Apostila de linguagem pascal

Por: Ítalo Herbert Siqueira Gabriel

Sumário

1 Apresentação

Se você tem vontade de aprender uma linguagem de programação e não sabe por onde começar, recomendo este curso gratuito sobre linguagem **PASCAL** que aborda, desde, conceitos iniciais para leigos em programação, até conceitos mais avançados.

1.1 Algumas sugestões

Abaixo algumas **sugestões** sobre o **aprendizado de programação**:

1. **Utilize um buscador** para pesquisar sempre que necessário
2. Compile e execute os programas apresentados aqui e **faça alterações neles** para visualizar o resultado após as alterações. Mais pra frente, nesta apostila, você aprenderá a compilar e executar os programas.
3. **Não tente apenas memorizar os programas**. Procure memorizar a sintaxe (e regras) da linguagem estudada e então **busque entender a lógica** contida em cada programa estudado.
4. Se sentir dificuldade no início, **não desanime**, é perfeitamente normal. Entenda que alunos de um curso de graduação na área têm em torno de um semestre inteiro para aprender o básico de uma linguagem de programação, entre outras cinco ou seis disciplinas.

2 Introdução

Para desenvolver software em linguagem Pascal, você vai precisar de um **compilador**, e uma **IDE**. Mas, o que seria um compilador? Seria um software que recebe o seu programa escrito (**código fonte**) em formato texto e, a partir dele, gera um outro arquivo contendo os dados binários (**código objeto**) correspondentes ao seu programa.

Agora que você sabe o que é um código fonte e um código objeto, está na hora de entender como que o Sistema Operacional (Windows/Linux ou outro) poderá executar o seu programa para você visualizar o resultado dele. Então, o Windows, por exemplo, executa programas em formato executável (**.exe**). Isso significa que, para você executar seu programa, será necessário gerar o arquivo executável a partir do **código objeto.** A este processo de "converter" o código objeto em executável, dar-se o nome de **linkagem**. Geralmente, ao baixar um compilador para Windows ou Linux, o mesmo já vem com o **linker** (programa que faz a linkagem).

A **IDE**, em alguns casos, e é o nosso, já traz o compilador e o linkador de modo que a gente nem percebe a distinção de cada um. Isto é, a **IDE** (Ambiente de Desenvolvimento Integrado) é um software que oferece um editor de texto e opções (na barra de ferramentas geralmente) para você compilar, gerar executável, rodar, etc.

Calma, **não se assuste** com esses **conceitos**, você poderia prosseguir com o curso até sem compreender eles de início. Embora, esses conceitos poderão ser importantes em seus estudos após concluir este curso.

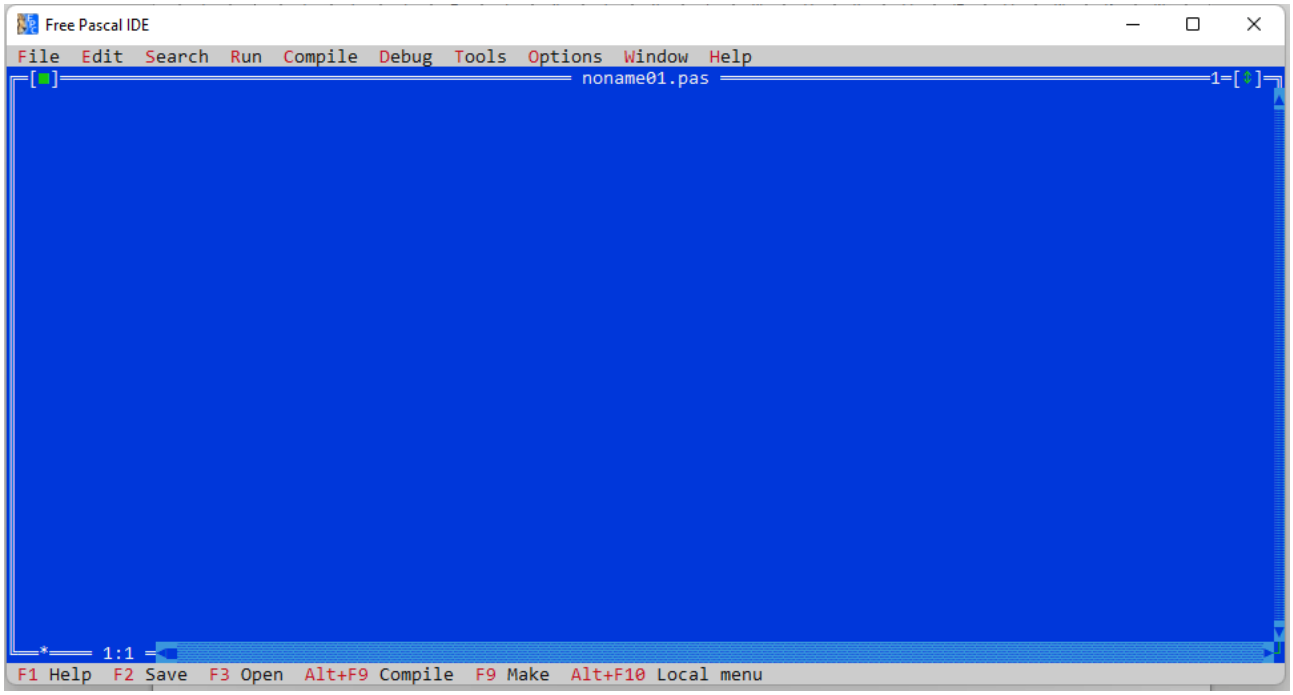
2.1 Primeiro programa

Existem várias opções, mas, a que utilizaremos é o FPC – (**Free Pascal Compiler**). Este software é, tanto compilador, quanto, IDE. Então, o que você tem a fazer é escrever seu programa nele e compilar e executar. Simples assim!

