Algoritmos e Estruturas de Dados I

Técnicas de Programação Usando a Linguagem Pascal

Ronaldo A. L. Gonçalves

PREFÁCIO

Hoje em dia os computadores são utilizados nas mais diversas atividades desenvolvidas pelo homem, desde para simples diversão até na solução de problemas complexos envolvendo cálculos gigantescos. Os computadores se tornaram mais atrativos economicamente devido ao avanço tecnológico seguido de aumento na competitividade entre os diferentes fabricantes e o conseqüente barateamento de preços. Milhares de pessoas espalhadas pelo mundo acessam Internet, trocam mensagens, jogam interativamente umas com as outras e até namoram virtualmente.

O poder computacional distribuído nas casas de pessoas comuns é muito grande. Se todas estas máquinas fossem usadas de forma inteligente poderiam juntas resolver problemas complexos tais como a decodificação genética, onde cada pessoa envolvida ficaria responsável por uma pequena parte do problema. É verdade que existe atualmente um esforço neste sentido mas infelizmente a maioria dos computadores é usada por pessoas leigas em atividades que exigem poucos conhecimentos de informática. Estes computadores quase não trabalham mais do que simples *videogames* ou agendas eletrônicas.

O conhecimento necessário para o domínio desta área infelizmente está longe de ser alcançado com a leitura de jornais ou revistas em quadrinhos ou mesmo assistindo programas de auditório na televisão. Quase sempre exige a realização de um curso superior específico e muita dedicação, embora seja possível estudar em cursos profissionalizantes em nível de segundo grau ou até mesmo em cursinhos de nível primário ou ginasial direcionados para público em geral onde o aprendizado é para uso doméstico. Existe atualmente formação em todos os níveis.

Os cursos superiores na área de computação têm crescido bastante no Brasil. São muitos os nomes usados e a diferença principal entre eles está no enfoque adotado para o perfil do profissional que se deseja formar. Entretanto, na maioria deles, a disciplina de Algoritmos e Estruturas de Dados tem sido ofertada como uma disciplina básica e essencial para uma boa formação na área. Alguns entendidos do assunto acreditam que quando um aluno de computação não consegue acompanhar esta disciplina é porque ele está no curso errado. É verdade dizer que este ponto de vista pode ser combatido pelo fato de que o campo de atuação de um profissional da área é extremamente grande e existe uma infinidade de atividades que não exige conhecimentos de algoritmos e estruturas de dados. Entretanto, estamos aqui nos referindo a formação básica onde o conhecimento é exigido, mesmo que não usado no futuro do profissional devido a certas especificidades de sua atuação.

A disciplina de Algoritmos e Estruturas de Dados envolve o ensino de técnicas capazes de resolver diferentes tipos de problemas pela execução de uma seqüência finita de passos, repetitivos ou não, os quais podem operar sobre dados organizados em diferentes estruturas. Estas técnicas podem ser implementadas usando linguagens abstratas ou linguagens reais de programação de computadores tais como a linguagem Pascal. Para aprender bem estas técnicas deve-se seguir uma metodologia de estudo crescente em grau de complexidade, iniciando com a solução de problemas simples tais como encontrar as

raízes reais de uma equação de segundo grau e seguindo para a solução de problemas mais complexos tais como ordenar um conjunto de dados usando árvores binárias balanceadas.

Este livro tem como objetivo principal apresentar técnicas algorítmicas básicas e avançadas e a aplicação destas na solução de problemas que podem ser resolvidos computacionalmente. O público alvo é composto pelos profissionais da área de informática que trabalham com programação de computadores e que queiram aperfeiçoar seus conhecimentos, alunos de cursos superiores na área de informática que queiram usar o livro como livro texto em suas disciplinas de Algoritmos e Estruturas de Dados e demais profissionais, estudantes e interessados com boa capacidade de concentração e raciocínio lógico que queiram desvendar as técnicas de programação algorítmica usando a linguagem Pascal padrão.

1. INTRODUÇÃO

Um computador é formado por 2 componentes: o *Hardware* e o *Software*. O *Hardware* representa os componentes físicos (processador, memória, placas controladoras etc.). O *Software* representa os programas (seqüência de instruções) que são executados pelo *Hardware* com o propósito de produzirem resultados desejados na solução de problemas. A Figura 1 mostra a visão externa simplificada de um computador.

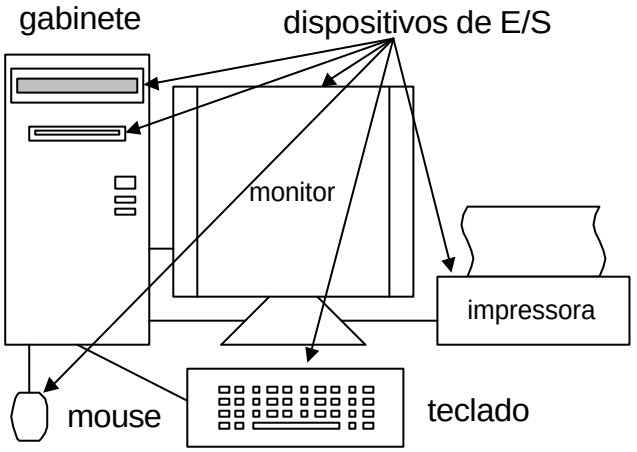


Figura 1 – Visão Externa Simplificada de um Computador

Dentro do gabinete, normalmente estão a fonte de energia (transformador), a placa mãe e provavelmente outras placas controladoras dos diferentes dispositivos periféricos (monitor de vídeo, placa de rede, fax modem etc.) que ficam conectadas na placa mãe (em conectores apropriados chamados *slots*). Em alguns computadores, estas placas controladoras podem vir integradas na própria placa mãe (chamada de placa mãe *on-board*).

A placa mãe é uma placa de circuito impresso que contém o *chip* (pastilha) do processador principal (ou UCP - Unidade Central de Processamento), um ou mais pentes de memória (pequenas placas contendo *chips* de memória conectadas nos *slots* de memória), um ou mais *slots* para conexão de placas controladoras, barramento (fiação impressa na superfície da placa mãe que serve de meio de comunicação entre os componentes da placa) e demais chips, componentes e conectores. A Figura 2 mostra o esquema de um exemplo fictício de placa mãe.

Quando o computador é ligado ele começa a executar ("rodar") automaticamente um programa que já vem armazenado na placa mãe, dentro do *chip* da BIOS (*Basic Input Output System* - memória que não se apaga quando o computador é desligado e que contém pequenos programas de checagem, de controle

de entrada/saída e de inicialização do computador). Este programa, conhecido como "*Startup*", localiza o programa de *boot* (pequeno programa responsável por fazer a inicialização do sistema operacional tal como o Windows ou o Linux) do sistema operacional que deve estar em um disquete ou no HD (*Hard Disk* – também conhecido como disco rígido, winchester ou simplesmente disco) e o coloca na memória para que este seja executado. O *boot* entra em execução e coloca o sistema operacional na memória. O sistema operacional começa a executar e a gerenciar o computador, aguardando os comandos do usuário. O sistema operacional fica instalado no computador, servindo de interface entre este e o usuário. Sempre que sentamos em frente a um computador e tentamos executar algo, nos deparamos com o sistema operacional a nossa frente, monitorando as nossas ações geralmente através de uma interface gráfica bem amigável.

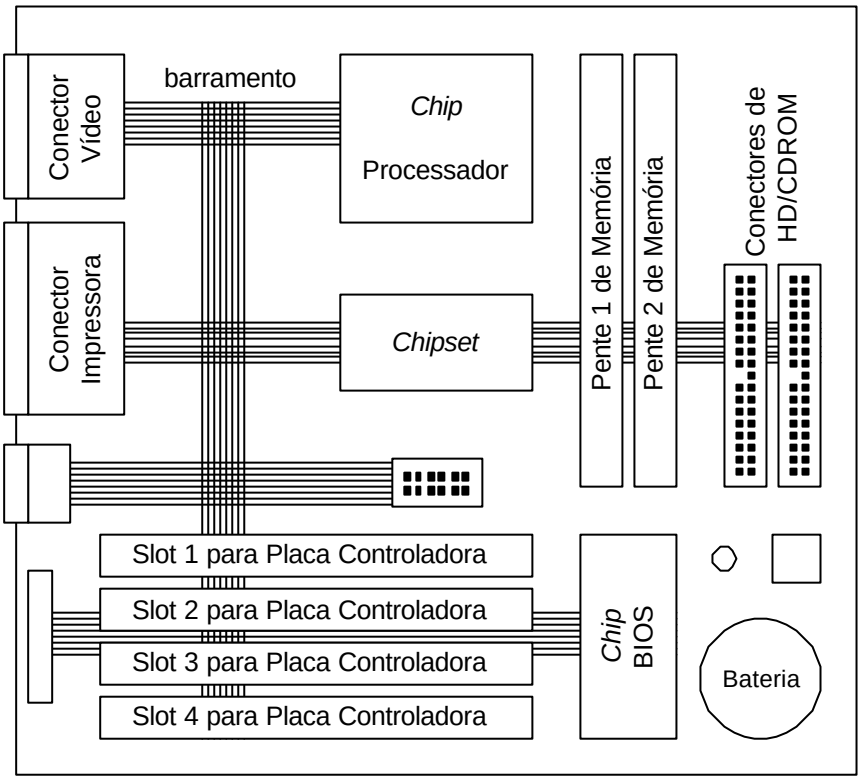


Figura 2 – Exemplo Fictício de uma Placa Mãe

Quando o usuário solicita a execução de um programa (por exemplo, quando clica com o *mouse* sobre um ícone da tela do Windows) o sistema operacional localiza o programa desejado no disquete ou no HD, coloca-o na memória e começa a executá-lo. De tempos em tempos o controle volta da aplicação para o sistema operacional para este possa continuar gerenciando o computador. Sempre que um outro programa precisa ser executado o sistema operacional deve carregá-lo antes na memória. Como já visto, o próprio sistema operacional é um programa que também deve estar na memória.

Bem, neste momento o leitor pode estar perguntando: Mas onde entra Algoritmos e Estruturas de Dados nesta história? Bem, este é o objetivo desta introdução. O autor acredita na capacidade de entendimento do leitor e na sua necessidade mínima de saber relacionar a teoria que irá aprender aqui com a prática que irá desenvolver posteriormente. Para tanto, de imediato faz-se necessário esclarecer o que é programa e o que é memória.

Um programa é um conjunto de instruções (operações) que deve ser executado em seqüência pelo processador. Uma instrução é um código numérico binário escrito com sinais elétricos do tipo "0" ou "1" (podemos entender aqui que o "1" indica a presença de carga elétrica e o "0" a ausência). Conforme a combinação destes "0"s e "1"s (por exemplo, "11001100" e "00111100") uma instrução pode ser interpretada de uma forma diferente pelo processador. Cada "0" ou "1" de uma instrução é chamado de

´bit´ (***bi****nary digi****t***). Uma vez colocado na memória, o processador executa continuamente as instruções do programa, em ordem, lendo a memória da posição inicial até a posição final do programa.

A memória pode ser vista como um local onde são armazenados os programas e os dados a serem manipulados pelos programas. Usando uma definição bastante simplificada, a memória pode ser vista como um conjunto seqüencial e enumerado de palavras (posições de armazenamento), tal como mostra a Figura 3. Uma palavra é um componente eletrônico capaz de armazenar um código binário, ou seja, um conjunto de sinais elétricos "0"s ou "1"s. O tamanho desta palavra pode ser medido em *bytes*, onde cada *byte* pode armazenar um código binário de 8 *bits*. Na Figura 3 podemos observar o exemplo de um trecho de programa localizado entre as posições 100 e 150 da memória. Nesta figura a palavra de endereço 101 está marcada. Ela contém o código binário "00101101" que para o processador terá algum significado como, por exemplo, escrever algo na tela do monitor de vídeo ou obter o código da tecla pressionada pelo usuário do computador ou realizar uma operação aritmética para somar duas outras palavras.

Memória

:	:	:	:	:	:	:	:	:
100	1	0	0	1	1	1	0	0
101	0	0	1	0	1	1	0	1
102	1	1	0	1	1	0	1	1
103	1	0	0	1	1	1	0	0
:	:	:	:	:	:	:	:	:
147	0	1	1	0	0	0	1	1
148	1	1	0	1	0	0	1	0
149	0	0	1	0	0	1	0	0
150	0	1	1	0	0	1	1	1
:	:	:	:	:	:	:	:	:

Figura 3 – Exemplo de um Trecho de Memória Contendo Código Binário

Como podemos ver, a linguagem que o processador pode entender é muito complexa para nós e seria quase impossível escrevermos programas usando "0"s e "1"s. Para facilitar esta tarefa foram criadas as "linguagens de programação" (Pascal, C, Java etc...) de alto nível, cada qual com suas características próprias mas todas parecidas com a nossa linguagem escrita. Assim, quando precisamos escrever um programa o fazemos usando uma dessas linguagens. Depois, o programa em linguagem de alto nível pode ser transformado para linguagem binária (executável) através de um outro programa chamado de compilador.

Por exemplo, podemos escrever um programa em linguagem Pascal (usando um editor simples tal como o Notepad - Bloco de Notas - do Windows) e depois usarmos o compilador Pascal para convertê-lo para um programa executável. A Figura 4 sintetiza estas idéias. Entretanto, para escrevermos programas corretos, eficientes, de fácil manutenção (correções e melhorias futuras) e de maneira fácil devemos fazer uso de técnicas algorítmicas adequadas, daí então a necessidade de aprendermos Algoritmos e Estruturas de Dados.

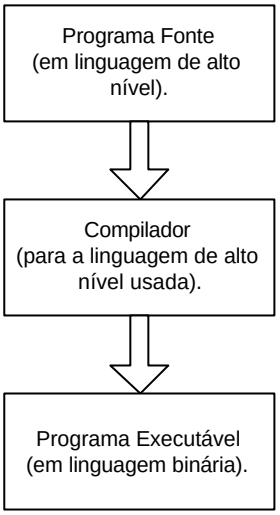


Figura 4 – Esquema da Compilação

2. FUNDAMENTOS BÁSICOS DO RACIOCÍNIO ALGORITMICO

Podemos dizer um algoritmo é uma seqüência de passos a ser executada para resolver um determinado problema. Sem que percebamos usamos algoritmos constantemente em nosso dia a dia. Executamos algoritmos sempre que precisamos realizar alguma tarefa, seja no trabalho, na rua ou em casa. Além disso, na tentativa de economizar esforços, sempre tentamos encontrar uma solução mais elaborada. Por exemplo, qual seria uma possível seqüência de passos para fazermos um café? Bem, sem muito raciocínio podemos pensar em 6 passos que devem ser executados um após o outro:

Passo 1: Lave a garrafa térmica
Passo 2: Coloque o suporte com o coador sobre a garrafa térmica
Passo 3: Coloque o pó de café no coador
Passo 4: Coloque a água para ferver
Passo 5: Quando a água ferver, passe-a sobre o pó
Passo 6: Tome o líquido passado.

Em uma primeira análise, observamos que esta explicação é suficiente para atender aos objetivos a que se propõe. Entretanto, alguém pode questionar: mas qual a quantidade de café? E a quantidade de água? Devemos mexer ou não o pó dentro do coador? Não tem açúcar? Podemos ver que o nível de detalhes pode ser ampliado caso a solução não esteja clara. Além disso, podemos observar que a solução também poder ser obtida invertendo-se a ordem das atividades. Por exemplo:

Passo 1: Coloque a água para ferver
Passo 2: Lave a garrafa térmica
Passo 3: Coloque o suporte com o coador sobre a garrafa térmica
Passo 4: Coloque o pó de café no coador
Passo 5: Quando a água ferver, passe-a sobre o pó
Passo 6: Tome o líquido passado.

Mas qual é a diferença entre as duas soluções? Analisando com maior profundidade concluímos que a segunda solução tende a ser mais eficiente em termos de economia de tempo pois você aproveita o tempo

de fervura da água para fazer outros passos. A idéia de "otimização" é essencial para quem deseja desenvolver uma boa solução algorítmica.

Em muitas situações, apesar de termos a idéia de quais serão os passos, não sabemos quantos serão necessários para resolver definitivamente o problema. É comum executarmos passos já executados anteriormente. Por exemplo, qual seria uma possível seqüência de passos para dividirmos uma quantidade desconhecida de balas entre três crianças, Paulo, João e Maria? Uma possibilidade seria executar os 4 passos a seguir.

Passo 1: Dê uma bala para o Paulo
Passo 2: Dê uma bala para o João
Passo 3: Dê uma bala para a Maria
Passo 4: Caso ainda reste pelo menos 3 balas então volte ao passo 1 novamente
Passo 5: Chupem

O leitor atento poderá perguntar: Quando o passo 5 será executado? A resposta é simples: quando passo 4 falhar, isto é, quando houver menos do que 3 balas. Mas então, e as balas que sobraram? Realmente, o algoritmo não analisa esta questão. Neste caso está implícito que as balas restantes ficarão com quem as distribuiu (existe uma quarta pessoa – o distribuidor – ou este algoritmo é executado por uma das 3 crianças). Lembre-se: estamos pensando em passos que devam ser executados seqüencialmente sem alteração da ordem em que foram estabelecidos.

