Lógica de Programação

Autor:
Fernandes José Rodrigues da Silva

Curso de Ciência da Computação

Apostila da Disciplina Introdução à Programação

Introdução à Programação

Índice

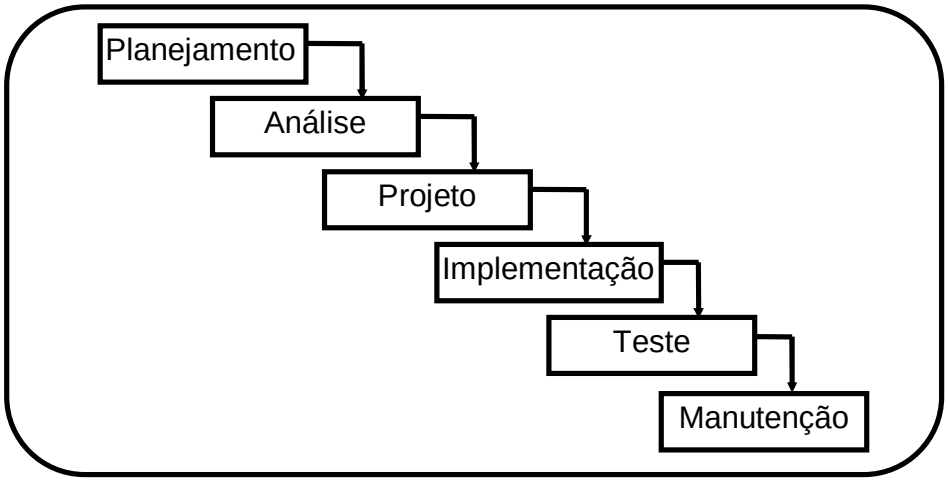
1. Uma Visão Geral do Processo de Desenvolvimento de Software

Antes de começarmos a estudar lógica de programação, é importante que tenhamos uma visão geral do processo de desenvolvimento de software, uma vez que estudaremos lógica de programação com o objetivo de conseguir um bom embasamento para a prática da programação de computadores.

O processo de desenvolvimento de software consiste basicamente num conjunto de atividades divididas em etapas, onde o objetivo é, ao executar estas etapas, chegar à efetiva construção de um software[1].

Podemos encontrar na literatura em Informática[2] várias formas de representação das etapas em que consiste o processo de desenvolvimento de software. Estas formas de representação podem variar tanto na quantidade de etapas como nas atividades que devem ser realizadas por cada etapa.

A seguir apresentaremos uma forma de representação do processo de desenvolvimento de software que é bastante encontrada na literatura:



Etapas do processo de desenvolvimento de software.

Na figura 1 podemos ver o processo de desenvolvimento de software dividido em 6 etapas. A seguir daremos uma rápida explicação das atividades realizadas por cada etapa.

- **Planejamento:** na etapa de planejamento é onde deve ser desenvolvido um plano inicial de desenvolvimento, levando em considerações questões como definição da abrangência do sistema, missão e objetivos do sistema, cronogramas de desenvolvimento, análise custo x benefício, levantamento inicial de informações etc.

[1] Conjunto de instruções (comandos) interpretáveis pelo computador.
[2] Mais especificamente na área de engenharia de software.

- **Análise:** também chamada de análise de requisitos, é onde deve se obter um claro entendimento sobre o sistema. A análise proporciona a base para uma boa implementação do software. Nesta etapa são construídos os modelos do sistema.

- **Projeto:** também chamada de especificação do projeto, é onde propomos uma arquitetura de implementação para o software, que atenda aos requisitos do sistema identificados na análise. Aqui passamos a nos preocupar com os diversos aspectos computacionais necessários para uma boa implementação do software. Os algoritmos dos programas a serem implementados são construídos nesta fase.

- **Implementação:** a etapa de implementação é onde os programas são efetivamente construídos, a partir da arquitetura de implementação feita na etapa anterior. Nesta etapa é onde a atividade de codificação ocorre de forma massiva.

- **Teste:** nesta etapa todos os programas construídos serão testados de forma exaustiva. Existe uma grande variedade de testes que são realizados, indo desde o teste unitário dos módulos de programas até o teste de integração de todo o sistema de software.

- **Manutenção:** é onde ocorrem ajustes do software implementado, que podem ser ocasionados por vários motivos: erros de projeto identificados após a implementação e o teste do software, inovações tecnológicas, evolução do sistema etc.

2. Introdução à Lógica de Programação

Como este é um curso de lógica de programação, vamos iniciar nossos estudos procurando entender o que é **lógica** de uma forma geral. A seguir serão dadas algumas definições que procuram elucidar o termo lógica.

Lógica:

- [do grego logiké, que significa "arte de raciocinar"]. Na tradição clássica, aristotélico-tomista, conjunto de estudos que visam a determinar os processos intelectuais que são condição geral do conhecimento verdadeiro. (1a definição encontrada no Dicionário Aurélio da Língua Portuguesa)

- Coerência de raciocínio, de idéias. (6a definição encontrada no Dicionário Aurélio da Língua Portuguesa)

- Maneira de raciocinar particular a um indivíduo ou a um grupo: *a lógica da criança, a lógica primitiva, a lógica do louco*. (7a definição encontrada no Dicionário Aurélio da Língua Portuguesa)

A lógica trata da correção do pensamento. Como filosofia ela procura saber por que pensamos de uma forma e não de outra. Poderíamos dizer também que a lógica é a arte de pensar corretamente e, visto que a forma mais complexa de pensamento é o raciocínio, a Lógica estuda ou tem em vista a "correção do pensamento". **A Lógica ensina a colocar Ordem no Pensamento.**

Exemplo:

1) Todo vulcano tem orelhas pontudas.
   Spock é vulcano.
   Logo, Spock tem orelhas pontudas.

2) Todo aluno que se formar em Ciência da Computação pela Unimep será um bom profissional em Informática.
   Se você se formar em Ciência da Computação pela Unimep, então você será um bom profissional em Informática. (mas para isso terá que estudar muito!!!)

Normalmente somos bem sucedidos na execução de uma tarefa quando empregamos raciocínio lógico (lógica). Se queremos ter bons resultados numa prova escolar, devemos estudar o assunto que será tratado na prova (isto é lógico, pois o objetivo da prova é fazer com que o aluno apreenda o conteúdo em pauta, ao passo que "colar" seria ilógico, pois o aluno está enganando a si mesmo, não apreendendo o conteúdo necessário). Se queremos ter sucesso numa modalidade esportiva, devemos treinar, descansar e nos alimentarmos adequadamente. Se queremos desenvolver bons programas de computador, devemos programa-lo logicamente, para que este possa resolver o problema desejado da forma mais otimizada possível, dado um conjunto de restrições. É neste ponto que entra o conceito de lógica de programação.

Lógica de Programação: raciocínio lógico empregado no desenvolvimento de programas de computador, fazendo uso ordenado dos elementos básicos suportados por um dado estilo de programação.

Uma boa lógica de programação é desenvolvida a partir de um conjunto de elementos, entre eles:

- Organização
- Criatividade
- Perseverança
- Padronização
- Otimização

3. Algoritmos

Definição: uma seqüência de passos (ações) que devem ser executados para se alcançar um determinado objetivo.

Embora a palavra pareça um pouco estranha, executamos algoritmos quotidianamente. Por exemplo: a rotina diária de um aluno, que se levanta de manhã, se prepara, pega um ônibus (ou carro) para vir até a UNIMEP, assiste as aulas, volta para a casa (de ônibus ou carro), estuda durante a tarde, toma banho, janta, estuda depois da janta, e em seguida vai dormir, é um algoritmo que a maioria dos alunos executa diariamente (ou pelo menos deveria executar). Um algoritmo também pode ser comparado como uma receita culinária. Se obedecermos passo a passo as instruções da receita, chegamos sempre ao mesmo resultado.

No desenvolvimento de programas, estaremos tratando constantemente com a **complexidade** inerente aos problemas que desejamos que nosso programa de computador resolva. Para tratarmos da complexidade dos problemas desenvolvemos algoritmos que devem expressar de forma objetiva e clara (**legibilidade**) a forma de resolvermos o problema. A partir do algoritmo desenvolvido fica fácil construirmos o programa, basta conhecermos a linguagem de programação que se pretende adotar. Uma vez construído o algoritmo, podemos transportá-lo para qualquer linguagem de programação, o que nos dá uma flexibilidade para a efetiva implementação do programa.

Existem três estruturas básicas para a construção de algoritmos: **sequenciação, seleção e repetição.** A combinação destas três estruturas permite-nos a construção de algoritmos para a resolução de problemas extremamente complexos. A programação estruturada se baseia nestas três estruturas básicas.

Imagine a seguinte situação: precisamos elaborar um algoritmo para trocar uma lâmpada.

Algoritmo 1:

Início
- pegue uma escada;
- coloque-a embaixo da lâmpada;
- busque uma lâmpada nova;
- suba na escada com a lâmpada nova;
- retire a lâmpada velha;
- coloque a lâmpada nova;
- desça da escada.

Fim

Observe que este algoritmo resolve o nosso problema da troca de lâmpada. No entanto, trata-se de um algoritmo bastante simples, que utiliza-se apenas da estrutura de sequenciação, ou seja, nenhuma seleção ou repetição de

procedimentos aparece no algoritmo. Uma **estrutura de seqüência**, caracteriza-se por possuir uma única seqüência de ações, que é executada apenas uma vez.

No entanto, antes de trocarmos a lâmpada devemos nos certificar de que ela realmente esteja queimada, para então trocá-la. Assim, podemos melhorar o nosso algoritmo.

Algoritmo 2:

Início
- ligue o interruptor;
- se a lâmpada não acender, então:
    - pegue uma escada;
    - coloque-a embaixo da lâmpada;
    - busque uma lâmpada nova;
    - suba na escada com a lâmpada nova;
    - retire a lâmpada velha;
    - coloque a lâmpada nova;
    - desça da escada.

Fim

Observe que agora o nosso algoritmo, além da estrutura de seqüência, passa a utilizar uma estrutura de seleção. Na **estrutura de seleção**, uma condição deve ser analisada, a partir do resultado da análise, um "caminho" do algoritmo será executado. Em outras palavras, uma estrutura de seleção seleciona ações a serem executadas a partir de uma condição (que pode ser simples ou composta).

Embora nosso algoritmo tenha melhorado, ainda podemos deixá-lo mais completo. Quando verificamos que a lâmpada está queimada, subimos para trocá-la, mas não consideramos a hipótese da lâmpada nova também estar queimada, e se isso ocorrer, precisaremos executar algumas ações novamente, até que possamos efetivamente resolver nosso problema.

Algoritmo 3:

Início
- ligue o interruptor;
- se a lâmpada não acender, então:
    - pegue uma escada;
    - coloque-a embaixo da lâmpada;
    - enquanto a lâmpada não acender, faça:
        - busque uma lâmpada nova;
        - suba na escada com a lâmpada nova;
        - retire a lâmpada velha;
        - coloque a lâmpada nova;
        - desça da escada.

Fim

Neste algoritmo somente iremos parar de trocar a lâmpada quando colocarmos uma lâmpada que acenda. Portanto, um conjunto de ações será executado repetidamente enquanto a condição de repetição for verdadeira. Assim, inserimos uma **estrutura de repetição** no nosso algoritmo, que passa a trabalhar com as três estruturas básicas de construção de algoritmos. É importante salientar que existem várias formas de se construir um algoritmo, pois as pessoas pensam de formas diferentes, no entanto, devemos sempre buscar a forma mais otimizada possível (dentro dos limites impostos pela situação).

Exercícios de Raciocínio

Elabore um algoritmo em linguagem natural para resolver as situações colocadas a seguir:

1) Um homem precisa atravessar um rio com um barco que possui capacidade de transportar apenas ele e mais uma de suas três cargas, que são: um cachorro, uma galinha e um saco de milho. O que o homem deve fazer para conseguir atravessar o rio sem perder as suas cargas ?

2) Uma Torre de Hanói é formada por três discos sobrepostos transpassados por uma haste. Tendo mais duas hastes e podendo mover um disco por vez, mas nunca deixando um disco maior sobre um disco menor, como podemos passar os discos para uma outra haste ?

3) Três jesuítas e três canibais precisam atravessar um rio. No entanto dispõem apenas de um barco com capacidade para duas pessoas. Por medida de segurança não se permite que em alguma das margens do rio a quantidade de jesuítas seja inferior à quantidade de canibais. Qual a seqüência de viagens necessárias para a travessia do rio com segurança para os jesuítas ?

3.1 Formas de Representação de Algoritmo.

Existem várias ferramentas que podem ser utilizadas para a representação de algoritmos, entre elas: linguagem natural, pseudocódigo, diagrama de Nassi-Shneiderman (ou Chapin), fluxograma etc..

Estas ferramentas procuram padronizar formas de representação, facilitando a posterior transformação do algoritmo para um conjunto de códigos[3].

Linguagem Natural

A representação de algoritmos através de linguagem natural é a forma mais espontânea de representação de algoritmos, pois descrevemos os passos do algoritmo utilizando o nosso linguajar quotidiano, escrevendo o algoritmo como um texto simples.

[3] Símbolos que fazem sentido dentro do contexto de uma linguagem de programação.

Por exemplo, queremos resolver o seguinte problema: Calcular a média de todos os alunos que cursaram uma disciplina X, a partir da leitura das notas da 1a e 2a prova, passando por um cálculo de média aritmética. Após a média calculada, devemos anunciar se aluno foi aprovado ou reprovado por nota. Somente estão aprovados alunos com média maior ou igual a 5,0.

Representação do algoritmo em linguagem natural: Para todos os alunos da disciplina X, faça: ler as notas da 1a e 2a prova, somar as notas e dividir por dois, chegando assim, ao resultado da média do aluno. Se a média do aluno for maior ou igual a 5,0, então o aluno está aprovado, senão o aluno está reprovado. Fazer para o próximo aluno.

