória é o dispositivo responsável por armazenar dados. Os vários tipos de memória no computador são classificadas de maneira geral de acordo com a sua capacidade de leitura, escrita e volatilidade. São divididas em:

RAM sigla para memória de acesso aleatório, é uma memória em que se pode ler e escrever, mas cujo conteúdo é perdido uma vez que o computador é desligado. É a memória principal do computador e a mais usada pelos aplicativos e sistema operacional.

[1] A sequência de tarefas descritas aqui constitui a base mínima de um algortimo: entrada de dados, processamento de dados e saída de dados. Mais sobre isso na Seção 2.2.

ROM sigla para memória somente-leitura, como o nome diz só é possível ler seu conteúdo, mas não alterá-lo. Não se altera se o computador é desligado.

Secundária são dispositivos usados para armazenar grandes quantidades de informação em caráter não volátil. Na maioria das vezes é muito mais lenta que a RAM. Exemplo são os discos rígidos.

1.2.3 Dispositivos de Entrada e de Saída

Os dispositivos de entrada e saída de dados (E/S) são de suma importância pois qualquer informação que deva entrar ou sair do computador será feita através deles. Dentre os dispositivos de entrada podemos citar: teclado, *mouse*, câmera, digitalizador. Os dispositivos de saída podem ser: monitor[2], impressora, saída de som, por exemplo.

Os dispositivos de E/S se comunicam com o computador através de portas específicas de comunicação, como porta paralela, porta serial, porta USB, porta SCSI, porta *Firewire*, porta PS/2, e assim por diante. Cada porta compreende um tipo de conector específico, porém mais do que isso um protocolo de comunicação entre dispositivos.

O dispositivo de **entrada padrão** é o teclado, enquanto que a **saída padrão** é o monitor. Isto significa que sempre que não for explicitamente especificado, um programa tentará ler do teclado e escrever para o monitor.

[2]Há monitores que são utilizados também como dispositivos de entrada, os chamados monitores touch-screen

Capítulo 2

Algoritmos

2.1 Conceito de Algortimo

Um algoritmo pode ser definido como **uma sequência finita de passos (instruções) para resolver um determinado problema**. Sempre que desenvolvemos um algoritmo estamos estabelecendo um padrão de comportamento que deverá ser seguido (uma norma de execução de ações) para alcançar o resultado de um problema.

Para o desenvolvimento de um algoritmo eficiente é necessário obedecermos algumas premissas básicas no momento de sua construção:

- ▷ Definir ações simples e sem ambiguidade;
- ▷ Organizar as ações de forma ordenada
- ▷ Estabelecer as ações dentro de uma sequência finita de passos.

O algoritmo 1 é um exemplo simples de algoritmo (sem condições ou repetições) para troca de um pneu.

Os algoritmos são capazes de realizar tarefas como:

1. Ler e escrever dados;
2. Avaliar expressões algébricas, relacionais e lógicas;
3. Tomar decisões com base nos resultados das expressões avaliadas;
4. Repetir um conjunto de ações de acordo com uma condição;

Algoritmo 1 Troca de pneu do carro.

```
1: desligar o carro
2: pegar as ferramentas (chave e macaco)
3: pegar o estepe
4: suspender o carro com o macaco
5: desenroscar os 4 parafusos do pneu furado
6: colocar o estepe
7: enroscar os 4 parafusos
8: baixar o carro com o macaco
9: guardar as ferramentas
```

No algoritmo 2 estão ilustradas as tarefas anteriormente mencionadas. Nas linhas de 2 a 4 pode-se observar a repetição de uma ação enquanto uma dada condição seja verdadeira, neste caso em específico, o algoritmo está repetindo a ação 'esperar ônibus' enquanto a condição 'ônibus não chega' permanecer verdadeira, assim que essa condição se tornar falsa (quando o ônibus chegar) o algoritmo deixará de repetir a ação 'esperar ônibus', e irá executar a linha 5.

Já nas linhas de 7 a 9, é possível observar um exemplo da execução (ou não execução) de uma uma ação com base na avaliação de uma expressão. Nesse trecho, o algoritmo avalia se a expressão 'não tenho passagem' é verdadeira e em caso positivo, executa a acão 'pegar dinheiro'. Caso a expressão 'não tenho passagem' seja falsa (ou seja, a pessoa tem passagem) então o algoritmo irá ignorar a ação 'pegar dinheiro' e irá executar a linha 10.

Estas estruturas de controle serão estudadas em detalhe nos capítulos 6 e 7.

2.2 Partes de Um Algoritmo

Um algortimo quando programado num computador é constituído pelo menos das 3 partes, sendo elas:

1. Entrada de dados;
2. Processamento de dados;
3. Saída de dados;

Na parte de entrada, são fornecidas as informações necessárias para que o algoritmo possa ser executado. Estas informações podem ser fornecidas no momento em que o programa está sendo executado ou podem estar embutidas dentro do mesmo.

Algoritmo 2 Pegar um onibus.

```
1: ir até a parada
2: enquanto ônibus não chega faça
3:         esperar ônibus
4: fim-enquanto
5: subir no ônibus
6: pegar passagem
7: se não há passagem então
8:         pegar dinheiro
9: fim-se
10: pagar o cobrador
11: troco ← dinheiro - passagem
12: enquanto banco não está vazio faça
13:         ir para o próximo
14: fim-enquanto
15: sentar
16: ...
```

Figura 2.1: Partes básicas de um algoritmo.

Na parte do processamento são avaliadas todas as expressões algébricas, relacionais e lógicas, assim como todas as estruturas de controle existentes no algoritmo (condição e/ou repetição).

Na parte de saída, todos os resultados do processamento (ou parte deles) são enviados para um ou mais dispositivos de saída, como: monitor, impressora, ou até mesmo a própria memória do computador.

Por exemplo, considere o algoritmo 3 que tem como objetivo calcular a área de uma circunferência dada por $A = \pi R^2$. Para calcular a área é necessário saber os valores do raio R e do π. Considerando que o valor de π é constante o mesmo poderá ser gravado (definido) dentro do próprio algoritmo, e a entrada para o processamento desse algoritmo consistirá nesse valor juntamente com o valor do raio R (que deve ser informado pelo usuário pelo teclado, por exemplo). O processamento do algoritmo será a realização do cálculo πR^2 e a atribuição do resultado dessa expressão para a variável A. A parte da saída consistirá na escrita do valor de A no monitor.

Algoritmo 3 Calcula Área de uma Circunferência.

1: $\pi \leftarrow 3.14$ {entrada para o processamento}
2: **leia** R {entrada para o processamento}
3: $A \leftarrow \pi * R^2$ {processametno}
4: **escreva** A {saída}

2.3 Representações de um Algoritmo

2.3.1 Fluxograma

Os fluxogramas são uma apresentação do algoritmo em formato gráfico. Cada ação ou situação é representada por uma caixa. Tomadas de decisões são indicadas por caixas especiais, possibilitando ao fluxo de ações tomar caminhos distintos.

A Figura 2.2 representa um algoritmo na forma de um fluxograma. O início e o fim do algoritmo são marcados com uma figura elíptica; as ações a serem executadas estão em retângulos; sendo que as estruturas de controle condicionais estão em losangos e indicam duas possibilidades de proseguimento do algoritmo, uma para o caso da expressão avaliada (condição) ser verdadeira e outra para o caso de ser falsa.

No exemplo da Figura 2.2, a primeira ação é executada ('abrir forno') e então a segunda expressão é avaliada ('fogo aceso?') como verdadeira ou falsa; caso seja verdadeira, o algoritmo prosegue para a ação à esquerda ('botar lenha'); caso seja falsa, o algoritmo executa a ação à direita ('acender fogo'). Em seguida, para qualquer um dos casos, a próxima ação a ser executada é ('assar pão').

2.4 Programas de Computador

2.5 Linguagens

Qualquer tipo de informação que deva ser transferida, processada ou armazenada deve estar na forma de uma linguagem. A linguagem é imprescindível para o processo de comunicação. Duas pessoas que se falam o fazem através de uma linguagem em comum, a linguagem natural. Da mesma forma, duas máquinas trocam informação por uma linguagem, que neste caso mais técnico e restrito, se chama protocolo. Do mesmo modo, um computador armazena suas instruções em código de máquina. Estas diferentes linguagens não podem ser traduzidas diretamente entre sí, pois além de serem representadas de modos diferentes, também referem-se a coisas muito distin-

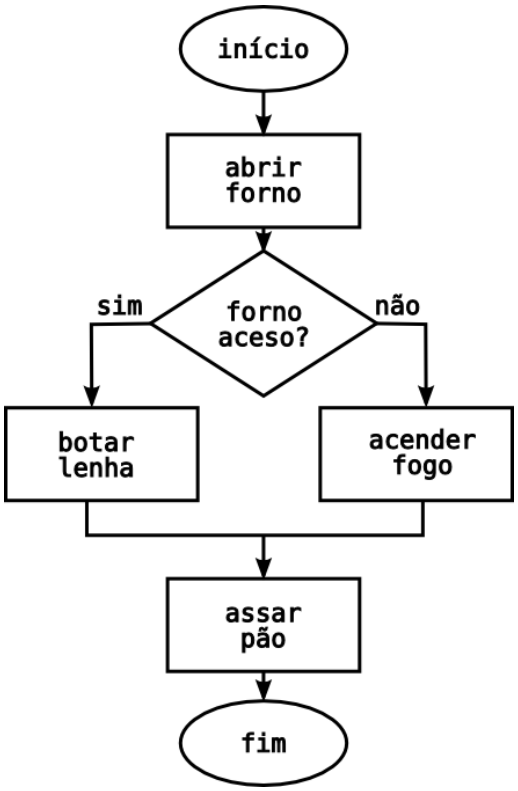

Figura 2.2: Algoritmo representado em forma de um fluxograma.

tas. Para que um ser humano possa programar, armazenar e buscar informações num computador, é necessário que saiba instruí-lo na sua linguagem de máquina ou numa linguagem intermediária (uma linguagem de programação) que possa ser facilmente traduzida para o computador.

2.5.1 Linguagem Natural

A linguagem natural é a maneira como expressamos nosso raciocínio e trocamos informação. Como é a expressão da cultura de uma sociedade, desenvolvida através das gerações e em diferentes situações, raramente constitui um sistema de regras rígidas que possa ser implementada numa máquina ou que possa ser transcrita logicamente. Além da linguagem falada, fazem parte da nossa comunicação gestos e posturas, que não podem ser diretamente adaptados para compreensão de uma máquina. Por fim, toda a comunicação eficiente pressupõe um conhecimento prévio comum entre os interlocutores, por exemplo a mesma língua, a mesma bagagem cultural e assim por diante.

Ao contrário dos seres humanos, as máquinas (dentre elas os computadores) são

projetados para executar tarefas bem determinadas a partir de determinadas instruções. Um computador não é por si só uma máquina inteligente no sentido que não pode aprender com a própria experiência para melhorar seu comportamente futuro[1]. Ao contrário, um computador é somente capaz de realizar estritamente as tarefas que lhe forem delegadas e que façam parte do conjunto daquelas ações que ele pode executar. Neste sentido, é necessário compreender que tipo de instruções podem ser executadas pelos computadores para que possamos programá-los — instruí-los com a sequência de ações necessárias para resolver um determinado problema — de modo que realizem a tarefa do modo desejado.

2.5.2 Linguagem de Máquina e Assembler

Além do fato de o computador necessitar que lhe instruam com ações bem específicas, estas ações devem ser passadas para o computador numa linguagem que ele possa entendê-las, chamada linguagem de máquina. Esta linguagem é composta somente por números, representados de forma binária, que, sob o ponto de vista do computador, representam as operações e os operandos que serão usados no processamento do programa. Para um ser humano, a linguagem de máquina é dificílima de se compreender. Assim, existe uma linguagem representada por comandos mas que reproduz as tarefas que serão executadas dentro do computador, a linguagem de montagem (*assembly*). Entretando, mesmo a linguagem de montagem é difícil de programar e os programas feitos para um determinado processador, por conterem instruções específicas deste, não funcionarão em um processador de outro tipo.