Bem, antes de prosseguirmos torna-se necessário esclarecer que quando desenvolvemos um algoritmo para um problema a ser resolvido computacionalmente, devemos ter em mente que quem irá executá-lo é o computador. Assim, este algoritmo será o raciocínio que o computador irá utilizar para solucionar um problema. Devemos então pensar pelo computador. Por exemplo, como o computador poderia encontrar as raízes reais de uma equação de segundo grau, isto é, quais os passos que o computador deveria executar para resolver este problema? Poderia ser:

Passo 1: Obter do usuário os coeficientes a, b e c pelo teclado
Passo 2: Calcular o valor de delta
Passo 3: Se delta é positivo calcular x1 e x2 e mostrar na tela
Passo 4: Se delta é negativo emitir mensagem na tela

Neste exemplo podemos ver que em determinados momentos da execução de um algoritmo os computadores precisam de informações a ser providas pelos usuários (usuários são pessoas que estão usando o computador, digitando dados e vendo resultados na tela) enquanto que em outros momentos os computadores precisam mostrar os resultados de seus algoritmos. De uma forma geral, os dados a serem providos ao computador pelos usuários são chamados de entradas.

As entradas podem ser feitas usando dispositivos chamados de "dispositivos de entrada", tais como teclado, disco e mouse, entre outros. Já os dados a serem mostrados aos usuários pelo computador são chamados de saídas. As saídas podem ser feitas por dispositivos chamados de "dispositivos de saída", tais como monitor de vídeo, impressora e disco, entre outros. Todos estes dispositivos juntos são chamados de dispositivos de entrada e saída ou de E/S ou de I/O do inglês *input/output*.

Exemplo para Fixação:

1. Escreva possíveis passos a serem executados por um pedreiro para erguer um muro. Faça 3 versões da solução. A primeira contendo 3 passos básicos. A segunda contendo 6 passos um pouco mais detalhados. A terceira contendo 10 passos mais detalhados ainda.

Primeira Versão

```
1.   efetuar as medições
2.   preparar a massa
3.   assentar os blocos
```

Segunda Versão

```
1.   efetuar as medições
2.   cavar a vala para o alicerce
3.   preparar o concreto
4.   concretar o alicerce
5.   preparar a massa
6.   assentar os blocos
```

Terceira Versão

```
1.   efetuar as medições
2.   cavar a vala para o alicerce
3.   pegar pedra, areia e cimento
4.   preparar o concreto
5.   preparar ferragem
6.   concretar o alicerce com ferragem
7.   pegar areia, cimento e cal
8.   preparar a massa
9.   assentar os blocos
10.  rebocar o muro
```

3. LINGUAGEM PASCAL: COMANDOS E ALGORITMOS BÁSICOS

Uma linguagem de programa consiste de um conjunto de regras (gramática) que define a sintaxe e a semântica dos algoritmos nela escritos. Ela contém cláusulas, comandos, identificadores, palavras reservadas e símbolos especiais que permitem especificar e desenvolver programas. Neste livro iremos desenvolver algoritmos usando a linguagem de programação Pascal, a qual é uma linguagem bastante estruturada e de fácil aprendizagem. A linguagem Pascal originou-se de uma simplificação da linguagem ALGOL, para propósitos educacionais, e foi idealizada por Niklaus Wirth em 1971, recebendo este nome em homenagem ao filósofo e matemático francês. De certa forma, um programa escrito em linguagem Pascal se parece com um algoritmo escrito em língua inglesa. Uma vez escrito um algoritmo em linguagem Pascal, este algoritmo poderá transformar-se em um programa binário usando para isto um compilador Pascal.

Entre as principais palavras reservadas podemos destacar em ordem alfabética: `and`, `array`, `begin`, `case`, `const`, `div`, `do`, `downto`, `else`, `end`, `file`, `for`, `function`, `if`, `in`, `mod`, `not`, `of`, `or`, `procedure`, `program`, `record`, `repeat`, `string`, `then`, `to`, `type`, `until`, `uses`, `var` e `while`. A estrutura básica de um programa em pascal pode ser visualizada no modelo a seguir, sendo composta por 2 partes principais: a primeira parte para a declaração (definição) de variáveis e estruturas de dados a serem manipulados pelo programa e a segunda parte para a especificação dos comandos que definem as ações que o programa deve executar. Todo programa em Pascal deve também ter um nome.

```
program  <nome_do_programa>;
var   <bloco_de_variáveis_e_estruturas_declaradas>
begin
      <bloco_de_comandos>
end.
```

O modelo acima mostra o formato geral segundo o qual devemos editar (digitar) um programa em linguagem Pascal, usando um editor de caracteres qualquer disponível no Windows, para que o mesmo possa posteriormente ser compilado. Muitas vezes, o próprio compilador Pascal é distribuído integrado com um editor próprio e com outras ferramentas dentro de um ambiente de programação tal como o Turbo Pascal da Borland. Ao tentar compilar um programa, que não esteja estritamente dentro das regras gramaticais definidas pela linguagem, recebemos uma mensagem de erro, mesmo que o erro seja a falta ou excesso de um único ponto.

3.1 Declaração de Variáveis

Uma variável é uma entidade que possui um nome e que poderá conter um dado a ser manipulado pelo programa, tal como as variáveis da matemática. Entretanto, na programação as variáveis podem ter tipos diferentes. Quando declaramos uma variável informamos o seu nome e o seu tipo. Os tipos básicos podem ser numéricos (`shortint`, `byte`, `integer`, `word`, `longint`, `single`, `real`, `double`, `comp` e `extended`), caractere (`char`), cadeias de caracteres (`string`) ou lógico (`boolean`). Esta diferenciação é feita porque cada tipo utiliza uma quantidade diferente de memória para armazenar seu respectivo dado. Quando um programa é compilado, os nomes das variáveis são transformados em endereços de memória onde os dados das respectivas variáveis serão manipulados. Todo espaço de memória necessário para as variáveis declaradas é reservado (alocado) durante a compilação juntamente com as instruções do programa executável. A Tabela 1 mostra os tipos de dados, seus respectivos valores permitidos e a quantidade de memória.

Tipo	Valores Permitidos	Memória
shortint	inteiros entre -128 e 127	1 *byte*
byte	inteiros entre 0 e 255	1 *byte*
integer	inteiros entre -32768 e +32767	2 *bytes*
word	inteiros entre 0 e 65535	2 *bytes*
longint	inteiros entre -2147483648 e +2147483647	4 *bytes*
single	reais entre 1.5 E-45 e 3.4E+38	4 *bytes*
real	reais entre 2.9E-39 e 1.7E+38	6 *bytes*
double	reais entre 5.0E-324 e 1.7E+308	8 *bytes*
comp	reais entre $-2^{63} + 1$ e $2^{63} - 1$	8 *bytes*

`extended`	reais entre 3.4E-4932 e 1.1E+4932	10 *bytes*
`char`	1 caractere	1 *byte*
`string[n]`	1 cadeia com n caracteres	± n+1 *bytes*
`boolean`	1 valor lógico (**`true`** ou **`false`**)	1 *byte* (1 *bit*)

Tabela 1 – Tipos de Dados da Linguagem Pascal

O modelo básico de declaração de uma variável é:

```
var <nome> : <tipo>;
```

Neste modelo, a declaração é feita iniciando com uma cláusula `var` seguida de um identificador de variável (nome que se deseja dar a variável), dois pontos, o tipo da variável (tal como definido na tabela 1) e a finalização com um ponto e vírgula. Também é possível declarar diversas variáveis de uma só vez, sendo todas de um mesmo tipo, tal como o modelo abaixo.

```
var <nome1>, <nome2>, <nome3> : <tipo>;
```

ou ainda:

```
var <nome1>, <nome2>, <nome3> : <tipo1>;
    <nome4>, <nome5> : <tipo2>;
      :    :    :     :
```

As variáveis numéricas podem conter somente números. Por exemplo, uma variável do tipo `integer` pode conter um número inteiro negativo, nulo ou positivo, mas jamais um número fracionário ou com casas decimais separadas por uma vírgula. Se o programa precisar manipular números fora desta faixa então deverá usar uma variável de outro tipo. Números fracionários ou com casas decimais são considerados números reais. O tipo `char` pode conter um caractere, não sendo necessário mais do que um *byte* para armazenar o código binário que representa o caractere. O tipo `string` pode conter uma cadeia de caracteres limitada a 255 também chamada de literal. Veremos que cada declaração deve especificar o número de caracteres envolvidos na cadeia. Além disso, normalmente o compilador um *byte* a mais em cada cadeia para armazenar o tamanho válido da mesma. O tipo `boolean` pode conter um entre dois valores possíveis (`true` - verdadeiro ou `false` - falso), não sendo necessário mais do que 1 *bit*, entretanto, utiliza-se um *byte* pela comodidade na compilação. A seguir é exemplificada uma seqüência de declarações.

```
var   i: integer;
      j, k: byte;
      nome: string[30];
      letra: char;
      resposta: boolean;
```

Observe que não é necessário utilizar mais do que uma palavra reservada **`var`** para declarar todo um conjunto de variáveis. Os nomes de variáveis, chamados de identificadores, não podem ser coincidentes com palavras reservadas da linguagem e existem regras para a sua formação, conforme segue. Um nome de variável não pode ser composto separado por espaço em branco (neste caso, pode-se usar um traço baixo `_` para criar nomes compostos tais como "número_primo"), nem começar com números ou com caracteres especiais (veja Tabela 2). Além disso, as letras maiúsculas e minúsculas são consideradas iguais e somente os primeiros 63 caracteres do nome são significativos. Preferencialmente, o nome de uma variável deve deixar claro qual é o significado do dado que ele armazena.

.	fim de programa, decimal, seleção de campo dentro do tipo `record`
( )	precedência, parâmetro
+	adição, concatenação
-	subtração, negação
/	divisão real
not	negação de um valor do tipo `boolean`
div	divisão inteira
mod	resto da divisão inteira
and	disjunção entre condições lógicas
or	conjunção entre condições lógicas
xor	ou exclusivo entre condições lógicas
=	igualdade, declaração
<>	desigualdade ou diferença
>	maior que
<	menor que
>=	maior ou igual
<=	menor ou igual
*****	multiplicação
:=	atribuição
in	pertence a ou está contido
#	prefixo que identifica o código ASCII de caractere (de 0 a 255)
,	separador de lista
'	delimitador de seqüência de caracteres
$	prefixo que indica um número hexadecimal
:	declaração de tipo
..	identifica intervalo em tipo ordinal simples ou faixa de valores
;	separa declarações e indica fim de definições, cabeçalhos, blocos, ...
[ ]	referência a um elemento de um `array`
{ }	delimita um comentário dentro de um programa
(* *)	delimita um comentário dentro de um programa

Tabela 2 – Caracteres e Símbolos Especiais da Linguagem Pascal

Exemplo para Fixação:

1. Declare as variáveis necessárias para armazenar as seguintes informações: título de livro, nome do autor, ano de publicação, número de página, preço unitário, quantidade disponível em estoque e se possui ou não capa dura.

```
var  titulo,nome:string[64];
     anopub,nropag,quant:integer;
     valor_uni:real;
     capadura:boolean;
```

3.2 Expressões

Uma expressão é uma combinação de operadores e operandos (variáveis e constantes) que depois de ser avaliada produz um valor. O tipo da expressão é dado pelo tipo do valor resultante da sua avaliação. Este tipo pode enquadrar-se dentro de um dos tipos aceitos pela linguagem. Mas para entender melhor precisamos ainda apresentar os conceitos de constantes e operadores. Uma constante representa o dado propriamente dito que poderá ser ou não armazenado dentro de uma variável. As constantes podem ser dos mesmos tipos que as variáveis. Assim, temos constantes numéricas, caracteres, literais e lógicas. A Tabela 3 mostra vários exemplos válidos e inválidos de constantes. Note que os caracteres e os literais são delimitados por aspas simples. Isto é feito para não confundi-los com os identificadores das variáveis.

tipo	exemplo válido	exemplo inválido
numérica	1 ; -6 ; 50,45621 ; 30E-10	´1´ ; ´234,67´ ; X ; Y
caractere	´a´ ; ´1´ ; ´$´ ; ´´´ ; ´ ´ ; ´;´	1 ; a ; @
literal	´begin´ ; ´a´ ; ´123,56´	34567,89 ; nome ; begin
lógico	`true` ; `false`	{qualquer outro}

Tabela 3 – Tipos e Exemplos de Constantes

Os operadores são os elementos ativos de uma expressão, ou melhor, são os elementos que exercem operações sobre as variáveis e constantes produzindo novas constantes. Quando um operador é aplicado a somente um operando ele é dito unário e quando é aplicado a dois operandos ele é dito binário. Os operadores podem ser aritméticos, lógicos ou relacionais.

Operadores Aritméticos

Os operadores aritméticos são parecidos com aqueles conhecidos da matemática e funcionam como tal. Os principais são: adição (`+`), subtração (`-`), multiplicação (`*`), divisão real (`/`), divisão inteira (`div`) e resto da divisão inteira (`mod`). Uma expressão contendo operadores aritméticos dever ser avaliada considerando a existência de prioridade entre eles. A prioridade dos operadores define a ordem segundo a qual eles devem ser avaliados. As operações de multiplicação e divisão devem ser resolvidas antes das operações de soma e subtração. O resultado da avaliação de uma operação aritmética é sempre um valor numérico. Quando na expressão existir parênteses, a parte interna de cada parêntese deve ser analisada antes da parte externa.

Exemplo1: Expressão cuja avaliação produz um valor real negativo.

(16 `mod` 7) * (18 `div` 5) + 21 / 5 – 12 =
(2) * (3) + 4,2 – 12 =
6 – 7,8 =
– 1,8

Operadores Lógicos

Os operadores lógicos são aplicados em expressões lógicas e produzem como resultados valores do tipo `boolean` (`true` ou `false`). Por questões de simplicidade usaremos as notações V e F para `true` e `false`, respectivamente. Os principais operadores lógicos são conjunção (`or`), disjunção (`and`) e negação (`not`), sendo que a ordem de maior prioridade é `not`, `and` e `or`. O funcionamento destes operadores pode ser visualizado na Tabela 4 (tabela verdade), para operações sobre dois valores lógicos A e B (os quais podem ser variáveis, constantes ou expressões lógicas).

Valores de Entrada		**Resultados da Aplicação dos Operadores**			
A	B	**not** A	**not** B	A **or** B	A **and** B
F	F	V	V	F	F
F	V	V	F	V	F
V	F	F	V	V	F
V	V	F	F	V	V

Tabela 4 – Tabela Verdade com Operações Básicas

Podemos observar na Tabela 4 que o operador `not` simplesmente inverte o valor do dado de entrada. Se o dado é `true` então o resultado será `false`, caso contrário será `true`. Para entender o funcionamento dos operadores `or` e `and` vejam os dois circuitos elétricos da Figura 5. O primeiro, o qual faz uma analogia a porta lógica `or`, contém duas chaves em paralelo ligando uma lâmpada a uma bateria e o segundo, o qual faz uma analogia a porta lógica `and`, é semelhante só que as chaves estão em série. No primeiro, para que a lâmpada acenda basta que qualquer uma das chaves esteja ligada (ou as duas). Neste caso, a lâmpada só não acenderá se as duas chaves estiverem simultaneamente desligadas. Já no segundo, a lâmpada somente acenderá quando as duas chaves estiverem simultaneamente ligadas. Neste caso, a lâmpada não acenderá quando qualquer uma das duas chaves estiver desligada (ou as duas).

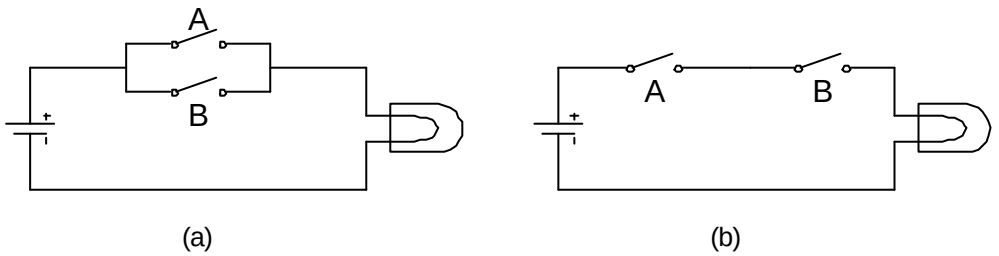


Figura 5 – Circuitos Elétricos Representativos dos Operadores (a) **`or`** e (b) **`and`**.

Exemplo 1: Para exemplificar a utilização dos operadores lógicos vamos avaliar a expressão "A `or` B `and not` C" para todos os valores de A, B e C. Para tanto, devemos montar uma tabela verdade e testar todas as possibilidades considerando as prioridades dos operadores. Podemos utilizar variáveis intermediárias X e Y para facilitar a análise, tal que X assuma o valor obtido na sub-expressão "A `or` B" e Y assuma o valor obtido na sub-expressão "`not` C" e para calcula-las usaremos a Tabela 4. Neste caso, a expressão se reduzirá a X `and` Y. Por questões de simplicidade usaremos V e F para `true` e `false`, respectivamente. O resultado é mostrado na Tabela 5.