Download em: https://www.freepascal.org/down/i386/win32.html

Esse download acima é para baixar a versão de 32 bits do software, mas, sistemas operacionais modernos suportam a versão de 64 bits que pode ser encontrada no mesmo site. Isto é, em: https://www.freepascal.org.

Instale o FPC após o download e, então, execute-o. Será mostrada uma tela como abaixo:



A tela acima é uma captura de tela do Free Pascal IDE, onde devemos escrever nossos programas e testar.

Muito bem, assumindo que você já baixou o software necessário e o ambiente já está preparado, é hora de fazer o primeiro programa. Então vamos lá:

Escreva o seguinte programa:

```
program PRIMEIROPROGRAMA;
uses crt;
begin
      clrscr;
      writeln('Meu primeiro programa');
      readln;
end.
```

Muito bem, agora vamos discutir as instruções em cada linha do programa:

Todo programa em pascal começa pela definição do nome do programa que você deve escolher conforme sua preferência. Neste caso, o nome escolhido foi “PRIMEIROPROGRAMA” e esse nome deve ficar após a palavra program separada por um ou mais espaços e finalizada com ponto e vírgula. Aliás, toda instrução em pascal deve ser finalizada com ponto e vírgula.

Então, a primeira linha ficou assim:

```
program PRIMEROPROGRAMA;
```

Por enquanto ignore a segunda linha e vamos para a terceira que contem apenas a palavra **begin**. Esta instrução apenas marca o início do bloco principal de instruções correspondentes ao programa em si. Logo, o fim de tal bloco de instruções é finalizado com a palavra **end**. Entenda que, traduzido para português, begin é início e end é final. Logo, o programa mais básico que se pode escrever em pascal é o seguinte:

```
program Qualquer;
begin
end.
```

No entanto, o programa acima não faz nada de útil, apenas executa o bloco principal (begin end), sem qualquer instrução e, então, não mostra nada. Por isso, devemos colocar as instruções do programa no **bloco de código principal** (entre o **begin** e o **end**). As instruções dentro deste bloco são chamadas de **programa principal**.

Logo, o programa que você compilou e rodou tem um nome, após o **program** e três instruções que compõem o programa principal:

```
clrscr;
writeln('Meu primeiro programa');
readln;
```

A segunda é um procedimento (depois explico o que é um procedimento). Mas, por enquanto, entenda apenas o segunte: **write** em português é **escreva** e **ln** é uma abreviação de **line** que em português é **linha.** Logo **writeln** é o mesmo que "escreva e pule de linha". Tenha atenção aos apóstrofos. Eles são obrigatórios neste caso e marcam o início e o final do texto a ser mostrado!

Em resumo, utilize a instrução **writeln** sempre que desejar imprimir uma mensagem na tela do computador!

A última é a instrução **readln** que significa "leia e pule de linha" que, em resumo, apenas faz com que o programa, após executadas as instruções anteriores, espere o usuário digitar uma tecla qualquer para, então finalizar o programa e voltar ao editor de código fonte.

Ah, a primeira instrução é a **clrscr**, esta instrução apenas limpa a tela onde a mensagem do writeln abaixo dela é mostrada. Se não tiver ela, a cada vez que você rodar o seu programa, a tela de resultado vai mostrando cada execução uma abaixo da outra. Se utilizar o **clrscr**, então, as execuções anteriores são apagadas para mostrar só a atual. Para utilizar o procedimento **clrscr**, é necessária a seguinte instrução após o a instrução program:

```
uses crt;
```

Muito bem, você já fez um programinha, compilou e rodou para ver o resultado. Se não compilou e rodou ainda o primeiro programa, faça isso!