Com ilustração, abaixo é mostrado o início de um programa que escreve a frase "Olá Mundo" no monitor. Na coluna da esquerda está o endereço relativo de memória, na coluna do centro o programa escrito em linguagem de máquina e na coluna da direita a representação em caracteres ASCII. Teoricamente, o programa poderia ser escrito diretamente em linguagem de máquina, como mostrado abaixo, entretando a sintaxe do mesmo é muito pouco compreensível e a probabilidade de erro para o seu desenvolvimento seria muito grande.

```
00000000   7F 45 4C 46  01 01 01 00  00 00 00 00  00 00 00 00  .ELF............
00000010   02 00 03 00  01 00 00 00  D0 82 04 08  34 00 00 00  ............4...
00000020   BC 0C 00 00  00 00 00 00  34 00 20 00  07 00 28 00  ........4. ...(.
```

[1]Diversos esforços vêm sendo despendidos dentro do meio científico para equipar computadores com esta capacidade, o campo de pesquisa que cuida desse tipo de tarefa é conhecido como Inteligência Artifical

Figura 2.3: Compilação: o programa em linguagem de programação é transformado em instruções em linguagem de máquina (que o processador pode executar).

```
00000030   24 00 21 00  06 00 00 00  34 00 00 00  34 80 04 08  ..!.....4...4...
00000040   34 80 04 08  E0 00 00 00  E0 00 00 00  05 00 00 00  4...............
```

2.5.3 Linguagens de Programação

Para facilitar a tarefa de programar um computador, foram criadas várias linguagens de programação. Estas linguagens são um maneira de tentar escrever as tarefas que o computador vai realizar de maneira mais parecida com a linguagem natural. Embora ainda seja muitas vezes complexo em comparação com a linguagem natural, um programa escrito em uma linguagem de programação é muito mais fácil de ser implementado, compreendido e modificado.

As linguagens de programação são um meio termo entre a linguagem de máquina e a linguagem natural. Deste modo são classificadas de acordo com o nível entre a linguagem natural ou de máquina que ocupam. As linguagens muito parecidas com linguagem de máquina são chamadas de linguagens **de baixo nível** e suas instruções parecem-se muito com aquelas que serão executadas pelo processador. As linguagens de **alto-nível** são as que guardam mais semelhanças com a linguagem natural. Exemplo de linguagens de baixo nível é a linguagem de montagem (*assembly*). Exemplos de linguagens de alto-nível são: Pascal, C, Fortran, Java, Perl, Python, Lisp, PHP, entre outras.

Como o processador não pode executar o código numa linguagem de programação, esta deve ser traduzida em código de máquina antes de ser executada. Este processo é chamado de textbfcompilação (representado na Figura 2.3) e é responsável por converter os comandos da lingugem de programação nas instruções em código de máquina que o processador poderá utilizar.

Por exemplo, o código de máquina da seção 2.5.2 foi gerado pelo programa a seguir, escrito na linguagem de programação C. Esse programa, depois de compilado, escreve frase "Olá Mundo" no monitor. A compilação, isto é, a tradução do programa em C para linguagem de máquina, produz algo parecido com o que foi é mostrado na seção 2.5.2, para o caso de um processador da família 80386, usados em PCs.

```
#include <stdio.h>
```

```
int main(){
   printf("Olá Mundo\n");
}
```

A primeira linha (`#include`) inclui algumas bibliotecas de instruções que facilitarão a programação. A linha seguinte indica que esta é a parte principal (`main`) do programa; o que estiver dentro do bloco delimitado por chaves { } será executado. Finalmente, a próxima linha imprime (`printf`) o argumento (‘‘`Olá Mundo`’’) no monitor.

Um programa escrito em linguagem de máquina, como contém instruções específicas de um processador, só poderá ser utilizado naquele processador ou em similares. Em contrapartida, uma linguagem de programação, como contém somente instruções abstratas do que fazer, pode ser compilado para qualquer código de máquina. Em resumo, ao invés de escrever um programa em código de máquina para cada família de processdores, escreve-se o mesmo código numa linguagem de programação e está é compilada por um compilador específico daquela arquitetura.

2.5.4 Pseudocódigo

O pseudocódigo é uma maneira intermediária entre a linguagem natural e uma linguagem de programação de representar um algoritmo. Ela utiliza um conjunto restrito de palavras-chave, em geral na língua nativa do programador, que tem equivalentes nas linguagens de programação. Além disso, o pseudocódigo não requer todo a rigidez sintática necessária numa linguagem de programação, permitindo que o aprendiz se detenha na lógica do algoritmos e não no formalismo da sua representação. Na medida que em se obtém mais familiaridade com os algoritmos, então o pseudocódigo pode ser traduzido para uma linguagem de programação.

Algoritmo 4 Exemplo de Pseudocódigo.

leia x, y {Esta linha é um comentário}
se $x > y$ **então**
    escreva "x é maior"
senão-se $x < y$ **então**
    escreva "y é maior"
senão
    escreva "x e y são iguais"
fim-se

Na listagem 4 é mostrado um exemplo de pseudocódigo escrito em português para escrever o maior valor entre, x ou y. As palavras **leia**, **se**, **então**, **senão**, **senão-se**, **fim-**

se e escreva são palavras-chave que representam estruturas presentes em todas as linguagens de programação. Entretanto, no pseudocódigo não é necessário se preocupar com detalhes de sintaxe (como ponto-e-vírgula no final de cada expressão) ou em formatos de entrada e saída dos dados. Deste modo, o enfoque no desenvolvimento do algoritmo fica restrito a sua lógica em si, e não na sua sintaxe para representação em determinada linguagem.

Por exemplo, considere o código do programa a seguir que implementa na linguagem de programação C o algoritmo 4. Veja como o mesmo requer uma sintaxe bem mais rígida do que o seu algoritmo correspondente Isso acontece pois para que o compilador C possa entender o programa desenvolvido, é necessário que sejam respeitadas algumas exigências da linguagem, como por exemplo:

- ▷ todo programa em C inicia sua execução na função `main()`, que é obrigatória;
- ▷ para que certas funções sejam acessíveis, é necessário incluir a bilbioteca `stdlib.h`;
- ▷ todas as linhas que contém instruções devem terminar com ponto-e-vírgula;
- ▷ os blocos de instruções são delimitados por chaves;
- ▷ linhas de comentários[2] são iniciadas por duas barras `//`;
- ▷ blocos de comentários são delimitados por `/*` e `*/`;

Este conjunto de regras demonstra como o compilador (nesse caso o compilador C) requer estruturas bem rígidas para poder processar (entender) o programa. Para facilitar o entendimento das estruturas algorítmicas que serão estudadas, os algoritmos apresentados aqui serão escritos em pseudocódigo, sendo fundamental que o estudante consiga entender a a correspondência entre os mesmos e a sua representação em uma linguagem de programação.

```
#include <stdio.h>

int main(){
  int x, y;    // isto é um comentário de linha

  /* isto é um comentário
     de bloco              */
  printf("\ndigite x:");
```

[2]Os textos escritos dentro de linhas e/ou blocos de comentários são ignoradoas pelo compilador e servem para que o programador mantenha o código documentado

```
  scanf("%i",&x);
  printf("\ndigite y:");
  scanf("%i",&y);

  if (x>y) {
    printf("x é maior\n");
  } else if (x<y) {
    printf("y é maior\n");
  } else {
    printf("x e y são iguais\n");
  }
}
```

Parte II

Dados

Capítulo 3

Representação de Dados

3.1 Representação Interna

Para que seja possível armazenar e manipular dados no computador é necessário representá-los internamente de alguma forma. Nós seres humanos, representamos nossos números usando um sistema que chamamos de **sistema decimal** (ou sistema na base 10). Esse sistema, que se originou do fato de utilizarmos os 10 dedos das mãos para realizarmos nossas contas, possui 10 diferentes dígitos para representar as infinitas quantidades e valores que desejamos (0 1 2 3 4 5 6 7 8 e 9).

Nos caso dos computadores digitais, a notação que é utilizada possui apenas 2 algarismos ou dígitos para representar uma quantidade desejada, o 0 e o 1. Esse sistema de representação é chamado de **sistema binário** (ou sistema na base 2) e utiliza a noção de ligado/desligado, ou verdadeiro/falso, ou finalmente $0/1$ [1] .

Pelo fato de um número precisar de muitos algarismos para ser expresso no sistema binário, outras formas de representação auxiliares também são utilizadas nos computadores, como por exemplo a representação pelo **sistema hexadecimal** (ou sistema na base 16) que utiliza 16 dígitos (0 1 2 3 4 5 6 7 8 9 A B C D E F), e a representação no **sistema octal** (ou sistema na base 8) que utiliza 8 dígitos (0 1 2 3 4 6 7 8).

Na Tabela 3.1 são mostradas as quantidades de 0 a 15 representadas nos diferentes sistemas mencionados

A quantidade de algarismos necessária para representar um determinado número varia de acordo com o sistema de representação utilizado. Se o sistema é decimal, o maior número que pode ser representado utilizando N algarismos será 10^N. Por

[1] Por esse motivo, o elemento mínimo capaz de armazenar a informação nos computadores foi apelidado de bit, uma contração do inglês **bi**nary digi**t** (dígito binário)

decimal	binário	hexadecimal	octal
0_d	0000_b	0_h	0_o
1_d	0001_b	1_h	1_o
2_d	0010_b	2_h	2_o
3_d	0011_b	3_h	3_o
4_d	0100_b	4_h	4_o
5_d	0101_b	5_h	5_o
6_d	0110_b	6_h	6_o
7_d	0111_b	7_h	7_o
8_d	1000_b	8_h	10_o
9_d	1001_b	9_h	11_o
10_d	1010_b	A_h	12_o
11_d	1011_b	B_h	13_o
12_d	1100_b	C_h	14_o
13_d	1101_b	D_h	15_o
14_d	1110_b	E_h	16_o
15_d	1111_b	F_h	17_o

Tabela 3.1: Equivalência entre sistemas numéricos de representação. O subscrito identifica em que base o número está escrito

exemplo, se nos restringimos a números de dois algarismos, no sitema decimal só poderíamos escrever 100 números, de 0 a 99 ($10^2 = 100$). Para números de três algarimos, poderíamos escrever 1000 números, de 0 a 999 ($10^3 = 100$) e assim por diante.

No sistema binário acontece da mesma forma. Se dispomos de 4 algarismos, poderemos escrever $2^4 = 16$ números, de 0 a 15. Se dispomos de 8 algarismos, poderemos escrever $2^8 = 256$ números, de 0 a 255 e assim por diante. Note que é por isso que a sequência de números

$$2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}, \ldots = 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, \ldots$$

aparece com tanta frequência na informática. Um único algarismo binário é chamado *bit*, uma sequência de 8 *bits* é um *byte* e uma sequência de 16 *bits* é uma *palavra*.

A lógica utilizada para realizar a conversão de números entre diferentes bases é similar a lógica que usamos para representar um número no sistema decimal, ou seja, os algarismos irão representar diferentes quantidades dependendo da sua posição no número em questão. Por exemplo, no número 25, o algarismo '5' representa a quantidade 5, já no número 58, o algarismo '5' representa a quantidade 50, pois $58 = 5\ 10^1 + 8\ 10^0$, sendo que os expoentes de 10 expressam a ordem que o algarismo ocupa no número.

Veja esse outro exemplo,

$$2679_d = 2\ 10^3 + 6\ 10^2 + 7\ 10^1 + 9\ 10^0$$

Em um sistema de representação binário acontece da mesma maneira. Por exemplo,

$$100110_b = 1\ 2^5 + 0\ 2^4 + 0\ 2^3 + 1\ 2^2 + 1\ 2^1 + 0\ 2^0 = 32 + 0 + 0 + 4 + 2 + 0 = 38_d$$

3.2 Tipos Primitivos

Os dados em um computador devem ser armazenados de acordo com o tipo de informação que se deseja representar e com o tipo de operação que será realizada com eles. A representação correta e adequada de uma informação permite otimizar os recursos computacionais disponíveis, além de acelerar o processamento. A seguir são definidos os tipos de dados mais comuns encontrados na maioria das linguagens de programação e que constituem a base de como qualquer informação será armazenada no mesmo.