A	B	C	X	Y	X **and** Y
F	F	F	F	V	F
F	F	V	F	F	F
F	V	F	V	V	V

F	V	V	V	F	F
V	F	F	V	V	V
V	F	V	V	F	F
V	V	F	V	V	V
V	V	V	V	F	F

Tabela 5 – Tabela Verdade Resultante

Exemplo 2: Um outro exemplo mais realista pode ser visto na seguinte situação cotidiana. A mãe de Paulinho diz a ele: "Filho, vá até a padaria e me traga 1 ou 2 litros de leite e 5 ou 10 pãezinhos; depois vá ao açougue e me traga 2 quilos de carne moída e 1 ou 2 quilos de bife de patinho ou alcatra". Como o garoto entendia um pouco de lógica, anotou tudo num papel e para executar as tarefas corretamente optou por fazer uma expressão lógica. Para tanto, ele nomeou as diferentes atividades com variáveis lógicas da seguinte maneira:

```
A = trazer 1 litro de leite da padaria
B = trazer 2 litros de leite da padaria
C = trazer 5 pãezinhos da padaria
D = trazer 10 pãezinhos da padaria
E = trazer 2 quilos de carne moída do açougue
F = trazer 1 quilo de bife de patinho do açougue
G = trazer 2 quilos de bife de patinho do açougue
H = trazer 1 quilo de bife de alcatra do açougue
I = trazer 2 quilos de bife de alcatra do açougue
```

Assim, a expressão ficou "(A `or` B) `and` (C `or` D) `and` (E) `and` (F `or` G `or` H `or` I)". Então, a partir dela, Paulinho soube tirar proveito da situação, já que sua mãe lhe havia autorizado: ele comprou o melhor para ele: 2 litros de leite, 10 pãezinhos, 2 quilos de moída de filé Mignone (já que ela não havia determinado o tipo) e 2 quilos de bife de alcatra. Desta forma, ele percebeu que atribuindo os seguintes valores para as variáveis: A = F, B = V, C = F, D = V, E = V, F = F, G = F, H = F e I = V, o resultado final ficaria verdadeiro. Caso o leitor estiver interessado, poderá praticar e montar toda a tabela verdade. Para o autor esta atividade seria muito tediosa, mesmo porque, neste caso, os possíveis resultados são bastante visíveis mesmo sem a tabela.

Operadores Relacionais

Uma outra classe de operadores é a dos operadores relacionais. Este tipo de operador deve ser aplicado em expressões comparativas, produzindo resultados booleanos. A Tabela 6 mostra os tipos de operadores relacionais existentes em Pascal. Após a tabela é possível observar sua utilização através de um exemplo.

Operador	Relação
=	Igual
< >	Diferente
<	Menor
>	Maior
<=	menor-igual
>=	maior-igual

Tabela 6 – Relacionamentos Possíveis

Exemplo 1: Considere as seguintes variáveis e valores:

```
A = true
B = false
X = 2.5
Y = 5.0
Nome1 = ´Joao´
Nome2 = ´Jose´
Nome3 = ´Joaozinho´
```

Usando as variáveis anteriores podemos avaliar algumas expressões comparativas tal como mostra a Tabela 7. Observe que o relacionamento entre valores literais é feito de acordo com a ordem alfabética.

Expressão	Resultado
(X > Y) <> B	F
not (X > Y)	V
Y = 2*X	V
Nome1 = Nome2	F
Nome1 < Nome2	V
Nome3 > Nome1	V
A **and** (X>Y/3)	V

Tabela 7 – Exemplos de Expressões Relacionais

Exercícios para Fixação:

1. Declare as variáveis necessárias para armazenar as seguintes informações: título de livro, nome do autor, ano de publicação, número de página, preço unitário, quantidade disponível em estoque e se possui ou não capa dura.

```
var  titulo,nome:string[64];
     anopub,nropag,quant:integer;
     valor_uni:real;
     capadura:boolean;
```

1. Avalie a expressão: `[(2+3*2)/(4-2)*2+12]*2`

```
= [(2+6)/(2)*2+12]*2 = [8/4+12]*2 = [2+12]*2 = 14*2 = 28
```

2. Avalie a expressão: `[(5+2)>5]and[2<(3-1)or(5>6)]`

```
= [7>5] and [2<2 or 5>6] = [V] and [F or F] = V and F = F
```

3.3 Especificação de Comandos

A partir desta seção são introduzidos os comandos da linguagem Pascal, que podem ser utilizados em seqüência para a execução de determinada tarefa.

3.3.1 Comando de Atribuição :=

Serve para atribuir um valor ou o resultado de uma expressão a uma variável. É representado pelos símbolos ´:=´ (dois pontos igual). A operação de atribuição deve ser finalizada com um ponto e vírgula. O tipo da variável deve ser compatível com o tipo do valor ou resultado da avaliação da expressão a ser atribuído. Sua sintaxe é:

```
<nome_da_var> ≔ <valor ou expressão> ;  {ponto e vírgula necessário}
```

Exemplo 1: algumas pequenas atribuições

```
nome ≔ ´Virginia Lane´;      {nome deve ser do tipo literal}
flag1 ≔ flag2;                     {flag1 e flag2 devem ser do mesmo tipo}
N1 ≔ 2;                            {N1 deve ser do tipo inteiro ou real}
N2 ≔ 24.67;                  {N2 deve ser do tipo real}
flag1 ≔ true;                      {flag1 deve ser do tipo lógico}
media ≔ (N1 + N2)/2;               {media deve ser do tipo real}
status ≔ (m ≥ 6) and (p ≥ 0.8);    {status deve ser lógico}
```

Exemplo 2: Um pequeno programa em Pascal utilizando atribuição.

```
{programa que realiza atribuições}
program simplex_1;
var  preco_unitario,          {armazena o preço individual de um produto}
     preco_total:real;        {armazena o preço total de vários produtos}
       quantidade:integer;     {armazena a quantidade de produtos}
begin
  preco_unitario:=5.0;        {valor de cada produto individualmente}
  quantidade:=10;             {estabelece a quantidade de produtos}
  preco_total:=quantidade*preco_unitario; {calcula o preço total}
end.
```

Importante! Talvez o leitor esteja preocupado em querer entender o significado destas variáveis, o porquê dos nomes, o objetivo deste programa quando for executado. Mas não se preocupe por enquanto, mesmo porque o significado funcional neste exemplo é desprezível. A idéia agora é somente exemplificar a utilização do comando de atribuição. Entretanto, várias características devem ser observadas neste exemplo e seguidas quando do desenvolvimento de programas. Note que os nomes das variáveis são sugestivos. Isto facilita o entendimento do programa durante o desenvolvimento e nas alterações futuras. Também, veja que o programa está comentado (comentários entre chaves). Esta prática deve ser uma

constante para os bons programadores e também visa facilitar o entendimento do programa durante o desenvolvimento e nas alterações futuras. Apesar disto, neste livro por questões de economia de espaço, somente usaremos comentários fundamentais para um melhor entendimento do leitor.

Podemos já neste momento questionar o leitor sobre o que acontecerá quando executarmos este programa. Deve estar claro que as ações a serem executadas pelo programa estão entre o **`begin`** e o **`end`**. A declaração de variáveis é apenas informativa para o compilador gerar o código corretamente. Então, podemos observar que o programa irá executar 3 comandos e depois terminará. Em um tempo *t1* o programa atribuirá o valor 5 para a variável `preco_unit`. No tempo *t2*, após *t1*, o programa atribuirá o valor 10 para a variável `quantidade`. No tempo *t3*, após *t2*, o programa atribuirá o resultado de uma expressão (multiplicação neste caso) para a variável `preço_total`. Finalmente, no tempo *t4*, após *t3*, o programa finalizará. O tempo necessário para executar cada comando depende da complexidade de cada um. Por exemplo, o terceiro comando irá levar um tempo maior porque é mais complexo do que os demais. O importante é termos ciência de que cada comando somente será executado após o término do comando anterior.

3.3.2 Comando de Saída de Dados `Write/Writeln`

Muitas vezes chamado de comando de escrita de dados. Serve para mostrar um dado (constante, valor de variável ou resultado de uma expressão) em um dispositivo de saída tal como a tela de vídeo, impressora ou disco rígido, para que o usuário do programa possa visualizá-lo. Nos algoritmos aqui apresentados será utilizada somente a tela de vídeo como dispositivo de saída. Sua sintaxe é mostrada abaixo:

```
write (<lista_de_argumentos>);
```

ou

```
writeln (<lista_de_argumentos>);
```

onde `<lista_de_argumentos>` pode ser um ou mais dados separados por vírgula.

Exemplo 1: Utilização típica com apenas um argumento literal.

```
write (´Pressione uma tecla ´);
writeln (´Pressione uma tecla ´);
```

Bem, de imediato podemos dizer que ambos comandos irão mostrar na tela do vídeo a mensagem "`Pressione uma tecla` ", mas para entender com maior clareza devemos antes introduzir o conceito de cursor. Cursor é aquela marca luminosa que aparece no vídeo do computador, podendo aparecer como um "traço deitado" ou "traço em pé" ou ainda como uma "caixinha". O cursor serve para marcar a posição da tela onde serão escritas as próximas mensagens. Assim, quando o programa executa um comando de escrita ou quando o usuário digita algum dado, estas informações aparecem a partir da posição do cursor. A cada caractere que é disparado na tela o cursor se desloca para a direita, onde o próximo caractere poderá ser disparado. O cursor sempre fica posicionado no final da informação que foi disparada, marcando assim a nova posição onde as próximas mensagens serão escritas. Bem, com relação aos comandos `write` e `writeln` a diferença é que no primeiro o cursor pára imediatamente após a

escrita das informações, ou seja, na mesma linha, enquanto que no segundo, o cursor pula para o início da próxima linha logo após a escrita das informações? Este recurso serve para evitar a confusão das informações na tela.

Dica: O comando `writeln` pode ser usado sem argumentos e sem parênteses para simplesmente deslocar o cursor para a linha de baixo. Vários comandos deste podem ser usados para deslocar o cursor várias linhas abaixo, tal como o trecho a seguir que desloca o cursor 3 linhas para baixo.

```
writeln;
writeln;
writeln;
```

Exemplo 2: Estendendo o programa anterior para mostrar o resultado do cálculo

```
{programa que calcula preço total e mostra o resultado na tela}
program     simplex_2;
var  preco_unitario,
     preco_total : real;
     quantidade : integer;
begin
   preco_unitario  := 5.0;
   quantidade  := 10;
   preco_total  := quantidade * preco_unitario;
   writeln(´Preco Unitario = ´, preço_unitario);
   writeln(´Quantidade = ´, quantidade);
   write(´Preco Total = ´, preço_total);
end.
```

Neste exemplo vemos o uso de dois argumentos nos comandos de escrita. O primeiro argumento é uma constante literal e o segundo é uma variável numérica.

3.3.3 Comando de Entrada de Dados `Read/Readln`

Muitas vezes chamado de comando de leitura de dados. Serve para que o programa possa obter dados para as suas variáveis internas. Geralmente estes dados são obtidos do usuário através do teclado (´mas pode ser obtido de outros dispositivos de entrada como o disco´). Sua sintaxe é mostrada abaixo:

```
read(<lista_de_variáveis>);
```

ou

```
readln(<lista_de_variáveis>);
```

onde `<lista_de_variáveis>` pode ser uma ou mais variáveis separadas por vírgula.

Exemplo 1: Lendo dois argumentos

```
read(x,y);
```

Neste exemplo, quando o computador for executar o comando `read` ele irá esperar até que 2 valores sejam digitados pelo usuário separados por um espaço em branco. Após a digitação dos 2 valores, para finalizar a entrada de dados, o usuário deverá apertar uma tecla especial chamada *<enter>*. Após o pressionamento desta tecla o programa prosseguirá normalmente para o próximo comando. Se o usuário digitar somente um dado e pressionar *<enter>*, o computador continuará esperando pelo segundo dado seguido de *<enter>*. Se o usuário digitar um dado que não seja compatível com o tipo da variável associada, o programa será interrompido pelo computador.

Exemplo 2: Melhorando o exemplo anterior com a leitura de dados

```
program simplex_3;
var  preco_unitario,preco_total:real;
     quantidade:integer;
begin
   read(preco_unitario,quantidade);
   preco_total:=quantidade*preco_unitario;
   write(preco_total);
end.
```

Neste exemplo, o computador solicita ao usuário que ele digite dois valores para serem atribuídos às variáveis `preco_unitario` e `quantidade`. Após o usuário digitar estes 2 valores ele deverá teclar *<enter>*, quando então o computador passará a calcular o valor de `preço_total` para depois escrever o resultado na tela (ou impressora etc...). Note que quando este programa for executado a primeira ação que ele irá fazer é solicitar uma entrada de dados. Mas como o usuário pode saber que o programa está parado esperando que ele digite algo?

Importante! Devemos pensar que quando um programa for executado pelo computador, o usuário pode não entender aquilo que o programa deseja que ele faça. Assim, sempre que o programa desejar alguma entrada de dados ele deve antes escrever uma mensagem informando aquilo que deseja que seja digitado. O mesmo acontece quando o computador for escrever um resultado qualquer no vídeo; antes ele deve informar o que está sendo escrito.

Assim podemos melhorar o exemplo anterior da seguinte forma:

```
program calcula_preco_total;
var  preco_unitario,preco_total:real;
     quantidade:integer;
begin
   write(´Digite o Preço Unitário e a Quantidade => ´);
   readln(preco_unitario,quantidade);
   preco_total := quantidade*preco_unitario;
   write(´O Preco Total é : ´);
   writeln(preco_total);
end.
```

Exemplo 3: Programa para calcular a área de um triângulo.

```
program calculo_area_triangulo;
var  base,altura:integer;
     area:real;
begin
   write(´Digite o valor da base : ´);
   readln(base);
   write(´Digite o valor da altura: ´);
   readln(altura)
   area:=(base*altura)/2;
   writeln(´Area = ´,area);
end.
```

Importante! Apesar de ser estritamente necessário disparar informações no vídeo para auxiliar o usuário na entrada de dados do programa, bem como na visualização clara e completa dos resultados do programa, esta prática é evitada neste livro devido às restrições de espaço de papel e pelo fato de não implicar em prejuízo no que se refere ao aprendizado das técnicas de programação. Este autor entende que escrever mensagens na tela não impede o desenvolvimento do raciocínio lógico e aproveita para incentivar a todos a se preocuparem com isto na prática.

Exercícios para Fixação:

1. Escrever um pequeno programa em linguagem Pascal. Primeiramente, dê um nome a ele. Agora, declare variáveis para manipular os seguintes dados: marca do veículo, modelo, ano de fabricação, placa, número de portas, tipo da pintura (M = metálico e N = normal) e valor venal.

```
program autocar;
var  marca,modelo:string[20]
     ano, portas: integer;
     placa:string[7];
     tipopint:char;
     valor:real;
```

2. Continue o programa e insira comandos. Faça atribuições iniciais as variáveis anteriormente declaradas.

```
program autocar;
var  marca,modelo:string[20]
     ano, portas: integer;
     placa:string[8];
     tipopint:char;
     valor:real;
begin
   marca:=´ford´;
   modelo:=´versailles´;
   ano:=1994;
   portas:=4;
   placa:=´CAZ 1763´;
```

```
   tipopint:=´N´;
   valor:=10000;
end.
```

3. Mantendo aquilo que já foi feito, na seqüência do código, insira comandos para mostrar todos os valores das variáveis na tela.

```
program autocar;
var  marca,modelo:string[20]
     ano, portas: integer;
     placa:string[8];
     tipopint:char;
     valor:real;
begin
   marca:=´ford´;
   modelo:=´versailles´;
   ano:=1994;
   portas:=4;
   placa:=´CAZ 1763´;
   tipopint:=´N´;
   valor:=10000;
   writeln(´Marca = ´,marca);
   writeln(´Modelo = ´,modelo);
   writeln(´Ano de Fabricação = ´,ano);
   writeln(´Numero de Portas = ´,portas);
   writeln(´Placa = ´,placa);
   writeln(´Tipo da Pintura = ´,tipopint);
   writeln(´Valor de Mercado = ´,valor);
end.
```

4. Mantendo aquilo que já foi feito, na seqüência do código, insira comandos para ler outros valores para as variáveis via teclado.

```
program autocar;
var  marca,modelo:string[20]
     ano, portas: integer;
     placa:string[8];
     tipopint:char;
     valor:real;
begin
   marca:=´ford´;
   modelo:=´versailles´;
   ano:=1994;
   portas:=4;
   placa:=´CAZ 1763´;
   tipopint:=´N´;
   valor:=10000;
   writeln(´Marca = ´,marca);
   writeln(´Modelo = ´,modelo);
   writeln(´Ano de Fabricação = ´,ano);
   writeln(´Numero de Portas = ´,portas);
   writeln(´Placa = ´,placa);
   writeln(´Tipo da Pintura = ´,tipopint);
   writeln(´Valor de Mercado = ´,valor);
   readln(marca);
```

```
  readln(modelo);
  readln(ano);
  readln(portas);
  readln(placa);
  readln(tipopint);
  readln(valor);
end.
```

5. Mantendo aquilo que já foi feito, na seqüência do código, insira comandos para mostrar todos os novos valores das variáveis na tela.

```
program autocar;
var  marca,modelo:string[20]
     ano, portas: integer;
     placa:string[8];
     tipopint:char;
     valor:real;
begin
  marca:=´ford´;
  modelo:=´versailles´;
  ano:=1994;
  portas:=4;
  placa:=´CAZ 1763´;
  tipopint:=´N´;
  valor:=10000;
  writeln(´Marca = ´,marca);
  writeln(´Modelo = ´,modelo);
  writeln(´Ano de Fabricação = ´,ano);
  writeln(´Numero de Portas = ´,portas);
  writeln(´Placa = ´,placa);
  writeln(´Tipo da Pintura = ´,tipopint);
  writeln(´Valor de Mercado = ´,valor);
  readln(marca);
  readln(modelo);
  readln(ano);
  readln(portas);
  readln(placa);
  readln(tipopint);
  readln(valor);
  writeln(´Marca = ´,marca);
  writeln(´Modelo = ´,modelo);
  writeln(´Ano de Fabricação = ´,ano);
  writeln(´Numero de Portas = ´,portas);
  writeln(´Placa = ´,placa);
  writeln(´Tipo da Pintura = ´,tipopint);
  writeln(´Valor de Mercado = ´,valor);
end.
```

3.3.4 Comando de Controle e Decisão `If/Then/Else`

Neste comando, o fluxo de instruções a ser executado é escolhido em função da avaliação de uma condição (expressão lógica). Dependendo do resultado obtido na avaliação da condição, um entre dois caminhos pode ser executado. Sua sintaxe é definida abaixo:

```
If <condição>
then begin
       <sequencia_de_comandos_1> {se a condição for verdadeira}
     end
else begin
       <sequencia_de_comandos_2> {se a condição for falsa}
     end;
```

ou somente

```
if <condição>
then begin
       <sequencia_de_comandos> {se a condição for verdadeira}
     end;
```

Se o resultado obtido com a avaliação da condição lógica `<condição>` for verdadeiro, então será executada a primeira seqüência de comandos (parte do `then`), caso contrário, se o resultado for falso, será executada a segunda seqüência de comandos (parte do `else`). Se a seqüência de comandos for composta por um único comando, então o `begin`/`end` pode ser omitido. Além disso, a parte do `else` é opcional.