O problema da representação de algoritmos com linguagem natural é que quanto maior a complexidade do problema, maior a dificuldade de entendermos o texto que procura descrever os passos do algoritmo, pois não se emprega nenhum recurso diagramático, e não há uma rigidez na estruturação das ações.

Pseudocódigo

O pseudocódigo vem sendo amplamente utilizado por projetistas de software e programadores, pois obriga o uso de estruturas que facilitam o entendimento do algoritmo, e também facilitam a transformação do mesmo em códigos reais. O pseudocódigo também recebe outros nomes, como: português estruturado, PDL (*Program Design Language*), pascalóide etc.. Utilizaremos neste curso o pseudocódigo como a forma de representação padrão para algoritmos. O exemplo anterior será representado através de pseudocódigo.

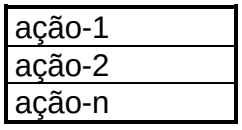
```
Início
   real: nota1, nota2, media;
   Enquanto não for fim da lista de alunos, faça
   Início
      Leia nota1;
      Leia nota2;
      média = (nota1 + nota2) / 2;
      Se média >= 5,0  então
      Início
         Escreva “Aluno aprovado”;
      Fim
      Senão
         Início
            Escreva “Aluno reprovado”;
         Fim
   Fim
Fim
```

Observe que no pseudocódigo, somos obrigados a utilizar algumas estruturas básicas de controle (seqüência, seleção e repetição), de forma a estruturar e organizar melhor os passos do algoritmo.

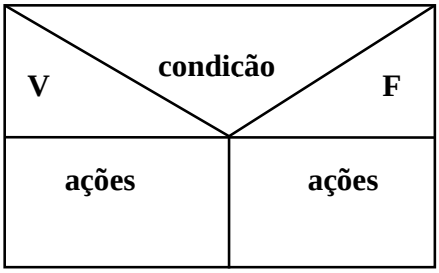
Diagrama de Nassi-Shneiderman

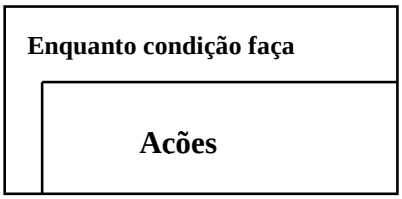
Também conhecido como diagrama Chapin, esta ferramenta de representação oferece grande clareza para a representação de sequenciação, seleção e repetição num algoritmo, utilizando-se de uma simbologia própria. A idéia básica deste diagrama é representar as ações de um algoritmo dentro de um único retângulo, subdividido-o em retângulos menores, que representam os diferentes blocos de seqüência de ações do algoritmo. Seleção e repetição também são representadas de forma gráfica, dentro dos retângulos.

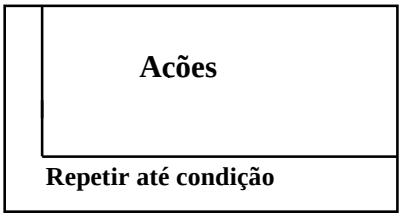
Seqüência



Seleção

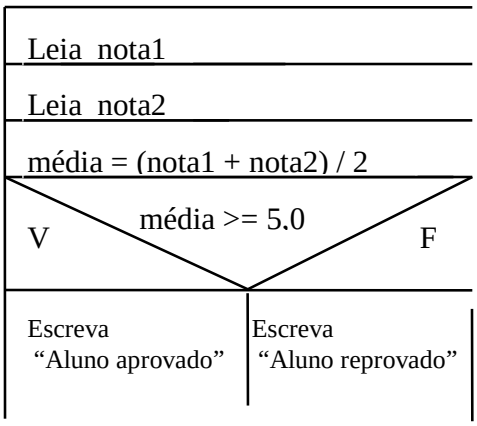






Exemplo:

Enquanto não for fim da lista de alunos faça



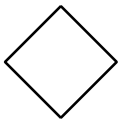
Embora os diagramas N-S ofereçam uma representação muito clara do algoritmo, à medida que os algoritmos vão se tornando mais complexos, fica difícil realizar os desenhos necessários numa única página, prejudicando a sua visualização.

Fluxograma

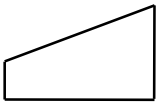
O fluxograma foi utilizado por muito tempo para a representação de algoritmos. No entanto, o seu grande problema é permitir o desenvolvimento de algoritmos não estruturados. Com o advento das linguagens de programação estruturada o fluxograma caiu em desuso.

O fluxograma utiliza-se de símbolos específicos para a representação de algoritmos. Existe uma certa variação na simbologia empregada, apresentamos a seguir uma simbologia tradicionalmente usada:

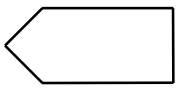
Processo



Decisão

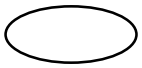


Leitura

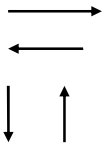


Escrita

Conector

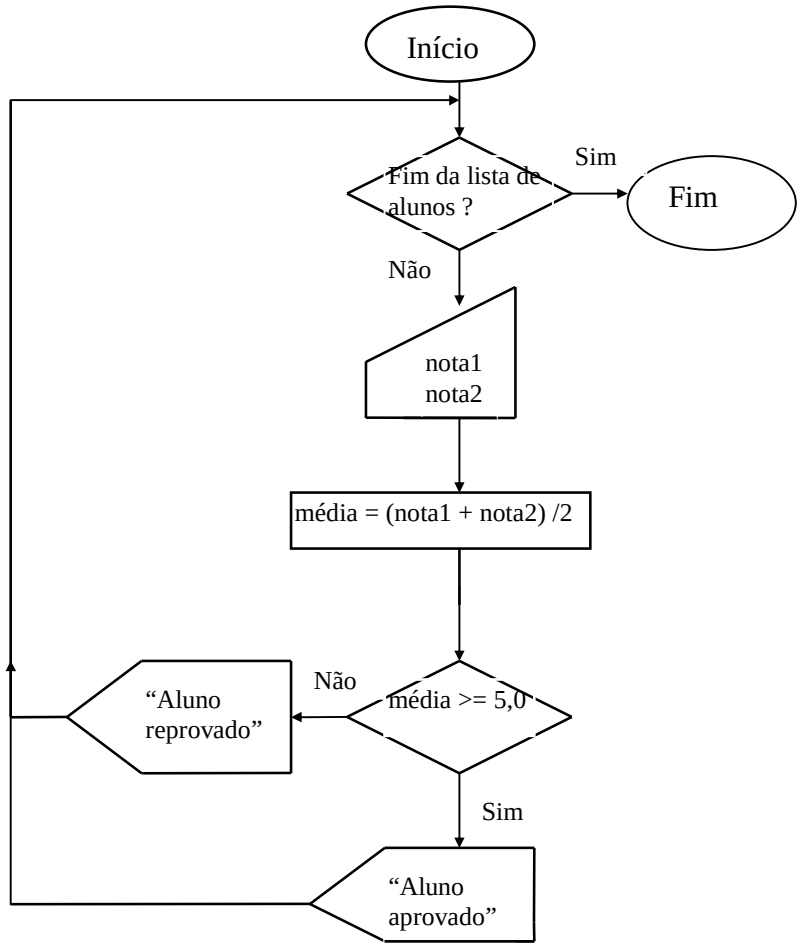


Terminal



Setas de fluxo
de controle

Exemplo de Representação com Fluxograma:



3.2 Variáveis e Constantes.

Variáveis e constantes são conceitos fundamentais para a construção de algoritmos e programas de computadores, pois são através deles que um algoritmo "guarda" os dados do problema.

Todo dado que tem a possibilidade de ser alterado (portanto, sofrer variação) no decorrer do tempo, deverá ser tratado como uma **variável** do problema, e portanto deverá ser definido como tal no algoritmo a ser desenvolvido.

Quando um dado não tem nenhuma possibilidade de variar com o decorrer do tempo, ele deverá ser tratado como uma **constante**, e dessa forma definido e tratado como tal no algoritmo.

Por exemplo, considere o desenvolvimento de um algoritmo para o seguinte problema: calcular as áreas de cinco triângulos com medidas diferentes. Sabemos que a fórmula para o cálculo da área de um triângulo é b.h/2, onde b é o valor da base e h é o valor da altura do triângulo. Sendo assim, b e h são dados que irão variar no decorrer do "tempo de execução" do algoritmo, portanto deverão ser tratados como variáveis, o número 2 da fórmula é um dado constante, não sofrerá variação no decorrer do "tempo de execução" do algoritmo, portanto deve ser tratado como uma constante.

Para manipularmos adequadamente os dados do problema no algoritmo, temos que identifica-los corretamente dentro deste. Para isso, devemos atribuir nomes para eles, estes nomes são chamados de **identificadores**. Seguiremos as seguintes regras para a formação de identificadores:

1) Devem começar por um caractere alfabético;
2) Podem ser seguidos por mais caracteres alfabéticos e/ou numéricos;
3) Não é permitido o uso de caracteres especiais, como: @, #, &, *, +, ? etc. (exceto o sublinhado).

Exemplos de identificadores válidos:
a) X b) X3 c) base d) altura1 e) teste_11 f) a1b2c3

Exemplos de identificadores inválidos:
a) 1X b) X 3 c) A%1 d) B-2 e) maior que 10 f) >10

3.3 Tipos Primitivos de Dados.

Todo dado a ser tratado num algoritmo deve pertencer a algum tipo, que irá determinar o domínio de seu conteúdo. Os tipos mais comuns de dados são conhecidos como tipos primitivos de dados, são eles: inteiro, real, caractere e lógico.

Inteiro: todo e qualquer dado numérico que pertença ao conjunto de números inteiros relativos (negativo, nulo ou positivo). Exemplos: 15, -5, 0, 234.

Real: todo e qualquer dado numérico que pertença ao conjunto de números reais (negativo, nulo ou positivo). Exemplos: 15,34 123,08 0,005 -12,0.

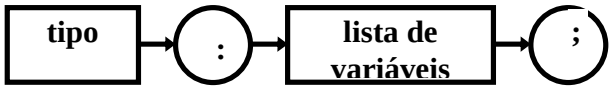
Caractere: todo e qualquer dado composto por um conjunto de caracteres alfanuméricos (números, letras e caracteres especiais). Exemplos: “Aluno Aprovado”, “10% de multa”, “Confirma a exclusão ??”, “S”, “99-3000-2”.

Lógico: todo e qualquer dado que só pode assumir duas situações (dados biestáveis, algo como verdadeiro ou falso).

3.4 Declaração de Variáveis.

Toda variável possui algum conteúdo, que será armazenado por ela e manipulado pelo algoritmo. As variáveis que serão utilizadas nos algoritmos devem ser declaradas inicialmente. A declaração de uma variável indica o tipo de dado que ela pode “guardar” no decorrer da execução do algoritmo (ou no decorrer da execução do programa que futuramente será construído).

Para declararmos uma variável, temos que criar um identificador para ela, que será o nome da variável no algoritmo, e também temos que definir o tipo de dado que a variável pode armazenar. Faremos a declaração de variáveis obedecendo ao seguinte padrão:



onde, **tipo** pode ser inteiro, real, caractere ou lógico, e **lista de variáveis** serão os nomes dos identificadores.

Exemplos:

```
inteiro     : X, RA;
real        : peso, altura;
caractere   : nome, end;
lógico      : resposta, z;
```

3.5 Operadores (aritméticos, relacionais e lógicos).

Quando construímos algoritmos é comum trabalharmos com expressões matemáticas para a resolução de alguns problemas. As expressões matemáticas podem fazer uso de operadores aritméticos e relacionais.

Chamamos de **operadores aritméticos** o conjunto de símbolos que representa as operações básicas da matemática, conforme tabela a seguir:

Operações	Operadores
adição	+
subtração	-
multiplicação	*
divisão	/
potenciação	**
radiciação	//

As variáveis (ou constantes) manipuladas pelos operadores são chamadas de operandos. Exemplos:

- A + B (A e B são operandos, e + é o operador da expressão)
- 2 * A
- A / B
- X ** 2
- 2 // 9
- C - 4

OBS: Criaremos um operador aritmético especial para obter o resto de uma divisão, chamaremos este operador de **resto**. Exemplo: 20 resto 3, o resultado obtido será 2. Pois o resto da divisão de 20 por 3 é 2.

Um outro grupo importante de operadores é formado pelos **operadores relacionais**. Quando queremos fazer comparações entre valores, ou entre expressões (tanto matemáticas como lógicas), precisamos utilizar esta categoria de operadores. Os operadores relacionais são:

Comparações	Operadores
igual	=
diferente	< >
maior	>
menor	<
maior ou igual	>=
menor ou igual	<=

O resultado de uma relação é sempre um valor lógico (verdadeiro ou falso).

Exemplos:

a) 2 + 5 = 3 + 2
7 = 5

Resultado: F (falso)

b) 3 * 5 < > 2 * 3
15 < > 6

Resultado: V (verdadeiro)

Uma terceira categoria de operadores que podemos utilizar na construção de algoritmos é chamada de **operadores lógicos**. Estes operadores funcionam como conectivos para a formação de novas proposições. Os principais operadores lógicos são:

Operações	**Operadores**
conjunção	e
disjunção (não-exclusiva)	ou
disjunção (exclusiva)	xou
negação	não

Os resultados das operações lógicas são sempre valores lógicos (verdadeiro ou falso).

Para trabalharmos adequadamente com operadores lógicos, temos que conhecer a tabela verdade para cada um dos operadores. Uma tabela verdade é o conjunto de todas as possibilidades combinatórias entre os valores das variáveis lógicas, conforme apresentado a seguir.