Inteiro São os números pertencentes ao conjunto dos Inteiros, isto é, que não possuem parte fracionária. Podem ser positivos, nulos ou negativos. Exemplos: 2 laranjas, calçado tamanho 42, 65535 grãos, 0 pessoas na fila, multa de -2 pontos no campeonato.

Real São os números pertencentes ao conjunto dos Reais, isto é, que podem possuir parte fracionária. Também são chamados de ponto flutuante devido à maneira como o computador os armazena. Exemplos[2]: 2.12 litros de combustível, -3.5°C, $\pi = 3.141592654$, saldo de R$ 10000.52, $e = 2.7182818284590451$.

Caractere São os valores pertencentes ao conjunto de todos os caracteres numéricos (0...9), alfabéticos (a...z,A...Z) e especiais (`! @ # $ % ^ & *`). Esse conjunto também é conhecido como conjunto de caracteres **alfanuméricos**. Os caracteres alfanuméricos são armazenados internamente no computador na forma numérica (binária) utilizando o padrão ASCII[3].

[2]Para ser coerente com a notação usada nos computadores, usaremos aqui o ponto como separador decimal

[3]**ASCII** significa American Standard Code for Information Interchange e é um conjunto de códigos usado pela indústria de computadores para representar em código binário (através de combinações de

A seguir são apresentados alguns dos caracteres existentes (representados no padrão ASCII entre o intervalo de 33 a 126):

```
! " # $ % & ’ ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H
I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _ ‘ a b c d e f g h i j k l m n o p
q r s t u v w x y z { | } ~
```

Exemplos de informações do tipo caractere[4]: "João Francisco", "Rua Ismael Soares", "Hotel Feliz", "?". Nestes exemplos, as aspas duplas (") são usadas para indicar o início e o fim das cadeias de caracteres, porém não fazem parte da informação contida nas mesmas. É importante ressaltar que o espaço em branco entre as palavras também é um caractere.

Lógico O tipo lógico é utilizado para representar informações que só podem assumir dois valores, o valor verdadeiro ($\mathbb{V}$) ou o valor falso ($\mathbb{F}$). Estes valores também podem ser entendidos como: ligado/desligado, $1/0$, alto/baixo, fechado/aberto, etc. Exemplos de informações que podem ser representadas utilizando o tipo lógico são: O fogão está apagado, a televisão está ligada, o portão está aberto, o produto foi encontrado.

3.3 Constantes e Variáveis

Dentro de um algoritmo podemos encontrar basicamente duas classes diferentes de dados, os dados **constantes** e os **variáveis**. Um dado é uma **constante** quando seu valor não se altera ao longo do tempo em que o algoritmo é executado, ou seja, permanece o mesmo desde o início até o final da execução. Já um dado que pode ter seu valor alterado durante a execução do programa é tido como uma **variável**.

Por exemplo, no cálculo da área de uma circunferência ($A = \pi\ r^2$), o valor de π é constante, pois é sempre igual a $3.1416...$, e o raio r é variável, pois pode assumir diferentes valores a cada cálculo da área. Seguindo a mesma lógica, a área A calculada para diferentes r também é variável.

8 bits) os diversos caracteres existentes

[4]Algumas linguagens de programação utilizam tipos diferentes para representar um caractere isolado e uma cadeia (ou sequência) de caracteres. Um exemplo comum é a linguagem **Pascal**, que utiliza o tipo **char** para representar um único caractere, e o tipo **string** para representar um conjunto de caracteres agrupados

3.4 Manipulação de Dados

3.4.1 Identificação

Para que os dados sejam manipulados no computador, é necessário que estes estejam associados a um nome, um identificador. O conteúdo deste identificador será o dado em si e o seu nome será usado para acessar o dado e realizar operações com o mesmo.

Uma analogia útil para entender o conceito e a necessidade de utilização do identificador seria pensar no mesmo como uma placa de sinalização que indica (delimita) uma determinada região ou espaço na memória do computador onde o dado (informação) desejado está localizado. Sendo assim, toda vez que se deseja acessar uma determinada informação utilizamos o nome dessa placa de sinalização e recuperamos o conteúdo que está localizado dentro do espaç delimitado pela mesma.

A nomeação dos identificadores deve obedecer a algumas regras, sendo elas:

1. Sempre começar com um caractere alfabético;
2. Pode ser seguido por um ou mais caracteres alfanuméricos;
3. Não conter caracteres especiais nem espaços com exceção do sublinhado '_' (essa exceção também vale para a regra do item 1, ou seja, é permitido iniciar a nomeação de um identificador com com '_').
4. Não é permitido utilizar palavras reservadas (palavras próprias da linguagem de programação, como os comandos, tipos de variáveis, etc).

Exemplos de identificadores válidos: `raio`, `_nome`, `R`, `num_clientes`, `BJ33f15`, `NumPessoasDoentes`.

Exemplos de identificadores inválidos: `(ee)`, `32-vr`, `mil*`, `12ddd`, `o:P`

Ao nomearmos os identificadores dos nossos dados é conveniente usarmos palavras mnemônicas, ou seja, palavras que nos façam lembrar o caráter do conteúdo armazenado. Isso facilita a leitura do código programado e possibilita uma melhor documentação do mesmo. Por exemplo, ao armazenarmos o nome completo, a idade e a quantidade de filhos de uma pessoa, é mais prático e coerente usarmos os identificadores `NomeSobrenome`, `Idade` e `NumFilhos` do que usarmos nomes aleatórios como `X`, `Y` e `Z`.

3.4.2 Definição

Como dito anteriormente, ao longo do programa o dado será manipulado através do nome do seu identificador, sendo assim, o primeiro passo para utilizarmos os dados é a nomeação do seu idenficador e a definição do seu tipo (no caso de identificadores variáveis), ou do seu valor (no caso de identificadores constantes). A definição dos dados em algoritmos também é conhecida como declaração.

Um identificador (sendo ele variável ou constante) declarado com um determinado tipo de dados ficará restrito a armazenar valores daquele tipo específico (inteiro, real, caractere, lógico). Na maioria dos casos, se houver uma tentativa de atribuir a um identificador um tipo diferente daquele para o qual ele foi definido irão ocorrer erros de compilação, de execução ou até mesmo perda de dados[5].

A definição de um identificador variável num algoritmo é feita da seguinte forma:

var <identificador1> [, <identificador2>,...]: <tipo1>;
    <identificador3> [, <identificador4>,...]: <tipo2>;

e a de um identificador constante da seguinte forma:

constante <identificador1> = <valor1>;
            <identificador2> = <valor2>;
            <identificador3> = <valor3>;

A seguir são apresentados alguns exemplos de declarações de ambos os tipos de identificadores:

var marca, modelo: **caractere**;
    ano: **inteiro**;
    preco: **real**;
    vendido: **lógico**;
constante PI=3.141592654;
            MAXIMO=100;

Ao declaramos um identificador variável, estamos reservando na memória do computador um espaço para armazenar valores do tipo declarado para o mesmo. Podemos comparar a memória de um computador com um móvel cheio de gavetas etiquetadas, onde cada gaveta marcada pela etiqueta corresponde a um identificador

[5]Há linguagens de programação que não exigem a definição de tipos para trabalhar com os dados, como é o caso do Python. Essas linguagens são classificadas como não-tipadas e normalmente aceitam atribuição de dados dos mais variados tipos para um mesmo identificador.

variável e o que está guardado dentro da gaveta corresponde ao valor do mesmo.

3.4.3 Atribuição

Após realizada a declaração de um identificador, é possível iniciar a manipulação dos dados que esse identificador irá representar a partir da atribuição de valores ao mesmo. Esse processo de atribuir ou alterar o valor dos dados de um identificador é chamado de **atribuição** e é representado pelo símbolo ← quando estivermos trabalhando com identificadores variáveis, e pelo símbolo = quando estivermos trabalhando com identificadores constantes [6].

A atribuição de valores a uma variável é feita da seguinte forma:

<identificador da variável> ← <valor do mesmo tipo da variável>;

<identificador da variável> ← <operações cujo resultado é do mesmo tipo da variável>;

Como pode ser visto acima, no lado esquerdo do operador ← será colocado o nome da variável que irá receber o valor, e do lado direito o valor que será armazenado na mesma. A seguir são apresentados alguns exemplos de atribuições de valores a variáveis:

```
TipoVeiculo ← "motocicleta";
Aceleracao ← 15.52;
Massa ← 12.3;
Forca ← Massa * Aceleracao;
Usado ← F;
```

Uma varíavel pode armazenar apenas um único valor por vez, sendo que sempre que um novo valor é atribuído a variável o valor anterior que estava armazenado na mesma é perdido. Por exemplo, consideremos o algoritmo 5 a seguir:

Algoritmo 5 Atribuições de valores a uma variável.

```
1: var numero: inteiro
2: numero ← 222
3: numero ← 1000
4: numero ← 23
```

[6] No caso dos identificadores constantes a atribuição é feita no momento da definição do mesmo (como apresentado na seção de declaração), sendo assim, os exemplos e explicações que seguem são restritas aos identificadores variáveis que chamaremos a partir de agora apenas de variáveis

Quando a execução passa pela linha 1 é reservado um espaço na memória do computador para armazenarmos valores do tipo inteiro, que acessaremos através da variável numero. É importante ressaltar que inicialmente, a variável numero não contém nenhum valor. Continuando a execução, passamos pela linha 2, onde o valor 222 é atribuído para a variável numero. Nesse momento, a variável numero está armazenando o valor 222. Ao passar pela linha 3 a variável numero recebe por atribuição o valor 1000, sendo que o valor anterior (222) é eliminado. Ao executarmos a linha 4, o valor 23 é atribuído para numero, e novamente o valor anterior que estava armazenado (1000) é perdido para dar espaço ao novo valor.

Capítulo 4

Expressões

4.1 Expressões Aritméticas

As expressões aritméticas são aquelas em que os operadores são aritméticos e os operandos são valores do tipo numérico (inteiro ou real). Esses valores numéricos podem ser acessados por meio de identificadores constantes ou por meio de variáveis.

As operações aritméticas fundamentais são: adição, subtração, multiplicação, divisão, potenciação, divisão inteira e o resto (módulo). A Tabela 4.1 apresenta os operadores para cada uma dessas operações aritméticas .

Operação	**Operador**	**Exemplo**
Adição	$+$	`2+3, 9+x`
Subtração	$-$	`x-1, f(x)-2`
Multiplicação	$*$	`2*1, x*y, 2*g(x)`
Divisão	$/$	`1/x, a/z, 2*f(x)`
Potenciação	$**$	`10**x, 2**3`
Resto (Módulo)	mod	`10%2, 120%10`
Divisão Inteira	$\div$	`10%2, 120%10`

Tabela 4.1: Operadores aritméticos básicos.

4.1.1 Precedência Geral dos Operadores Aritméticos

Quando uma expressão aritmética precisa ser avaliada num algoritmo, o analisador processa a expressão dando prioridade para certos operadores. As sub-expressões que contém estes operadores serão avaliadas primeiro e seu valor substituído pela sub-expressão inteira. A seguir a próxima sub-expressão na ordem é avaliada e assim

por diante até que toda a expressão corresponda a um só valor. A Tabela 4.2 mostra a ordem de prioridade na avaliação dos operadores numa expressão aritmética, chamada de precedência de operadores.

Ordem	**Operação**	**Símbolo**
1ª	Parênteses	`()`
2ª	Potenciação	`**`
3ª	Multiplicação, Divisão, Resto e Divisão Inteira	`*`, `/`, `mod`, `div`
4ª	Adição, Subtração	`+`, `-`

Tabela 4.2: Precedência Geral de Operadores Aritméticos

Conforme a tabela, as primeiras sub-expressões a serem resolvidas serão os parênteses mais internos, depois as potências, depois as multiplicações e divisões, e assim por diante. A maneira de alterar a ordem de execução das operações numa expressão aritmética é através de parênteses, sendo que eles são executados antes de tudo, a partir dos mais internos para os mais externos.

Por exemplo, considere a seguinte expressão aritmética:

```
(5+3)**2 * (5-2) + 8
  8**2   *   3   + 8
   64    *   3   + 8
        192      + 8
             200
```

A mesma é avaliada pelo computador da seguinte maneira: primeiro os parênteses mais internos são avaliados, depois a potenciação[1], depois a multiplicação e depois a soma. Se ignorássemos a precedência dos operadores (ou dos parênteses) teríamos um resultado completamente diferente e consequentemente errado.