Importante! O final da parte do `then` não deve ter "ponto e vírgula" quando houver o `else`.

Exemplo 1: Programa que verifica se uma pessoa é ou não maior de idade.

```
program verifica_idade;
var idade:integer;
begin
  write(´Digite a idade => ´);
  readln(idade);
  if idade>=18
     then writeln(´Voce eh maior de idade´)
     else writeln(´Voce eh menor de idade´);
end.
```

Exemplo 2: Programa que encontrar as raízes de uma equação do segundo grau.

```
program encontra_raizes;
var a,b,c,delta,x1,x2:real;
begin
  write(´Digite os coeficientes a, b e c => ´);
  readln(a,b,c);
  delta:=b*b–4*a*c;
  if delta<0
     then writeln(´Nao existe raizes reais´)
     else begin
            x1:=(-b+sqrt(delta))/(2*a); {sqrt = raiz quadrada}
            x2:=(-b-sqrt(delta))/(2*a);
```

```
            writeln(´As raizes sao: x1 = ´,x1,´ x2 = ´,x2);
          end;
end.
```

Exemplo 3: Programa que classifica triângulos conforme os lados.

```
program classifica_triangulos;
var a,b,c:real;
begin
  write(´Digite os Lados A, B e C => ´);
  readln(a,b,c);
  if (a=b)and(b=c)
      then writeln(´Triangulo Equilatero´)
      else if (a<>b)and(b<>c)and(a<>c)
             then writeln(´Triangulo Escaleno´)
             else writeln(´Triangulo Isosceles´);
end.
```

Dica: Observe neste exemplo que as condições são analisadas de forma a simplificar o programa. Se tentássemos primeiramente descobrir se o triangulo é Isósceles, o algoritmo seria mais complicado, pois a condição seria mais extensa. A questão é que para saber se o triangulo tem apenas dois lados iguais (nem 3 lados iguais, nem 3 lados diferentes) um maior número de comparações é necessário (Sugiro que o leitor tente resolver esta condição para praticar seu raciocínio). Assim, no caso de usar "`if`s aninhados" (`if` dentro de `if`), é comum deixarmos a pior condição por último, para evitar a sua elaboração explícita no algoritmo, colocando assim as condições mais simples nos primeiros `if`s do aninhamento.

Exemplo 4: Programa que obtém 2 números diferentes quaisquer e os imprime em ordem crescente. Supondo que os números sejam n1 e n2, existem 2 possibilidades de ordenação. Ou n1 < n2 ou n2 < n1. Baseado nisto, podemos escrever o programa conforme segue.

```
program ordena_dois;
var  n1,n2:integer;
begin
  write(´Digite o primeiro numero => ´);
  readln(n1);
  write(´Digite o segundo numero => ´);
  readln(n2);
  if (n1<n2)
    then writeln(´numeros ordenados: ´,n1,´ ´,n2);
  if (n2<n1)
    then writeln(´numeros ordenados: ´,n2,´ ´,n1);
end.
```

Observamos neste exemplo que os dois comandos `if`s são independentes e são ambos executados. Entretanto, como as duas condições são mutuamente exclusivas (impossível ambas serem satisfeitas ao mesmo tempo), também podemos escrever o programa conforme segue.

```
program ordena_dois;
var  n1,n2:integer;
```

```
begin
   write(´Digite o primeiro numero => ´);
   readln(n1);
   write(´Digite o segundo numero => ´);
   readln(n2);
   if (n1<n2)
      then writeln(´numeros ordenados: ´,n1,´ ´,n2)
      else writeln(´numeros ordenados: ´,n2,´ ´,n1);
end.
```

Exemplo 5: Programa que obtém 3 números diferentes quaisquer e os imprime em ordem crescente. Supondo que os números sejam n1, n2 e n3, existem 6 possibilidades (permutações) de ordenação, conforme segue.

1. n1 < n2 < n3	3. n2 < n1 < n3	5. n3 < n1 < n2
2. n1 < n3 < n2	4. n2 < n3 < n1	6. n3 < n2 < n1

Assim, o problema pode ser resolvido com uma seqüência de 6 comandos `if`s, conforme o programa que segue.

```
program ordena_tres;
var  n1,n2,n3:integer;
begin
   write(´Digite o primeiro numero => ´);
   readln(n1);
   write(´Digite o segundo numero => ´);
   readln(n2);
   write(´Digite o terceiro numero => ´);
   readln(n3);
   if (n1<n2)and(n2<n3)
       then writeln(´numeros ordenados: ´,n1,´ ´,n2,´ ´,n3);
   if (n1<n3)and(n3<n2)
       then writeln(´numeros ordenados: ´,n1,´ ´,n3,´ ´,n2);
   if (n2<n1)and(n1<n3)
       then writeln(´numeros ordenados: ´,n2,´ ´,n1,´ ´,n3);
   if (n2<n3)and(n3<n1)
       then writeln(´numeros ordenados: ´,n2,´ ´,n3,´ ´,n1);
   if (n3<n1)and(n1<n2)
       then writeln(´numeros ordenados: ´,n3,´ ´,n1,´ ´,n2);
   if (n3<n2)and(n2<n1)
       then writeln(´numeros ordenados: ´,n3,´ ´,n2,´ ´,n1);
end.
```

Importante! Convém ressaltar que, embora somente um dos comandos `then` será executado, todos os `if`s serão testados, consumindo tempo desnecessário de processamento. Uma ineficiência aparente com esta solução é a repetição de operações relacionais (< ou >). Uma observação minuciosa nos permite notar que cada relação ocorre duas vezes. Isto pode ser resolvido com aninhamento de `if`s.

Exemplo 6: Com relação ao exercício do exemplo anterior, todas as permutações podem ser colocadas em uma árvore hierárquica de decisão, conforme mostra a figura 6. Nesta figura cada nó representa um conjunto de variáveis que se relacionam em uma expressão lógica comparativa. Os arcos representam o

tipo do relacionamento e as caixas representam a escrita ordenada disparada no vídeo, a qual depende do caminho percorrido pelo programa. Obviamente, o caminho percorrido depende dos valores de entrada n1, n2 e n3.

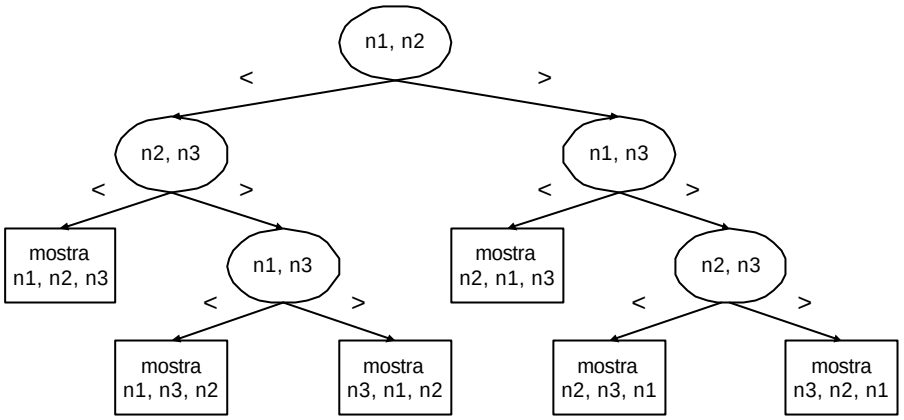


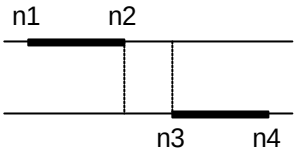
Figura 6 – Árvore Hierárquica de Decisão

A idéia desta árvore é comparar os 3 elementos dois a dois, aninhando os `if`s e eliminando a repetição de operações relacionais. Utilizando a árvore de decisão acima, o programa anterior pode ser melhorado pela redução do número de passos de processamento. Para isto, o programa deverá começar pela avaliação da relação mais alta da hierarquia. Dependendo do resultado desta avaliação, um entre os dois caminhos abaixo será seguido. Com isso, o número de passos restantes é dividido pela metade. Em cada um destes dois caminhos uma nova relação será avaliada e da mesma forma um novo caminho será selecionado e assim sucessivamente até que opção de ordenação correta seja alcançada. O programa a seguir resolve o problema de ordenação de 3 números usando o raciocínio da árvore hierárquica de decisão acima.

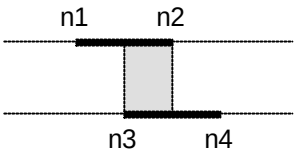
```
program ordena_tres;
var  n1,n2,n3:integer;
begin
  write(´Digite n1, n2, n3 => ´);
  read(n1,n2,n3);
  if (n1<n2)
    then begin
           if (n2<n3)
             then write(n1,n2,n3)
             else if (n1<n3)
                     then write(n1,n3,n2)
                     else write(n3,n1,n2)
         end
    else begin
           if (n1<n3)
             then write(n2,n1,n3)
             else if (n2<n3)
                     then write(n2,n3,n1)
                     else write(n3,n2,n1)
         end;
end.
```

Exemplo 7: Programa que encontra a intersecção entre duas sim-retas [n1,n2] e [n3,n4]. Neste exemplo supomos que n1<n2, n3<n4 e que todos são diferentes entre si. Temos então, várias possibilidades de intersecção:

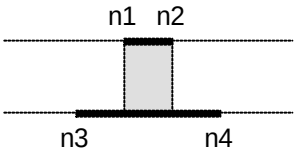
1° Caso: n2<n3 → Intersecção = Vazia



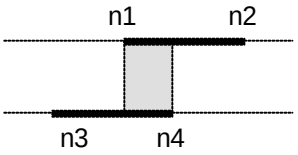
2° Caso: n1<n3<n2<n4 → Intersecção = [n3,n2]



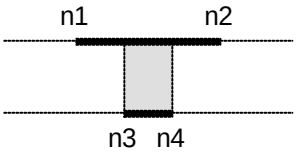
3° Caso: (n3<n1) e (n2<n4) → Intersecção = [n1,n2]



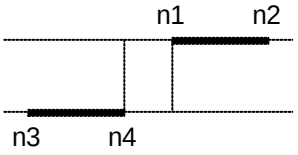
4° Caso: n3<n1<n4<n2 → Intersecção = [n1,n4]



5° Caso: (n1<n3) e (n4<n2) → Intersecção = [n3,n4]



6° Caso: n4<n1 → Intersecção = Vazia



Este exemplo é bastante útil para percebermos que a questão mais importante no desenvolvimento de um programa é o entendimento completo do problema. O desenvolvimento de um programa sem o prévio conhecimento das possibilidades de tratamento do problema não passará de uma tentativa frustrada ou no máximo de uma implementação. Então, podemos concluir que a dificuldade maior na programação está em conhecer previamente qual a solução para resolver o problema que queremos. Esta idéia é a essência de

um algoritmo, enquanto que a implementação em si é mero “esforço braçal”. Olhando as possibilidades acima, o programa que soluciona o problema em questão segue abaixo.

```
program acha_interseccao;
var n1,n2,n3,n4:integer;
begin
  write(´Digite o Primeiro Intervalo [n1,n2] com n1 < n2 => ´);
  readln(n1,n2);
  write(´Digite o Segundo Intervalo [n3,n4] com n3 < n4 => ´);
  readln(n3,n4);
  if (n2<n3)or(n4<n1)
     then writeln(´Interseccao Vazia!´)
     else begin
       if (n1<n3)and(n2<n4) {nao precisa ver se n3<n2 – não eh vazia}
         then writeln(´Interseccao = [ ´,n3,´,´,n2,´ ]´)
         else begin
           if (n3<n1)and(n2<n4)
             then writeln(´Interseccao = [ ´,n1,´,´,n2,´ ]´)
             else if (n3<n1)and(n4<n2) {tambem nao precisa ver se n1<n4}
               then writeln(´Interseccao = [ ´,n1,´,´,n4,´ ]´)
               else writeln(´Interseccao = [ ´,n3,´,´,n4,´ ]´);
         end;
     end;
end.
```

Exemplo 8: Programa que resolve o esquema de contratação de uma empresa segundo a árvore hierárquica de decisão da figura 7.

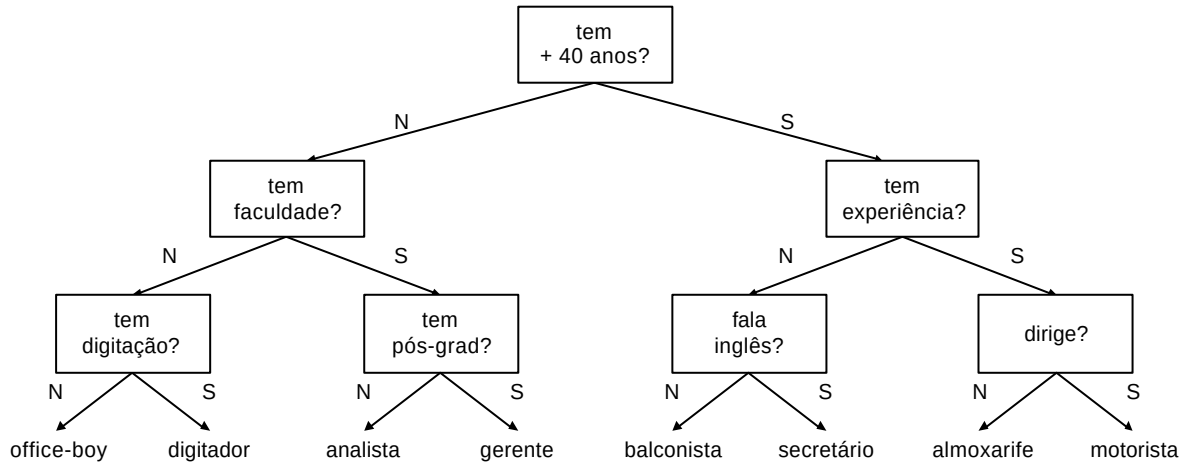


Figura 7 – Árvore Hierárquica de Decisão Empresarial

```
program contrata;
var  resp:char;
begin
  write(´Voce tem mais de 40 anos? <s> ou <n>? ´);
  read(resp);
  if (resp=´s´)
     then begin
         write(´Voce tem experiência? <s> ou <n>? ´);
         if (resp=´s´)
           then begin
```

```
              write(´Voce dirige? <s> ou <n>? ´);
              if (resp=´s´)
                 then writeln(´Voce sera motorista.´)
                 else writeln(´Voce sera almoxarife.´)
           end
           else begin
              write(´Voce fala inglês? <s> ou <n>? ´);
              if (resp=´s´)
                 then writeln(´Voce sera secretário.´)
                 else writeln(´Voce sera balconista.´)
           end
      end
      else begin
          write(´Voce tem faculdade? <s> ou <n>? ´);
          if (resp=´s´)
             then begin
               write(´Voce tem pos-graduação? <s> ou <n>? ´);
               if (resp=´s´)
                  then writeln(´Voce sera gerente.´)
                  else writeln(´Voce sera analista.´)
             end
             else begin
               write(´Voce tem digitação? <s> ou <n>? ´);
               if (resp=´s´)
                  then writeln(´Voce sera digitador.´)
                  else writeln(´Voce sera office-boy.´);
             end;
      end;
end.
```

Exemplo 9: Programa que lê as 4 notas bimestrais de um aluno e diz em quantos bimestres ele ficou abaixo da média de aprovação (6). No final, informa sua situação: aprovado ou reprovado.

```
program avalia_escola;
var  n1,n2,n3,n4,media:real;
     quant:integer;
     nome:string[64];
begin
   quant:=0; {inicializa contador}
   write(´Qual eh o seu nome? ´);
   readln(nome);
   write(´Qual a nota do primeiro bimestre? ´);
   readln(n1);
   if (n1<6)
     then quant:=quant+1; {incrementa contador em 1}
   write(´Qual a nota do segundo bimestre? ´);
   readln(n2);
   if (n2<6)
     then quant:=quant+1; {incrementa contador em 1}
   write(´Qual a nota do terceiro bimestre? ´);
   readln(n3);
   if (n3<6)
     then quant:=quant+1; {incrementa contador em 1}
   write(´Qual a nota do quarto bimestre? ´);
   readln(n4);
   if (n4<6)
     then quant:=quant+1; {incrementa contador em 1}
   media:=(n1+n2+n3+n4)/4;
```

```
  writeln(nome,´, voce teve ´,quant,´ notas abaixo da media´);
  if (media>=6)
    then writeln(nome,´, voce foi aprovado com media = ´,media)
    else writeln(nome,´, voce foi reprovado com media = ´,media);
end.
```

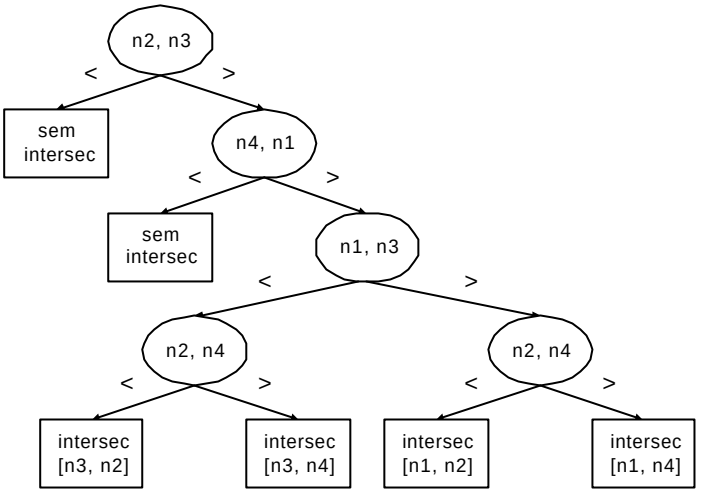
Importante! Este exemplo introduz uma técnica muito importante para efetuar contagens: o uso da variável "`contador`". Você deve inicializá-la com 0 e cada vez que determinada condição for satisfeita ela deve ser incrementada em uma unidade. Este incremento é feito fazendo-se com que a variável `contador` receba seu próprio valor acrescido de 1. Note que, neste exemplo, existem quatro possibilidades de ocorrência de notas menores que a média. Quaisquer possibilidades, independente da ordem entre elas, se satisfeitas, serão consideradas no cômputo da variável "`contador`".