2.2 Pascal é case insensitive

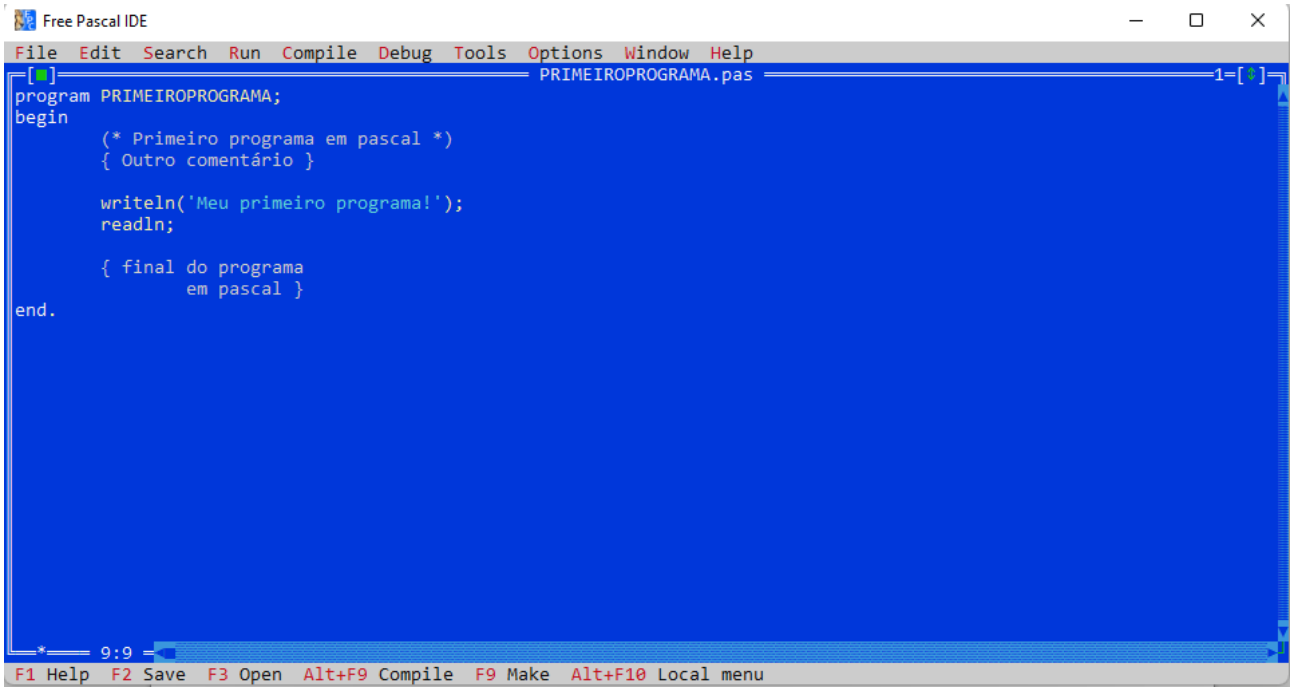
A linguagem Pascal não diferencia maiúsculas de minúsculas, por isso ela é conhecida como **case insensitive**. Há outras linguagens que são **case sensitive** (diferenciam maiúsculas de minúsculas). Então, em Pascal, os programas abaixo fazem exatamente a mesma coisa:

```
PROGRAM a;
begin
End.

PrOGram    B; BEGIN
End.

program
      c; begin end;
```

Perceba também o espaçamento. Perceba que no segundo programa há um espaçamento maior que um entre o **program** e o nome do programa. O compilador entende que, entre o program e o nome deve haver um ou mais espaços ou quebras de linha, etc.

2.3 Comentários

Os comentários são textos que o compilador ignora no processo de compilação do código fonte. Você pode inserir comentários em seu código fonte através de {} ou (**). Conforme abaixo:



3 Variáveis

As variáveis geralmente têm um tipo e um valor associado. Logo, há um bloco especifico de programas em Pascal para declaração de variáveis. Mas, o que vem a ser uma variável? Você pode se perguntar. Então, uma variável é uma alocação de memória, em que, se pode ler ou alterar os dados gravados em tal posição de memória através de uma linguagem de programação como a linguagem Pascal. Entenda que, para manipular variáveis em pascal, você deve associar um nome a ela após sua declaração.

3.1 Variáveis inteiras

Veja o programa abaixo:

```
program Exemplo;
uses crt;
var
      a:integer;
begin
      clrscr;
      write('Informe um numero: ');
      readln(a);
      writeln('Você informou o numero ',a);
      readln;
end.
```

O programa acima apenas solicita ao usuário que ele informe um número e, após o usuário digitar o número, tal número é armazenado na variável inteira de nome "a" e, após isto, é impressa outra mensagem exibindo também o número informado para, então finalizar o programa após o usuário digitar qualquer tecla.

Calma, a explicação acima é superficial. Vamos por partes:

O bloco de declaração de variáveis é o "var". Após o var e antes do begin do programa principal, você pode declarar suas variáveis. No programa exemplo acima, apenas uma variável foi declarada. A sintaxe é: "nome:tipo", logo o nome da variável é "a" e o tipo é "integer" que significa inteiro. Isto é, o programa tem uma declaração de uma variável que pode armazenar valores inteiros.

Agora vamos ao programa principal:

A primeira instrução é o **clrscr**. Limpa a tela.

A segunda é o **write** (sem **ln**), logo, apenas escreve sem pular de linha e posicionar o cursor na linha seguinte. Neste caso, após a mensagem do **write** ser mostrada, o programa vai esperar que o usuário digite uma tecla e ela é mostrada na frente da mensagem. Se tivesse o **ln**, o valor digitado seria mostrado na linha abaixo.

A terceira instrução é a instrução de leitura, logo, entre os parênteses dela está a variável onde deve ser armazenado o valor digitado pelo usuário.