A	**B**	**A e B**
F	F	F
F	V	F
V	F	F
V	V	V

A	**B**	**A ou B**
F	F	F
F	V	V
V	F	V
V	V	V

A	B	A xou B
F	F	F
F	V	V
V	F	V
V	V	F

A	não A
F	V
V	F

Exemplos:

a) 3 > 6 ou 4 < 5
F ou V

Resultado: V

b) 4 < 7 e 5 > 9
V e F

Resultado: F

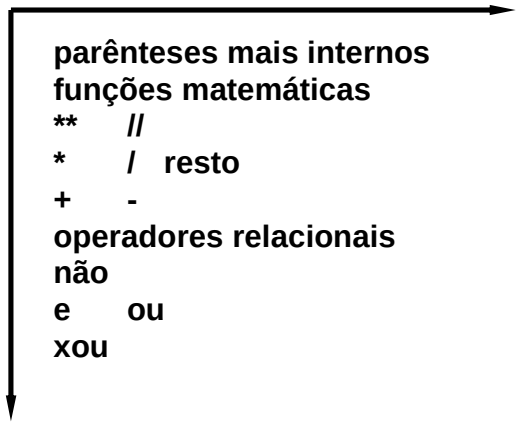
Além dos operadores citados acima, também podemos usar algumas funções matemáticas em nossos algoritmos[4], conforme tabela a seguir:

Função	Resultado
sen(x)	seno de x
cos(x)	cosseno de x
tg(x)	tangente de x
arcsen(x)	arco cujo seno é x
arccos(x)	arco cujo cosseno é x
arctg(x)	arco cuja tangente é x
abs(x)	valor absoluto (módulo) de x
int(x)	a parte inteira de x
Frac(x)	a parte fracionária de x
Ard(x)	Arredondamento de x
Sinal(x)	-1, +1 ou zero
Rnd(x)	valor randômico de x

Exemplos: sinal(-44) = -1 ard(3,50) = 4,00
abs(2) = 2 frac(1,78) = 78
rnd(10) = 8 (pode resultar qualquer número inteiro do intervalo de 0 a 9)

[4] Utilizaremos as principais funções matemáticas fornecidas pela maioria dos compiladores.

Na resolução das expressões aritméticas, lógicas e relacionais, os operadores e as funções matemáticas possuem uma hierarquia de prioridade.



Exemplo:

não 2 ** 3 < 4 **2 ou abs(int(15.0/-2)) < 10
não 8 < 16 ou abs(int(-7,5)) < 10
não 8 < 16 ou abs(-7) < 10
não 8 < 16 ou 7 < 10
não V ou V
F ou V
Resultado: V

4. Comandos de Atribuição, Entrada e Saída de Dados.

Um comando (ou instrução) pode ser definido como sendo uma ação a ser executada num dado momento pelo algoritmo.

4.1 Comando de Atribuição.

O comando de atribuição permite-nos atribuir um valor para uma certa variável, onde o tipo do dado atribuído para a variável deve ser compatível com o tipo declarado para a variável. A sintaxe utilizada será:

identificador ← expressão;

onde **identificador** é o nome da variável que receberá o valor da **expressão**.

Exemplo:

```
inteiro      : X, Y;
real         : A;
caractere    : nome;
lógico       : Z;
X ← 0;
Y ← 10 + 7;
A ← 0.089;
nome ← “exemplo de atribuição”;
Z ← verdadeiro;
```

4.2 Comando de Entrada de Dados.

Na prática de construção de programas, será muito comum o uso de comandos que proporcionam a entrada de dados para o computador. Assim, devemos ter uma representação correspondente em nível de algoritmo para a entrada de dados. Utilizaremos o comando **leia** para efetuar a entrada de dados para o algoritmo, conforme sintaxe abaixo.

leia(variável);

onde **variável** receberá um valor vindo de “fora” do algoritmo para que algum processamento ocorra.

Exemplo:

```
inteiro      : X;
real         : A;
caractere    : nome;
leia(X);
leia(A);
leia(nome);
```

4.3 Comando de Saída de Dados.

Assim como para entrada de dados, na prática de construção de programas será muito comum o uso de comandos que proporcionam a saída de dados gerados pelo computador. Assim, devemos ter uma representação correspondente em nível de algoritmo para a saída de dados. Utilizaremos o comando **escreva** para efetuar a saída de dados do algoritmo, conforme sintaxe abaixo:

escreva(variável, constante, expressão);

onde o algoritmo mostrará os valores de variáveis, constantes e/ou expressões.

Exemplo:

```
Início
   inteiro      : X,Y;
   real         : A,B;
   caractere  : nome;
   escreva(“Entre com o valor de X:”);
   leia(X);
   escreva(“Entre com o valor de A:”);
   leia(A);
   escreva(“Entre com o nome da pessoa:”);
   leia(nome);
   Y ← X * 3;
   B ← (A * 2,4) / 3;
   escreva(“A partir dos valores de X e A, Y e B valem: “,Y, B);
Fim
```

5. Estrutura de Seqüência.

A estrutura de seqüência é a estrutura mais simples que utilizamos na construção de algoritmos estruturados. Ela é formada por um conjunto de instruções (ou comandos) que serão executadas numa seqüência linear de cima para baixo e da esquerda para a direita, ou seja, da mesma forma como elas foram escritas. Utilizamos as palavras **início e fim** para delimitar o bloco de seqüência, conforme sintaxe abaixo:

```
Início
   instrução 1;
   instrução 2;
   instrução 3;
      .
      .
      .
   instrução N;
Fim.
```

Exemplo:

```
Início
   real: A,B,C;
   leia(A);
   leia(B);
   C ← 2 // (A**2 + B**2);
   escreva(C);
Fim.
```

6. Estrutura de Seleção.

Uma estrutura de seleção permite a escolha de um conjunto de ações e/ou estruturas que serão executadas a partir do resultado de uma condição (simples ou composta), representada por uma expressão lógica.

Para estrutura de seleção do tipo **se-senão** utilizaremos a seguinte sintaxe:

```
Se <condição> então
   Início {bloco verdade}
      comando-1;
      comando-2;
        .
        .
        .
      comando-n;
   Fim
Senão
   Início {bloco falso}
      comando-1;
      comando-2;
        .
        .
        .
      comando-n;
   Fim
```

Na estrutura apresentada acima, o primeiro bloco será executado se o resultado da condição for verdade, o segundo bloco somente será executado se o resultado da condição for falso. É importante observar que se não houverem comandos a serem executados caso o resultado da condição seja falso, basta não utilizarmos a parte **senão** da estrutura.

Tanto para a parte **se** como para a parte **senão** da estrutura apresentada, quando houver apenas um comando a ser executado, podemos eliminar os delimitadores de bloco (início e fim).

Uma outra estrutura de seleção que utilizaremos será a estrutura de **seleção de múltipla escolha** (também chamada de estrutura de caso). A seguinte sintaxe será adotada:

```
Escolha  X
   caso V1: C1;
   caso V2: C2;
   caso V3: C3;
      .
      .
      .
   caso Vn: Cn;
   caso contrário: Cn+1;
Fim
```

Nesta estrutura **X** é a variável que será verificada para cada **caso**, ou seja, se o conteúdo de **X** for igual ao valor de **V** para algum dos casos, então **C** (que representa um comando primitivo) será executado. Se precisarmos que vários comandos ou estruturas sejam executados após o resultado da comparação de **X** com **V**, então devemos trabalhar com delimitadores de bloco (início e fim).

V pode assumir um valor único ou vários valores para cada **caso**. Para trabalharmos com mais de um valor por **caso** podemos separar os valores por vírgulas **(,)** ou por dois pontos seguidos **(..)**.

Exemplo:

```
Escolha  opção
   caso 1: escreva("teste 1");
   caso 2: escreva("teste 2");
   caso 3,4,5: escreva("teste 3");
   caso 6..10: escreva("teste 4");
Fim
```

No exemplo acima não foi usado a opção **caso contrário.** Está opção somente deve ser empregada se quisermos que algum comando seja executado quando nenhum dos casos acima tenha ocorrido.

7. Estrutura de Repetição

A estrutura de repetição permite que uma seqüência de comandos seja executada um certo número de vezes até que uma determinada condição seja satisfeita. Por exemplo, pode-se citar o caso em que se deseja realizar o mesmo processamento para um conjunto de dados diferentes, como a folha de pagamento de uma empresa de 100 funcionários. Neste caso o mesmo cálculo é efetuado para cada um dos funcionários. Para solucionar este problema precisaríamos escrever o algoritmo em questão uma vez para cada funcionário, ou seja, sendo 100 funcionários teríamos que escrevê-lo 100 vezes, o que se tornaria inviável. Outro modo de resolver essa questão seria utilizar a mesma seqüência de comandos, ou seja, fazer a repetição de um conjunto de comandos 100 vezes sem ter que rescrevê-lo novamente.

As estruturas de repetição são muitas vezes chamadas de **Laços** ou **Loops** e se dividem em:

- **Laços Condicionais**: quando não se conhece o número de vezes que um conjunto de comandos no interior do laço será repetido. A repetição ou não dos comandos dependerá do resultado de uma condição. As estruturas de repetição que implementam esse tipo de laço também são conhecidas como: **repetição com teste no início e repetição com teste no final do laço.**

- **Laços Contados:** quando se conhece previamente quantas vezes o conjunto de comandos será executado. Esse tipo de estrutura também é conhecida como **repetição com variável de controle.**

Repetição Com Teste no Início

Caracteriza-se por uma estrutura que efetua um teste lógico no início do laço. Se o resultado do teste for verdadeiro o laço é executado retornando novamente ao teste lógico e assim o processo será repetido enquanto a condição testada for verdadeira.

Para realizar a repetição com teste no início, utilizamos a estrutura **enquanto**, conforme sintaxe abaixo:

Enquanto <condição> **Faça**
    Comando;

```
Enquanto  <condição> Faça
Início
    Comando1;
    Comando2;
    ....
    Comando n;
Fim
```

Exemplo:- Dada a descrição de um produto e o preço desenvolver um algoritmo que calcule e mostre o novo preço do produto com um aumento de 30%. Repetir o processo enquanto o usuário desejar.

```
Início
   Caracter   : resposta, desc;
   Real       : preco, n_preco;
   resposta ← 'S';
   Enquanto (resposta = 'S')  ou (resposta = 's')  Faça
   Início
      Leia(desc,preco);
      n_preco ← preco * 1.30;
      Escreva ('O novo preço de ',desc,' é = ', n_preco);
      Leia(resposta);
   Fim
Fim.
```

No momento em que o resultado do teste lógico <condição> for falso o processamento é desviado para o fim do laço. Se o resultado do teste lógico já for falso no primeiro teste o laço não é executado nenhuma vez.

Repetição Com Teste no Final

Caracteriza-se por uma estrutura que permite que um laço seja executado até que o resultado do teste lógico seja verdadeiro. Neste caso o laço é executado pelo menos uma vez e então a condição é testada, se o resultado for falso o laço é executado novamente e este processo é repetido até que o resultado da condição seja verdadeiro. A diferença que existe dessa estrutura para a estrutura do **enquanto** é o fato do laço ser executado pelo menos uma vez antes de passar pela condição de teste.

Para realizar a repetição com teste no final, utilizamos a estrutura **repita**, conforme sintaxe abaixo:

```
Repita
    Comando1;
```

```
        Comando2;
        ...
        Comando n;
    Até <condição>
```

Exemplo:- O mesmo do anterior.

```
Início
   Caracter   : resposta,desc;
   Real       : preco, n_preco;
   Repita
      Leia(desc,preco);
      n_preco ← preco * 1.30;
      Escreva ('O novo preço de ',desc,' é = ', n_preco);
      Leia(resposta);
   Até (resposta = 'N')  ou (resposta = 'n')
Fim.
```

No momento em que o resultado da condição de teste for verdadeiro o processamento de repetição é interrompido.

Repetição Com Variável de Controle

É utilizada quando se conhece previamente o número de vezes que se deseja executar um determinado conjunto de comandos. Esse tipo de laço nada mais é que uma estrutura dotada de mecanismos próprios para contar o número de vezes que o laço vai ser executado obedecendo os limites fixados.

Para realizar a repetição com variável de controle utilizados a estrutura **para**, conforme sintaxe abaixo:

```
Para  V de Vi até VF passo p faça
   Comando;

Para  V de Vi até VF passo p faça
Início
   Comando1;
   Comando2;
      ...
   Comando n;
Fim
```

Onde: V é a variável de controle
Vi : é o valor inicial da variável V
Vf : é o valor final da variável V
p : é o valor do incremento dado a variável V

Exemplo:- O mesmo do anterior para 50 produtos.

```
Início
   Caracter   : descricao;
   Real       : preco, n_preco;
   Inteiro    : x;
   Para x de 1 até 50 passo 1 faça
   Início
      Leia(descricao,preco);
      n_preco ← preco * 1.30;
      Escreva ('O novo preço de ',descricao,' é = ', n_preco);
   Fim
Fim.
```

O processo de repetição é executado enquanto a variável V tenha um valor menor ou igual ao valor da variável Vf. O laço é finalizado quando a variável V ultrapassar o valor de Vf.

O algoritmo abaixo será representado utilizando-se as três estruturas de repetição. Enunciado: dados 10 valores quaisquer desenvolver um algoritmo que calcule e mostre quantos desses valores são positivos e quantos são negativos.