Exemplo 10: Programa que lê 2 horários no formato ´hora:min:seg´ e calcula a diferença entre eles. O tempo resultante é mostrado no formato ´hora:min:seg´. Para efetuar este cálculo, o programa usa a seguinte técnica: converte os horários em segundos, calcula a diferença entre eles e depois converte esta diferença no formato (hora, min, seg) novamente.

```
program dif_horarios;
var    hora1,min1,seg1,hora2,min2,seg2,hora,min,seg:integer;
       total1,total2,dif:real;
begin
  write(´Digite o primeiro horario => ´);
  readln(hora1,min1,seg1);
  write(´Digite o segundo horario => ´);
  readln(hora2,min2,seg2);
  total1:=hora1*360+min1*60+seg1;
  total2:=hora2*360+min2*60+seg2;
  if (total1>total2)
    then dif:=total1-total2
    else dif:=total2-total1;
  hora:=(dif div 360);
  min:=(dif mod 360) div 60;
  seg:=(dif mod 360) mod 60;
  writeln(´tempo resultante ´, hora,´:´,min,´:´,seg);
end;
```

Exercícios para Fixação:

1. Refazer o exemplo 7 de forma alternativa. Utilize apenas uma operação simples na condição de cada comando `if`. Note que as operações (N3<N1) e (N2<N4) aparecem duas vezes cada. Faça primeiramente uma árvore hierárquica de decisão e posteriormente o programa.




```
program acha_interseccao;
var n1,n2,n3,n4:integer;
begin
  write(´Digite o Primeiro Intervalo [n1,n2] com n1 < n2 => ´);
  readln(n1,n2);
  write(´Digite o Segundo Intervalo [n3,n4] com n3 < n4 => ´);
  readln(n3,n4);
  if (n2<n3)
    then writeln(´Interseccao Vazia!´)
    else if (n4<n1)
           then writeln(´Interseccao Vazia!´)
           else begin
                  if (n1<n3)
                    then if (n2<n4)
                           then writeln(´Interseccao=[´,n3,´,´,n2,´]´)
                           else writeln(´Interseccao=[´,n3,´,´,n4,´]´)
                    else if (n2<n4)
                           then writeln(´Interseccao=[´,n1,´,´,n2,´]´)
                           else writeln(´Interseccao=[´,n1,´,´,n4,´]´)

           end;
end.
```

2. Fazer programa que leia 2 triângulos quaisquer e diga o número de lados em comum entre eles. O usuário deverá fornecer os tamanhos dos lados de cada triângulo.

Primeira solução: considerando que cada triângulo possui lados diferentes

```
program compara_triang;
var    a1,b1,c1,a2,b2,c2:real;
       nlados:integer;
begin
readln(a1,b1,c1); {supondo que a1<>b1<>c1}
readln(a2,b2,c2); {supondo que a2<>b2<>c2}
nlados:=0;
if (a1=a2)or(a1=b2)or(a1=c2)
   then inc(nlados);
if (b1=a2)or(b1=b2)or(b1=c2)
   then inc(nlados);
if (c1=a2)or(c1=b2)or(c1=c2)
```

```
   then inc(nlados);
writeln(nlados);
end.
```

Segunda solução: considerando que cada triângulo pode ter lados iguais

```
program compara_triang_2;
var    a1,b1,c1,a2,b2,c2:real;
       nlados:integer;
begin
readln(a1,b1,c1);
readln(a2,b2,c2);
if  [(a1=a2)and(b1=b2)and(c1=c2)]or
    [(a1=a2)and(b1=c2)and(c1=b2)]or
    [(a1=b2)and(b1=a2)and(c1=c2)]or
    [(a1=b2)and(b1=c2)and(c1=a2)]or
    [(a1=c2)and(b1=a2)and(c1=b2)]or
    [(a1=c2)and(b1=b2)and(c1=a2)]
    then nlados:=3
    else if [(a1=a2)and(b1=b2)]or[(a1=a2)and(b1=c2)]or
            [(a1=b2)and(b1=a2)]or[(a1=b2)and(b1=c2)]or
            [(a1=c2)and(b1=a2)]or[(a1=c2)and(b1=b2)]or

            [(a1=a2)and(c1=b2)]or[(a1=a2)and(c1=c2)]or
            [(a1=b2)and(c1=a2)]or[(a1=b2)and(c1=c2)]or
            [(a1=c2)and(c1=a2)]or[(a1=c2)and(c1=b2)]or

            [(b1=a2)and(c1=b2)]or[(b1=a2)and(c1=c2)]or
            [(b1=b2)and(c1=a2)]or[(b1=b2)and(c1=c2)]or
            [(b1=c2)and(c1=a2)]or[(b1=c2)and(c1=b2)]

            then nlados:=2
            else if (a1=a2)or(a1=b2)or(a1=c2)or
                    (b1=a2)or(b1=b2)or(b1=c2)or
                    (c1=a2)or(c1=b2)or(c1=c2)
                    then nlados:=1
                    else nlados:=0
writeln(nlados);
end.
```

3. Fazer um programa que leia uma coordenada (x,y) e posteriormente leia uma tecla com um caractere de direcionamento. Se a tecla for “e” a coordenada deve subir. Se for “x” deve descer. Se for “s” deve ir para a esquerda. Se for “d” deve ir para a direita.

```
program trata_coordenada;
var  x,y:integer;
     tecla:char;
begin
   readln(x,y);
   readln(tecla);
   if (tecla = ´e´)
    then inc(y)
    else if (tecla = ´x´)
            then dec(y)
            else if (tecla = ´s´)
                    then dec(x)
                    else inc(x);
```

```
  writeln(x,y);
end.
```

4. Melhore o programa anterior colocando um comando de repetição que permita ao programa continuamente ler uma nova tecla de direcionamento e assim mudar a posição x,y. Além disso, use o comando `gotoxy` (que posiciona o cursor na posição x,y da tela) para realmente imprimir na tela, em cada nova posição de x e y, a letra O. Entretanto, sempre que uma nova posição for calculada, o programa deverá apagar a letra até então escrita para depois escreve-la na nova posição. Também, para que o programa não pare a cada nova interação para esperar que uma nova tecla de direcionamento seja digitada, use o comando `keypressed`, o qual retorna `true` quando uma tecla é pressionada. Quando o programa detectar que uma tecla foi pressionada ele deverá obtê-la usando o comando `readkey`. O programa deve parar quando a tecla ´z´ for digitada. **OBS**: normalmente a tela possui 80 colunas e 24 linhas onde a coordenada (0,0) é posicionada no canto superior esquerdo. Considere (40,20) como posição inicial do cursor.

```
program trata_coordenada_2;
var  x,y:integer;
     tecla:char;
begin
  x:=40;
  y:=20;
  tecla:=´s´;
  while (tecla <> ´z´) do
  begin
    gotoxy(x,y);
    write(´O´);
    gotoxy(x,y);
    if (keypressed)
        then begin
               write(´ ´);
               tecla:=readkey;
               case tecla of
                  ´s´: dec(x);
                  ´d´: inc(x);
                  ´e´: dec(y);
                  ´x´: inc(y);
               end;
             end;
  end;
end.
```

5. Faça um programa que faça 5 perguntas para uma pessoa sobre um crime. As perguntas são “telefonou para a vitima”, “esteve no local”, “mora perto”, “devia para a vítima” e se “já trabalhou com a vítima”. O programa deve no final emitir uma classificação sobre a participação da pessoa no crime. Se a pessoa responder positivamente a 2 questões ela deve ser classificada como “suspeita”, entre 3 e 4 como “cúmplice” e 5 como “assassino”. Caso contrário, ele será classificado como “inocente”.

```
program detecta_crime;
var    resp:char;
       grau:integer;
begin
  grau:=0;
  writeln(´Voce telefonou para a vitima <s>im ou <n>ao?´);
```

```
  readln(resp);
  if resp=´s´
    then inc(grau);
  writeln(´Voce esteve no local da vitima <s>im ou <n>ao?´);
  readln(resp);
  if resp=´s´
    then inc(grau);
  writeln(´Voce mora perto da vitima <s>im ou <n>ao?´);
  readln(resp);
  if resp=´s´
    then inc(grau);
  writeln(´Voce devia para a vitima <s>im ou <n>ao?´);
  readln(resp);
  if resp=´s´
    then inc(grau);
  writeln(´Voce ja trabalhou com a vitima <s>im ou <n>ao?´);
  readln(resp);
  if resp=´s´
    then inc(grau);
  if (grau=1)
    then writeln(´Inocente´)
    else if (grau=2)
           then writeln(´Suspeita´)
           else if (grau=5)
                  then writeln(´Assassino´)
                  else writeln(´Cumplice´);
end.
```

3.3.5 Comando de Decisão `Case`

Este comando é estruturado e funciona de forma equivalente a uma seqüência de comandos `if`s aninhados, de forma que somente uma condição será satisfeita. Sua forma geral é mostrada abaixo.

```
Case <variavel> of
  <valor_1>:  begin
               <seq_comandos_1>
              end;
  <valor_2>:  begin
               <seq_comandos_2>
              end;
      :           :
  <valor_n>:  begin
               <seq_comandos_n>
              end;
  else begin  {*opcional*}
         <outros_comandos>
       end;
end;
```

Na estrutura acima, os valores `<valor_1>`, `<valor_2>` ... `<valor_n>` devem ser constantes do tipo `integer` ou `char`, e do mesmo tipo da variável `<variavel>`. A variável `<variavel>` será comparada com estes valores de acordo com a ordem de ocorrência. Quando ocorrer a primeira igualdade, a seqüência de comandos correspondentes será executada. Assim, se o valor da variável `<variavel>` for igual ao valor `<valor_k>`, a seqüência de comandos `<seq_comandos_k>`

será executada. Após a execução de uma seqüência, o restante do `case` será saltado e o próximo comando após o `case` será executado. Caso a seqüência contenha um único comando, o `begin/end` pode ser omitido. Se nenhum dos valores for igualado, a seqüência de comandos `<outros_comandos>` será executada independente do valor da variável (isto se houver a opção de comando `else`).

Exemplo 1: Programa que calcula o reajuste salarial em função da profissão e mostra no vídeo o resultado final, de acordo com a seguinte regra: Técnico => 50% de reajuste, Engenheiro => 30% de reajuste e Outros => 10% de reajuste.

```
program reajuste_salarial;
var  salario:real;
     profissao:char;
begin
   write(´Entre com a sua profissao e salário => ´);
   readln(profissao,salario);
   case profissao of
      ´T´: salario:=1.5*salario;
      ´E´: salario:=1.3*salario;
      else salario:=1.1*salario;
   end;
   writeln(´Salario Reajustado = ´,salario);
end.
```

No exemplo acima, o usuário pode não entender o que de fato precisa ser digitado e informar incorretamente a profissão desejada, fazendo que o cálculo seja efetuado de maneira incorreta. Para facilitar o usuário, o programa pode fazer uso da técnica de ´Menu de Opções´, conforme mostra o exemplo a seguir. Neste exemplo, o programa dispara na tela de vídeo informações sobre qual tecla pressionar para que o usuário saiba com clareza como agir.

```
program reajuste_salarial_2;
var    salario:real;
       opcao:integer;
begin
   writeln(´Menu de Opcoes´);
   writeln(´<1> Tecnico´);
   writeln(´<2> Engenheiro´);
   writeln(´<3> Outros´);
   write(´Digite a opcao => ´);
   readln(opcao);
   write(´Entre com o salario => ´);
   readln(salario);
   Case opcao of
      1 : salario:=1.5*salario;
      2 : salario:=1.3*salario;
      else salario:=1.1*salario;
   end;
   writeln(´Salario Reajustado = ´, salario);
end.
```

3.3.6 Comando de Repetição `For`

Este comando também é estruturado. Ele permite que um trecho do programa seja executado várias vezes (*loop* ou laço). Sua forma geral compõe-se de uma variável de contagem, um início e um fim, que podem ser constantes, variáveis ou expressões, além da seqüência de comandos entre o **`begin/end`**, conforme mostrado abaixo.

```
for <variavel>:=<inicio> to <fim> do
begin
   <sequencia_comandos>;
end;
```

A seqüência de comandos é executada n vezes, quando n > 0, onde n é igual a "`<fim>` - `<inicio>` + 1". A cada execução da seqüência de comandos dize-se ter ocorrido uma iteração do **`for`**. Seu funcionamento é o seguinte. Quando o comando **`for`** é encontrado, a variável de contagem recebe o valor `<inicio>`. Então, seu valor é comparado com o valor `<fim>`. Se ele é menor ou igual, a seqüência de comandos é executada. Quando é concluída a execução destes comandos, a variável de contagem é incrementada de uma unidade e uma nova comparação com o valor `<fim>` é feita. Se é novamente menor ou igual, os seqüência de comandos será executada novamente e assim por diante, até que o valor da variável de contagem seja maior que o valor `<fim>`, ou seja, até que n = 0. Se de imediato, na primeira vez que o comando `for` é encontrado, n = 0, o comando `for` é desviado e o próximo comando após o `for` é executado. Normalmente, a variável de contagem é do tipo `integer` e os valores `<inicio>` e `<fim>` são constantes ou variáveis do tipo `integer`. Se a seqüência de comandos tiver apenas um comando, o `begin/end` pode ser omitido. A figura 8 mostra o esquema gráfico representativo deste comando.

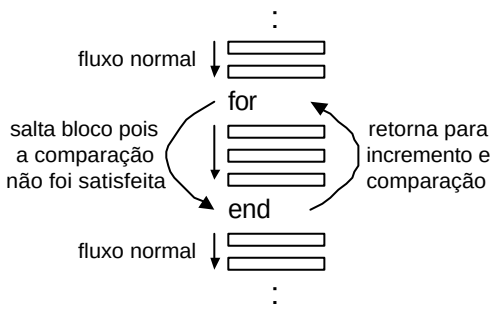


Figura 8 – Esquema Gráfico Representativo do `For`

Exemplo 1: Programa que lê um valor inteiro qualquer n e imprime todos os números de 1 até n. A idéia aqui é utilizar um comando **`for`** de 1 até n e mostrar no vídeo os valores da variável de contagem em cada iteração do laço.

```
program lista_numeros;
var n,i:integer;
begin
   write(´Digite o valor de n => ´);
   readln(n);
   for i:=1 to n do
      write(i,´ ´);
   writeln;
end.
```

Exemplo 2: Programa que lê n números reais e imprime a somatória entre eles.

```
program somatoria;
var    n,i:integer;
       valor,soma:real;
begin
  write(´Digite o valor de n => ´);
  readln(n);
  soma:=0;
  for i:=1 to n do
    begin
      write(´Digite um valor ´);
      readln(valor);
      soma:=soma+valor;
    end;
  writeln(´Somatoria = ´,soma);
end.
```

Neste exemplo, a utilização da técnica de somatória, a qual é similar aquela usada pela variável `contador`, mostrada anteriormente, permite o acumulo de valores. Uma variável `soma` é inicializada com 0 e a cada iteração do laço ela recebe seu próprio valor acrescido de outro valor digitado. Este modelo de acumulador pode ser estendido para acumular valores sempre que determinada situação ocorrer, tal como aquela mostrada no próximo exemplo.