Em seguida vem outro **write**, sendo que com o **ln**. Neste caso é impressa a mensagem e o valor da variável. Perceba que o texto está entre apóstrofos, mas, há uma vírgula após o segundo apóstrofo e, em seguida, há o nome da variável que, neste caso, não deve estar entre apóstrofos, ou,

caso contrário, o compilador entenderia que deveria imprimir o nome da variável, e não o valor armazenado nela.

Ao final, vem um **readln** para dar uma parada na tela até o usuário teclar algo.

Pronto, está explicado! Agora brinque um pouco com esse programa. Altere as mensagens! Coloque mais **writeln**(s) com textos quaisquer de sua preferência, alguns com **write** sem **ln** e outros com o **ln** e, então, visualize o resultado de cada alteração!

4 Constantes

A constante predefinida mais básica e utilizada em pascal é a constante PI. A constante PI tem valor em torno de: 3,14. Claro, quanto maior o número de casas decimais, mais preciso se torna o número. Neste caso, estamos considerando o valor das constantes com precisão de apenas duas casas decimais. PI pode ser também: 3,14159. O que significa que o número agora está representado com precisão de 5 casas decimais.

Você pode utilizar em suas expressões numéricas quaisquer constantes. Isto é, as constantes são como variáveis que você não pode alterar seu valor. Apenas é possível acessar o valor de uma constante. Veja o exemplo abaixo:

```
program Matematica;
uses crt;
const E:real=2.71;
var angulo:real;
begin
      angulo:=2*PI;
      writeln('360 graus em radianos corresponde a: ',angulo:0:2);
      writeln('O número de Euler corresponde a: ',E:0:2);
      readln;
end.
```

No programa acima, o dobro de PI é armazenado na variável ângulo e mostrado como valor real. No segundo writeln, é impressa a constante E diretamente, também como valor real, isto é, sem atribuir seu valor a uma variável. Como pascal é case insensitive, você pode declarar a constante e com letra maiúscula ou minúscula, ou com outro nome se desejar.

5 Expressões aritméticas

As expressões aritméticas podem ser usadas de várias formas. Uma delas é através da atribuição de seu resultado a uma variável que pode ser inteira ou número com vírgula, conforme os operadores aritméticos da expressão.

O operador de atribuição é o dois pontos seguido de igual (:=).

Outra coisa importante de se entender é a precedência que há entre os operadores. Por exemplo: Em uma expressão aritmética, são executadas primeiramente as multiplicações e divisões do que as somas e subtrações. Inclusive, as operações de mod e div têm maior precedência que as multiplicações e divisões. Entre parênteses ficam subexpressões que são executadas primeiro que qualquer outra parte. Veja o exemplo abaixo:

```
2+5*10-3 mod 2/(3 div 2*2)
```

Primeiramente são executados os parênteses, e depois, os mods e divs, resultando em:

```
2+5*10-1/(1*2)
```

Depois as multiplicações e divisões, resultando em:

```
2+50-1/2 = 2+50-0.5 = 52–0.5 = 51.5
```

O resultado da expressão acima é: 51.5. Se testar a expressão, não esqueça de atribuir o resultado a uma variável real e formatar no writeln com :0:1 ou conforme a precisão desejada.

5.1 Operadores aritméticos

Os operadores aritméticos suportados pela linguagem pascal são os seguintes:

Operador	Descrição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão (Fracionária)
mod	Resto de divisão
div	Quociente de divisão

Agora veja um programa exemplo:

```
program Exemplo;
uses crt;
var
      a,b,c:integer;
      soma,sub,mult,resto,quociente:integer;
      divisao:real;
begin
      clrscr;
      a := 3;
      b := 2;
      soma := a + b;
      sub := a – b;
      mult := a * b;
      divisao := a / b;
      resto := a mod b;
      quociente := a div b;
      writeln(‘Soma=’,soma);
      writeln(‘Subtracao=’,sub);
      writeln(‘Multiplicacao=’,mult);
      writeln(‘Divisao=’,divisao);
      writeln(‘Resto=’,resto);
      writeln(‘Quociente=’,quociente);
      readln;
end.
```

Primeiramente vamos focar à declaração de variáveis. Isto é, foram declaradas as variáveis a,b e c do tipo inteiro e, em seguida, as variáveis: soma, sub, mult, quociente, resto, também do tipo

inteiro. Inclusive, seria possível colocar todas essas variáveis na mesma linha, isto é, após a declaração da variável “c”.

Bom, perceba que foi declarada também uma variável do tipo **Real**, me refiro a variável “divisao”.

Um **número do tipo real** é um número com vírgulas e casas decimais após a vírgula (**Números de ponto flutuante**). Mas, você pode se perguntar: Por que a necessidade de declarar o resultado de uma divisão (Com utilização do operador **“/”**) em uma variável **real**? A resposta é simples! Entenda o seguinte: **3 dividido por 2** é igual a **1,5** e este número não é um número inteiro. Logo, sempre que utilizar o operador de divisão, terá que atribuir o resultado da divisão a uma variável **Real**.

Perceba também o seguinte: enquanto o resultado da expressão 3/2 é 1,5, o resultado da expressão 3 mod 2 é 1 porque se dividirmos 3 por 2, teremos como resto da divisão o valor 1 e, no caso de 3 div 2, teremos o quociente da divisão: neste caso, também 1.