- Utilizando a Estrutura com teste no início:

```
Início
    Real    :valor;
    Inteiro :totalp,totaln,cont;
    cont    ← 0;
    totaln  ← 0;
    totalp  ← 0;
    Enquanto cont < 10 Faça
    Início
       Leia (valor);
       Se valor >=0 então totalp ← totalp + 1;
       Senão totaln ← totaln + 1;
       cont ← cont + 1;
    Fim
    Escreva ( 'Total de números positivos = ',totalp);
    Escreva ( 'Total de números negativos = ',totaln);
Fim.
```

- Utilizando a Estrutura com teste no final:

```
Início
    Real    :valor;
    Inteiro :totalp,totaln,cont;
    cont    ← 0;
```

```
    totaln  ← 0;
    totalp  ← 0;
    Repita
       Leia (valor);
       Se valor >=0 então totalp ← totalp + 1;
       Senão totaln ← totaln + 1;
       cont ←  cont + 1;
   Até cont =10;
   Escreva ( 'Total de números positivos = ',totalp);
   Escreva ( 'Total de números negativos = ',totaln);
Fim.
```

- Utilizando a Estrutura com variável de controle:

```
Início
    Real    :valor;
    Inteiro  :totalp,totaln,cont;
    cont    ← 0;
    totaln  ← 0;
    totalp  ← 0;
    Para cont de 1 até 10 passo 1 Faça
    Início
       Leia (valor);
       Se valor >=0 então totalp ← totalp + 1;
       Senão totaln ← totaln + 1;
    Fim
    Escreva ( 'Total de números positivos = ',totalp);
    Escreva ( 'Total de números negativos = ',totaln);
Fim.
```

LISTA DE EXERCÍCIOS

1) Determine qual é o tipo primitivo das informações:

a) “Pare”
b) 2
c) 100.59
d) “Preserve o meio ambiente”
e) 'S'
f) falso
g) –57.3
h) -5000
i) “UNIMEP”
j) 0.001

2) Assinale os identificadores válidos:

( ) (X)
( ) ab*c
( ) k2
( ) C&A
( ) 'CLIENTE'
( ) #49
( ) x(3)
( ) 1A
( ) nº
( ) quantidade

3) Encontre os erros da seguinte declaração de variáveis:

inteiro : endereco, nfilhos;
caracter: idade, resposta;
real : peso, altura, nome;
logico : situacao;

4) Supondo que as variáveis COD, NOME, SEXO, DEPTO, FUNÇÃO, SAL, NDEP, ADIC, VALE, TIPO, sejam utilizadas para armazenar informações referentes a um funcionário de uma empresa, a saber: código, nome, sexo, nome do departamento em que trabalha, função que exerce, salário mensal, número de dependentes, adicional de função, se tem direito a vale transporte e tipo sangüíneo, declare-as corretamente.

5) Supondo A, B e C variáveis do tipo inteiro, com valores 5, 10 e -8, e D variável do tipo real com valor 1.5, quais os resultados das expressões abaixo:

a) 2 + A * ABS(3) + C
b) 3 + 2 // (2 * ABS(C)) / 2
c) (B -6 * 2)**2 * D + 2
d) B RESTO 3 + A RESTO 2
e) 2 * A RESTO 3-C
f) 2//(2*ABS(C)) / 4
g) (FRAC(A/B)+SINAL(C))**3
h) ARD(ABS(C/2+D))-INT(A/2)
i) 3+(3//(C+16))*((3RESTOA)+0.5*2)
j) (A+B) / A *ARD(SINAL(C)+D)-INT(D*2)

6) Determine os resultados (verdadeiro ou falso) obtidos na avaliação das expressões lógicas seguintes, sabendo que A, B, C, D e E contém respectivamente 2, 7, 3.5, 'noite' e 'frio' e que existe uma variável lógica L cujo valor é falso:

a) (B = A * C) e L ou verdadeiro
b) ('dia' = D) xou ('frio' <> 'clima')
c) L e (B / A >= C) ou não (A <= C)
d) L ou (C >= B / A) ou não A<= C
e) 2//(7**2)=14/A xou (B-3 <= C+0.5)
f) não L ou verdadeiro e sinal(C) >= A/A
g) ABS(B+(-2)) = ard(((2*C)**2)/10.0)

7) Para o enunciado a seguir foi elaborado um algoritmo em pseudocódigo que contém erros, identifique-os:

Tendo como dados de entrada o nome, a altura e o sexo (M ou F) de uma pessoa, calcule e mostre seu peso ideal, utilizando as seguintes fórmulas:
para sexo masculino: peso ideal = (72.7 * altura) - 58
para sexo feminino: peso ideal = (62.1 * altura) - 44.7

```
inicio
  caractere: nome, alt, sexo;
  {preparar tela}
  leia (nome);
  leia (sexo);
  se sexo = M
     então peso_ideal ← (72.7 * altura) - 58
     senão peso_ideal ←(62.1 * altura) - 44.7;
  escreva (peso_ideal);
fim.
```

8) Seja o seguinte algoritmo:

```
início
   inteiro: x,y,z;
   caractere: resposta;
   {preparar a tela}
    leia (x);
    leia (y);
    z ← (x*y) + 5;
    se z  <= 0
        então resposta ← 'A'
    senão se z <= 100
                então resposta ← 'B'
            senão resposta ← 'C';
                 escreva (z,resposta);
 fim
```

Faça um teste de mesa e complete o quadro a seguir para os seguintes valores:

x	y	z	resposta
3	2		
150	3		

7	-1
-2	5
50	3

9) Seja o seguinte algoritmo:

```
inicio
   inteiro: a, b, c;
   caractere: mens;
   {preparar tela}
   leia (a);
   leia (b);
   leia (c);
   se (a<b+c) e (b<a+c) e (c<a+b)
        então se (a=b) e (b=c)
              então mens ← 'Triângulo Equilátero'
              senão se (a=b) ou (b=c) ou (a=c)
                    então mens ← 'Triângulo Isósceles'
                    senão mens ← 'Triângulo Escaleno'
    senão mens ← 'Não e possível formar um triângulo';
    escreva (mens);
fim.
```

Faça um teste de mesa e complete o quadro a seguir para os seguintes valores:

a	b	c	mens
1	2	3	
3	4	5	
2	2	4	
4	4	4	
5	3	3	

Elabore um algoritmo em pseudocódigo para os enunciados a seguir:

10) Ler dois números positivos (>0) e executar as seguintes operações sobre eles: adição, subtração, multiplicação e divisão.

11) Dados dois pontos quaisquer do plano, de coordenadas (x1,y1) para o primeiro ponto e (x2,y2) para o segundo ponto, determine a distância entre eles. A fórmula que efetua tal cálculo é:

distância = raiz quadrada $((x2-x1)^2 + (y2-y1)^2)$

12) Dados: número da conta do cliente, saldo, débito e crédito, elabore um algoritmo que calcule e mostre saldo atual = saldo - débito + crédito. Se saldo atual for maior ou igual a zero mostre a mensagem 'Saldo Positivo', senão mostre a mensagem 'Saldo Negativo'.

13) Calcular e escrever o valor de y, sabendo-se que:

$y = ax^2 + bx + c;$ para x < a
$y = ax + b;$ para x = a
$y = ax^2 + bx - c;$ para x > a

14) Determinar e mostrar o valor de z, dados x e y, sabendo-se que:
se x>y e y>0 então z=x+y;

se x<y ou y<o então $z = \frac{2x-y}{2}$;
se x=y então z=3x

Se nenhuma das situações ocorrer então z=0.

15) Calcular e escrever o valor de z para:

$$z = (a + b)^2 + c \cdot x + y - \frac{a}{b^2}$$

Obs: Não é possível dividir por zero. Caso isto ocorra escrever uma mensagem.

16) Sejam três números inteiros diferentes, desenvolver um algoritmo que coloque-os em ordem crescente.

17) Dados a descrição do produto, a quantidade adquirida e o preço unitário, calcular e escrever o total (total = quantidade adquirida x preço unitário), o desconto e o total a pagar (total a pagar = total - desconto), sabendo-se que:
se quantidade <= 5 o desconto será de 2%
se quantidade > 5 e quantidade <=10 o desconto será de 3%
se quantidade > 10 o desconto será de 5%

18) Dados quantidade em estoque, quantidade máxima em estoque e quantidade mínima em estoque de um produto, calcular e exibir a quantidade média = (quantidade máxima + quantidade mínima)/2. Se a quantidade em estoque for maior ou igual a quantidade média exibir a mensagem 'Não efetuar compra', senão exibir a mensagem 'Efetuar compra'.

19) Dada um caracter qualquer verificar e escrever se ele é vogal (a, e, i, o, u).

20) Dado um número entre 1 e 12, escrever o número de dias do mês correspondente.
Obs: Se o número escolhido for 2, correspondente ao mês de fevereiro, considerar 28 dias.

21) Calcular e mostrar o perímetro de retângulos sabendo-se que perímetro = 2 * (Comprimento + Largura). Repetir o processo enquanto comprimento e largura forem positivos.

22) Uma empresa quer verificar se um empregado está qualificado para a aposentadoria. Para estar em condições, um dos seguintes requisitos deve ser satisfeito:

Ter no mínimo 65 anos de idade.
Ter trabalhado, no mínimo 30 anos.
Ter no mínimo 60 anos e ter trabalhado no mínimo 25 anos.

É dado o número do empregado, o ano de seu nascimento e o ano de seu ingresso na empresa. O programa deverá escrever a idade e o tempo de trabalho do empregado e a mensagem 'Requerer aposentadoria' ou 'Não requerer'. Repetir o processo enquanto número do empregado > 0. Obs: É possível resolver utilizando apenas uma decisão.

23) Calcule a área de círculos através da fórmula $A=PI*R^2$, onde R representa o raio e PI o número 3,14159. Repetir o processo enquanto R for positivo.

24) Os empregados de uma empresa recebem por hora trabalhada. Para as primeiras quarenta horas da semana, eles recebem o salário hora multiplicado pelo número de horas trabalhadas. Para as horas que excederem quarenta horas eles recebem o dobro por hora. É fornecido o número de cada empregado, o salário hora e o total de horas trabalhadas. Determinar o salário bruto. Parar o processo quando o número do empregado for zero.

25) Dado o nome e duas notas de provas de um aluno, calcular e escrever a média do aluno, que é ponderada, ou seja, a primeira prova tem peso 4 e a segunda prova tem peso 6. Calcular também a media da classe, que é a média aritmética das médias dos alunos. Repetir o processo enquanto nome do aluno <> '' (vazio).

26) Dado o nome e a idade de um nadador, classifique-o em uma das seguintes categorias:

Infantil A	---	de 5 a 7 anos
Infantil B	---	de 8 a 10 anos
Juvenil A	---	de 11 a 13 anos
Juvenil B	---	de 14 a 17 anos
Sênior	---	maiores de 17 anos

Repetir o processo até que nome = ‘ ’(vazio).

27) Dado a opção de pagamento, número da nota e valor da compra, determine o valor a pagar, considerando:

Opção	Situação para pagamento
1	descontar 2% do valor da compra
2	manter valor da compra e dividir em 2 parcelas
3	acrescentar 2% do valor da compra e dividir em 3 parcelas
4	acrescentar 5% do valor da compra e dividir em 5 parcelas
5	parar o processo.

Obs: Apresentar: ... parcela(s) de R$ ....
totalizando R$ ....

28) Dado o nome e o salário bruto de 70 funcionários de uma empresa, calcular e exibir o novo salário com reajuste de 15%.

29) Calcular e escrever a área de uma figura geométrica. O usuário poderá escolher a figura, sabendo-se que:

FIGURA	ÁREA
1-Retângulo	comprimento x largura
2-Círculo	PI x raio^2
3-Quadrado	lado ^{2}
4-Triângulo	(base x altura) / 2
5-Fim	

Para calcular a área da figura escolhida são dados os valores necessários (fazer crítica para aceitar apenas valores > 0). Fazer crítica também na entrada da figura que poderá ser 1, 2, 3, 4 ou 5.

30) Dado o número de inscrição do aluno, ano de ingresso e ano de conclusão de um curso universitário, elaborar um algoritmo para calcular:
- o tempo (em anos) que cada aluno levou para concluir o curso;
- a média aritmética do tempo (em anos) para conclusão do curso.

Fazer crítica dos dados:
ano de ingresso - permitir somente >=1993;
ano de conclusão - permitir somente >=1995.
Repetir o processo enquanto número de inscrição <> 0.

31) A conversão de graus Farenheit para Centígrados é obtida pela fórmula c=5/9(f-32). Calcular e escrever uma tabela de graus Centígrados em função de graus Farenheit, com graus Farenheit variando de 50 a 150 de um em um.

32) Uma empresa pretende enviar, para outra cidade, via aérea, 50 mercadorias. Calcular a tarifa de embarque cada mercadoria, sabendo-se que é cobrado 0,5% do valor da mercadoria. Calcular também a tarifa total para envio de todas as mercadorias.

33) O salário líquido de um empregado é obtido calculando o seu salário bruto, e então deduzindo os descontos. O salário bruto é obtido pela fórmula salário bruto = horas trabalhadas * salário por hora. O desconto é calculado pela fórmula desconto = salário bruto * percentual de desconto. Escreva um algoritmo para calcular o salário líquido de 30 funcionários. O algoritmo deverá ler para cada funcionário o seu número, o salário por hora, o percentual de descontos e as horas trabalhadas. A seguir, calcular e exibir o salário bruto, os descontos e o salário líquido. Fazer crítica onde necessário.

34) Considere dados referente a altura e o sexo de 40 pessoas. Fazer um algoritmo que calcule e escreva:
- a maior e a menor altura do grupo
- a média de altura das mulheres
- o número de homens.

35) Dado o código do produto e preço de 15 produtos, elaborar um algoritmo para calcular e exibir:
- o maior preço;
- a média aritmética dos preços dos produtos.

36) Escreva um algoritmo para calcular w = 3a + 2b + 5, para todas as possibilidades a seguir:

a) variando a 1 a 4 com incrementos de 1;
b) variando b de 0.5 a 2.5 com incrementos de 0.5.
Mostrar na tela a, b e w.