Vejamos agora um exemplo de uma expressão que contém apenas variáveis (cujos valores ainda não foram informados):

$$(x + y)/((2 * y + (z - w)).$$

A expressão será executada na seguinte ordem:

1. $(x + y)$;
2. $(z - w)$

[1]grande parte das linguagens de programação não possui um operador aritmético específico para identificar a potenciação.

3. $2 * y$

4. adição do resultado de 2. comăoăresultado de 3.

5. divisão do resultado de 1. com o resultado de 4.

4.1.2 Escrita de Operações Aritméticas

Cada operação aritmética em um algoritmo deve ser escrita em apenas uma única linha. Sendo assim, quando uma expressão matemática usual é escrita precismamos utilizar parênteses para garantir que todas as operações sejam executadas na ordem adequada. Por exemplo, considere a expressão:

$$\frac{9 + \sqrt{23 + 2}}{4 + 3} + 23$$

cujo valor é 25, deve ser escrita e resolvida pelo analisador da seguinte forma

```
((9+(23+2)**(1/2))/(4+3))+23
((9+  25**(0.5)  )/ 7   )+23
((9+     5       )/ 7  )+23
(       14         / 7  )+23
               2         +23
                    25
```

Ao desenvolvermos um algoritmo é bastante comum deixarmos parênteses não pareados nas expressões aritméticas, o que é um erro difícil de se localizar posteriormente. Um teste prático para evitarmos esse tipo de contratempo consiste em contar na expressão quantos parênteses esquerdos e direitos existem, e conferir se eles estão em mesmo número.

4.1.3 Exceções em Expressões Aritméticas

Para a maioria das expressões aritméticas executadas em um algoritmo é possível associar um valor definido, ou seja, o resultado da expressão proprimamente dito. Por exemplo, a expressão $2 + 3$, depois de avaliada, tem um valor definido igual a 5, e a expressão $2 * *10$ tem um valor definido de 1024. Entretanto, nem todas as expressões aritméticas possuem um valor definido matematicamente, é o caso de divisões de números pelo valor 0 (zero) ou de raízes quadradas de números negativos. A avaliação

desse tipo de expressão deve ser sempre evitada a partir da verificação dos valores que farão parte das mesmas, ou seja, se um denominador é nulo ou se o número cuja raiz será extraída é negativo, a operação não deve ser realizada

4.1.4 Simplificação de Expressões Aritméticas

É importante ressaltar que expressões as aritméticas podem ser simplificadas, ou escritas de maneira diferente se observarmos as igualdades existentes entre as operações. Observe que a subtração é equivalente a soma de um número negativo; a divisão é equivalante a multiplicação pelo inverso do número, e a radiciação é idêntica a potenciação com o inverso do expoente. Ou seja,

$$\begin{aligned} x - y &= x + (-y) \\ \frac{x}{y} &= x\,\frac{1}{y} \\ \sqrt[n]{x} &= x^{\frac{1}{n}}. \end{aligned}$$

4.2 Expressões Lógicas

As expressões lógicas são aquelas cujo valor só pode ser verdadeiro ou falso. São compostas por operadores relacionais, operadores lógicos, e por identificadores variáveis ou constantes do tipo lógico.As expressões lógicas também podem ser compostas por resultados de expressões aritméticas.

4.2.1 Operadores Relacionais

Os operadores relacionais são aqueles que comparam dois valores do mesmo tipo. O retorno da expressão relacional indica se o resultado da comparação foi verdadeiro ou falso. Por exemplo, a expressão $2 < 3$ é uma expressão lógica válida cujo valor é verdadeiro. Em contrapartida, a expressão $2 = 8$ é uma expressão lógica também válida, mas cujo valor é falso. A tabela

Operador	Símbolo
Igual a	$=$
Maior que	$>$
Menor que	$<$
Maior ou Igual a	$>=$
Menor ou Igual a	$<=$
Diferente de	$<>$

Tabela 4.3: Operadores Relacionais

4.2.2 Operadores Lógicos

Os operadores lógicos são usados para representar situações lógicas que não podem ser representadas por operadores aritméticos. Também são chamados conectivos lógicos por unirem duas expressões simples numa composta. Podem ser operadores binários, que operam em duas sentenças ou expressões, ou unário que opera numa sentença só.

O primeiro deles é o operador binário de conjunção ou *e* lógico, representado por $\wedge$ ou AND. Quando duas expressões são unidas por este operador, a expressão resultante só é verdadeira se ambas expressões constituintes também são. Por exemplo "chove *e* venta" só é verdadeiro se as duas coisas forem verdadeiras, "chove" e também "venta". Se uma das sentenças não ocorrer, a sentença como um todo é falsa.

O segundo operador é o operador binário de disjunção ou *ou* lógico, representado por $\vee$ ou OR. Neste caso, se qualquer uma das expressões constituintes for verdadeira, a expressão completa também será. Por exemplo, "vou à praia *ou* vou ao campo" é um sentença verdadeira caso qualquer uma das duas ações acontecer, ou ambas. É verdadeira, se eu for a praia e não ao campo, se eu for ao campo e não a praia e se eu for a ambos.

Para o caso em que deve-se garantir que somente uma das sentenças aconteça, define-se o operador *ou*-exclusivo, cujo símbolo é $\oplus$ ou XOR. Como o nome diz, é semelhante ao operador *ou* com exclusividade na veracidade dos operandos, isto é, somente um dos operandos pode ser verdadeiro. No exemplo anterior, se o concectivo fosse o *ou*-exclusivo, a sentença composta só seria verdadeira se fosse à praia ou ao campo, mas não ambos.

O último dos operadores é o operador unário *não* lógico, representado por $\neg$. Sua função é simplesmente inverter valor lógico da expressão a qual se aplica.

Exemplificando, considere a expressão

$$(2 < 3) \wedge (5 > 1).$$

Tanto a parte $(2 < 3)$ como $(5 > 1)$ são verdadeiras, logo a expressão completa também é. A primeira parte é verdadeira *e* a segunda é verdadeira, logo toda a expressão é verdadeira. Na linguagem natural não damos tanta importância para a diferença entre *e* e *ou*. Se uma das partes fosse falsa, toda a expressão, ligadas por *e*, seria falsa. Outra maneira de avaliar o valor de uma expressão lógica é substituindo suas subexpressões por $\mathbb{V}$ ou $\mathbb{F}$, assim

$$\begin{array}{ccc} (2 < 3) & \wedge & (5 > 1) \\ \mathbb{V} & \wedge & \mathbb{V} \\ & \mathbb{V} & \end{array}$$

O operador *ou* é complementar ao operador *e*. Ele indica que a primeira expressão pode ser verdadeira *ou* a segunda expressão pode ser verdadeira. Assim, desde que um dos operandos seja verdadeiro, toda a expressão é verdadeira. Considere a expressão

$$\begin{array}{ccc} (5 < 4) & \vee & (12 > 2) \\ \mathbb{F} & \vee & \mathbb{V} \\ & \mathbb{V} & \end{array}$$

Existe uma gama finita de possíveis valores resultantes das operações executadas com operadores lógicos, pois estes só podem assumir V ou F. O resumo destas operações é o que se chama tabela-verdade dos operadores lógicos e está apresentada na Tabela 4.4

P	Q	$P \wedge Q$	$P \vee Q$	$P \oplus Q$	$\neg P$
$\mathbb{V}$	$\mathbb{V}$	$\mathbb{V}$	$\mathbb{V}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{V}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{V}$	$\mathbb{V}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{V}$	$\mathbb{F}$	$\mathbb{V}$	$\mathbb{V}$	$\mathbb{V}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{V}$

Tabela 4.4: Tabela verdade dos operadores lógicos. P e Q são sentenças lógicas quaisquer.

Ainda sobre os operadores relacionais, os operadores $<>$, $>=$ e $<=$ são redundantes, pois poderiam ser substituídos por uma composição de outros mais simples. Por exemplo,

$$(x >= b) \iff (x > b) \vee (x = b).$$

Da mesma forma,

$$(x <= b) \iff (x < b) \vee (x = b).$$

Ainda,

$$(x <> b) \iff (x < b) \vee (x > b).$$

Deste modo, somente os operadores $>, <, =$ seriam suficientes para expressar todas as expressões lógicas relacionais.

Capítulo 5

Comandos de Entrada e Saída

5.1 Saída

Para imprimirmos algum tipo de informação na tela do computador utilizamos o comando **escreva** seguido da informação que será escrita. Dessa forma, se quisermos imprimir uma mensagem como por exemplo "Ola mundo!", isto seria feito com a instrução

escreva "Olá Mundo"

As aspas servem para delimitar uma sequência de caracteres, uma constante, mas não fazem parte do conteúdo a ser impresso. Para imprimir o valor de uma variável, basta colocar o seu identificador diretamente. O fragmento de código

ttt ← 123

escreva ttt

imprime 123 na saída. Como a sequência ttt não tem aspas, durante a execução o algoritmo considera ttt como sendo o identificador de uma variável e o substitui pelo seu conteúdo, neste caso 123. Se por outro lado, colocássemos

ttt ← 123

escreva "ttt"

seria impresso "ttt" na saída. ttt é essencialmente diferente de "ttt". O primeiro indica o identificador de uma variável. O segundo, com aspas, simplesmente uma sequência de letras. Em resumo, as aspas previnem que o algoritmo interprete o conteúdo da cadeia de caracteres. Opcionalmente, pode-se colocar parênteses para indicar o argumento da função **escreva**.

É possível escrever valores de qualquer tipo existente, como valores reais, valores lógicos, valores inteiros, do tipo sequência de caracteres, resultados de expressões

aritméticas, resultados de expressões lógicas, resultados de expressões relacionais. A instrução

escreva $(8 < 9)$

irá escrever o valor da expressão relacional $8 < 9$, neste caso $\mathbb{V}$. Se quiséssemos escrever literalmente 8<9 sem interpretação, deveríamos delimitá-la com aspas. O código

escreva ("8<9")

escreve `8<9` na tela. Diversos dados a serem escritos podem ser informados numa mesma instrução separando-os por vírgula.

5.2 Entrada

Da mesma maneira que necessitamos enviar informações de dentro do algoritmo para a saida padrão (em geral a tela), também necessitamos receber informações de fora do algoritmo, a partir da entrada padrão (em geral o teclado). Considere por exemplo um sistema de locadora, sempre que alugamos um filme, o sistema irá necessitar de algumas informações como, por exemplo: o nosso código de cliente (ou o nome) e o nome da fita que estamos locando. Essas informações são fornecidas pelo sistema a partir de comandos de entrada de dados.

Para realizarmos a entrada de dados utilizaremos o comando **leia**. Ao utilizar o comando **leia** o programador deve saber de antemão qual a variável que irá armazenar o valor que será fornecido pelo usuário. No caso do exemplo anterior, os valores que seriam fornecidos pelo usuário são referentes ao código do cliente e ao nome da fita que o mesmo está locando. Sendo assim, é necessário declarar variáveis que possam armazenar valores que sejam compatíveis com as informações solicitadas ao usuário. Por exemplo, a informação do código do cliente pode ser um valor do tipo inteiro, então é necessário que declaremos no algoritmo uma variável desse tipo, seguindo esse mesmo raciocínio, a informação do nome da fita pode ser uma informação do tipo caractere, sendo também necessário que declaremos no algoritmo uma outra variável para receber essa informação.

Após declaradas as variáveis que receberão os valores fornecidos pelo usuário podemos utilizar o comando **leia** para receber esses valores. Para isso devemos escrever **leia** seguido da variável que receberá os valores entre parênteses. No algoritmo 6

No algoritmo 6, quando o algoritmo passar pela linha 3, o usuário do algoritmo (ou do sistema) deverá digitar um valor do tipo inteiro, e ao teclar <ENTER> esse

Algoritmo 6 Locadora: exemplo de entrada de dados.

```
1: var codigo_cliente: inteiro
2:     nome_fita: caractere
3: leia(codigo_cliente)
4: leia(nome_fita)
```

valor será armazenado na variável codigo_cliente. Logo em seguida, na linha 4, o usuário deverá digitar um valor do tipo caractere, e ao teclar <ENTER> esse valor será armazenado na variável nome_fita.