Entenda que quando dividimos 3 por 2, o quociente é 1 e o resto também é 1. Já 6 dividido por 4 tem resto 2 e quociente 1.

E, como você já deve ter entendido, os writeln(s) são para imprimir o resultado. Atenção para o modo que as variáveis são colocadas no comando write ou writeln. Você pode inserir nessa instrução entre parênteses vários valores do tipo texto ou variáveis ou constantes, sendo que, separadas por vírgula e, o que é texto, deve iniciar e finalizar com aspas e o que é variável, deve ficar sem as aspas. Exemplo:

```
program Exemplo;
uses crt;
var a, b, c:integer;
begin
      clrscr;
      a:=1;
      b:=2;
      c:=a+b;
      writeln(a,' + ',b,' = ',c);
      writeln(1,' + ',2,' = ',(1+2));
      writeln(1,' + ',2,' = ',3);
      readln;
end.
```

O programa acima imprime três vezes a mesma coisa. Isto é:

```
1 + 2 = 3
1 + 2 = 3
1 + 2 = 3
```

6 Tipos de dados

As linguagens de programação geralmente oferecem meios de se armazenar dados após operação de leitura. Isso significa que, você pode solicitar ao usuário que digite valores para, então, armazenar os valores digitados em variáveis para fazer verificações ou outro tipo de processamento sob os valores armazenados em tais variáveis.

Veja abaixo os principais tipos de variáveis em Pascal:

Tipo	Descrição	Tamanho em memória
integer	Armazena números inteiros negativos ou positivos ou zero	2 bytes
real	Armazena números reais com ou sem casas decimais separadas por ponto. (Não por vírgula).	4 bytes
char	Armazena um caractere entre aspas	1 byte
string	Armazena um conjunto (array) de caracteres que formam um texto	127 bytes
boolean	Armazenam valores resultantes de expressões booleanas. Isto é, “false” ou “true”.	1 byte
longint	Armazena números inteiros longos	4 bytes

Veja o programa exemplo abaixo:

```
program Exemplo;
uses crt;
var
      x:integer;
      y:longint;
      z:real;
      ch:char;
      str:string;
      b:boolean;
begin
      clrscr;
      x:=-1;
      y:=1000000;
      z:=1.5;
      ch:='A';
      str:='Ola mundo!';
      b:=true;
      writeln('Valor de x: ',x);
      writeln('Valor de y: ',y);
      writeln('Valor de z: ',z:0:2);
      writeln('Valor de ch: ',ch);
      writeln('Valor de str: ',str);
      writeln('Valor de b: ',b);
      readln;
end.
```

No programa acima, foram declaradas uma variável de cada um dos seis tipos abordados nesta seção. Perceba que:

1. “x” é do tipo integer, logo, pode assumir um valor inteiro negativo ou positivo.
2. “y” é do tipo longint, logo, pode receber números longos. Se o número 1000000 tivesse sido atribuído a uma variável inteira, o programa geraria um erro porque variáveis do tipo integer só têm capacidade para armazenar números entre: -32768 até 32767, e 1000000 é maior que 32767, logo, está fora da faixa de números do tipo integer.
3. O “z” é do tipo real, logo, pode assumir valores reais com casas decimais separadas por um ponto (não por vírgula).
4. A variável “ch” é do tipo char (caractere), logo, pode assumir qualquer caractere do teclado, sendo que, se definido explicitamente, como é o caso no programa acima, o caractere precisa estar entre aspas simples.
5. A variável “str” é do tipo string (conjunto de caracteres), logo, pode receber uma sequência de caracteres entre aspas simples. Por isso é um conjunto de chars (caracteres).
6. A variável “b” é do tipo boolean. Isso significa que se pode atribuir a ela, expressões booleanas ou os valores true ou false.

Importante!

Perceba o seguinte: veja a linha abaixo:

```
writeln(‘2 menos 1 eh igual a: ‘,(2-1));
```

O comando writeln, dentro de parênteses que o sucede, recebe uma ou mais variáveis, expressões ou constantes. Perceba que as variáveis, constantes ou expressões são separadas por vírgula. No writeln acima, há primeiro uma “string”, entre aspas, e depois, há uma vírgula para, depois, vir uma expressão inteira entre parênteses que retorna um valor inteiro. Logo, o compilador entende que há uma string e um inteiro, separados por vírgula e passados para o comando.

Seria possível fazer conforme o programa abaixo:

```
program Exemplo;
begin
      writeln(‘2 menos 1 eh igual a: ‘,(2-1));
      writeln(‘2 menos 1 eh igual a: ‘,1);

      a:=‘2 menos 1 eh igual a: ‘;
      x:=2-1;
      writeln(a,x);

      readln;
end.
```

O programa acima imprime três vezes a mesma mensagem:

```
2 menos 1 eh igual a: 1
2 menos 1 eh igual a: 1
2 menos 1 eh igual a: 1
```

7 Expressões Booleanas

Muito bem, agora que você já sabe imprimir o resultado de uma operação aritmética envolvendo variáveis, é hora de aprender sobre operações lógicas com expressões booleanas.