37) Uma loja de discos está fornecendo descontos na compra de discos e fitas. Elabore um algoritmo que mostre uma tabela contendo: quantidade de discos, quantidade de fitas e porcentagem de desconto, sabendo-se que:

quantidade de discos varia de 1 a 5;
quantidade de fitas varia de 1 a 5;
porcentagem de desconto = 5 + quantidade.

8. Crítica ou Consistência de Dados

Fazer crítica ou consistência de dados em um algoritmo significa verificar se a informação digitada no momento da entrada de dados é válida ou não, isto é, no caso da crítica para entrada de um valor verificar se o mesmo se encontra dentro dos limites estabelecidos.

Exemplos: a nota de uma prova que deve ser entre 0 e 10.00 não devendo ser aceito valores fora desta faixa como –1 ou 11; o sexo de uma pessoa que deverá ser 'F' ou 'M' não devendo ser aceitas outras letras; a idade de uma pessoa que só pode ser um número inteiro positivo, etc.

A crítica é usada principalmente para checar a validade dos dados de forma a garantir a execução correta do algoritmo. Quando ocorre uma entrada de dados errada o algoritmo deverá permitir uma nova entrada de dados até que os mesmos estejam dentro das especificações. Para isso podemos utilizar uma estrutura de repetição com teste no final.

Exemplo: dados RA e as notas de duas provas de um aluno calcular a média final, sabendo-se que média = (1ª nota * 4 + 2ª nota * 6) /10. Repetir o processo até que RA = '' (vazio).

```
Início
  Caractere: ra;
  Real      :n1, n2, med;
  Leia(ra);
  Enquanto ra <> '' Faça
  Início
   Repita
      Leia(n1);
   Até (n1>=0)  e (n1<=10);
   Repita
      Leia(n2);
   Até (n2>=0)  e (n2<=10);
   med ← (n1 * 4 + n2 * 6) / 10;
   Escreva ('A média = ',med);
   Leia (ra);
    Fim
Fim.
```

9 - Modularização

A complexidade de um algoritmo está diretamente ligada à aplicação a que ele se destina. Em geral, problemas mais complexos exigem algoritmos extensos. Quando nos deparamos com tal situação muitas vezes o resultado é um amontoado de ações que não ficam muito claras, além de que muitas vezes pode apresentar trechos com instruções repetidas em vários pontos do algoritmo.

Uma solução para este problema é dividir este algoritmo mais complexo em pequenos algoritmos conhecidos como: **módulos ou subalgoritmos**.

Ao modularizar um algoritmo, buscamos aumentar a funcionalidade das partes do conjunto, facilitando o seu entendimento e possibilitando a reutilização destas partes.

Um algoritmo completo é dividido em módulo principal e diversos módulos ou subalgoritmos tantos quantos forem necessários. O módulo principal é aquele por onde a execução sempre se inicia, podendo invocar (ativar ou chamar) outros módulos que por sua vez poderão invocar outros ou até a si mesmos.

A chamada ao módulo, representa a execução das ações contidas nele, em seguida a execução retorna ao ponto da sua chamada (que poderá ser o módulo principal ou outros módulos). Não existe ordem para definição dos módulos.

Declaração Geral:-

```
início
    {declarações  globais}
    módulo principal;
    {declarações  locais}
    início
       {corpo do algoritmo principal}
    fim;

    {definições de outros  módulos ou subalgoritmos}
fim.
```

onde:-

declarações globais:- são variáveis, tipos e/ou constantes declaradas no início do algoritmo que podem ser utilizadas por qualquer módulo inclusive pelo módulo principal;

módulo principal:- é por onde inicia a execução do algoritmo;

declarações locais:- são variáveis, tipos e ou constantes que só podem ser utilizadas dentro do módulo no qual foram declaradas;

corpo do algoritimo:- conjunto de ações ou comandos;

definição dos módulos ou subalgoritmos :- compreende um cabeçalho chamado módulo seguido de um nome, declarações locais e o corpo do subalgoritmo.

Declaração de um módulo**:-**

```
módulo identificador;
{declarações locais}
   início
      {corpo do subalgoritmo}
   fim;
```

EXEMPLO:- Dadas as informações de um aluno, calcular a média e a situação desse aluno.

```
Início
      Caractere: nome;
      Real      : p1,p2,media;
      Caracter: sit;
      Inteiro   : falta;
      Módulo Principal;
 Início
   Leia(nome);
   Leia(p1);
   Leia(p2);
   Leia(falta);
```

```
    calculo;
Fim;

Módulo calculo;
   Início
    media ← (p1 *4 + p2 * 6) / 10;
    Se (media >=5) e (falta <=20) então sit ← ‘Apr’
                                senão sit ← ‘Rep’
    escreva(media,sit);
  Fim;
Fim.
```

PASSAGEM DE PARÂMETROS

São canais pelos quais se estabelece uma comunicação bidirecional entre um módulo e outro (módulo principal ou outros módulos). Nesse caso os dados locais a um módulo podem ser enviados para um outro módulo, que por sua vez poderá utilizar estes dados e alterá-los ou não. Este mecanismo é chamado de passagem de parâmetros ou argumentos e pode ser de dois tipos: passagem de parâmetros por valor (ou cópia) e passagem de parâmetros por referência.

- passagem de parâmetros por valor:- Na passagem de parâmetros por valor é feita uma cópia do conteúdo das variáveis locais para um outro módulo. As modificações efetuadas nos dados do parâmetro não retornam ao módulo chamador. Esse tipo de mecanismo reserva um espaço diferente em memória para os parâmetros serem copiados e assim não causarem modificações nos dados originais.

- passagem de parâmetros por referência:- Neste caso é feita a cópia do endereço da memória onde a variável esta armazenada. Nesse mecanismo uma outra variável ocupando um outro espaço diferente na memória não armazena o dado em si, e sim o endereço onde ele se localiza na memória. Sendo assim todas as modificações efetuadas nos dados do parâmetro estarão sendo feitas no conteúdo original da variável.

Um mesmo algoritmo pode utilizar os dois mecanismos de passagem de parâmetros, para diferenciar um do outro convencionamos utilizar um prefixo **ref** antes da definição dos parâmetros no módulo receptor.

EXEMPLO1:- No exemplo abaixo é fornecido um valor em real e convertido para o corresponde em dólar.

```
início
   módulo principal;
   início
     entrada;
   fim;

   módulo entrada;
   real: re,dl;
   caractere: op;
   início
      repita
         leia(re);
         conv(re, dl);
         escreva(dl);
         leia(op);
      até op='N';
   fim;

   módulo conv(real:r; ref real:d);
   início
      d ← r/1.20;
   fim;
fim.
```

EXEMPLO2:- Dados dois valores positivos calcular e exibir o resultado das operações aritméticas efetuadas utilizando passagem por parâmetros.

```
início
   módulo principal;
         real: x, y, res;    {variáveis locais}
          início
     leia(x);
     leia(y);
```

```
      se (x>0) e (y>0) então
       início
          calc(x,y,'+', res);
           escreva(res);
          calc(x,y,'-', res);
          escreva(res);
          calc(x,y,'*', res);
          escreva(res);
          calc(x,y,'/', res);
          escreva(res);
       fim;
        fim;
   módulo calc(real:a,b;caracter:oper; ref real:re);
   início
     escolha oper
      caso '+' : re ← a + b;
      caso '-' : re ← a - b;
      caso '*' : re ← a * b;
      caso '/' : re ← a / b;
     fim;
   fim;
fim.
```

Ainda podemos ter um módulo que tenha a função de calcular um resultado, como exemplo as funções matemáticas. Este tipo de módulo tem o objetivo de retornar uma única informação ao módulo chamador, como se fosse uma resposta. Esta resposta será feita através do comando **retorne():**

Declaração:- retorne(expressão);

onde:

expressão: pode ser uma informação numérica, alfanumérica ou uma expressão aritmética.

EXEMPLO3 :- O mesmo do exemplo 1 modificado.

```
início
   módulo principal;
   início
     entrada;
   fim;

   módulo entrada;
   real:re;
   caracter:op;
   início
      repita
         leia(re);
         escreva(conv(re));
         leia(op);
      até op='N';
   fim;

   módulo conv(real:r);
   início
      retorne(r/1.20);
   fim;
fim.
```

EXEMPLO4 :- O mesmo do exemplo 2 modificado usando retorne. Observe que sempre irá retornar um único resultado de cada vez.

```
início
   módulo principal;
        real: x, y;    {variáveis locais}
```

```
        início
      leia(x);
      leia(y);
      se (x>0) e (y>0) então
       início
         escreva(calc(x,y,'+'));
         escreva(calc(x,y,'-'));
         escreva(calc(x,y,'*'));
         escreva(calc(x,y,'/'));
          fim;
        fim;
   módulo calc(real:a,b;caracter:oper);
   início
     escolha oper
      caso '+' : retorne(a + b);
      caso '-' : retorne(a – b);
      caso '*' : retorne(a * b);
      caso '/' : retorne (a / b);
     fim;
   fim;
fim.
```

EXEMPLO5: Dados três números quaisquer desenvolver um algoritmo que receba via parâmetro estes números, calcule o quadrado de cada um e retorne o resultado para o módulo chamador(passagem de parâmetro por referência).

```
Início
  Módulo principal;
    Real: n1,n2,n3,q1,q2,q3;
    Início
       Leia(n1,n2,n3);
       Quadrado(n1,n2,n3,q1,q2,q3);
```

```
        Escreva( 'O quadrado de n1 =',q1);
        Escreva( 'O quadrado de n2 =',q2);
        Escreva( 'O quadrado de n3 =',q3);
    Fim;

Módulo quadrado (real:n1,n2,n3; ref real:q1,q2,q3);
    início
        q1← n1 **2;
        q2← n2 **2;
        q3← n3 **2;
    Fim;
Fim.
```

EXEMPLO6: O mesmo exercício usando retorne.

```
Início
  Módulo principal;
    Real: n1,n2,n3;
    Início
        Leia(n1,n2,n3);
        Escreva( 'O quadrado de n1 =',quadrado(n1));
        Escreva( 'O quadrado de n2 =',quadrado(n2));
        Escreva( 'O quadrado de n3 =',quadrado(n3));
    Fim;

Módulo quadrado (real:n);
    início
        retorne(n **2);
    fim;
Fim.
```

LISTA DE EXERCÍCIOS - Modularização

1) Dado um número qualquer entre 1 e 10 (fazer crítica), calcular e exibir a tabuada do número dado, se ele é par ou ímpar e se é múltiplo de 3. O algoritmo deverá ser repetido até que a resposta à pergunta "Deseja Continuar <S/N>" seja igual a "N".

2) Desenvolver um algoritmo para calcular a equação do 2º grau para delta maior que zero, e exibir as raízes ou a mensagem de que não existem raízes reais. O algoritmo deverá conter:

a) Um módulo para entrada de dados, os valores de a,b e c;
b) Um módulo para cálculo do delta (delta = quadrado de (b)-4*a*c e um para exibir as raízes reais (x1=(-b - raíz quadrada(delta))/(2*a) e x2=(-b+raíz quadrada(delta))/(2*a) ou a mensagem "Não existem raízes reais)".

Repetir o processo enquanto o usuário desejar.

3) Faça o mesmo algoritmo do exercício anterior retornando o resultado (comando retorne( )) para calcular o delta.

4) Dadas as seguintes informações: R.A., nota da 1ª prova, nota da 2ª prova e número de faltas. Elabore um algoritmo que contenha:

a) Um módulo para entrada de dados;
b) Um módulo para cálculo da média = ((p1*4) + (p2*6))/10. (Utilizar passagem de parâmetros).
c) Um módulo para cálculo da situação do aluno que estará “Aprovado” se média >=5 e número de faltas<=16, caso contrário aluno “Reprovado”. Utilizar passagem de parâmetros.

O algoritmo deverá ser repetido para vários alunos até que RA=‘’. Utilizar variáveis locais e passá-las por parâmetro onde necessário.

5) Dado o modelo do veículo, a marca e o custo de fabricação, desenvolver um algoritmo que calcule e exiba o preço de venda do veículo, sabendo-se que terá um acréscimo de 30 %. O programa deverá conter os seguintes módulos:

a) Para cálculo do preço de venda.
b) Para entrada de dados. As variáveis deverão ser locais e passadas por parâmetro se necessário. Repetir o processo até que modelo do veículo=‘’.

6) Dado o número do telefone de uma residência, o número de pulsos registrados para chamadas locais e o valor total de todas as chamadas interurbanas, fazer um algoritmo que controle a conta telefônica de 50 residências contendo os seguintes módulos:

a) Para entrada dos dados (variáveis locais);
b) Para cálculo do valor total da conta telefônica. Sabendo-se que o valor total da conta telefônica = número de pulsos locais x 0.127 + valor total de interurbanos;

7) Tendo o nome e o rendimento de um funcionário de uma empresa, calcular o imposto de renda devido, sabendo-se que o imposto de renda = rendimento * alíquota - valor a deduzir e considerando a tabela abaixo:

Rendimento	Alíquota	Valor a Deduzir
até 900.00	0	0
acima de 900.00 até 1800.00	0.15	135.00
acima de 1800.00	0.25	315.00

Obs: Utilizar módulos e variáveis locais.

8) Desenvolver um algoritmo para armazenar o número da nota, o número do vendedor, o valor da venda e o tipo de pagamento (V- à vista; P – à prazo). O algoritmo deverá calcular e exibir o valor final a pagar sabendo-se que à vista o valor da compra terá um desconto de 5%. O algoritmo deverá armazenar também o total vendido por cada vendedor sabendo-se que a loja possui 3 vendedores. Repetir o processo enquanto número da nota > 0.

9) Desenvolver um algoritmo que receba 3 números inteiros diferentes via parâmetro. Ordene esses números em ordem crescente e retorne os números ordenados.