É possível avisarmos o usuário sobre qual tipo de informação o algoritmo está precisando, para isso, utilizamos o comando de saída de dados **escreva** imediatamente antes do **leia** que informa ao usuário o que deve entrar. Por exemplo, o algoritmo 7 informaria ao usuário o que digitar antes de executar o comando **leia** e esperar pela entrada do usuário.

Algoritmo 7 Locadora 2: exemplo de aviso para entrada de dados.

```
1: var codigo_cliente: inteiro
2:     nome_fita: caractere
3: escreva("digite código do cliente e tecle <ENTER>")
4: leia(codigo_cliente)
5: escreva("digite o nome da fita e tecle <ENTER>")
6: leia(nome_fita)
```

Parte III

Estruturas de Controle

Capítulo 6

Estruturas de Condição

Num processo geral de execução de um algoritmo implementado em uma linguagem de programação, a execução começa na primeira linha e vai avançando sequencialmente executando o código linha após linha até chegar no final. Entretanto, frequentemente surge a necessidade de colocar instruções dentro de um programa que só serão executadas caso alguma condição específica aconteça. Para esta finalidade a maioria das linguagens possui **estruturas de condição** para realizar esta tarefa. Neste capítulo examinaremos o seu funcionamento e suas peculiaridades.

Nos capítulos anteriores foram apresentados alguns conceitos básicos sobre as estruturas e comandos que são utilizados para construir um algoritmo simples. Como visto, podemos solicitar valores de entrada aos usuários do sistema utilizando o comando leia(), e podemos ainda enviar valores de saída do sistema por meio do comando escreva(). Entretanto, as possibilidades de construção de algoritmos que temos até o presente momento são bastante limitadas, pois ainda não estamos aptos a tomar decisões durante o tempo de execução do algoritmo, ou até mesmo de classificar determinados valores de variáveis.

Por exemplo, considere que precisamos desenvolver um algoritmo que classifique uma determinada pessoa entre **maior de idade** ou **menor de idade**. Para esse problema sabemos que precisamos avaliar a idade da pessoa, e que **se** essa idade for maior (ou igual) que 18 anos a pessoa é considerada **maior de idade**. Neste caso, para um intervalo de valores da idade o algoritmos executa um conjunto de ações e para outro intervalo executa um outro conjunto de ações. Neste tipo de situação, onde um determinado valor é avaliado para a partir do resultado dessa avaliação executar alguma ação, utilizamos as **estruturas de condição**.

6.1 Estrutura de Condição Simples: se-então.

A **estrutura de condição** mais simples é a se-entao, utilizada da seguinte forma:

```
se <expressão-lógica> então:
    <bloco de comandos>
fim-se
```

A <expressão-lógica> é uma expressão que deverá retornar um valor de verdadeiro ($\mathbb{V}$) ou de falso ($\mathbb{F}$), e caso o resultado dessa expressão for **verdadeiro**, será executado o bloco de comandos que está dentro da estrutura. Caso seja **falso**, a execução do programa ignora o bloco de comando e continua na linha seguinte à estutura de condição. Alguns exemplos de expressões lógicas já foram vistos anteriormente, a seguir temos mais alguns exemplos:

- 18 > 20, cujo resultado será **falso**.
- 45 = 45, cujo resultado será **verdadeiro**.
- média > 7 , cujo resultado dependerá do valor da variável média. Por exemplo, se média vale 5 o bloco **não** é executado; se média for 15, o bloco será executado.

O <bloco de comandos> é uma sequência de código que será executado somente quando o resultado da expressão lógica for verdadeiro. Por fim, a instrução fim-se indica que a estrutra se-entao chegou ao final, servido para delimitar o bloco de instruções.

Voltando ao nosso problema de classificar uma pessoa como **maior de idade** ou **menor de idade**, podemos utilizar a estrutura de condição se-então da seguinte maneira:

- Solicitamos ao usuário que digite a sua idade, e utilizamos o comando leia() para armazenar o valor digitado na variável idade.
- Depois de termos o valor da idade, avaliamos se esse valor é maior ou igual a 18.
- Se o resultado dessa avaliação for **verdadeiro** escreveremos na tela a frase “você é maior de idade”, como mostra o algoritmo 8.

Algoritmo 8 Condição: maior ou menor de idade.

```
1: var idade: inteiro
2: escreva "digite a sua idade"
3: leia(idade)
4: se idade >= 18 então
5:       escreva "você é maior de idade"
6: fim-se
```

6.2 Estrutura de Condição Composta: se-então-senão

O algoritmo 8 resolve o nosso problema quando a pessoa é maior de idade, porém não nos dá nenhum retorno para quando a mesma for menor de idade. Para contornar esse tipo de situação, a estrutura de condição se-então, oferece a possibilidade de executarmos uma determinada ação ou comando se o resultado da expressão lógica for verdadeiro e de executarmos uma ação diferente se o resultado da expressão lógica for falso. Para essas situações é utilizado o comando senão, como mostrado abaixo.

se <expressão-lógica> **então**:
    <bloco de comandos verdade>
senão:
    <bloco de comandos falsidade>
fim-se

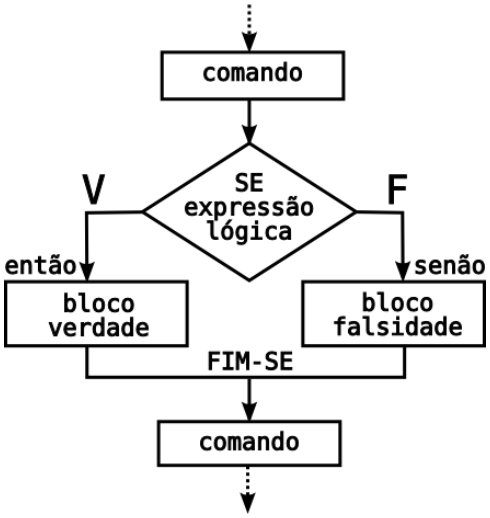

Figura 6.1: Estutura de um comando se-então.

Na estrutura se-então-senão, o bloco verdade é executado se a expressão lógica é

verdadeira e o bloco falsidade, que vem após senão, é executado se a expressão lógica é falsa. A Figura 6.1 mostra esquematicamente como funciona uma estrutura de condição se-então. de modo geral. Quando o processamento do algoritmo chega numa estrutura se-então a expressão lógica nela contida é avaliada; se o seu valor é verdadeiro ($\mathbb{V}$) então o bloco de comandos seguinte ao se, chamado bloco-verdade, é executado. Caso o resultado da expressão lógica seja o bloco de comandos posterior ao comando senão, chamado de bloco-falso, é executado. Se a estrutura de condição não possui uma cláusula senão, então no caso da expressão lógica ser falsa, a execução do algoritmo continua na linha subsequente ao bloco se-então.

O algoritmo 9 mostra como escrever que a pessoa é maior de idade quando tem 18 anos ou mais, ou que a pessoa é menor de idade quando essa condição não é atendida.

Algoritmo 9 Condição: maior ou menor de idade com se-então-senão.

```
1: var idade: inteiro
2: escreva "digite a sua idade"
3: leia(idade)
4: se idade >= 18 então
5:       escreva "você é maior de idade"
6: senão
7:       escreva "você é menor de idade"
8: fim-se
9: escreva "Boa Sorte"
```

Neste caso, a expressão lógica é (idade>=18). Se for verdadeira então o bloco-verdade (linha 5) será executado; isto depende da variável idade lida na linha 3. Caso contrário, o bloco-falso (linha 7) seria executado. A seguir, o processamento segue na linha seguinte à estrutura de condição, neste caso a linha 9 seria executada independente do valor da variável idade.

Expressões lógicas compostas em Estruturas de Condição Como vimos na seção 4.2, é possível compor expressões lógicas utilizando operadores relacionais $<, >, <>, =, <=, >=$, mas também é possível compor expressões lógicas utilizando os operadores lógicos $\wedge$ (conjunção), $\vee$ (disjunção) e $\neg$ (negação). Nesse sentido, a expressão lógica que será avaliada na estrutura de condição se-entao também pode ser formada por uma expressão lógica composta.

Por exemplo, considere a situação de um determinado aluno em uma disciplina. Sabe-se que para ser aprovado, é necessário que nota $>= 7.0$ e que frequencia > 75, ao mesmo tempo, isto é, uma conjunção lógica representada pelo operador *e* lógico ($\wedge$).

O fragmento do algoritmo que avalia a situação está mostrado na algoritmo 10. Neste caso, para que a expressão lógica como um todo seja verdadeira, é necessário que as duas expressões lógicas que a compõem também o sejam.

Algoritmo 10 Expressão lógica composta.

```
1: se (nota >= 7 ∧ frequencia > 0.75) então
2:       escreva "O aluno está aprovado"
3: senão
4:       escreva "O aluno está reprovado"
5: fim-se
```

6.3 Estruturas de Condição Encadeadas

Dentro de uma estrutura se-então-senão é perfeitamente possível utilizarmos mais de uma linha de comando, ou até mesmo outras estruturas se-então-senão . Existem situações em que os caminhos para a tomada de uma decisão acabam formando uma espécie de árvore com diversas ramificações, onde cada caminho é um conjunto de ações. Nesses casos podemos recorrer à utilização de várias estruturas se-então-senão embutidas umas dentro das outras, comumente chamadas de **ninhos**.

Algoritmo 11 Estrutura de condição composta.

```
se EL1 então
      se EL2 então
            BV2
      senão
            BF2
      fim-se
senão
      se EL3 então
            BV3
      senão
            BF3
      fim-se
fim-se
```

Nas estruturas de decisão encadeadas, uma estrutura de condição é aninhada dentro de outra, como bloco verdade ou falsidade. Neste caso, para que a estrutura de condição mais interna seja avaliada, é necessário que uma determinada condição seja

satisfeita na estrutura de condição mais externa. Considere a estrutura aninhada no algoritmo 11, onde EL significa expressão lógica, BV bloco verdade e BF bloco falsidade. Para que BV2 seja executado é necessário que a EL1 juntamente com EL2 sejam verdadeiras. Se EL1 for verdadeira mas EL2 for falsa, então é BF2 quem será executado. A tabela 6.1, chamada tabela de decisão, mostra as diferentes alternativas possíveis para este caso.

EL1	EL2	EL3	execução
$\mathbb{V}$	$\mathbb{V}$	–	BV2
$\mathbb{V}$	$\mathbb{F}$	–	BF2
$\mathbb{F}$	–	$\mathbb{V}$	BV3
$\mathbb{F}$	–	$\mathbb{F}$	BF3

Tabela 6.1: Tabela de decisão para a estrutura de condição composta mostrada no algoritmo 11.

Por exemplo, suponha que desejemos refinar um pouco mais o problema referente às médias dos alunos de uma dada disciplina. Sabemos que um aluno é aprovado caso apresente média maior ou igual a 7.0 e frequencia maior ou igual a 75%. Na verdade, em uma situação real, se o aluno obtiver a frequência minima exigida e uma média entre 3 e 7, ainda teria direito a uma última avaliação de recuperação. Como faríamos para resolver o problema em questão utilizando apenas estruturas de condição se-então-senão? Poderíamos começar avaliando a frequencia do aluno, e se a mesma for menor que 75% o aluno já estaria reprovado, porém caso a frequencia respeite o mínimo exigido, começariamos a avaliar a média para saber se está aprovado, em recuperação ou reprovado. No momento em que é verificado que a frequencia é menor que 0.75 (75%) o aluno já está imediatamente reprovado, mas caso a frequencia seja maior ou igual a esse valor, devemos continuar com o algoritmo para avaliar em que situação que o aluno se encontra. Enfim, agora é necessário avaliar a média do mesmo, verificando se está acima de 7.0 (aprovado), entre 3 e 7.0 (recuperação), ou abaixo de 3.0 (reprovado). Estes condicionais estão mostrados no algoritmo 12.