As operações booleanas são operações lógicas que envolvem os conectivos AND, OR, NOT, XOR, traduzidas respectivamente para o português, temos: E, OU, NÃO, OU Exclusivo. Toda expressão booleana retorna um resultado que pode assubir um entre dois valores: false (FALSO) ou true (VERDADE).

7.1 Conector AND – E

Veja a expressão em português abaixo:

“João comprou a casa e reformou a casa do filho”

Perceba o conector “e” que liga duas afirmações que, juntas, compõe uma outra afirmação. Então, vamos analisar a afirmação acima:

“João comprou a casa”, é uma afirmação que pode ser verdadeira ou falsa.

“João reformou a casa do filho”, é outra afirmação que pode ser verdadeira ou falsa.

Então, com o uso do conector ***“E”*** a terceira afirmação é verdadeira se:

“João comprou a casa” for verdade e se **“João reformou a casa do filho”** for verdade também

Vamos considerar a primeira afirmação como A1 e a segunda afirmação como A2:

Se **A1 e A2 forem verdade**, a afirmação é **verdadeira**
Se **A1 for verdade** e **A2 for falsa**, a afirmação é **falsa**
Se **A1 for falsa** e **A2 for verdade**, a afirmação é **falsa**
Se **A1 e A2 forem falsas**, a afirmação também é **falsa**

Outro jeito de analisar:

```
verdade e verdade = verdade
verdade e falso   = falso
falso   e verdade = falso
falso   e falso   = falso
```

Ou ainda:

```
true and true = true
true and false = false
false and true = false
false and false = false
```

Logo, caso uma das duas afirmações seja falsa, a firmação inteira também assume valor falso, e verdade, somente se as duas forem também verdade.

7.2 Conector OR - OU

Agora vamos analisar a mesma afirmação com o conector **“OR”**:

```
“João comprou a casa ou reformou a casa do filho”
```

Entenda que as subafirmações são as mesmas, mudou apenas o conector que as liga, formando uma outra afirmação. Então, analise as possibilidades:

```
verdade ou verdade = verdade
verdade ou falso   = verdade
falso   ou verdade = verdade
falso   ou falso   = falso
```

Em pascal seria:

```
true  or true  = true
true  or false = true
false or true  = true
false or false = false
```

Logo, duas expressões lógicas são ligadas pelo conector E (AND) são verdadeiras somente se as duas partes são também verdadeiras.

Duas expressões lógicas litadas pelo conector OU (OR) são verdadeiras caso as duas, ou pelo menos, uma das duas, for verdadeira. Isto é, só é falsa, em caso de as duas serem também falsas.

7.3 Conector NÃO – NOT

O operador NÂO não liga duas expressões como os conectores E e OU, mas, sim, tem efeito em apenas uma afirmação. Entenda também que uma afirmação pode ser a combinação de duas cada afirmação dela pode ser composta de outras subafirmações.

Então, veja como o conector “NÃO” é simples:

```
não verdade = falso
não falso = verdade
```

Ou ainda:

```
not true  = false
not false = true
```

7.4 Conector XOR – OU Exclusivo

O conector XOR, assim como os conectores AND e OR, também é um conector que conecta duas condições ou, em português, duas afirmações.

Então veja a seguir as possibilidades para o operador XOR – Ou Exclusivo:

```
true  xor true  = false
true  xor false = true
false xor true  = true
false xor false = false
```

Logo, o resultado de uma operação XOR é verdade se: Uma, e somente uma, das duas afirmações ligadas pelo conector, tiver valor verdade. É falso caso as duas afirmações sejam falsas ou as duas sejam verdadeiras.

O OR é quase o mesmo que o XOR, diferindo apenas quando as duas afirmações são verdadeiras. Isto é:

```
true or  true é true
true xor true é false
```

7.5 Conectores em pascal

Em pascal os operadores são: AND, OR, XOR e NOT que correspondem respectivamente em português a: E, OU, OU Exclusivo, NÂO.

E os valores possíveis para uma condição (afirmação), isto é: verdade e falso, correspondem respectivamente a: **true** e **false**.

Logo, veja a relação abaixo:

```
E – AND
OU – OR
NÃO – NOT
OU Exclusivo – XOR
VERDADE – TRUE
FALSO – FALSE
```

Atenção para o seguinte, há regras de precedência entre os operadores booleanos também. Logo, em uma expressão booleana, é resolvido primeiramente às sub-expressões entre parênteses e, após, são executadas as operações seguindo a seguinte ordem: primeiro os “xor”(s) depois os “and”(s) e depois os “or”(s). Veja o exemplo abaixo:

```
(2>3 or 3<4 and 4+10<13) or true xor (3>0)
```

Primeiramente resolve-se o que está entre parênteses:

```
(2>3 or 3<4 and 4+10<13)  = false or true and false = false
(3>0) = false
```

Após isto, é executado o xor:

```
true xor (3>0) = true xor false = true
```

Depois o or:

```
false or true = true
```

Logo, a expressão retorna valor true apõs sua execução!