10 - Tipos Estruturados

São organizações de dados construídas a partir da composição dos tipos primitivos já existentes (caracter, inteiro, real, lógico). Esses novos tipos podem armazenar um conjunto de dados conhecidos como variáveis compostas. As variáveis que armazenam esses tipos de dados podem ser classificadas em: variáveis compostas homogêneas e variáveis compostas heterogêneas.

10.1 Variáveis Compostas Homogêneas

São variáveis que armazenam vários elementos do mesmo tipo primitivo.
Exemplo: alcatéia (vários lobos), conjunto de números inteiros, e outros.

As variáveis compostas são estruturas de dados (armazenam um conjunto de dados). Esses dados podem ser armazenados em dois tipos de variáveis: as variáveis unidimensionais (conhecidas como vetores) e as variáveis multidimensionais (conhecidas como matriz).

variáveis compostas unidimensionais - VETORES

São variáveis que necessitam de apenas um índice para individualizar um elemento do conjunto.

Declaração: tipo: identificador[n];

onde: identificador – é o nome de uma variável;
n – é o tamanho do vetor

Exemplo:
real: media[40];

Estrutura que se forma :

media

8.0	7.0	5.5	9.5	6.4	9.9	1.0	.....	4.8
0	1	2	3	4	5	6	.....	39

Para acessar cada elemento da variável composta unidimensional (vetor) é necessário especificar o nome da variável seguido do número do elemento que se deseja acessar (deverá estar entre colchetes). Este número tem o nome de índice. O índice pode varia de 0 até n -1

```
    Leia (media[0]);
    media [2] ← 8.5;
    escreva(media[3]);
    .....
```

Exemplo 1: Dados 5 valores numéricos elaborar um algoritmo que calcule e exiba a média dos valores dados e quantos desses valores estão acima da média.

Resolução sem utilizar vetor:

```
início
    inteiro:n1,n2,n3,n4,n5,cont;
    real:media;
     cont← 0;
     leia (n1);
     leia (n2);
     leia (n3);
     leia (n4);
     leia (n5);
     media ← (n1+n2+n3+n4+n5)/5;
     escreva(media);
     se (n1>media) então cont ← cont + 1;
     se (n2>media) então cont ← cont + 1;
     se (n3>media) então cont ← cont + 1;
     se (n4>media) então cont ← cont + 1;
     se (n5>media) então cont ← cont + 1;
     escreva(cont);
fim;
```

Resolução utilizando variável composta:

```
Início
Inteiro:n[5];
    inteiro:i,cont;
    real:media;
     cont ← 0; media ←0;
     Para i de 0 até 4 passo1 faça
     início
        leia(n[i]);
        media← media + n[i];
     fim;
     media ← media/5;
     escreva(media);
     Para i de 0 até 4 passo1 faça
           se (n[i]>media) então cont ← cont +1;
      escreva(cont);
fim.
```

<u>variáveis compostas multidimensionais - MATRIZ</u>

São variáveis que utilizam mais de um índice para acessar cada um de seus elementos. Podemos ter duas ou mais dimensões.

Declaração: tipo: identificador[n][m];

onde: identificador – é o nome de uma variável;
n - é o número de linhas da matriz;
m – é o número de colunas da matriz;

Exemplo:
Caractere: m[3][5];

Estrutura que se forma :

	0	1	2	3	4
0					
1					
2					

Para acessar cada elemento da matriz:

```
Leia (m[1][1]);
m[1][2] ← 'X';
escreva(m[1][3]);
.....
```

Exemplo : Dada uma matriz de ordem 5x5, desenvolver um algoritmo que multiplique sua diagonal principal por um valor dado.

```
Início
   Inteiro: mt[5][5];
   inteiro:i,j,num;
   leia (num);
   Para i de 0 até 4 passo1 faça
    Início
        Para j de 0 até 4 passo1 faça
            Início
              leia(mt[i][j]);
              se i=j então mt[i][j] * num;
              escreva(mt[i][j]);
            Fim;
```

```
    Fim;
Fim.
```

LISTA DE EXERCÍCIO - VETORES E MATRIZES

1) Dados dois vetores, um contendo a quantidade e o outro o preço de 20 produtos, elabore um algoritmo que calcule e exiba o faturamento que é igual a quantidade x preço. Calcule e exiba também o faturamento total que é o somatório de todos os faturamentos, a média dos faturamentos e quantos faturamentos estão abaixo da média.

2) Dado um vetor contendo 10 elementos numéricos, elabore um algoritmo que verifique se um outro valor dado pertence ou não ao vetor.

3) Dado um vetor contendo letras do alfabeto, elabore um algoritmo para verificar quantas vezes ocorreu a letra 'A'.
OBS: Fazer crítica na entrada do caractere para aceitar somente letras.

4) Dado um vetor X numérico contendo 5 elementos, fazer um algoritmo que crie e exiba na tela um vetor Y. O vetor Y deverá conter o mesmo conteúdo do vetor X na ordem inversa, de acordo com o exemplo abaixo:

VETOR NORMAL	VETOR INVERTIDO
9	4
3	1
5	5
1	3
4	9

5) Dada uma matriz de ordem 3x5 contendo valores numéricos reais, fazer um algoritmo que calcule e exiba a soma dos números positivos e a soma dos números negativos.

6) Dado um valor numérico VAL e uma matriz A 3x4 elabore um algoritmo que calcule e exiba uma outra matriz B que deverá conter cada elemento da matriz A dividido pelo valor numérico VAL.

7) Dada uma matriz A de dimensão 2x3 contendo números inteiros positivos fazer um algoritmo que gere uma matriz b de caracteres, tal que se o número da posição de A[i][j] for par o elemento correspondente na matriz B[i][j] deverá conter a informação 'P', caso contrário deverá conter a informação 'I'. Exibir as duas matrizes.

8) Escreva um algoritmo que leia duas matrizes reais de dimensão 3X5, calcule e exiba a soma das matrizes.
C[i][j] = A[i][j] + B[i][j]

9) Elabore um algoritmo que a partir de uma matriz quadrada de ordem 4X4 contendo elementos reais determine uma outra matriz que é obtida através da

divisão dos elementos de cada linha pelo elemento da diagonal principal. Exibir as duas matrizes.

10) Dadas duas matrizes numéricas A e B de dimensão 4x3, fazer um algoritmo que gere uma matriz lógica C, tal que o elemento C[i][j] seja verdadeiro se os elementos nas posições respectivas das matrizes A e B forem iguais, e falso caso contrário. Exibir as matrizes A, B e C.

EXEMPLO:

	2 4 6		2 5 8		verdadeiro	falso	falso
A =	1 5 9	B =	1 9 7	C =	verdadeiro	falso	falso
	3 7 2		3 7 1		verdadeiro	verdadeiro	falso
	4 6 8		4 5 8		verdadeiro	falso	verdadeiro

11) Dada a matriz MAT abaixo:

	0	1	2	3
0	O	Q	*	I
1	E	*	E	S
2	R	E	U	T
3	A	*	*	S

Conteúdo de MAT

	0	1	2	3
0				
1				
2				
3				

- Descreva qual será o conteúdo de MAT (quadro acima) depois de executado o seguinte trecho do algoritmo:

```
início
caractere: mat[4][4];
inteiro: i,j;
caractere:aux;
    {prepar a tela}
   para i de 0 até 2 passo1 faça
     início
       para j de i+1 até 3 passo1 faça
         início
           aux← mat [i][ j];
           mat [ i][ j] ← mat [ j][ i];
           mat [ j][ i] ← aux;
         fim;
     fim;
   aux← mat [0][0];
   mat [0][0] ← mat [3][3];
   mat [3][3] ← aux;
   aux← mat [1][1];
   mat [1][1] ← mat [2][2];

   mat [2][2] ← aux;
fim
```

12) Dada uma matriz quadrada de ordem m x m elaborar um algoritmo que determine o menor valor da matriz. Exibir a matriz, o menor valor, e a linha e a coluna onde o valor se encontra.

13) Dada uma matriz bidimensional contendo 4 notas de 10 alunos, elaborar um algoritmo que calcule e exiba um outra matriz unidimensional que deverá conter a média aritmética das 4 notas de cada aluno.

14) Faça um algoritmo que leia dois vetores de 10 elementos numéricos cada um e intercale os elementos deste em um outro vetor de 20 elementos.

15) Uma pessoa deseja comprar um eletrodoméstico, para isto percorreu 5 lojas. Dados dois vetores, um contendo o nome das lojas e o outro o preço, desenvolver um algoritmo que calcule e mostre qual foi o menor preço encontrado e o nome da loja que tem o menor preço.

16) Desenvolver um algoritmo que leia os elementos da 1ª linha de uma matriz 4x4 numérica e a partir desses elementos calcule e mostre os outros elementos da matriz. Sabe-se que os elementos da 2ª linha são os elementos da 1ª linha x 2, os elementos da 3ª linha são os elementos da 1ª linha x 3 e assim por diante.

17) Dada uma matriz A numérica 3x3 desenvolver um algoritmo que gere uma outra matriz B 3x4, que contém os mesmo elementos da matriz A menos a 4ª coluna que deverá conter a média aritmética dos elementos das três colunas.

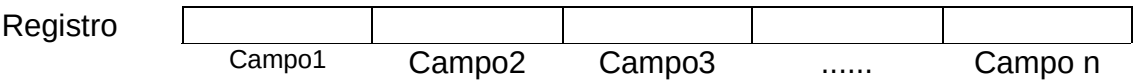
10.2 Variáveis Compostas Heterogêneas

São variáveis que armazenam vários dados de tipos primitivos diferentes.
Exemplo: mamíferos (cachorro, baleia, gato,...), conjunto numérico (inteiros e reais).

Uma das estruturas de dados utilizadas para implementar este tipo de variável é o registro.

Registro

É uma forma de armazenamento de dados que permite o agrupamento de vários tipos primitivos (real, caracter, inteiro ou lógico) em uma mesma variável. Cada um dos dados armazenados no registro são conhecidos como campos do registro.



Imagine por exemplo as informações de um aluno, tais como: registro acadêmico, nome, curso, valor da mensalidade e ano de ingresso. Todas essas informações agrupadas formam um registro onde cada uma delas representa um campo do registro.

Declaração: Tipo identificador = **registro**
Tipo1:campo1;
Tipo2:campo2;
....
Tipon:campon;
Fim;
Identificador : lista de variáveis;

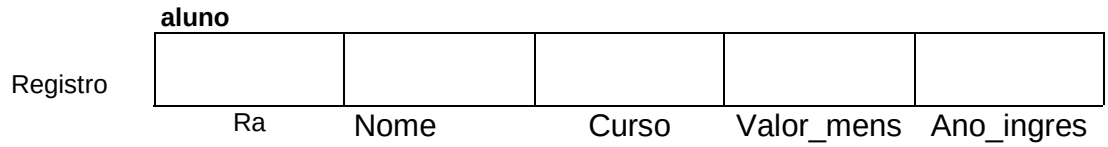
Onde: identificador: representa o nome associado ao tipo registro;
tipo1,tipo2..tipon: representam os tipos primitivos de cada campo;
campo1,campo2..campon: representam os nomes associados a cada um dos campos do registro;
lista de variáveis: são os nomes dados as variáveis que irão armazenar o tipo registro;

Exemplo1: .....
Tipo reg_aluno = registro
Caracter : ra;
Caracter : nome;

```
            Caracter : curso;
            Real     : valor_mens;
            Inteiro  : ano_ingres;
      Fim
   reg_aluno : aluno;
```

O exemplo corresponde à definição de um tipo de dado chamado reg_aluno e a variável aluno que armazena 5 informações dos tipos definidos no registro.

Estrutura que se forma:



Para acessar cada uma das informações contidas no registro, isto é, para acessar cada campo do registro, é necessário especificar o nome da variável seguido do caracter ponto (**.**) seguido do nome do campo, como mostra os exemplos abaixo.

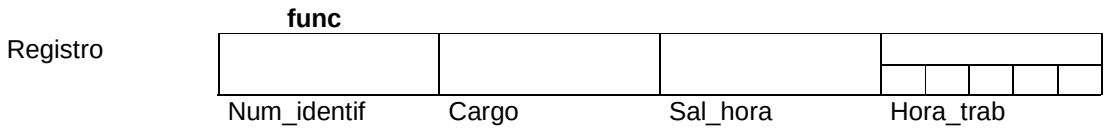
```
Leia (aluno.ra);
Leia ( aluno.nome);
Escreva(aluno.curso);
Aluno.valor_mens ← aluno.valor_mens – 0.10 * aluno.valor_mens;
....
```

Exemplo2:-

```
      Tipo reg_func = registro
                                 Caracter : num_identif;
                                 Caracter : cargo;
                                 Real     : sal_hora;
                                 Real     : hora_trab[5];
            Fim

      reg_func : func;
      Inteiro  : x;
```

Estrutura que se forma:



Para acessar os campos do registro usamos:

```
Leia(func.num_identif);
Leia(func.cargo);
Leia (func.sal_hora);
Para x de 0 até 4 faça
    Leia (func.hora_trab[x]);
```

Podemos verificar com o exemplo acima que os campos do registro podem armazenar qualquer tipo de dado inclusive um tipo vetor ou matriz.

Conjunto de Registros

Na estrutura vista anteriormente as informações armazenadas são referentes a um único registro. Supondo que quiséssemos armazenar informações em vários registros utilizaremos então um conjunto de registro, isto é, podemos armazenar um vetor composto por registros ou mesmo uma matriz de registros, onde os registros seriam referenciados por índices.