6.4 Comando caso

Uma outra alternativa para trabalhar com comandos condicionados a um determinado valor é a estrutura caso. Nessa estrutura o valor de uma determinada variável é avaliado e caso esse valor coincida com determinado valor pré-estabelecido um de-

Algoritmo 12 Verifica aprovação de alunos.

```
 1: var frequencia, media: real
 2: escreva "digite a media e a frequencia"
 3: leia(media, frequencia)
 4: se (frequencia >= 0.75) então
 5:       se (media >= 7) então
 6:             escreva "voce esta APROVADO"
 7:       senão
 8:             se (media >= 3) então
 9:                   escreva "voce esta em RECUPERACAO"
10:             senão
11:                   escreva "voce esta REPROVADO POR MEDIA"
12:             fim-se
13:       fim-se
14: senão
15:       escreva "voce esta reprovado por FALTAS"
16: fim-se
```

terminado comando é executado. A estrutura de condição caso é utilizada da forma mostrada no algoritmo 13.

Algoritmo 13 Estrutura caso.

```
caso variável seja:
    valor1:
        bloco de comandos 1
    valor2:
        bloco de comandos 2
      ⋮
    valorN:
        bloco de comandos N
    padrão:
        bloco de comandos padrão
```

Da mesma maneira que a estrutura de condição se-então possibilita que executemos algum comando quando a expressão avaliada não é verdadeira, a estrutura de condição caso também nos oferece essa opção, chamada opção padrão. O bloco de comandos dentro da opção padrão será executado caso nenhuma dos casos fornecidos seja contemplado. A sintaxe para utilizarmos essa opção é mostrada no exemplo do algoritmo 14: a variável n do tipo inteiro é testada, e caso tenha valor 1 é escrito na tela "um", caso tenha valor 2 é escrito na tela "dois" e caso não tenha nenhum desses valores será escrito na tela "outro valor".

Algoritmo 14 Exemplo de caso: mostra o numero.

```
var n: inteiro
escreva "digite n"
leia(n)
caso n seja:
    1:
        escreva "você escolheu 1"
    2:
        escreva "você escolheu 2"
    padrão:
        escreva "outro valor"
```

Capítulo 7

Estruturas de Repetição

Uma das principais características que consolidaram o sucesso na utilização dos computadores para a resolução de problemas foi a sua capacidade de repetir o processamento de um conjunto de operações para grandes quantidades de dados. Exemplos de conjuntos de tarefas que repetimos diversas vezes dentro de uma situação específica podem ser observados largamente no nosso dia a dia.

As estruturas de repetição provém uma maneira de repetir um conjunto de procedimentos até que determinado objetivo seja atingido, quando a repetição se encerra. Todas as estruturas de repetição tem em comum o fato de haver uma **condição de controle**, expressa através de uma expressão lógica, que é testada em cada ciclo para determinar se a repetição prossegue ou não.

Por exemplo, consideremos que uma determinada loja de calçados efetue uma venda no crediário para um cliente que ainda não está registrado em seu sistema. Para realizar essa venda, é necessário cadastrar o cliente, solicitando informações básicas como: nome, endereço, CPF, RG, etc. Essas etapas para realizar o cadastro seguirão sempre a mesma ordem para cada novo cliente que aparecer na loja. Caso precisássemos desenvolver um sistema para efetuar os cadastros de clientes de uma loja, não haveria lógica que programássemos novamente essas etapas para cada cliente novo, bastaria que desenvolvêssemos uma única vez a seqüência de etapas e que a cada novo cliente usássemos a seqüência previamente definida.

As estruturas de repetição são basicamente três: enquanto-faça, faça-enquanto e para-faça. A diferença básica é que enquanto-faça primeiro testa a condição para depois realizar o bloco de comando, ao contrário de faça-enquanto que primeiro executa o bloco para depois realizar o teste. A estrutura para-faça tem embutida um mecanismo de controle para determinar quando o laço deverá ser terminado.

7.1 Teste no Início: enquanto-faça.

No algoritmo 15 está mostrada o formato básico da estrutura de repeticão enquanto-faca. Antes de entrar na estrutura de repetição, a expressão lógica é avaliada, caso o resultado da mesma seja verdadeiro os comandos que estão dentro da estrutura serão executados e ao final volta-se a avaliar a expressão lógica novamente. Caso o resultado da expressão lógica seja falso, o algoritmo sai da estrutura de repetição e segue na próxima linha.

Na estrutura enquanto-faça não está pré-determinado quantas vezes o bloco de comandos será executado, sendo determinado dutante a execução do algoritmo. De maneira geral, o mecanismo que altere o valor da expressão lógica que controla o laço deve estar embutido dentro do bloco de comandos ou depender de alguma variável externa, caso contrário o laço ficará executando indefinidamente.

A estrutura enquanto-faça é usada principalmente quando não se sabe de antemão quantas repetições serão realizadas. Por exexmplo, vamos analisar de maneira mais clara uma situação onde não conseguimos determinar a quantidade de vezes que executaremos um conjunto de comandos. Por exemplo, suponha que estamos oferecendo ao usuário 3 opções de menu sendo que uma dessas 3 opções seria a opção de sair do programa. Caso desejemos que o usuário possa executar várias vezes as opções dispostas no menu, não temos como adivinhar quando o usuário irá optar por sair do algoritmo, sendo assim, não podemos limitar a repetição a determinado número de vezes.

Considere um problema mais específico onde necessitamos fazer a leitura de vários nomes de pessoas e a cada nome que é lido devemos escrever na tela a frase "O nome digitado foi nome", onde nome é a variável. A princípio isso deve ser feito inúmeras vezes e quando o usuário digitar um nome igual a “fim” o algoritmo deve parar. Da mesma maneira que no exemplo anterior não podemos definir quando o usuário irá digitar “fim”, e não temos como precisar a quantidade de vezes que o algoritmo deverá repetir esse conjunto de ações.

Nessas situações, a repetição de um conjunto de comandos é determinada pela avaliação de uma expressão lógica, ou seja, se a expressão lógica é verdadeira o conjunto de comandos continua a ser executado, caso seja falsa a estrutura de repetição é abortada.

Algoritmo 15 Estrutura de repetição enquanto-faça.

```
enquanto <expressão lógica> faça
    <bloco de comandos>
fim-enquanto
```

7.2 Teste no Fim: faça-enquanto

A estrutura faça-enquanto difere de enquanto-faça somente por executar o bloco de comando antes de testar se a condição é verdadeira. Assim, utilizando o faça-enquanto o bloco de comandos será executado pelo menos uma vez, mesmo que a expressão de controle seja falsa. Seu formato é mostrado no algoritmo 16.

Algoritmo 16 Estrutura de repetição faça-enquanto.

```
faça
    <bloco de comandos>
enquanto <expressão lógica>
```

7.3 Repetição com Controle: faça-para

A estrutura para-faça é composta de um mecanismo de controle que estabelece de antemão quantas vezes o laço será executado. A estrutura é mostrada no algoritmo 17. A <sequência> é uma lista de valores que a variável assumirá ao longo da execução, em geral no formato início,fim,incremento. Por exemplo a expressão x=0,10,2 significa que x assumirá os valores 0,2,4,6,8,10. Logo, o laço seria executado 5 vezes.

Algoritmo 17 Estrutura de repetição para-faça.

```
para variável = <sequência> faça
    <bloco de comandos>
fim-para
```

Algoritmo 18 Exemplo de estrutura de repetição.

```
var j: inteiro
para j = 1,100,1 faça
    escreva "não vou mais fazer bagunça"
fim-para
```

7.4 Contadores e Acumuladores

Em situações onde é necessário realizarmos contagens de ocorrências, ou somatórios e produtórios de valores dentro de um conjunto de dados, devemos utilizar variáveis específicas para fazer o armazenamento dos resultados. Chamamos de **contadores** para as variáveis que realizam a contagem de ocorrências de um determinado valor (ou situação) e de **acumuladores** para as variáveis responsáveis por armazenar os resultados de somatórios e produtórios de valores.

7.4.1 Contadores

Os **contadores** são normalmente inicializados com valor 0 (zero) e incrementados em 1 (um) a cada vez que uma nova ocorrência (ou situação) é observada.

Algoritmo 19 Contadores 1

```
var contador: inteiro
contador ← 0
...
contador ← contador + 1
```

Por exemplo, considere que dentro de um conjunto de informações referentes a idades e sexos de 50 pessoas, desejemos saber quantas dessas pessoas são do sexo feminino e possuem 18 anos ou mais. Para isso, é necessário inserir um contador para armazenar a quantidade de ocorrências da condição definida no enunciado. Esse contador deve ser inicializado com 0 e incrementado em 1 sempre que o sexo de uma dada pessoa é feminino e sua idade é maior ou igual a 18, como no Algoritmo 20.

7.4.2 Acumuladores

Como comentado anteriormente, os **acumuladores** são utilizados em dois tipos de situações, para a realização de somatórios e para a realização de produtórios. No caso dos **somatórios**, o acumulador é normalmente inicializado com o valor 0 e incrementado no valor de um outro termo qualquer, dependendo do problema em questão.

Considere que no problema anterior, ao invés de desejarmos calcular a quantidade de pessoas que são do sexo feminino e possuem 18 anos, desejemos calcular a soma das idades das pessoas que estão nessa situação. Nesse caso, precisamos inserir no algoritmo um acumulador, que deve ser inicializado em 0, e incrementado no valor da idade da pessoa em quetão. Veja no algoritmo 22.

Algoritmo 20 Contadores 2

var $nome$: **cadeia**
var $idade$, i, n: **inteiro**
$n \leftarrow 0$ {a variável n será o contador que armazenará o número de pessoas que pertencem ao conjunto solicitado no enunciado, ela é inicializada com um valor neutro, nesse caso 0}
para i de 0 ate 50 passo 1 **faça**
    escreva "digite sexo ('M' ou 'F') e idade da pessoa"
    leia($sexo$, $idade$)
    se $sexo$ = "M" $\underline{e}$ $idade$ >= 18 **então**
        $n \leftarrow n+1$ {aumenta em 1 a quantidade de pessoas que pertencem ao conjunto}
    fim-se
fim-para
escreva "A quantidade de pessoas do sexo feminino com 18 anos ou mais é :", n

Algoritmo 21 Acumuladores 1

var $acumulador$: **inteiro**
$acumulador \leftarrow 0$
...
$acumulador \leftarrow acumulador + termo$

Algoritmo 22 Acumuladores 2

var $nome$: **cadeia**
var $idade$, i, $soma$: **inteiro**
$soma \leftarrow 0$ {a variável soma irá armazenar o somatório das idades das pessoas que pertencem ao conjunto solicitado no enunciado, ela é inicializada com um valor neutro, nesse caso 0}
para i de 0 ate 50 passo 1 **faça**
    escreva "digite sexo ('M' ou 'F') e idade da pessoa"
    leia($sexo$, $idade$)
    se $sexo$ = "M" $\underline{e}$ $idade$ >= 18 **então**
        $soma \leftarrow soma + idade$ {aumenta o somatório no valor da idade da pessoa em questão}
    fim-se
fim-para
escreva "A soma das idades das pessoas do sexo feminino com 18 anos ou mais é :", $soma$

Um algoritmo para calcular a média das idades das pessoas do sexo feminino com 18 anos ou mais, pode ser facilmente desenvolvido utilizando um contador para armazenar a quantidade de pessoas que pertencem a esse conjunto e um acumulador para armazenar a soma das idades dessas pessoas.

No caso de utilizarmos **acumuladores** para armazenar **produtórios** é necessário a inicialização do mesmo com o valor neutro da multiplicação (o número 1). A cada iteração o acumulador é então multiplicado por um outro termo qualquer, dependendo do problema em questão.

Algoritmo 23 Acumuladores 3

```
var acumulador: inteiro
acumulador ← 1
...
acumulador ← acumulador * termo
```

Por exemplo, para calcular o fatorial de um determinado número, devemos escrever o Algoritmo 24.