Exemplo 3: Programa que verifica em uma sala de 30 alunos, quantos foram aprovados e quantos foram reprovados no ano, a partir das 4 notas bimestrais de cada um.

```
program controle_academico;
var n1,n2,n3,n4,media:real;
i,nro_aprov,nro_reprov:integer;
begin
  nro_aprov:=0;
  nro_reprov:=0;
  for i:=1 to 30 do
  begin
    write(´Entre com as 4 notas bimestrais => ´);
    readln(n1,n2,n3,n4);
    media:=(n1+n2+n3+n4)/4;
    if(media<6)
      then nro_reprov:=nro_reprov+1
      else nro_aprov:=nro_aprov+1;
  end;
  writeln(´Numero de Aprovados = ´,nro_aprov);
  writeln(´Numero de Reprovados = ´,nro_reprov);
end.
```

Exemplo 4: Programa que calcula n! (fatorial de n), onde n! = n*(n-1)*(n-2)*.....*1

```
program fatorial;
var  n,i:integer;
       fat:real;
begin
```

```
  write(´Digite o valor de n => ´);
  readln(n);
  if (n<0)
     then writeln(´Nao Existe Fatorial de Numero Negativo´)
     else begin
       fat:=1;
       for i:= 2 to n do
         fat:=fat*i;
       writeln(´Fatorial de ´,n,´ = ´,fat);
     end;
end.
```

Exemplo 5: Programa que imprime todos os n primeiros elementos da seqüência de Fibonacci. De uma forma geral, cada elemento desta seqüência é igual a soma dos dois elementos anteriores, com exceção dos dois primeiros elementos que são iguais a 1. Formalmente, f(1)=1; f(2)=1 e f(n) = f(n-1)+f(n-2), o qual é o n-ésimo elemento da seqüência. Assim, tem-se a seguinte seqüência 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ...

```
program fibonacci;
var  i,n:integer;
     fn,fn_1,fn_2:real;
begin
  write(´Digite o numero de elementos => ´);
  readln(n);
  fn:=1;
  fn_1:=1;
  fn_2:=1;
  if (n>=1)
     then begin
       write(1);
       if (n>=2)
         then write(´, ´,1);
       for i:=3 to n do
       begin
         fn:=fn_1+fn_2;
         fn_2:=fn_1;
         fn_1:=fn;
         write(´, ´,fn);
       end;
     end
     else writeln(´Numero invalido´);
end.
```

Exemplo 6: Programa que simula um relógio com hora, minuto e segundo, usando um aninhamento de comandos `for` (3 comandos `for`, um interno ao outro). O relógio simula apenas um dia.

```
program relogio;
uses crt;
var hora,mim,seg:integer;
begin
  for hora:=1 to 24 do
    for min:=1 to 60 do
      for seg:=1 to 60 do
      begin
        clrscr;
```

```
          write(hora,´:´,min,´:´,seg);
          delay(1000);
        end;
end;
```

Neste exemplo, para cada iteração do comando `for` mais externo (das horas), o comando `for` intermediário (dos minutos) executa 60 iterações. Da mesma forma, para cada iteração do comando `for` intermediário, o comando `for` mais interno (dos segundos) executa 60 iterações. Assim, as horas são mostradas 86400 vezes (24*60*60), dentro do `begin/end` do comando `for` mais interno, sendo atualizada a cada vez. Neste exemplo, introduzimos o uso da biblioteca `crt`, a qual permite o uso de comandos `clrscr` e `delay`, que não existem na linguagem Pascal padrão.

O comando `clrscr` limpa toda a tela vídeo, permitindo que as horas sejam disparadas sobrepostas sempre na mesma posição, simulando um relógio digital em constante modificação. O comando `delay` simula uma parada no processamento durante a sua execução. O valor passado como parâmetro do `delay` determina o tempo de parada, mas varia de máquina para máquina, devendo ser testado na prática. Este comando é usado para forçar uma parada após a escrita das horas, cada vez que ela é atualizada, para permitir a visualização do relógio, caso contrário, o relógio andaria rápido o suficiente para não podermos perceber as mudanças de horário. Para uma simulação mais próxima da real, o `delay` deveria ser configurado para parar o processamento cerca de 1 segundo.

Importante! De uma forma geral, podemos dizer que o comando `for` permite que o fluxo de execução retorne de uma posição à frente do código a uma posição atrás do código, sempre que determinada condição seja satisfeita. Este comando é, em essência, um comando de salto condicional das instruções do programa. Da mesma forma que o comando `for`, os comandos `repeat` e `while` também permitem o retorno do fluxo de execução, cada qual a sua maneira, sendo os mesmos explicados a seguir.

3.3.7 Comando de Repetição `Repeat`

Este também é um comando de repetição. Enquanto o comando `for` executa um número fixo de vezes, o comando `repeat` executa tantas vezes for necessário até que uma condição seja satisfeita. Sua forma geral é mostrada abaixo:

```
repeat
  <sequencia_comandos>;
until  <condição>;
```

Os comandos `<sequencia_comandos>` serão executados pelo menos uma vez e a repetição será interrompida caso a condição `<condicao>` seja verdadeira.

Exemplo 1: Laço infinito. Neste exemplo, a condição de parada jamais será satisfeita, pois ela contém uma expressão simples cujo valor é um constante `false`. Jamais será `true`. Neste caso, o programa entra em *loop* e somente poderá ser interrompido mediante intervenção do usuário. Em sistemas computacionais mais antigos e/ou mais simples, esta situação somente pode ser resolvida com o desligamento do equipamento. Cuidado!!!

```
repeat
  writeln(´Nunca Paro!!! Estou em loop!!!´);
until (false);
```

Exemplo 2: Programa que força o usuário a entrar sempre com números maiores que zero.

```
program leitura_forcada;
var n:integer;
begin
  repeat
    write(´Digite um numero maior que zero => ´ );
    readln(n);
  until (n>0);
end.
```

Importante! Este exemplo, apesar de simples, introduz uma técnica bastante útil quando queremos forçar o usuário a entrar com dados sob certas restrições. É comum ver usuários tentando teclar valores fora do padrão solicitado pelo programa, simplesmente pelo prazer de ver o programa "dar pau". Um programa seguro deve monitorar as entradas do usuário. Suponha, por exemplo, que um programa solicita a idade do usuário, usando uma variável do tipo `integer` para ler o valor digitado por ele, por entender que as idades sejam números inteiros pequenos. Entretanto, imagina o que acontece se o usuário digitar um número real, com casas decimais, ou mesmo um conjunto de letras ou caracteres quaisquer. O programa simplesmente será "abortado" pelo sistema operacional.

Exemplo 3: Programa que calcula a média anual de uma turma de alunos. O programa lê continuamente as médias anuais individuais de cada aluno até que seja digitada uma média negativa (condição de parada, que deve ser desconsiderada no cálculo da média final). A média final e o número de alunos são mostrados no vídeo.

```
program media_turma;
var  nro_alunos:integer;
     somatoria,media_indiv,media_final:real;
begin
  nro_alunos:=0;
  somatoria:=0;
  repeat
    write(´Entre com uma média individual => ´);
    readln(media_indiv);
    if media_indiv>=0
       then begin
         nro_alunos:=nro_alunos+1;
         somatoria:=somatoria+media_indiv;
       end;
  until (media_indiv<0);
  media_final:=somatoria/nro_alunos;
  writeln(´Sao ´,nro_alunos,´ alunos. A media da turma eh ´,media_final);
end.
```

Importante! Neste exemplo, podemos perceber um problema típico aplicado comumente entre os programadores despreocupados com a eficiência dos programas. Observe que existe um comando `if`

dentro do comando `repeat`. Este comando foi inserido para desconsiderar a média negativa no calculo da média final. A cada iteração do comando `repeat`, o comando `if` perde tempo em verificar a condição `media_indiv>=0`, que somente será verdadeira na última iteração. Obviamente, este exemplo é simples o suficiente para não ter sérias implicações no desempenho. Mas, este uso ineficiente se for aplicado em programas com milhares de iterações poderá implicar em uma execução perceptivelmente mais lenta. Uma possível solução é mostrada no exemplo a seguir.

Exemplo 4: Programa que elimina a ineficiência do exemplo anterior.

```
program media_turma_2;
var  nro_alunos:integer;
     somatoria,media_indiv,media_final:real;
begin
   nro_alunos:=0;
   somatoria:=0;
   repeat
     write(´Entre com uma média individual => ´);
     readln(media_indiv);
     nro_alunos:=nro_alunos+1;
     somatoria:=somatoria+media_indiv;
   until (media_indiv<0);
   somatoria:=somatoria-media_indiv;
   nro_alunos:=nro_alunos-1;
   media_final:=somatoria/nro_alunos;
   writeln(´Sao ´,nro_alunos,´ alunos. A media da turma eh ´,media_final);
end.
```

Neste exemplo, nenhuma restrição é aplicada a cada média individual que está sendo lida. Todas elas são consideradas no cálculo da média final, até mesmo a média negativa. Entretanto, como o laço pára somente quando uma média negativa for digitada, o programa simplesmente subtrai o último valor somado e diminui de 1 o contador do número de alunos. Resumidamente, a condição que antes era checada a cada iteração foi substituída pela execução única de apenas dois comandos simples de subtração. Ganho na certa.

Observe no exemplo anterior que a leitura das médias individuais não restringe valores negativos ou fora do alcance de validade (por exemplo, de 0 a 10). O esquema de leitura forçada pode ser aplicado aqui para resolver esta questão, tal como o exemplo seguinte.

Exemplo 5: Programa que restringe a entrada de valores do exemplo anterior.

```
program media_turma_3;
var    nro_alunos:integer;
       somatoria,media_indiv,media_final:real;
       resp:char;
begin
   nro_alunos:=0;
   somatoria:=0;
   repeat
     repeat
       write(´Entre com uma média individual => ´);
       readln(media_indiv);
     until (media_indiv>=0)and(media_indiv<=10);
     nro_alunos:=nro_alunos+1;
```

```
     somatoria:=somatoria+media_indiv;
     write(´Deseja continuar <s>im ou <n>ao ? => ´);
     readln(resp);
   until (resp=´n´);
   media_final:=somatoria/nro_alunos;
   writeln(´Sao ´,nro_alunos,´ alunos. A media da turma eh ´,media_final);
end.
```

Exemplo 6: Programa que calcula a média aritmética de todos os números pares que são fornecidos pelo usuário. O usuário pode fornecer tanto números pares quanto ímpares. A condição de parada será a leitura do número 0 (zero) .

```
program media_pares;
var  quant,valor:integer;
     somatoria,media_final:real;
begin
   quant:=0;
   somatoria:=0;
   repeat
     write(´Entre com um número, par ou ímpar => ´);
     readln(valor);
     if (valor mod 2)= 0
        then begin
          quant:=quant+1;
          somatoria:=somatoria+valor;
        end;
   until (valor=0);
   quant:=quant-1;
   media_final:=somatoria/quant;
   writeln(´Foram digitados ´,quant,´ numeros pares.´);
   writeln(´A soma entre eles = ´,media_final);
end.
```

Neste exemplo, o comando `mod` foi usado. Ele retorna o resto da divisão inteira de um número pelo outro. Quando aplicamos N `mod` 2, se o resto da divisão inteira é 0, tem-se que N é par. Entretanto, observe que a condição de parada (valor 0 digitado) também é considerada um número par, pois seu resto também é zero, e por isso é considerada no cálculo da somatória. Este fato não altera o valor da somatória, mas a quantidade de números armazenada na variável `quant` fica com 1 a mais. Assim, após o `repeat`, a variável `quant` é diminuída de 1 unidade. Note também que neste caso é impossível remover o comando `if` de dentro do comando `repeat`, pois ele é executado um número indeterminado de vezes e a condição por ele avaliada não é a mesma que força a parada do comando `repeat`.

Exemplo 7: Programa que simula o relógio digital usando aninhamento de comandos `repeat`.

```
program relogio_2;
uses crt;
var   hora,minuto,segundo:integer;
begin
   repeat
     hora:=0;
     repeat
       minuto:= 0;
       repeat
          segundo:=0;
```

```
         repeat
           clrscr;
           write(hora,´:´,minuto,´:´,segundo);
           delay(1000);
           segundo:=segundo+1;
         until (segundo=60);
         minuto:=minuto+1;
       until (minuto=60);
       hora:=hora+1;
     until (hora=24);
  until (false);
end.
```

Observe que neste exemplo, o comando `repeat` mais externo é um laço infinito, o qual não permite que o relógio pare após marcar o dia todo. Este seria um exemplo simples daquilo que é implementado nos relógios digitais verdadeiros. O comando `repeat` mais interno cuida dos segundos, o comando `repeat` intermediário cuida dos minutos e o `repeat` externo a este cuida das horas. Note que, antes de cada `repeat`, a variável associada a ele é inicializada com 0, para que a contagem das horas ou minutos ou segundos seja feita dentro do intervalo desejado.

Exemplo 8: Programa que simula o relógio digital usando somente um comando `repeat`.

```
program relogio_3;
uses crt;
var  hora,minuto,segundo:integer;
begin
  segundo:=0;
  minuto:= 0;
  hora:=0;
  Repeat
    clrscr;
    write(hora,´:´,minuto,´:´,segundo);
    segundo:=segundo+1;
    if segundo=60
      then begin
        segundo:=0;
        minuto:=minuto+1;
        if minuto=60
          then begin
            minuto:=0;
            hora:=hora+1;
            if hora=24
              then hora:=0;
          end;
      end;
  until (false);
end.
```

Neste exemplo, somente um comando `repeat` executa o retorno do fluxo de controle. Com isso, o incremento de cada variável `hora`, `minuto` ou `segundo` teve que ser controlado por um comando condicional `if`. Somente após os segundos atingirem sessenta, o minuto pode ser incrementado. Somente após os minutos atingirem sessenta, a hora pode ser incrementada. Somente após as horas atingirem 24, a contagem pode ser recomeçada.

3.3.8 Comando de Repetição `while`

Este comando é equivalente ao comando `repeat`, diferenciando na posição da verificação da condição. Enquanto no comando `repeat` a condição é testada no final, no comando `while` a condição é testada no começo. Sua forma geral é mostrada abaixo.

```
while <condicao> do
begin
   <sequencia_comandos>;
end;
```

No comando `repeat`, os comandos internos ao laço são executados até que a condição seja verdadeira, ou seja, enquanto a condição for falsa. No comando `while`, os comandos internos ao laço serão executados enquanto a condição for verdadeira, ou seja, até que a condição seja falsa.

Exemplo 1: Programa do relógio usando aninhamento de comandos `while`.

```
program relogio_4;
uses crt;
var   hora,minuto,segundo:integer;
begin
   while (true) do
   begin
      hora:=0;
      while (hora<>24) do
      begin
         minuto:=0;
         while (minuto<>60) do
         begin
            segundo:=0;
            while (segundo<>60) do
            begin
               clrscr;
               write(hora,´:´,minuto,´:´,segundo);
               delay(1000);
               segundo:=segundo+1;
            end;
            minuto:=minuto+1;
         end;
         hora:=hora+1;
      end;
   end;
end.
```

Observe que este programa e aquele que implementa o `relogio_2` são extremamente parecidos. A diferença está simplesmente no uso dos comandos `repeat` e `while`, utilizados por um e pelo outro, respectivamente. Agora, olhe mais atentamente para os dois programas. O que podemos concluir com relação às condições de verificação dos comandos `repeat` e `while`? Uma é exatamente oposta à outra, certo?

Importante! De uma forma geral, todo comando `repeat` pode ser substituído por um comando `while`, e vice-versa, bastando trocar a condição de verificação pela sua oposta. O que determina qual comando devemos usar no programa é basicamente uma questão de gosto pessoal. Entretanto, existem situações que a programação poderá ser ligeiramente facilitada se um ou outro comando for utilizado. Estas situações serão percebidas no decorrer deste livro.

Exemplo 2: Programa que permite fazer um levantamento do estoque de vinhos de uma adega, de acordo com os seus tipos (branco, tinto e rosé). O programa fica continuamente lendo os tipos de vinhos até que um finalizador seja digitado. O finalizador é entendido como sendo a digitação de um tipo não definido. No final, o programa mostra o total de vinhos existentes e a porcentagem de ocorrência de cada tipo de vinho. Observe o uso do comando `inc`, o qual incrementa o valor passado em uma unidade.

```
program controle_estoque;
uses crt;
var  nvt,            {numero de vinho tinto}
     nvb,            {numero de vinho branco}
     nvr,            {numero de vinho rose}
     tipo:integer;   {tipo de vinho (1=tinto, 2=branco, 3=rose)}
     total,          {total de vinhos computados}
     pvt,            {% de vinho tinto sobre o total}
     pvb,            {% de vinho branco sobre o total}
     pvr:real;       {% de vinho rose sobre o total}
     acabou:boolean; {controla a finalizacao do laco}
begin
   nvt:=0;
   nvb:= 0;
   nvr:=0;
   acabou:=false;
   while (not acabou) do
   begin
     clrscr;
     writeln(´Opcoes´);
     writeln(´<1> Vinho Tinto´);
     writeln(´<2> Vinho Branco´);
     writeln(´<3> Vinho Rose´);
     write(´Digite sua opcao => ´);
     readln(tipo);
     case tipo of
       1: inc(nvt);
       2: inc(nvb);
       3: inc(nvr);
       else acabou:=true;
     end;
   end;
   total:=nvt+nvb+nvr;
   pvt:=(nvt*100)/total;
   pvb:=(nvb*100)/total;
   pvr:=(nvr*100)/total;
   writeln(´Total de Vinhos = ´,total);
   writeln(´Total de Vinhos Tintos = ´,pvt,´%´);
   writeln(´Total de Vinhos Brancos = ´,pvb,´%´);
   writeln(´Total de Vinhos Roses = ´,pvr,´%´);
end.
```

Exemplo 3: Programa que lê a altura de 2 pessoas e seus respectivos valores de crescimento anual. O programa deve calcular quantos anos levará para que a menor fique mais alto que a maior.

```
program calcula_crescimento;
uses crt;
var  alt1,alt2,
     cresc1,cresc2,nanos:integer;
begin
  nanos:=0;
  clrscr;
  write(´Digite as alturas 1 e 2 =>´);
  readln(alt1,alt2);
  write(´Digite as medidas de crescimento 1 e 2 =>´);
  readln(cresc1,cresc2);
  if [(alt1>alt2)and(cresc1>cresc2)]or[(alt2>alt1)and(cresc2>cresc1)]
    then writeln(´Nunca o menor alcancarah o maior´)
    else begin
      if (alt1>alt2)
        then while (alt1>alt2) do
               begin
                 alt1:=alt1+cresc1;
                 alt2:=alt2+cresc2;
                 inc(nanos);
               end
        else while (alt2>alt1) do
               begin
                 alt1:=alt1+cresc1;
                 alt2:=alt2+taxa2;
                 inc(nanos);
               end;
      writeln(´Alcancarah em ´,nanos, ´ anos´);
    end;
end.
```

Exemplo 4: Programa que lê um conjunto de valores inteiros e positivos, determinando quais são o maior e o menor valores do conjunto. O final do conjunto de valores deve ser o número 0 (finalizador da entrada), o qual também é um número válido.

```
program calcula_maior_menor;
var maior,menor,nro:integer;
begin
  menor:=32767;
  maior:=-32768;
  nro:=1;
  while (nro<>0) do
  begin
    read(nro);
    if (nro<menor)
      then menor:=nro
      else if (nro>maior)
             then maior:=nro;
  end;
  write(menor,maior);
end.
```

Este exemplo apresenta uma técnica para a localização do maior e menor elementos de um conjunto, usando comparação seguida de substituição. O programa usa duas variáveis, `maior` e `menor`, para armazenar o maior e o menor valores do conjunto, respectivamente. A idéia básica é que cada novo valor digitado seja comparado com os valores maior e menor até então armazenados. Caso este novo valor seja maior que o maior ou menor que o menor, uma substituição deve ser feita. A inicialização das variáveis `maior` e `menor` é de extrema importância. O objetivo é inicializar a variável `menor` com o maior número possível (+infinito seria o ideal), de forma que o mesmo seja substituído imediatamente. Da mesma forma, a variável `maior` deve ser inicializada com o menor número possível (-infinito seria o ideal).