8 Operadores de comparação

Os operadores de comparação podem fazer comparações com variáveis e constantes numéricas veja abaixo os operadores de comparação para dados numéricos:

Operador	Descrição
=	Igual
<>	Diferente
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou

Os operadores: = e <> (igual e diferente) podem ser utilizados para comparar expressões (ou variáveis ou constantes) quaisquer. Desde que, o a expressão/variável/constante do lado esquerdo do igual (ou do diferente) seja de um tipo compatível com a expressão do lado direito do igual (ou do diferente).

Veja abaixo algumas comparações válidas:

```
a=2, com a do tipo numérico
a<>b, com a e b de mesmo tipo
b>4, com b do tipo numérico
3<4
4<=x, com x do tipo numérico
x>=5, com x do tipo numérico
```

Veja algumas comparações inválidas:

```
a=b, com a de tipo diferente do tipo de b
a<>b, com a de tipo diferente do tipo de b
b>4, com b de tipo não numérico
4<=x, com x de tipo não numérico
x>5, com x de tipo não numérico
```

9 Bloco de código

Os blocos de código são quaisquer instruções executadas entre o início do bloco (begin) e o final do bloco (end). Finalizados com ponto e vírgula (;). Com exceção ao bloco de código do programa principal, isto é, o programa principal também inicia com um “begin” e finaliza com um “end”, sendo que o “end” do programa principal antecede um ponto final (.), não um ponto e vírgula (;).

Adiante você verá as seguintes instruções:

```
IF
IF, ELSE
IF, ELSE IF, ELSE
CASE,
FOR
WHILE
PROCEDURE
```

FUNCTION

As **procedures** e **functions** são subprogramas, é aí onde faz-se diferença entre um programa principal e um subprograma. E, obrigatoriamente, tanto o programa principal, quanto os subprogramas, necessitam que seu conjunto de instruções esteja entre um **begin** e **end**.

Diferente dos programas principais e subprogramas, as demais instruções citadas acima podem ter um **begin** e **end** que são obrigatórios quanto o bloco de instruções tem duas ou mais instruções e opcional quando tem apenas uma instrução.

10 Estruturas condicionais

A partir de agora, você vai aprender sobre como mudar o fluxo de execução de acordo com o resultado de uma expressão booleana (operação lógica).

10.1 Comando IF

O IF (ou SE em inglês), corresponde a instrução que recebe na sequência uma condição. Esta instrução tem um, dois ou até vários blocos de código que só são executados em caso de sua condição (expressão booleana) retornar verdade. Isto é, true. Veja abaixo a sintaxe do comando IF:

```
if ( condição ) then
  begin
      INSTRUÇÃO 1
      INSTRUÇÃO 2
      …
      INSTRUÇÃO N
  end;
```

Perceba que agora podemos, com uso do “if”, executar um conjunto de instruções inseridos dentro do begin e end, apenas em caso do resultado da condição ser verdade, isto é, em caso da condição ser satisfeita. Veja o programa abaixo:

```
program exemplo;
uses crt;
var numero : integer;
begin
      clrscr;
      write(‘Informe um numero: ‘);
      readln(numero);

      if (numero = 0) then
         begin
            writeln(‘Voce digitou zero!’);
         end;

      writeln(‘Tecle enter para sair…’);
      readln;
end.
```

O programa acima mostra uma mensagem, espera o usuário digitar um número e, após o usuário digitar, o número digitado é armazenado na variável “numero”. Depois do valor digitado pelo usuário ser armazenado em “numero”, o fluxo de execução do programa chega até a instrução IF. Ou seja, se a condição (que neste caso é: numero = 0) for satisfeita, então são executadas todas as instruções contidas no begin end do comando if. Caso a condição não seja satisfeita, então, o que está entre o begin e end do if não é executado. Depois o restante do programa é executado.

Perceba que o begin e end do if é são opcionais no caso do programa acima, dado que, dentro do begin e end, tem apenas uma instrução: um **writeln** que exibe algo na tela.

O que o programa anterior, em resumo, faz é solicitar que o usuário digite um número e, caso o usuário digite o número 0, então é mostrada a mensagem “Voce digitou zero”. Se o número digitado pelo usuário não for zero, então, esta mensagem “Voce digitou zero” não é mostrada.

10.2 Comando IF/ELSE

O comando else como parte do if pode ser traduzido para (caso contrário). Logo, seu bloco de código (com ou sem begin end) só será executado caso a condição do if seja falsa.

Veja o programa abaixo:

```
program exemplo;
uses crt;
var numero : integer;
begin
      clrscr;
      write('Informe um numero: ');
      readln(numero);

      if (numero = 0) then
         writeln('Voce digitou zero!')
      else
         writeln('Você não digitou zero!');

      writeln('Tecle enter para sair…');
      readln;
end.
```

Perceba que o programa acima é quase igual ao anterior, com exceção que os begins ends do IF e do ELSE não foram colocados, dado que há apenas uma instrução associada a cada bloco. Logo, se o usuário digitar o número zero, é mostrada a mensagem "Voce digitou zero!". Caso contrário, o bloco do else é executado, mostrando a mensagem "Voce nao digitou zero!".

Obs: perceba que o else não pode ser precedido de um ponto e vírgula (;), logo a instrução anterior a ele não deve conter (;).

10.3 Comando IF/ELSE IF

O "if" pode ser sucedido também de um "else if then". Que quer dizer: caso contrário (caso a condição do if for falsa), então, execute o bloco de código.