Algoritmo 24 Acumuladores 4

```
var n, i, fat: inteiro
escreva "digite o número inteiro para calcular o fatorial"
leia(n)
fat ← 1 {a variável fat irá armazenar o fatorial da variável n, ela deve ser inicializada com
um valor neutro para a multiplicação, nesse caso 1}
para i de 1 ate n passo 1 faça
    fat ← fat * i {a cada iteração o valor do fatorial é acumulado em fat e multiplicado
    pelo contador i}
fim-para
escreva "O fatorial de n é igual a ", fat
```

Parte IV

Estrutura de Dados e Modularização

Capítulo 8

Variáveis Compostas Homogêneas

8.1 Vetores Unidimensionais

Vetores são varíaveis compostas que podem armazenar um conjunto de valores. Todos estes valores são referenciados através do **nome do vetor** (o mesmo para todo o conjunto de valores) e de um **índice** (distinto para cada valor.) As variáveis vetoriais são análogas aos vetores usados na matemática e na física, em que um vetor, por exemplo

$$\vec{x} = (x_1, x_2, x_3),$$

é constituído por três valores x_1, x_2 e x_3; neste caso o nome do vetor é x e os índices são 1, 2 e 3. Ao contrário de um escalar que possui só um valor, x é uma variável composta por 3 valores. As variáveis vetoriais na prática são constituídas por um grande número de valores.

As valores armazenados numa variável vetorial são todos do mesmo tipo, por isso os vetores são chamados de **variáveis compostas homogêneas**.

Os vetores são imprescindíveis quando se quer armazenar diversos valores de um mesmo tipo e referenciá-los com o mesmo nome. Por exemplo, para armazenar as idades de vários alunos de uma turma, poderia-se criar um vetor idade com 8 posições; cada índice de 0 a 7 corresponderia a um funcionário. A Figura 8.1 ilustra a variável **idade**, os respectivos valores armazenados (na ordem, 23, 22, 18, 34, 23, 21, 25, 39) e os índices de cada elemento.

Os vetores são declarados anexando-se ao nome da variável um colchete com o número de posições que o vetor porerá conter:

int idade[100].

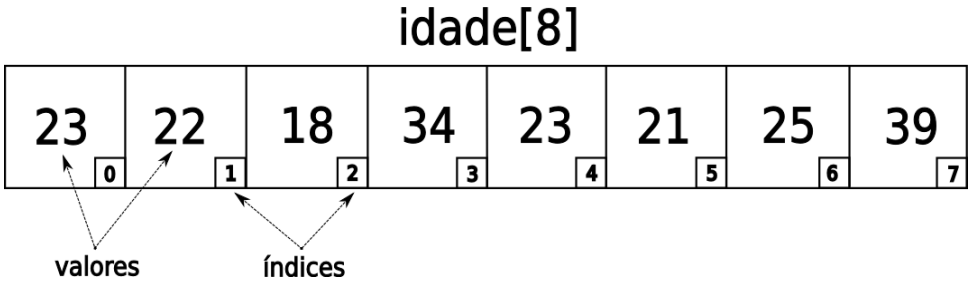

Figura 8.1: Vetor idade[8] com seus valores e índices.

Neste caso será criada uma variável idade[] que conterá 8 posições – índices 0 a 7 – onde poderão ser armazenados números inteiros. Os vetores podem ser de qualquer tipo alfanumérico.

Cada uma das posições do vetor são referenciadas através do nome do vetor seguido do respecivo índice colocado entre colchetes. O Algoritmo 25 mostra como definir todos os valores da variável idade[].

Algoritmo 25 Definindo os valores da variável **idade**.

```
1: var idade[8] int
2: idade[0] = 23
3: idade[1] = 22
4: idade[2] = 18
5: idade[3] = 34
6: idade[4] = 23
7: idade[5] = 21
8: idade[6] = 25
9: idade[7] = 39
```

É importante notar que uma variável de N posições possui índices de 0 a $N-1$. Na variável idade de 8 posições usam-se os índices 0 a 7; qualquer índice fora desta faixa resulta em erro.

A grade vantagem de se usar índices dentro do nome da variável é a possibilidade de referenciar um dado elemento do vetor através de um índice variável. Por exemplo, para imprimir todos os valores da variável idade, ao invés de colocar **escreva** idade[0], **escreva** idade[1], ..., é muito mais simples colocar a instrução que se repete (no caso escreva) dentro de uma estrutura de repetição, como mostra o Algoritmo 26.

Algoritmo 26 Imprimindo todos os valores da variável idade[].

```
// valores de idade[] já definidos
para i = 0,7,1 faça
    escreva idade[i]
fim-para
```

EXEMPLOS

exemplo: imprimindo os valores na ordem inversa.

exemplo: média de 100 valores.

8.2 Vetores Bidimensionais

exemplo: jogo de damas.

exemplo: somando matrizes.

8.3 Vetores Multidimensionais

exemplo: arestas de um cubo.

Capítulo 9

Módulos

9.1 Modularização

9.2 Retorno de Valores

9.3 Escopo de Variáveis

9.4 Passagem de Parâmetros

9.4.1 Por Valor

9.4.2 Por Referência

Referências Bibliográficas

[1] Marco Medina, Cristina Fertig, ALGORITMOS E PROGRAMAÇÃO: TEORIA E PRÁTICA, São Paulo, Novatec Editora, 2005.

[2] Marco Antônio Furlan de Souza et al., ALGORITMOS E LÓGICA DE PROGRAMAÇÃO, São Paulo, Thomson Learning, 2006.

[3] Wikipedia, `http://www.wikipedia.org`, visitado em 23/abril/2007

[4] André Luiz Villar Forbellone, Henri Frederico Eberspächer, LÓGICA DE PROGRAMAÇÃO, São Paulo, Pearson Prentice Hall, 2005

[5] Álvaro Borges de Oliveira, Isaias Camilo Boratti, INTRODUÇÃO À PROGRAMAÇÃO DE ALGORITMOS, Florianópolis, Bookstore, 1999.

[6] Irenice de Fátima Carboni, LÓGICA DE PROGRAMAÇÃO, São Paulo, Thomson Learning, 2003.

Parte V

Apêndice

Apêndice A

Exercícios

A.1 Introdução

1.. Diferencie um algoritmo de um programa.

2.. Crie algortimos simplificados para executar cada uma das tarefas a seguir:

- ▷ Tomar um banho
- ▷ Fazer um bolo
- ▷ Tirar uma fotografia
- ▷ Ligar um automóvel
- ▷ Cadastrar de um cliente
- ▷ Tricotar uma blusa
- ▷ Ler uma revista

3.. Represente 2 dos algoritmos acima na forma de um fluxograma.

4.. Pense em um problema existente na sua rotina diária (particular ou no trabalho) e monte um algoritmo de acordo com os passos que você normalmente utiliza para resolvê-lo.

5.. Considere um algoritmo que calcule a área de um triângulo $A = (b * h)/2$. Quais são os valores de entrada? Qual será o processamento do algoritmo? Qual o valor

de saída? Quais são os valores que variam e quais permanecem constantes ou fixos durante a execução do algoritmo?

6.. Considere que se deseja desenvolver um algoritmo para calcular a quantidade de azulejos que são necessários para cobrir uma determinada parede. Quais são os valores de entrada? Qual será o processamento do algoritmo? Qual o valor de saída? Quais são os valores que variam e quais permanecem constantes ou fixos?

A.2 Dados

1.. No final da execução do fragmento de código abaixo, qual é o valor de n1, n2 e n3?

```
var n1,n2: inteiro
n1 ← 10
n2 ← 30
n3 ← n1
n1 ← n2
n2 ← n3
```

2.. Quais os valores de v1, v2 e v3 no final da execução do código abaixo?

```
var v1,v2,v3: logico
v1 ← 8 > 9
v2 ← 8 < 9
v3 ← v1 ∨ v2
v1 ← v3 ∧ v2
```

3.. Que problema existe no algoritmo abaixo?

```
var num1: inteiro
num1 ← (20 > 9)
```

4.. O que será impresso para cada um das instruções abaixo?

1. **escreva**("numero= ", 78)
2. **escreva** ("veja esse resultado")
   escreva (78 + 2)

3. **escreva** (76 > 8 , 8*2, "três")
4. **escreva**("ola", "ola de novo", 3<9)
5. **escreva**("verdadeiro, 8 = 8")
6. **escreva**(8+8, 12<8, "bla bla")
7. n1 ← 4
   n2 ← 10
   escreva(n1, n2)
8. n1 ← 4
   n2 ← 10
   escreva(n1+n2)
9. n1 ← 4
   n2 ← 10
   escreva("n1+n2")
10. n1 ← 4
    n2 ← 10
    escreva("n1+n2")
11. n1 ← 4
    n2 ← 10
    escreva(n1>n2)
12. n1 ← 4
    n2 ← 10
    Escreva("n1 + n2 =", n1 + n2)
13. n1 ← 4
    n2 ← 10
    Escreva(n1, "+", n2, " =", n1 + n2)

A.3 Estruturas de Condição

1. Elabore um algoritmo que leia um número e imprima uma das mensagens: é múltiplo de 3, ou, não é múltiplo de 3.

2. Desenvolva um algoritmo que classifique um número de entrada fornecido pelo usuário como *par* ou *ímpar*.

3. Elabore um algoritmo que leia um número, e se ele for maior do que 20, imprimir a metade desse número.

4. Elabore um algoritmo que leia dois números inteiros e efetue a adição; caso o resultado seja maior que 10, imprima-o.

5. Elabore um algoritmo que leia um número e, se ele for positivo, imprima a metade desse número, caso contrário imprima o número ao quadrado.

6. O sistema de avaliação de determinada disciplina é composto por três provas. A primeira prova tem peso 2, a segunda tem peso 3 e a terceira tem peso 5. Considerando que a média para aprovação é 7.0, Faça um algoritmo para calcular a média final de um aluno desta disciplina e dizer se o aluno foi aprovado ou não.

7. Elabore um algoritmo que leia dois números e responda se a divisão do primeiro pelo segundo é exata (o resto da divisão deve ser igual a 0). Se for, o algoritmo deve imprimir a mensagem "A divisão de (1^o numero) por (2^o número) é exata".

8. Elabore um algoritmo que leia o nome e o peso (em real) de duas pessoas e imprima os dados da pessoa mais pesada.

9. Elabore um algoritmo que leia um número e informe se ele é ou não divisível por 5.

10. Elabore um algoritmo que indique se um número digitado está compreendido entre 20 e 90, ou não.

11. Um comerciante comprou um produto e quer vendê-lo com um lucro de 45% se o valor da compra for menor que R$ 20,00; caso contrário, o lucro será de 30%. Elabore um algoritmo que leia o valor do produto e imprima o valor de venda para o produto.

12. Segundo uma tabela médica, o peso ideal está relacionado com a altura e o sexo. Elabore um algoritmo que leia a altura e o sexo de uma pessoa, calcule e imprima seu peso ideal, utilizando as seguintes fórmulas.
Para homens (72.7*altura)–58
Para mulheres (62.1*altura)–44.7

13. Elabore um algoritmo para testar se uma senha digita é igual a "Patinho Feio". Se a senha estiver correta escreva "Acesso permitido", do contrario emita a mensagem "Você não tem acesso ao sistema".

14. Elabore um algoritmo que leia dois números e imprima qual é maior, qual é menor, ou se são iguais.

15. Uma empresa qualquer decidiu conceder um aumento de salários a seus funcionários de acordo com a tabela abaixo:

Salário Atual	Aumento
0 – 400,00	15%
400,01 – 700,00	12%
700,01 – 1.000,00	10%
1.000,01 – 1.800,00	7%
1.800,01 – 2.500,00	4%
acima de 2.500,00	Sem aumento

Escrever um algoritmo que leia o salário atual de um funcionário e escreva o percentual de seu aumento e o valor do salário corrigido a partir desse aumento. Utilize a estrutura de condição se-então aninhadas.

16. Considerando o sistema de avaliação das médias colocado a seguir, escreva um algoritmo que avalie a média de um aluno, o seu conceito correspondente e escreva a mensagem: "APROVADO" se o conceito for A, B, ou C e "REPROVADO" se o conceito for D ou E. Utilize a estrutura de condição se-entao-senao aninhadas.

Média	Conceito
>= 9.0	A
>= 7.5 e < 9.0	B
>= 6.0 e < 7.5	C
>= 4.0 e < 6.0	D
< 4.0	E

17. Elabore um algoritmo que leia o nome, nota da avaliação 1 e nota da avaliação 2 de um aluno. Ao final, imprima o nome do aluno, suas notas, a média aritmética e uma das mensagens: Aprovado, Reprovado ou em Prova Final (a média é 7,0 para aprovação, menor que 3,0 para reprovação e as demais em prova final).