Exemplo1: .....

```
Tipo   reg_aluno  =  registro
            Caracter : ra;
            Caracter : nome;
            Caracter : curso;
            Real       : valor_mens;
            Inteiro    : ano_ingres;
      Fim
       reg_aluno: aluno[40];
       inteiro       : i;
```

Estrutura que se forma:

aluno

Registro 0					
Registro 1					
....					
Registro 39					
	Ra	Nome	Curso	Valor_mens	Ano_ingres

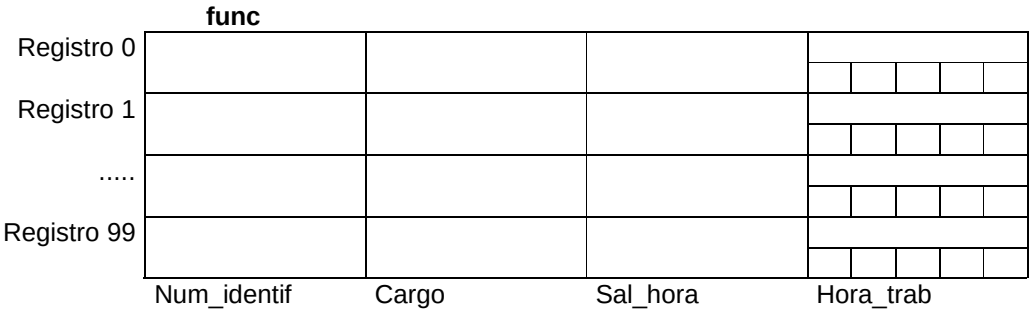
Para acessar as informações contidas em um conjunto de registros usamos:

```
Para i de 0 até 39 faça
    Início
        Leia (aluno[ i ].ra);
                Leia (aluno[ i ]. nome);
                Leia (aluno[ i ]. Curso);
                Leia (aluno[ i ]. Valor_mens);
                Leia (aluno[ i ]. Ano_ingres);
            Fim;
```

Exemplo2:-

```
        Tipo reg_func = registro
                                    Caracter : num_identif;
                                    Caracter : cargo;
                                    Real     : sal_hora;
                                    Real     : hora_trab[5];
              Fim
        reg_func = func[100];
        Inteiro      : i,x;
```

Estrutura que se forma:



Para acessar os campos do registro usamos:

```
Para i de 0 até 99 faça
   início
      Leia(func[ i ].num_identif);
      Leia(func[ i ].cargo);
      Leia (func[ i ].sal_hora);
      Para x de 0 até 4 faça
          Leia (func[ i ].hora_trab[x]);
   Fim;
```

Observamos que quando possuímos um vetor composto de registros primeiro é necessário declarar o tipo do registro para depois declararmos o vetor que tem aquele tipo.

EXEMPLO1:- Dado um conjunto de 40 registros contendo informações de alunos calcular a média e a situação desses alunos. O algoritmo deverá exibir uma lista ordenada dos alunos contendo a média e a situação.

```
Início
  Tipo R_aluno = registro
       Caractere: nome;
       Real      : p1,p2,media;
       Caracter: sit;
       Inteiro   : falta;
       Fim;
  R_aluno: aluno[40];
  Inteiro: x;
```

```
Módulo Principal;
 Início
   Para x de 0 até 39 faça
     Início
    Leia(aluno[x].nome);
    Leia(aluno[x].p1);
    Leia(aluno[x].p2);
    Leia(aluno[x].falta);
    calculo;
   Fim;
   Relatório;
Fim;

Módulo calculo;
   Início
    Aluno[x].media ← (aluno[x].p1 *4 + aluno[x].p2 * 6) / 10;
    Se (aluno[x].media >=5) e (aluno[x].falta <=20) então aluno[x].sit ← 'Apr'
                              senão aluno[x].sit ← 'Rep'
  Fim;

Módulo ordena;
  R_aluno : aux;
  Inteiro       : y;
  Início
     Para x de 0 até 38 faça
        Início
          Para y de x+1 até 39 faça
        Início
        Se aluno[x].nome > aluno[y].nome então
            Início
              Aux ← aluno[x];
              Aluno[x] ← aluno[y];
```

```
            Aluno[y] ← aux;
          Fim;
      Fim;
      Fim;
Fim;


Módulo Relatório;
Início
   Ordena;
    Para x de 0 até 39 faça
        Escreva (aluno[x].nome, aluno[x].média, aluno[x].sit);
Fim;
Fim.
```

EXEMPLO2: Tendo o registro contendo: número do prontuário, nome, categoria (A- auxiliar, M - mestre e D - doutor) e departamento de um professor, desenvolver um algoritmo para calcular qual é a categoria que possui mais professores. Mostrar também os nomes e os departamentos dos professores que pertencem a categoria doutor. O algoritmo deverá permitir o cadastramento de vários professores (máximo de 2000) até que usuário digite número do prontuário = -1 para finalizar.

```
Início
    Tipo professor = registro
            Caracter:pront,nome,cat,dep;
            Fim;
    professor: p[2000];
    inteiro    : i,x,contA,contM,contD,maior;

     contA ← 0; contM ← 0; contD ← 0;
     i ← 1;
     maior ← 0;
     repita
    leia( p[ i ].pront);
    se p[ i ].pront <> -1 então
       início
        leia( p[ i ].nome);
        repita
            leia( p[ i ].cat);
        até (p[ i ].cat in ['A','M','D']);
        leia(p[ i ].dep);
        escolha p[ i ].cat
          caso 'A': contA ← contA + 1;
          caso 'M': contM ← contM + 1;
                       caso 'D': contD ← contD + 1;
                    fim;
                    i ← i + 1;
        fim;
```

```
    até (p[ i ].pront = -1) ou (i > 1999);
    se contA > contM então maior  ← contA
                  senão maior ← contM;
    se contD > maior então maior ← contD;
    escreva(maior);
    Para x de 0 até i-1 faça
            Início
               Se p[ x ].cat = 'D' então
                        Escreva(p[ x ].nome,p[ x ].dep);
      Fim;
Fim.
```

LISTA DE EXERCÍCIOS – CONJUNTO DE REGISTROS

1) Dados os seguintes campos de um registro: nome, dia de aniversário e mês de aniversário, desenvolver um algoritmo que mostre em cada um dos meses do ano quem são as pessoas que fazem aniversário, exibir também o dia. Considere um conjunto de 40 pessoas.

2) Uma pessoa cadastrou um conjunto de 15 registros contendo o nome da loja, telefone e preço de um eletrodoméstico. Desenvolver um algoritmo que permita exibir qual foi a média dos preços cadastrados e uma relação contendo o nome e o telefone das lojas cujo preço estava abaixo da média.

3) Tendo um registro contendo RA do aluno, tipo de participação (A, B, C ou D) e sócio da SBC (S-sim ou N-não), desenvolver um algoritmo para calcular o valor que cada aluno vai pagar para participar da semana de informática, sabendo-se que:

Tipo de Participação	Valor a Pagar
A - 1 curso	R$ 30,00
B - 2 cursos	R$ 60,00
C - 3 cursos	R$ 90,00
D - outros	R$100,00

Para os sócios da SBC o valor a pagar terá um desconto de 50%. O algoritmo deverá permitir a entrada de vários registros (no máximo 1000) até que uma condição de finalização seja satisfeita. Calcular e exibir também o total geral arrecadado com o evento e quantos alunos se matricularam em cada um dos tipos de participação.

4) Um provedor de acesso à *Internet* mantém o seguinte cadastro de clientes: código do cliente, *e-mail*, número de horas de acesso, página (S-sim ou N-não). Elaborar um algoritmo que calcule e mostre um relatório contendo o valor a pagar por cada cliente, sabendo-se que as primeiras 20 horas de acesso é R$35,00 e as horas que excederam tem o custo de R$2,50 por hora. Para os clientes que têm página adicionar R$40,00. Inserir um conjunto de registros (máximo 500).

5) Uma determinada biblioteca possui obras de ciências exatas, humanas e biológicas , totalizando 1500 volumes, distribuídos em cada uma das áreas. O proprietário resolveu agrupar as informações de cada livro no seguinte registro:

Código de catalogação
Doação (S/N)
Nome da obra
Nome do autor
Editora
Área

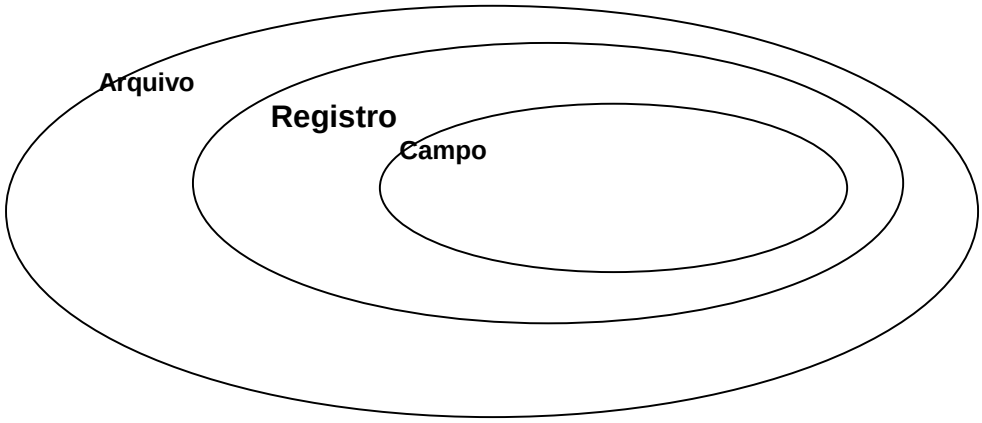
Construir um algoritmo que:
a) cadastre todos os volumes de cada uma das áreas em três vetores distintos;
b) permita ao usuário fazer consulta às informações cadastradas fornecendo o código de catalogação e a área. Existindo tal livro as informação são exibidas, caso contrário enviar mensagem de aviso. A consulta se repete até que o usuário digite código finalizador = -1.

11 - ARQUIVOS

Quando desenvolvemos programas de computadores é muito comum precisarmos "guardar" dados para posterior recuperação. A recuperação dos dados pode ser alguns minutos depois, dias depois, ou até mesmo anos depois do armazenamento inicial. Sendo assim dados que precisam ser guardados por um tempo indeterminado ficam armazenados em arquivos.

Até o momento somente trabalhamos com o conceito de dados armazenados de forma temporária (em memória principal). Quando desejamos armazenar dados por um período de tempo indeterminado, podendo reutilizá-los, modificá-los e ou continuar a armazenar mais dados utilizamos arquivos.

ARQUIVO :- É um conjunto de registros logicamente organizados armazenados em um dispositivo de memória secundária (disco rígido, disquetes, fitas magnéticas, cd, etc), onde cada registro compreende um conjunto de informações denominadas campos. Em um arquivo é possível armazenar um volume grande de dados.



Um arquivo é formado por um ou mais registros, que por sua vez, como já sabemos, é formado por um conjunto de campos.

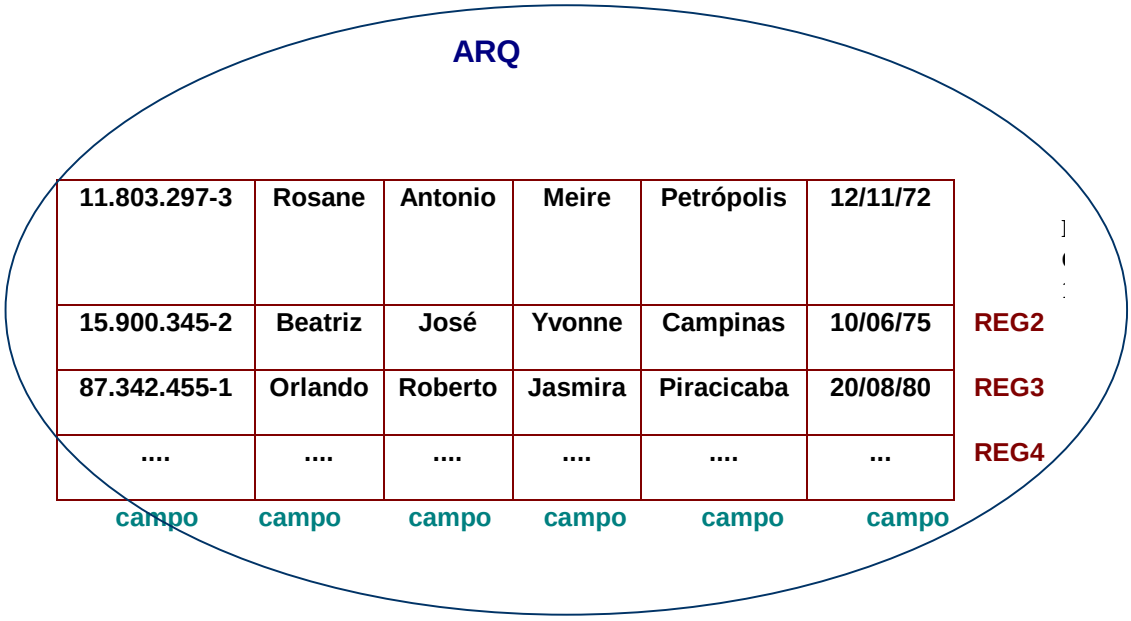
Para que o algoritmo possa fazer acesso aos dados armazenados no arquivo é necessária a declaração do registro de dados que o arquivo irá armazenar e a declaração do arquivo propriamente dito.

Forma Geral:- Tipo identificador = **arquivo composto de** tipo_registro;
identificador: lista de variáveis;
onde: identificador - representa o nome do tipo do arquivo;
tipo_registro - é o nome do registro previamente definido;

Exemplo:-

```
Tipo pessoa = REGISTRO
        caracter:rg;
        caracter:nome,pai,mãe;
        caracter:naturalidade,nascimento;
```

```
        fim;
Tipo ar_pessoas = ARQUIVO COMPOSTO  de pessoa;
Pessoa       :reg;   //variáveis que irão armazenar os tipos definidos
ar_pessoas:arq;
```



Um arquivo é uma unidade separada, o que o torna independente, isto é pode ser criado, consultado, processado e removido por algoritmos distintos, sendo uma estrutura fora do ambiente do algoritmo.