Entretanto, existe certa dificuldade para o programador saber quais seriam os valores limites máximo e mínimo para esta inicialização, os quais dependem do tipo da variável, do compilador e do processador utilizado. Para resolver este problema, uma outra possibilidade de inicialização é utilizar o primeiro número válido do conjunto para ser simultaneamente o maior e o menor elemento, tal como mostra o próximo exemplo.

Exemplo 5: Programa similar ao anterior mas que utiliza o primeiro número lido para inicializar as variáveis `maior` e `menor`.

```
program calcule_maior_menor_2;
var maior,menor,nro:integer;
begin
  read(nro);
  menor:=nro;
  maior:=nro;
  while (nro<>0) do
    begin
      read(nro);
      if (nro<menor)
        then menor:=nro
        else if (nro>maior)
                then maior:=nro;
    end;
  write(menor,maior);
end.
```

Importante! Os programas anteriores estão simplificados para não escreverem mensagens na tela. Esta prática é adotada na seqüência deste livro.

Exemplo 6: Programa que lê continuamente números reais e calcula a soma somente daqueles que são inteiros. A condição de parada deve ser a leitura do número 888. **DICA**: o comando `int` retorna somente a parte inteira de um número real.

```
program soma_inteiros;
var  nro,soma:real;
begin
  soma:=0;
  repeat
    read(nro);
    if (int(nro)=nro)
```

```
       then soma:=soma+nro;
  until (nro=888);
  soma:=soma-888;
  writeln(´soma = ´,soma);
end.
```

Exemplo 7: Programa para calcular os `n` primeiros termos de uma PA de elemento inicial `a0` e razão `r`. O i-ésimo termo é representado por `ai`. O programa força a entrada de um número `n` positivo.

```
program PA;
var  ai,a0,r:real;
     i,n:integer;
begin
  repeat
    read(n);
  until n>0;
  read(a0,r);
  ai:=a0;
  for i:=1 to n do
  begin
    write(ai);
    ai:=ai+r;
  end;
end.
```

Exemplo 8: Programa que calcula a função $f = \frac{[\sum_{i=1}^{n}(x_i + y_i)^2]}{n}$. O programa lê `n` e as variáveis `xi` e `yi`. **OBS**: Esta representação indica uma somatória de n termos (onde i vai de 1 a n) que é dividida por n. Isto representa uma média de n termos. Cada termo é constituído por uma soma (x+y) ao quadrado. Para cada valor de i têm-se valores diferentes de x e y, representados por x_i e y_i.

```
program somatoria;
var  f,soma,termo,xi,yi:real;
     i,n:integer;
begin
  read(n);
  soma:=0;
  for i:=1 to n do
  begin
    read(xi,yi);
    termo:=(xi+yi)*(xi+yi);
    soma:=soma+termo;
  end;
  f:=soma/n;
  write(f);
end.
```

Exemplo 9: Programa que calcula o reajuste salarial de uma empresa que possui `n` funcionários, de acordo com os seguintes critérios: os funcionários com salário inferior a 1000 devem receber 55% de reajuste; os funcionários com salário entre 1000 e 2500 devem receber 30% de reajuste e os funcionários com salário acima de 2500 devem receber 20% de reajuste.

```
program reajuste_salarial;
var  salario:real;
     nfunc,i:integer;
begin
   write(´quantos funcionarios sua empresa tem ?´);
   read(nfunc);
   for i:=1 to nfunc do
   begin
     readln(salario);
     if salário<1000
        then salário:=1.55*salario
        else if salario 2500
                then salario:=1.30*salario
                else salario:=1.2*salario;
     write(´salario reajustado = ´,salario);
   end;
end.
```

Exemplo 10: Programa que lê continuamente pares de números inteiros `a` e `b` até que ambos sejam iguais (condição de parada). Esta última leitura não é considerada. No final, o programa mostra quais foram os pares com a menor e a maior diferença numérica (sem sinal; módulo) entre os números `a` e `b` de todos os pares digitados.

```
program diferenca;
var dif_maior,dif_menor,a_maior,b_maior,a_menor,b_menor,a,b,dif:integer;
begin
   dif_menor:=32767;
   dif_maior:=-32768;
   repeat
     read(a,b);
     dif:=abs(a-b);
     if (dif<dif_menor)and(dif>0)
        then begin
           dif_menor:=dif;
           a_menor:=a;
           b_menor:=b;
        end
        else if (dif>dif_maior)and(dif>0)
               then begin
                    dif_maior:=dif;
                    a_maior:=a;
                    b_maior:=b;
               end;
   until (dif=0);
   writeln(´par maior = (´,amaior,bmaior,´)´);
   writeln(´par menor = (´,amenor,bmenor,´)´);
end.
```

O leitor pode observar que existe neste programa um laço maior controlado pela condição de parada. Enquanto não forem digitados dois números iguais o programa continuará em laço lendo pares de números `a` e `b`. Após a leitura de cada par, o programa calcula a diferença e procede a análise de maiores e menores. Note que a comparação é entre as diferenças e não entre os valores de `a` e `b` propriamente ditos. Como o último par a ser digitado é a condição de parada, este não pode ser considerado, motivo pelo qual o programa inclui a condição *(dif>0)* dentro de seus comandos `if`s, para evitá-lo. Entretanto, a inclusão desta condição causa esforço desnecessário do processador já que ela será válida somente na última vez. Isto ocorre porque a leitura dos números `a` e `b` é feita antes de sua análise. Uma solução para

resolver esta ineficiência é retirar para fora do laço a primeira leitura e colocar as próximas leituras no final do laço, imediatamente antes na análise da condição de parada. O programa a seguir implementa esta solução.

Exemplo 11: Idem ao anterior, entretanto, eliminando uma comparação desnecessária.

```
program diferenca_2;
var dif_maior,dif_menor,a_maior,b_maior,a_menor,b_menor,a,b,dif:integer;
begin
  read(a,b);
  dif:=abs(a-b);
  if (dif>0)
     then begin
       dif_menor:=dif;
       dif_maior:=dif;
       a_maior:=a;
       a_menor:=a;
       b_maior:=b;
       b_menor:=b;
       repeat
         if (dif<dif_menor)
            then begin
              dif_menor:=dif;
              a_menor:=a;
              b_menor:=b;
            end
            else if (dif>dif_maior)
                   then begin
                     dif_maior:=dif;
                     a_maior:=a;
                     b_maior:=b;
                   end;
         read(a,b);
         dif:=abs(a-b);
       until (dif=0);
       writeln(´par maior = (´,amaior,bmaior,´)´);
       writeln(´par menor = (´,amenor,bmenor,´)´);
     end
     else writeln(´Não existe pares validos para analise´);
end.
```

Com isso, caso o par seja de números iguais, o laço será finalizado sem que o mesmo seja analisado. O único prejuízo desta nova solução é que a primeira verificação entre maiores e menores é desnecessária pois esta será feita com os próprios valores da inicialização.

Exemplo 12: Programa que calcula a potência X^n (X elevado a n). O algoritmo deve ler os valores inteiros de X e n e aceitar que tanto X quanto n possa ser negativo. Observe que:

$4^3 = 4*4*4 = 64$ e $4^{-3} = (4^3)^{-1} = 1/(4^3) = 1/64$

A partir desta análise percebe-se que quando o n é positivo a potência é calculada por um produto do valor de X por ele mesmo, n vezes. Entretanto, mesmo que n seja negativo, o cálculo deste produto

também é necessário, com a diferença de que o resultado final é obtido dividindo-se 1 pelo valor obtido no produto. O programa a seguir implementa esta solução. Note que a função `abs` retorna o valor absoluto de um número.

```
program potencia;
var  pot,x:real;
     i,n:integer;
begin
   read(x,n);
   pot:=1;
   for i:=1 to abs(n) do
     pot:=pot*x;
   if n<0
     then pot:=1/pot;
   write(pot);
end.
```

Exercícios para Fixação:

1. Faça um programa para calcular a somatória, a soma dos quadrados e a média entre os n primeiros números naturais.

```
program soma_natural;
var  somatoria,soma_quad,media:real;
     i,n:integer;
begin
   read(n);
   somatoria:=0;
   soma_quad:=0;
   for i:=1 to n do
     begin
       somatoria:=somatoria+i;
       soma_quad:=soma_quad+i*i;
     end;
   media:=somatoria/n;
   writeln(somatoria,soma_quad,media);
end.
```

2. Escreva um programa para calcular os n primeiros termos de uma PG de elemento inicial A0 e razão R.

```
program pg;
var  an,a0,r:real;
     i,n:integer;
begin
   writeln(´Quantos elementos?´);
   readln(n);
   writeln(´Quais sao o elemento inicial e a razão?´);
   readln(a0,r);
   an:=a0;
   for i:=1 to n do
   begin
     write(an,´ ´);
     an:=an*r;
   end;
```

```
end.
```

3. Faça um programa que calcule a função $f = \sum_{i=1}^{n}[\sum_{j=i}^{m}(i * y_j^2)]$.

```
program somatoria1;
var  soma_ext,soma_int,y:real;
     i,j,n,m:integer;
begin
   readln(n,m);
   soma_ext:=0;
   for i:=1 to n do
   begin
     soma_int:=0;
     for j:=i to m do
       begin
         read(y);
         soma_int:= soma_int+i*y*y;
       end;
     soma_ext:=soma_ext+soma_int;
   end;
   writeln(soma_ext);
end.
```

4. Faça um programa que calcule a função $f = \sum_{i=1}^{n}(x_i + \sum_{j=i}^{m} y_j^2)$

```
program somatoria2;
var  soma_ext,soma_int,x,y:real;
     i,j,n,m:integer;
begin
   readln(n,m);
   soma:=0;
   for i:=1 to n do
   begin
     read(x);
     soma_int:=x;
     for j:=i to m do
       begin
         read(y);
         soma_int:= soma_int+y*y;
       end;
     soma_ext:=soma_ext+soma_int;
   end;
   writeln(soma_ext);
end.
```

5. Faça um programa que calcule a função $f = \sum_{i=1}^{n}[x_i^{(n-i)} + \sum_{j=1}^{m}(x_i + y_j^2)]$

```
program somatoria3;
var  soma_ext,soma_int,pot,x,y:real;
     i,j,k,n,m:integer;
begin
   readln(n,m);
   soma:=0;
   for i:=1 to n do
   begin
```

```
     read(x);
     pot:=x;
     for k:=2 to (n-i) do
       pot:=pot*x;
     soma_int:=pot;
     for j:=1 to m do
       begin
         read(y);
         soma_int:= soma_int+x+y*y;
       end;
     soma_ext:=soma_ext+soma_int;
   end;
   writeln(soma_ext);
end.
```

6. Uma empresa decide presentear seus funcionários com um bônus de natal, cujo valor é definido conforme segue. Os funcionários com tempo de casa superior a 5 anos terão direitos a um bônus de 10% do seu salário. Os funcionários com tempo de casa superior a 10 anos terão direitos a um bônus de 20% do seu salário. Os funcionários com tempo de casa superior a 20 anos terão direitos a um bônus de 30% do seu salário. Os demais funcionários terão direitos a um bônus de 5%. Elabore um programa para calcular o valor do bônus concedido a cada funcionário e o impacto de tal atitude no orçamento da empresa (o montante total de bônus concedido).

```
program bonus_natal;
var  tempo:integer;
     salario,bonus,montante:real;
begin
   montante:=0;
   bonus:=0;
   tempo:=30;
   repeat
     if (tempo>20)
     then bonus:=0,30*salario;
     else if (tempo>10)
            then bonus:=0,20*salario
            else if (tempo>5)
                   then bonus:=0,10*salario
                   else bonus:=0,05*salario;
     salario:=salario+ bonus;
     montante:=montante+ bonus;
     writeln(bonus); {o primeiro bonus não eh considerado}
     read(salario,tempo);
   until (salario<0); {salahrio negativo eh condicao de parada}
   writeln(montante);
end.
```

7. Para cada uma das 200 mercadorias diferentes com que um armazém trabalha dispõe-se os seguintes dados: nome, preço e a quantidade vendida de cada mercadoria por mês. Elabore um programa para calcular o faturamento bruto total mensal do armazém. Além disso, o programa deverá informar quais foram os produtos mais e menos vendidos do armazém.

```
program faturamento;
var  nome,mome_mais,nome_menos:string[32];
```

```
     preço,preço_mais,preço_menos,fatura_total:real;
     quant,i:integer;
begin
   read(nome,preço,quant); {o primeiro lido eh usado para inicializacao}
   nome_mais:=nome;
   nome_menos:=nome;
   quant_mais:=quant;
   quant_menos:=quant;
   fatura_total:=quant*preço;
   for i:=2 to 200 do  {o primeiro jah foi lido antes do for}
   begin
     read(nome,preço,quant);
     fatura_total:=fatura_total+quant*preço;
     if (quant>quant_mais)
        then begin
          quant_mais:=quant;
          nome_mais:=nome;
        end
        else if (quant<quant_menos)
               then begin
                  quant_menos:=quant;
                  nome_menos:=nome;
               end;
   end;
   writeln(fatura_total,nome_mais,quant_mais,nome_menos,quant_menos);
end.
```

8. Faça um programa que leia números continuamente até que seja digitado o número zero (0). Cada número digitado deve ser somado ou subtraído no total de números digitados, da seguinte forma: se o número é maior que a somatória atual ele deve ser somado na somatória, aumentando o valor desta, mas se o número é menor que a somatória, ele deve ser subtraído da somatória, diminuindo o valor desta. Números iguais ao valor da somatória devem ser desconsiderados. Além disso, sempre que um número localizado em uma posição múltipla de 5 (por ex: o quinto número digitado, o décimo número digitado, o 15º, o 20º etc.) for digitado ele deverá ser desconsiderado.

```
program operacao;
var  nro,soma:real;
     pos:integer;
begin
   soma:=0;
   read(nro);
   pos:=1;
   while (nro<>0) do
     begin
       if (nro>soma)and[(pos mod 5)<>0]
         then soma:=soma+nro
         else if (nro<soma)and[(pos mod 5)<>0]
                  then soma:=soma-nro;
       read(nro);
       inc(pos);
     end;
end.
```

9. Considere uma circunferência de raio R centrada em (0,0) no plano xy. Faça um programa que leia continuamente uma coordenada (x,y) e diga se ela está fora ou dentro da circunferência e em que

quadrante ela se encontra. Os quadrantes podem ser 1, 2, 3 e 4. O programa deve ler o valor de R. O programa deve parar de ler coordenadas quando a mesma se encontrar sobre qualquer um dos eixos.

```
program quadrantes;
var  x,y:integer;
     r,d:real;
begin
   read(r);
   repeat
     read(x,y);
     d:= sqrt(x*x+y*y); {raiz quadrada para calculo da distancia)
     if (d<=r)
        then writeln(´dentro´)
        else writeln(´fora´);
     if (x=0)
     then writeln(´sobre o eixo y´)
     else if (y=0)
            then writeln(´sobre o eixo x´)
            else begin
                if (x>0)
                   then begin
                     if (y>0)
                         then writeln(´quadrante 1´)
                         else writeln(´quadrante 4´)
                   end
                   else begin
                        if (y>0)
                         then writeln(´quadrante 2´)
                         else writeln(´quadrante 3´)
                   end;
            end;
   until (x=0)or(y=0);
end.
```