18. Elabore um algoritmo que leia o salário de uma pessoa e imprima o desconto do INSS segundo a tabela a seguir utilizando a estrutura de condição se-então aninhadas:

Faixa de salário	Desconto
Menor ou igual a R$ 600,00	Isento
Maior que R$ 600,00 e menor ou igual a R$ 1200,00	20%
Maior que R$ 1200,00 e menor ou igual a R$ 2000,00	25%
Maior que R$ 2000,00	30%

20. Construa a tabela de decisão (veja Tabela 6.1) para o algoritmo 27 abaixo, conforme os possíveis valores de q1, q2, q3, especificando o que será impresso em cada caso ao final da execução do algoritmo.

Algoritmo 27 Problema 20

```
var q1, q2, q3: logico
escreva "A"
se (q1) então
    escreva "B"
    escreva "C"
fim-se
se (q2) então
    escreva "D"
fim-se
escreva "E"
se (q3) então
    escreva "F"
senão
    escreva "G"
fim-se
escreva "H"
```

21. Construir um algoritmo que tome como entrada três valores e os imprima em ordem crescente.

22. Elabore um algoritmo que leia um número e informe se ele é divisível por 10, por 5, por 2, ou se não é divisível por nenhum deles.

23. Elabore um algoritmo para ler três valores e verificar se eles podem ser os comprimentos dos lados de um triângulo, e se forem dizer o tipo de triângulo. Para ser um triângulo é necessário que qualquer um dos lados seja menor que a soma dos outros

dois lados, ($A < B + C$) ou ($B < A + C$) ou ($C < A + B$). Utilize a estrutura de condição se-entao aninhadas. Equilátero é aquele que tem os três lados iguais ($A = B = C$.) Isósceles é aquele que tem dois lados iguais ($A = B$) ou ($A = C$) ou ($B = C$). Escaleno é aquele que tem todos os lados diferentes ($A <> B <> C$)

24. Criar um algoritmo que leia dois números inteiros, e que solicite ao usuário qual a operação deseja realizar entre esses números. Caso o usuário digitar o caractere "*" será realizada uma multiplicação, caso seja digitado o caractere "/" será realizada uma divisão, caso seja digitado o caractere "+" será realizado uma adição, e caso seja digitado o caractere "–" será realizada uma subtração. Utilize a estrutura de condição caso.

25. Elabore um algoritmo que leia um número inteiro entre 1 e 12 e imprima o mês correspondente. Caso seja digitado um valor fora desse intervalo, deverá ser exibida uma mensagem informando que não existe mês com esse número.

A.4 Estruturas de Condicao – exercícios avançados

1.. Desenvolva um algoritmo que solicite ao usuário duas variáveis de valor inteiro (variáveis A e B) e que ao final do processamento a variável A contenha o valor da variável B dividido por 2, e a variável B contenha o valor da variável A dividido por 3.

2.. Sabe-se que a direção de uma determinada escolinha faz a distribuição de seus alunos de acordo com as idades dos mesmos. Dessa forma, os alunos são distribuídos nas seguintes turmas de acordo com a classificação a seguir:

TURMA – Faixa de Idade
TURMA A – de 4 a 5 anos
TURMA B – de 6 a 8 anos
TURMA C – de 9 a 10 anos
SEM TURMAS – abaixo de 4 anos, acima de 10 anos

Desenvolva que leia a idade de uma única criança e informe em qual turma a mesma irá ter aulas. O algoritmo deve se preocupar em responder para o usuário que a escolinha não possui turmas para a criança caso a mesma tenha menos que 4 anos ou mais que 10 anos.

3.. Numa loja de eletrodomésticos, as compras têm um preço à vista, ou acréscimo de 10 % para pagamentos em 2 vezes, ou ainda, acréscimo de 20% para pagamento em 3 vezes. O programa deve pedir para o usuário entrar com o valor da compra a vista e a opção de compra. O programa deve exibir qual o valor final a ser pago.

4.. Faça um programa para calcular a conta final de um hóspede de um hotel fictício, considerando que:

- Devem ser lidos o nome do hóspede, o tipo do apartamento utilizado (A, B, C ou D), o número de diárias utilizadas pelo hóspede e o valor do consumo interno do hóspede;
- O valor da diária é determinado pela seguinte tabela:

TIPO DO APTO – VALOR DA DIÁRIA (R$)
A – 150.00
B – 100.00
C – 75.00
D – 150.00

- O valor total das diárias é calculado pela multiplicação do número de diárias utilizadas pelo valor da diária;
- O subtotal é calculado pela soma do valor total das diárias e o valor do consumo interno;
- O valor da taxa de serviço equivale a 10% do subtotal;
- O total geral resulta da soma do subtotal com a taxa de serviço.

Escreva a conta final contendo: o nome do hóspede, o tipo do apartamento, o número de diárias utilizadas, o valor unitário da diária, o valor total das diárias, o valor do consumo interno, o subtotal, o valor da taxa de serviço e o total geral.

A.5 Estruturas de Repeticao

1.. Desenvolva um algoritmo que calcule as tabuadas dos números divisíveis por 2 que encontram-se no intervalo entre 1 e 10.

2.. Desenvolva um algoritmo que calcule o valor de π a partir da seguinte expressão matemática:

$$\pi = 4\sum_{k=0}^{n}(-1)^k(\frac{1}{(2k+1)})$$

onde n deve ser informado pelo usuário e corresponde ao grau de precisão no cálculo do valor de π

Algoritmo 28 Calculo do pi

```
var k, n: inteiro
var pi: real
escreva "digite n"
leia(n)
para k de 0 ate n passo 1 faça
    se k % 2 = 0 então
        pi ← pi + 1/(2 * k + 1)
    senão
        pi ← pi − 1/(2 * k + 1)
    fim-se
fim-para
```

3.. Resolva o exercício anterior sem a utilização da estrutura de condição se-então-senão

4.. Um mês antes das eleições municipais, um determinado partido político encomendou uma pesquisa de opinião sobre as intenções de votos dos eleitores. Foram entrevistas 50 pessoas que indicaram suas intenções de acordo com as seguintes opções: (A) candidato A, (B) candidato B, (C) indeciso. Desenvolva um algoritmo que faça a leitura das intenções de votos dessas 50 pessoas e que informe ao final a porcentagem de intenções para cada uma das opções existentes (candidatos A e B, e indecisos).

5.. Desenvolva um algoritmo para calcular e imprimir o valor de S na expressão a seguir:

$$S = \frac{1}{1} - \frac{3}{2} + \frac{5}{3} - \frac{7}{4} + ... - \frac{99}{50}$$

6.. Desenvolva um algoritmo que calcule o valor de S para um determinado valor de n informado pelo usuário a partir da seguinte expressão:

$$S_n = \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + ... + \frac{n}{n+1}$$

7.. Desenvolva um algoritmo capaz de calcular o resultado da seguinte expressão aritmética, onde o valor de n é informado pelo usuário:

$$S_n = 1^1 + 2^2 + 3^3 + ... + n^n$$

8.. Desenvolva um algoritmo que realize o sorteio de um número inteiro pertencente ao intervalo de 1 a 100, e que solicite ao usuário qual o valor que foi sorteado. O algoritmo deve informar se o valor que o usuário digitou é maior, menor ou igual ao valor sorteado. O algoritmo deve parar quando o usuário acertar o valor sorteado e deve informar ao final a quantidade de tentativas que o usuário utilizou para acertar o número. Considere a existência de um comando chamado SORTEIO que retorna um valor aleatório de 1 até um número informado da seguinte forma:

Algoritmo 29 Adivinhacao

```
var sorteado: inteiro
sorteado ← SORTEIO(100)  {o número 100 indica que o sorteio será de 1 a 100}
```

9.. Uma determinada empresa fez uma pesquisa de mercado para saber se as pessoas gostaram ou não de um novo produto que foi lançado. Para cada pessoa entrevistada foram coletados os seguintes dados: Sexo (M ou F) e Resposta (Gostou ou Não Gostou). Sabendo-se que foram entrevistados N pessoas, faça um programa que forneça:

- ▷ Número de pessoas que gostaram do produto
- ▷ Numero de pessoas que não gostaram do produto
- ▷ Percentagem de pessoas do sexo masculino que não gostaram do produto
- ▷ Informação dizendo em que sexo o produto teve uma melhor aceitação.

10.. Desenvolva um algoritmo que calcule o valor de X que é dado por:

$$X_n = n + \frac{n-1}{2} + \frac{n-2}{3} + ... + \frac{1}{n}$$

A.6 Vetores

1.. Desenvolva um programa que solicite a idade, o nome e o sexo de 10 pessoas e armazene esses dados em vetores. O programa deve oferecer um menu que permita ao usuário as seguintes opções:

1. Informar os dados das 10 pessoas.
2. Sair

Após o usuário ter informado os dados das 10 pessoas (caso a opção 1 seja selecionada), o programa deve oferecer as seguintes opções em um segundo menu:

1. Consultar o nome da pessoa mais nova
2. Consultar a idade do homem mais idoso
3. Consultar a média das idades das mulheres
4. Sair

A cada consulta realizada o programa deverá apresentar novamente o menu com as opções disponíveis e só deverá ser encerrado quando o usuário escolher a opção Sair.

2.. Desenvolva um algoritmo que ofereça ao usuário as seguintes opções:

1. Inserir números inteiros em um vetor de até 10 posições. Os números devem ser inseridos de modo que o vetor nunca fique desordenado (em nenhum momento). Ao se tentar inserir um número em um vetor cheio o programa deve acusar que não será possível realizar a inserção. Após a inserção, o algoritmo deve imprimir a quantidade de elementos do vetor e os respectivos elementos.
2. Excluir um elemento do vetor a partir de seu valor. O usuário deverá entrar com o número que deseja excluir do vetor e o mesmo deverá ser retirado. Os demais elementos que se localizam após o elemento excluído devem ser realocados para suas novas posições. O algoritmo deve informar quando não existirem mais elementos para excluir.
3. Imprimir na tela os elementos do vetor em ordem CRESCENTE.

4. Imprimir na tela os elementos do vetor em ordem DECRESCENTE.

5. Sair do programa.

Obs: O objetivo do item 1 não é o de ordenar o vetor, mas sim de manter o vetor ordenado a cada inserção, sendo assim, o vetor nunca chegará a estar desordenado, ou seja, antes de inserir cada elemento, o algoritmo deve procurar em qual posição o mesmo deve ser inserido, e depois realizar a inserção exatemente naquela posição.

A.7 Matrizes

1.. Desenvolva um algoritmo que solicite ao usuário a ordem de duas matrizes A e B (máximo 10x10) e seus respectivos elementos. Após a inserção dos elementos das matrizes A e B o programa deve oferecer ao usuário as seguintes opções:

1. mostrar as duas matrizes

2. multiplicar as duas matrizes e mostrar a matriz resultante. Caso a multiplicação das matrizes não seja possível o programa deve informar ao usuário o motivo da impossibilidade.

3. Sair do programa.

Obs: Os elementos da matriz devem ser do tipo inteiro ou do tipo real.

A.8 Modularização

1.. Para um grupo de valores reais, determinar o valor do desvio padrão destes valores em relação a média dos valores. O desvio padrão de um grupo de valores pode ser obtido por:

$$dp = \sqrt{\frac{\sum_{i=1}^{n} X_i^2 - \frac{(\sum_{i=1}^{n} X_i)^2}{n}}{n-1}}$$

Para esse problema devem ser desenvolvidas 3 funções que irão receber como parâmetros o conjunto de elementos e a quantidade de elementos desse conjunto, e que farão os seguintes processamentos:

1. Cálculo da soma dos quadrados dos elementos do grupo.
2. Cálculo do quadrado da soma dos elementos do grupo.
3. Cálculo do Desvio Padrão.

O programa principal deve oferecer para o usuário as seguintes opções:

1. Informar a quantidade e os valores dos elementos do grupo.
2. Calcular o desvio padrão do grupo.
3. Sair.