Para que o algoritmo possa fazer acesso aos dados armazenados em um arquivo é necessária a utilização de algumas operações básicas que serão descritas a seguir.

OPERAÇÕES BÁSICAS COM ARQUIVO

Para utilizarmos a estrutura de dados do tipo arquivo fazemos o uso de operações básicas como: abrir, fechar, apagar, copiar e renomear.

ABRIR O ARQUIVO:- Para podermos utilizar as informações contidas no arquivo ou para guardá-las nesse arquivo devemos primeiramente abrir o arquivo através do comando:

Abra (arquivo_físico);

onde arquivo é a variável de arquivo previamente definida.

Após a operação de abertura, a informação que estará disponível é sempre a do primeiro registro que foi armazenado, para onde o apontador de registro estará sinalizando.

FECHAR ARQUIVO:- Depois de utilizar as informações de um arquivo devemos fechá-lo para garantir a integridade dos dados armazenados.

Feche(arquivo);

APAGAR ARQUIVO:- Deleta um arquivo existente.

Apaga(arquivo_físico);

RENOMEAR ARQUIVO:- Permite mudar o nome do arquivo existente para um outro nome diferente.

Renomeia(arquivo1_físico,arquivo2_físico);

COPIAR ARQUIVO:- Permite fazer uma cópia exatamente igual do arquivo existente para um outro arquivo.

Copia (arquivo1_físico,arquivo2_físico);

OPERAÇÕES BÁSICAS COM REGISTRO DE UM ARQUIVO

Quando já temos um arquivo aberto, podemos tanto **ler** os registros já armazenados no arquivo como **escrever** (gravar) novos registros no arquivo.

LER REGISTRO DO ARQUIVO:- quando lemos um registro de um arquivo estamos transferindo os dados do registro do arquivo, armazenados em memória secundária, para a memória principal do computador (memória RAM). Para isto utilizamos a seguinte instrução:

Leia(arquivo,registro);

Onde:
arquivo: é o nome da variável do tipo de arquivo definido previamente.
registro: é o nome da variável do tipo registro definida anteriormente.

ESCREVER UM REGISTRO NO ARQUIVO (GRAVAR) :- quando escrevemos (gravamos) um registro num arquivo, estamos transferindo os dados do registro, armazenados em memória principal, para a memória secundária do computador (disquete, disco rígido, CD, etc). Utilizamos a seguinte instrução:

Escreva(arquivo,registro);

Onde:
arquivo: é o nome da variável do tipo de arquivo definido previamente.
registro: é o nome da variável do tipo registro definida para compor o arquivo.

ORGANIZAÇÃO DE ARQUIVOS

As operações realizadas com registros em uma estrutura de dados do tipo arquivo podem ser resumidas em: inserção de um novo registro (inclusão), obtenção de um registro do arquivo (consulta), modificação dos dados armazenados no arquivo (alteração) ou remoção de um registro do arquivo (exclusão). Dependendo do tipo de problema estas operações poderão ocorrer em maior ou escala.

Neste caso é necessário que seja analisada qual é a melhor forma de acessar esses registros. Basicamente existem duas formas diferentes de acesso aos registros de um arquivo: o acesso seqüencial e o acesso direto (também conhecido como randômico). Existem outras formas de acesso que são variações destas.

Acesso Seqüencial: Quando desejamos ter acesso aos dados de um determinado registro mas não sabemos a localização desse registro no arquivo devemos vasculhar o arquivo desde o início em busca do registro desejado até o final do arquivo se necessário. Esta operação deverá ser repetida seqüencialmente, isto é, avançando pelos registros armazenados até que se encontre o registro procurado. Isto significa que para acessar um registro específico precisamos obedecer a ordem com que os registros foram armazenados no arquivo, o que implica em percorrer todos os registro que o antecedem.

Acesso Direto ou Randômico: Neste caso para acessarmos o registro desejado é necessário que saibamos a sua posição física (posição do registro no arquivo no instante da gravação). Dessa forma é possível se posicionar instantaneamente no registro desejado sem ter que percorrer todos os registros que o antecedem. Para isto, é necessário que um campo do registro armazene a informação do número da posição física do registro dentro do arquivo. Este campo é chamado de **chave**. A chave de um registro deve ser única, pois nunca dois registros diferentes poderão ter a mesma localização.

COMANDOS E FUNÇÕES PARA MANIPULAÇÃO DE REGISTROS

Para posicionar o apontador de registros no registro cuja posição física seja igual a chave usamos o comando:

Posicione (arquivo,chave);
Onde:

arquivo: é o nome da variável do tipo arquivo;
chave: é um número inteiro (constante ou variável) que indica a posição do registro que se deseja acessar.

Para movimentar o apontador de registro para o próximo registro usamos o comando:

Avance(arquivo);

Onde:
arquivo: é o nome da variável do tipo arquivo;

Para indicar o final do arquivo usamos a função FDA( ) que retorna verdadeiro quando o apontador de registro estiver apontando para o próximo registro após o último.

Fda(arquivo);

Onde:
arquivo: é o nome da variável do tipo arquivo;

Para indicar quantos registros estão gravados no arquivo usamos o comando:

Tamanho(arquivo);

Onde:
arquivo: é o nome da variável do tipo arquivo;

EXEMPLO1: Considere um arquivo de alunos, com registros já gravados. O arquivo possui organização seqüencial e armazena os seguintes dados: RA, nome, endereço e telefone de vários alunos. O algoritmo deverá permitir a consulta do telefone de um determinado aluno dado o nome do aluno.

```
Início
   Tipo regaluno = registro
      Caracter:ra,nome,endereco,fone;
      Fim;
   Tipo arqaluno = arquivo composto de regaluno;
   Regaluno: reg;
   Arqaluno: arq;
   Caracter: no;
   Logico:achou;
   Abra(arq);
```

```
    Leia(no);
    achou← falso;
    Repita
       Leia(arq,reg);
       se (reg.nome = no) então
          Início
             Escreva(reg.ra,  reg.fone, reg.endereco);
             Achou ←  verdadeiro;
          Fim;
       Avance(arq);
    Até (fda(arq))  ou (achou=verdadeiro);
    Se achou=falso então escreva(‘ Registro não Cadastrado’);
    Feche(arq);
Fim.
```

EXEMPLO2: Desenvolver um algoritmo que faça uma cópia de backup de um arquivo já existente. O registro do arquivo armazena os seguintes dados: código, valor e situação.

```
Início
    Tipo produto = registro
             inteiro:codigo;
                real:valor;
                caracter:sit;
       Fim;
    Tipo arqprod = arquivo composto de produto;
    produto: pr;
    Arqprod: arqpr,arqprb;
    Abra(arqpr);
    Abra(arqprb);
    Repita
       Leia(arqpr,pr);
       Escreva(arqprb,pr);
       Avance(arqpr);
```

```
        Avance(arqprb);
    Até (fda(arqpr));
    Feche(arqpr);
    Feche(arqpra);
Fim.
```

EXEMPLO3: O algoritmo abaixo permite a inclusão, consulta, alteração e exclusão (lógica) de registros de um arquivo já existente utilizando o acesso direto.

```
Início
    Tipo dados = Registro
            Caracter:codigo;
            Caracter:nome;
            Caracter:e_mail;
        Fim;
    Tipo A_dados = arquivo composto de dados;
    Dados:reg;
    A_dados:arq;

Módulo principal;
    Início
        Abra(arq);
        Menu;
        Fecha(arq);
    Fim;

Módulo menu;
    Caracter:opcao;
    Início
        Repete
            Repete
                Leia(opcao);
            Ate (opcao>=1) e (opcao<=5);
```

```
        Se opcao<>5 então
            Escolha opcao
                Caso 1:inclusao;
                Caso 2:consulta;
                Caso 3:alteracao;
                Caso 4:exclusao;
            Fim;
    Até opcao=5;
Fim;

Módulo inclusão;
   Caracter:op;
Início
   Se tamanho(arq) > 0 então posicione (arq,tamanho(arq+1));
   Repete
      Reg.codigo ← tamanho(arq);
      Leia(reg.nome);
      Leia(reg.e_mail);
      Escreva(arq,reg);
      Avance(arq);
      Escreva('Deseja Continuar <S/N> ? ');
      Repita
         Leia(op);
      Até (op='S') ou (op='N');
   Até op='N';
Fim;

Módulo consulta;
   Caracter:op;
   Inteiro   :num_reg;
Início
  Repete
```

```
    Repete
       Leia (num_reg);
    Até (num_reg>=0) e (num_Reg<=tamanho(arq));
    Posicione(arq,num_reg);
    Leia(arq,reg);
    Escreva(reg.codigo,reg.nome.reg.e_mail);
    Escreva('Deseja Continuar <S/N> ? ');
       Repita
          Leia(op);
       Até (op='S') ou (op='N');
  Até op='N';
Fim;

Módulo alteracao;
    Caracter:op,aux;
    Inteiro   :num_reg;
Início
  Repete
    Repete
       Leia (num_reg);
    Até (num_reg>=0) e (num_Reg< tamanho(arq));
    Posicione(arq,num_reg);
    Leia(arq,reg);
    Escreva(reg.codigo,reg.nome.reg.e_mail);
    Leia (aux);
    Se aux <> '' então reg.nome ← aux;
    Leia(aux);
    Se aux <> '' então reg.e_mail ← aux;
    Posicione(arq,num_reg);
    Escreva(arq,reg);
    Escreva('Deseja Continuar <S/N> ? ');
       Repita
```

```
        Leia(op);
      Até (op='S') ou (op='N');
  Até op='N';
Fim;

Módulo exclusao;
   Caracter:op;
   Inteiro   :num_reg;
Início
Repete
   Repete
      Leia (num_reg);
   Até (num_reg>=0) e (num_Reg< tamanho(arq));
   Posicione(arq,num_reg);
   Leia(arq,reg);
   Se reg.nome <> '*****' então
      Início
      Escreva(reg.codigo,reg.nome.reg.e_mail);
      Escreva ('Confirma a Exclusão <S/N> ?');
      Repita
         Leia(op);
      Até (op='S') ou (op='N');
      Se op='S' então
            Início
               Reg.nome ← '*****';
               Posicione(arq,num_reg);
               Escreva(arq,reg);
            Fim;
      Fim;
      Senão
         Escreva('Código já excluído !!');
      Escreva('Deseja Continuar <S/N> ? ');
```

```
        Repita
            Leia(op);
        Até (op=’S’) ou (op=’N’);
  Até op=’N’;
Fim;
Fim.
```

LISTA DE EXERCÍCIOS - ARQUIVOS

1) Seja a seguinte estrutura de registro:
   R.A.
   NOME

Fazer um algoritmo para cadastrar alunos em um arquivo novo. Parar o processo quando R.A. = “

2) Seja a seguinte estrutura de registro:

   NÚMERO DE INSCRIÇÃO
   NOME
   SEXO
   CURSO

   a) Cadastrar dados sobre candidatos ao vestibular em um arquivo novo, fazendo crítica para entrada de sexo. Parar o processo quando nome = ‘’.
   b) Dado o número de inscrição mostrar na tela os dados do candidato (consulta de acesso seqüencial).

3) A seção de controle de produção de uma fábrica mantém o arquivo de registros de produção por máquinas. Cada registro contém o número da máquina, a data e o número de peças produzidas no dia. Supondo que a fábrica possua três máquinas, escrever um algoritmo que separe o arquivo em três outros arquivos, um para cada máquina. O novo arquivo não precisa conter o número da máquina. Utilizar:

MAQ	número da máquina - numérico inteiro 1,2 ou 3
DATA	dd/mm/aa - 8 caracteres
PECAS	número de peças - numérico inteiro.

4) Uma instituição de pesquisa recolheu amostras em três regiões a respeito do nível de vida da população. Cada amostra constitui um registro com os seguintes campos: sexo, idade, salário e grau de instrução. Em cada região os dados foram armazenados em um arquivo sequencial. Escrever um algoritmo que junte estes arquivos em um único arquivo. O novo arquivo deverá conter em cada registro um campo com o número da região. Utilizar:

SEXO	sexo
ID	idade
SAL	salário
GRAU	grau de instrução
REGIAO	número da região

5) Considerando o arquivo contendo registros gravados com os seguintes campos:

CODMAT	código do material - numérico inteiro
NOMAT	nome do material
QTEST	quantidade em estoque - numérico inteiro
QTMIN	quantidade mínima - numérico inteiro
PRUNIT	preço unitário - numérico real.

a) Fazer a alteração automática do preço unitário de todos os materiais. O usuário deverá fornecer o índice de reajuste.
b) Exibir na tela código e nome do material cuja quantidade em estoque seja inferior ou igual a quantidade mínima.

6 Seja o arquivo contendo a seguinte estrutura de registro:

CODIGO	numérico inteiro
NOME	
SETOR	caracter (letra)
FUNÇÃO	
SALÁRIO	numérico real.

a) Permitir ao usuário escolher o código do funcionário e caso este esteja cadastrado, apresentar os dados na tela e permitir alterar um ou mais campos entre: nome, setor e função. O usuário poderá parar o processo quando desejar.
b) Criar e gravar um arquivo de cópia de segurança do arquivo original.

7) Seja um arquivo contendo registros com a seguinte estrutura:

MARCA	
MODELO	
COR	
ANO	numérico inteiro
PREÇO	numérico real.

a) Dado MARCA e MODELO, apresentar na tela todos os veículos cuja MARCA e MODELO coincidam com as escolhas.

b) Considerando o mesmo arquivo, dado o número do registro, mostrar as informações do veículo na tela e solicitar confirmação para exclusão. Em caso afirmativo regravar o registro com o campo MARCA contendo *** (exclusão lógica). Repetir o processo até que se digite N para opção "mensagem -> Continua exclusão <S/N> ?".