10. Fazer um programa que leia inicialmente 2 números A e B. Caso um dos números seja menor, o programa deverá ler um outro número para substituir o menor. Então, o programa deve fazer novamente uma verificação e se existir ainda um número menor, este deverá novamente ser substituído por outro número a ser lido, e assim sucessivamente até que os dois números A e B sejam iguais, quando então o programa deverá parar de ler números. O programa deve mostrar o número de substituições feitas.

```
program substitui;
var a,b,ns:integer;
begin
   read(a,b);
   ns:=0;
   while (a<>b) do
   begin
     inc(ns);
     if (a<b)
        then read(a)
        else read(b);
   end;
   writeln(ns);
end.
```

11. Faça um programa para simular um elevador. O programa deverá ler inicialmente o número do andar inicial (qualquer número não negativo). Depois disto, o programa deve continuamente ler o próximo andar e escrever ´sobe´ ou ´desce´ caso este andar seja superior (número maior) ou inferior (número menor). O programa deve parar quando o próximo andar for igual ao andar em que o elevador já se encontrar. O programa deve forçar a leitura de andares válidos não negativos. No final, o programa deverá mostrar o número de andares percorridos.

```
program elevador;
var andar,prox,dist:integer;
begin
  dist:=0;
  read(andar);
  repeat
    read(prox);
    if (prox>andar)
      then writeln(´sobe´)
      else if (prox<andar)
             then writeln(´desce´);
    dist:=dist+abs(prox-andar);
  until (andar=prox);
end.
```

12. Faça um programa para controlar um caixa eletrônico. Existem 4 tipos de notas: de 5, de 10, de 50 e de 100. O programa deve inicialmente ler uma quantidade de notas de cada tipo, simulando o abastecimento inicial do caixa eletrônico. Depois disto, o caixa entra em operação contínua atendendo um cliente após o outro. Para cada cliente, é lido o valor do saque a ser efetuado. Como resultado da operação, o programa deverá então escrever na tela a quantidade de notas de cada tipo que será dada ao cliente a fim de atender ao seu saque. Sempre que um saque for efetuado por um cliente, a quantidade inicial de dinheiro que foi colocada no caixa é decrementada. O programa deve pagar sempre com as maiores notas possíveis. Sempre que não for possível pagar somente com notas de 100, então o programa tentará complementar com notas de 50, e caso não seja suficiente, tentará complementar também com notas de 10, e por último, tentará usar também as notas de 5. Antes de efetuar um saque para um cliente, ou seja, escrever na tela as notas que ele irá receber, o programa deve ter certeza que é possível pagá-lo, senão emitirá uma mensagem do tipo ´Não Temos Notas Para Este Saque´. Caso o caixa fique abaixo de um certo mínimo, o algoritmo deverá parar de atender aos clientes e emitir uma mensagem do tipo ´Caixa Vazio: Chame o Operador´.

```
program caixa_eletronico;
var  n5b,n10b,n50b,n100b, {quantias do banco}
     n5c,n10c,n50c,n100c: integer;{quantias do cliente}
     minimo,saque,saldo,restante: integer;
begin
  read(n5b,n10b,n50b,n100b);
  read(minimo);
  saldo:=n5b*5+n10b*10+n50b*50+n100b*100; {inicializacao}
  while (saldo>minimo) do
    begin
      read(saque);
      if (saque<=saldo)
        then begin
          n100c:=(saque div 100);
          if (n100c>n100b)
```

```
            then n100c:=n100b;
          saque:=saque-n100c*100;
          restante:=saldo-n100b*100;
          if (saque<restante)
            then begin
              n50c:= (saque div 50);
              if (n50c>n50b)
                then n50c:=n50b;
              saque:=saque-n50c*50;
              restante:=restante-n50b*50;
              if (saque<restante)
                then begin
                    n10c:= (saque div 10);
                    if (n10c>n10b)
                      then n10c:=n10b;
                    saque:=saque-n10c*10;
                    restante:=restante-n10b*10;
                    if (saque<restante)
                    then begin
                      n5c:=(saque div 5);
                      if (n5c<=n5b)
                        then begin
                          writeln(´Saque Aprovado´);
                          writeln(n100c,n50c,n10c,n5c);
                          n100b:=n100b-n100c;
                          n50b:=n50b-n50c;
                          n10b:=n10b-n10c;
                          n5b:=n5b-n5c;
                          saldo:= n5b*5+n10b*10+n50b*50+n100b*100;
                        end
                        else writeln(´Valor Exagerado´);
                    end
                    else writeln(´Valor Impróprio para Troco´);
                end
                else writeln(´Valor Improprio para Troco´)
            end
            else writeln(´Valor Improprio para Troco´)
        end
        else writeln(´Valor Exagerado´);
    end;
end;
```

13. Sabendo-se que “Combinação de p em q” é definida da seguinte forma:

$$C_q^p = \frac{p*(p-1)*(p-2)*...(p-q+1)}{1*2*3...*q}$$

ex1: $C_3^5 = \frac{5*4*3}{1*2*3} = 10$

ex2: $C_4^9 = \frac{9*8*7*6}{1*2*3*4} = 126$

Faça um algoritmo que calcule “Combinação de p em q” , com p e q $\in N^+$.

```
program combinação;
var  comb,num,den:real;
```

```
     p,q,i:integer;
begin
   read(p,q);
   num:=1; {termo de cima}
   den:=1; {termo de baixo}
   for i:=1 to q do
   begin
       num:=num*(p-i+1);
       den:=den*i;
   end;
   comb:=num/den;
   write(comb);
end.
```

14. Faça um programa em Pascal que simule um relógio usando 2 comandos **`for`** e 2 comandos **`repeat`**.

```
program relogio_5;
var h,m,s:integer;
begin
   repeat
     h:=0;
     repeat
        for m:=0 to 59 do
           for s:=0 to 59 do
              begin
                clrscr;
                write(h,´ ´,m,´ ´,s);
                delay(1000);
              end;
        inc(h);
     until (h>=24); {aqui existe um exagero na condição}
   until (true);
end.
```

15. Idem ao 14 usando 2 comandos **`while`** e 2 comandos **`repeat`**.

```
program relógio_6;
var h,m,s:integer;
begin
   while (true) do
   begin
     h:=0;
     while (h<24) do
     begin
        m:=0;
        repeat
           s:=0;
           repeat
              clrscr;
              write(h,´ ´,m,´ ´,s);
              delay(1000);
              inc(s);
           until (s>=60);
           inc(m);
```

```
      until (m>=60);
      inc(h);
    end;
  end;
end.
```

16. Faça um programa que simule um relógio usando os 3 comandos de repetição conhecidos e que desperte sempre de n em n minutos, desde após o início da contagem. O valor de n deve ser lido no início do programa.

```
program relógio_7;
var h,m,s,n,cont:integer;
begin
  cont:=0;
  read(n);
  repeat
    h:=0;
    while (h<24) do
    begin
      for m:=0 to 59 do
      begin
        for s:=0 to 59 do
          begin
            clrscr;
            write(h,´ ´,m,´ ´,s);
            delay(1000);
          end;
        inc(cont);
        if (cont=n)
          then begin
            cont:=0;
            writeln(´hora de despertar!!!´);
          end;
      end;
      inc(h);
    end;
  until (true);
end.
```

17. Idem ao 16, mas o despertador somente deverá funcionar dentro do intervalo de tempo contido entre dois horários fornecidos inicialmente. Os horários devem ser compostos de hora e minuto.

```
program relógio_8;
var h,m,s,n,cont,hi,mi,hf,mf:integer;
begin
  cont:=0;
  read(hi,mi);
  read(hf,mf);
  read(n);
  repeat
    h:=0;
    while (h<24) do
    begin
      for m:=0 to 59 do
      begin
```

```
          for s:=0 to 59 do
            begin
              clrscr;
              write(h,´ ´,m,´ ´,s);
              delay(1000);
            end;
          if  (hi*60+mi*60)<(h*60+m*60)and
              (h*60+m*60)<(hf*60+mf*60)
            then begin
                inc(cont);
                if (cont=n)
                  then begin
                      cont:=0;
                      writeln(´hora de despertar!!!´);
                  end;
            end;
        end;
        inc(h);
      end;
   until (true);
end.
```

18. Faça um programa em Pascal que simule uma calculadora de operações básicas de adição, subtração, divisão e multiplicação. A calculadora deverá permitir a entrada de quantos números o usuário desejar, sempre separados por uma das 4 operações citadas. O usuário deve começar sempre com um número e posteriormente intercalar entre operação e número. Após pressionar o sinal de igual (=) o resultado final será mostrado.

```
program calculadora;
var   nro:integer;
      total:real;
      op:char;
begin
   while (true) do
   begin
     total:=0;
     op:=´+´;
     repeat
       read(nro);
       case op of
          ´+´: total:=total+nro;
          ´-´: total:=total-nro;
          ´*´: total:=total*nro;
          ´/´: total:=total/nro;
       end;
       read(op);
     until (op=´=´);
     write(total);
   end;
end;
```

19. Faça um programa em Pascal que sorteie um número aleatório de 1 a 3 e dependendo do resultado ele faz uma entre 3 perguntas. Cada pergunta deve ter 3 alternativas onde somente uma é verdadeira. O programa deve informar se a resposta está certa ou errada. **Nota**: o programa deverá ter 3 perguntas mas somente uma será feita.

```
program pergunta;
uses crt;
var  n:integer;
     resp:char;
begin
  n:=random(3);
  case n of
  0:  begin
        writeln(´Em que ano ocorreu a Sabinada?´);
        writeln(´A – Em 1880´);
        writeln(´B – Em 1980´);
        writeln(´C – Em outro ano´);
        write(´= > ´);
        readln(resp);
        if (resp=´c´)or(resp=´C´)
          then writeln(´Sua resposta esta EXATA´)
          else writeln(´Sua resposta esta ERRADA´);
      end;
  1:  begin
        writeln(´Quem descobriu o Brasil?´);
        writeln(´A – Pietro Albanes Sobral´);
        writeln(´B – Pedro Alvares Cabral´);
        writeln(´C – Outra pessoa´);
        write(´= > ´);
        readln(resp);
        if (resp=´b´)or(resp=´B´)
          then writeln(´Sua resposta esta EXATA´)
          else writeln(´Sua resposta esta ERRADA´);
      end;
  2:  begin
        writeln(´Quem presidiu o Brasil no ano 2000?´);
        writeln(´A – Fernando Henrique Cardoso´);
        writeln(´B – Luis Inácio Lula da Silva´);
        writeln(´C – Outro presidente´);
        write(´= > ´);
        readln(resp);
        if (resp=´a´)or(resp=´A´)
          then writeln(´Sua resposta esta EXATA´)
          else writeln(´Sua resposta esta ERRADA´);
      end;
end.
```

20. Estenda o exemplo anterior de forma que o usuário passa ter 2 coringas caso não saiba a resposta. Os coringas são assim chamados: 1) “Dica”: se escolhido este coringa uma dica sobre a resposta verdadeira é dada; 2) “Carta”: se escolhido este coringa uma resposta errads é eliminada.

```
program pergunta_2;
uses crt;
var  n:integer;
     resp:char;
     dica, carta, volta:boolean;
begin
  dica:=true;
  carta:=true;
  n:=random(3);
  case n of
```

```
  0:  begin
        volta:=true;
        repeat
          clrscr;
          delay(1000);
          writeln(´Em que ano ocorreu a Sabinada?´);
          writeln(´A – Em 1880´);
          writeln(´B – Em 1980´);
          writeln(´C – Em outro ano´);
          if (dica)
            then writeln(´D – Quero uma Dica´);
             else
          if (carta)
            then writeln(´E – Quero uma Carta´);
          write(´= > ´);
          readln(resp);
          case resp of
          ´c´: begin
                 writeln(´Sua resposta esta EXATA´);
                 volta:=false;
               end
          ´d´: if(dica)
                 then begin
                   writeln(´DICA: A resposta eh difícil´);
                   dica:=false;
                 end
                 else writeln(´A dica jah foi usada. Tente de novo´)
          ´e´: if(carta)
                 then begin
                   writeln(´CARTA: A resposta A eh incorreta´);
                   carta:=false;
                 end
                 else writeln(´A carta jah foi usada. Tente de novo´)
          else begin
              volta:=false;
              writeln(´Sua resposta esta ERRADA´);
          end;
        until (not volta);
      end;
  1:  begin
           ... {idem, modificando pergunta, resposta, dica e carta}
      end;
  2:  begin
           ... {idem, modificando pergunta, resposta, dica e carta}
      end;
end.
```

DICAS DE APARÊNCIA:

Os comandos `textbackground(cor)` e `textcolor(cor)` podem ser usados para a seleção da cor do fundo da tela, onde o texto será escrito, e da cor do texto propriamente dito, respectivamente. Não são comandos gráficos, mas podem ser usados em modo texto (modo de caracteres) para melhorar a aparência dos programas. Comumente são usados em conjunto com os comandos `clrscr()` e `gotoxy()`.

Lista de Atividades Complementares:

1. Estenda o programa do exemplo 20 de forma a implementar um Mini-Show do Milhão. O programa deve passar por 10 estágios eliminatórios. Em cada estágio é feita uma pergunta aleatória da mesma forma que no exercício anterior. Caso a resposta esteja correta, o próximo estágio é alcançado e uma nova pergunta é feita. Caso a resposta esteja errada o usuário é eliminado e recebe um prêmio. O prêmio é inicialmente nenhum mas é aumentado a cada estágio alcançado, de forma acumulativa. Os estágios iniciais valem menos e os estágios finais valem mais. Assim, as perguntas iniciais devem ser mais fáceis e as perguntas finais mais difíceis. **Nota**: O programa deverá ter 50 perguntas, mas somente 10 serão feitas, no máximo.

2. Faça um programa em Pascal para resolver a seguinte função: $f = \sum_{i=1}^{n}[(i^n + i^{(n-1)} + ... + i^0) * \sum_{j=i}^{i^n}(j)]$

3. Fazer um programa em Pascal que simule uma bomba relógio. O programa deve inicialmente ler um horário (composto de hora, min e seg) e um código de segurança (string). O horário lido indicará o tempo que deverá decorrer para que a bomba exploda (escreverá a mensagem "BUMMM!!" na tela e finalizará a execução do programa). Um relógio digital deverá ser mostrado na tela continuamente em ordem decrescente de horário, ou seja, iniciará com o horário estabelecido e irá diminuindo a cada segundo até atingir o zero total (hora, min e seg iguais a zero), quando então a bomba explodirá. Mas se alguém digitar o código de segurança correto, antes do horário ter decorrido a zero total, a bomba será desativada (escreverá a mensagem "BOMBA DESATIVADA!!!" e a execução do programa será finalizada). O programa não deve parar a cada iteração do relógio para pedir que o código seja digitado. O programa deve usar o comando **keypressed** para determinar se o usuário pressionou alguma tecla e somente neste caso o programa poderá pedir pelo código usando o comando **readln**. Se o código digitado for igual ao código de segurança, a bomba será desativada e o programa finalizará.

4. Fazer um programa em Pascal para ficar em *loop* lendo caracteres continuamente com o comando **readkey** (permite a entrada de um caractere sem a necessidade de digitar <enter>**)**. A leitura deverá ser feita a partir de uma posição específica da tela, feita pelo comando **gotoxy**(x,y). A coordenada (x,y) deve ser lida no início do programa. O programa somente manterá escrito na tela os caracteres que forem digitados em ordem alfabética. Assim, para cada caractere que for digitado fora de ordem alfabética (menor que o anterior válido), o **cursor** deverá voltar uma posição para a esquerda do vídeo e um espaço em branco deverá ser escrito sobre o caractere digitado incorretamente. Depois de apagado, o **cursor** deverá voltar novamente uma posição para a esquerda do vídeo para ficar posicionado no local correto da próxima leitura, a qual deverá ser analisada novamente. O programa deverá ser finalizado após a escrita do caractere ´z´ ou após a escrita atingir o extremo lado direito do vídeo (coluna 80).

5. Fazer um programa em Pascal que simule um editor de texto na tela do vídeo. O programa deverá ser iniciado com o cursor na posição (0,0). Use **gotoxy**() para controlar o posicionamento na tela. Considere que a tela vai de (0,0) no canto superior esquerdo até (79,23) no canto inferior direito. O programa fica em loop esperando que quaisquer teclas sejam digitadas, inclusive as teclas <enter>, <backspace> e setas direcionais <←>, <↑>, <→> e <↓>, onde a tecla <enter> faz com que o cursor pule para o início da próxima linha; a tecla <backspace> deleta (apaga) o último caracter antes da posição atual do cursor e retorna uma posição para a esquerda e as setas direcionais apenas caminham sobre as posições do vídeo. Além disso, o programa deverá aceitar uma combinação de duas teclas: <alt><del> para deletar a linha inteira onde esta o cursor. Após deletar esta linha, o cursor deverá estar posicionado na posição inicial da linha que foi deletada. Use o comando **readkey** para ler os caracteres sem que o usuário precise digitar <enter> após cada um. Para reconhecer as teclas especiais, compare o caracter lido com o código retornado pelo comando ORD, com a seguinte suposição:

ORD(<backspace>) = 30 ORD(←) = 50 ORD(→) = 52 ORD(<alt>) = 60

ORD(<enter>) = 32 ORD(↑) = 51 ORD(↓) = 53 ORD(<del>) = 61

6. Fazer um programa em Pascal para calcular a somatória abaixo, a qual contém n elementos.

S = (1 + 4/2 + 9/4 + 16/8 + 25/16 + 36/32 + 49/64 ...) / [n*(n-1)*(n-2)*...*1]