Veja o exemplo abaixo:

```
program exemplo;
uses crt;
var numero : integer;
begin
      clrscr;
      write('Informe um numero: ');
      readln(numero);

      if (numero = 0) then
         writeln('Voce digitou zero!')
      else if (numero = 1) then
         writeln('Você digitou o numero 1!')
      else if (numero = -1)
         writeln('Você digitou um numero negativo: o menos um');

      writeln('Tecle enter para sair…');
      readln;
end.
```

Perceba que o programa acima é quase igual ao anterior, tendo o bloco eles sido substituído pelos else ifs. Veja outro exemplo:

```
program exemplo;
uses crt;
var numero : integer;
begin
      clrscr;
      write(‘Informe um numero: ‘);
      readln(numero);

      if (numero = 0) then
         writeln(‘Voce digitou zero!’)
      else if (numero > 0) then
        begin
         writeln(‘Voce digitou um numero positivo!’);
         writeln(‘O numero positivo digitado eh o ‘,numero);
        end
      else
         writeln(‘Você digitou um numero negativo’);

      readln;
end.
```

Perceba no programa acima que o segundo “if” é um “else if” que tem um begin end. Neste caso obrigatório porque há mais que uma instrução que devem ser executadas em caso da condição ser satisfeita. E, ao final, um “else”, cujo conteúdo só é executado, caso todas as outras condições tenham sido falsas, isto é, caso nenhuma seja satisfeita.

10.4 Comando case

O comando case pode ser utilizado como alternativa ao “if” em alguns casos. Isto é, pra todo “case” há um “if” equivalente, mas, para nem todo “if”, há um “case” equivalente. Veja o exemplo abaixo de uma calculadora de áreas:

```
program Area_Case;
uses crt;
var base, altura:integer;
    area:real;
    op:char;
begin
      clrscr;
      write('Informe a base: ');
      readln(base);
      write('Informe a altura: ');
      readln(altura);
      writeln;
      writeln('(********** Menu **********)');
      writeln('  (1) Retangulo');
      writeln('  (2) Triangulo');
      writeln('  (3) Metade do triangulo');
      writeln;
      write('Informe uma opcao: ');
      readln(op);
      case (op) of
          '1': begin
                 area := base * altura;
                 writeln('A area do retangulo eh: ',area:0:2);
               end;
          '2': begin
                 area := ( base * altura ) / 2;
```

```
                writeln('A area do triangulo eh: ',area:0:2);
              end;
         '3': begin
                area := ( base * altura ) / 4;
                writeln('A metade da area do triangulo eh: ',area:0:2);
              end;
         else
           begin
             writeln('Opcao invalida.');
           end;
     end;

     writeln;
     writeln('Tecle enter para sair.');
     readln;
end.
```

Perceba que no programa acima é como se fossem utilizados “ifs” para implementar o seguinte algoritmo:

```
Se opção escolhida igual a ‘1’, calcular e exibir a área do retângulo;
Se opção escolhida igual a ‘2’, calcular e exibir a área do triângulo;
Se opção escolhida igual a ‘3’, calcular e exibir a área de metade do triângulo;
Caso contrário, exibir mensagem indicando que a opção informada é inválida.
```

Perceba também que o “begin end” do else é opcional, dado que há apenas uma instrução no bloco de código dele que corresponde a instrução que se segue após dele.

Perceba também que a variável “area” é do tipo real, não inteira. Esse tipo de dado foi escolhido para a variável “area” porque, em alguns pontos do programa, a variável “area” recebe o valor de uma expressão que envolve o operador de divisão “/”, o que significa que, o resultado não necessariamente é um número inteiro, podendo ser um número com vírgula.

Repare também como que a variável “area” é exibida no writeln: tem “:0:2” após o nome da variável. O primeiro número “0” corresponde a um deslocamento na tela (não importante entender agora), mas, o que mais importa agora mesmo é o segundo número, o “2” que indica que o número deve ser formatado com apenas duas casas decimais após a vírgula. **Obs**: em pascal, números reais são separados por ponto, não por vírgula. Logo “2,54” em pascal corresponde a “2.54”.

11 Estruturas de repetição

A linguagem pascal dispõem de três estruturas de repetição. São elas: FOR, WHILE e REPEAT/UNTIL. Traduzidas respectivamente ficariam: Para, Enquanto, Repita/Até Que.

11.1 Comando FOR

A sintaxe básica de uma instrução for é:

```
for <CONTADOR>:= <INI> to <FIM> do
  begin
    <INST 1>
    <INST 2>
      …
    <INST N>
  end;
```

Veja o exemplo abaixo:

```
program Exemplo;
uses crt;
var i : integer;
begin
      clrscr;
      for i := 1 to 10 do
          write(i,' ');
      readln;
end.
```

O programa acima apenas tem uma declaração de uma variável contador nomeada de “i” do tipo integer e utiliza o for para repetir de 1 até dez a instrução write. Entenda que nesse caso o begin end para o for é opcional. Então, após cada vez que o código de dentro do for (do bloco de código) for executado a variável “i” é incrementada em uma unidade. Logo, o programa acima apenas imprime os números de 1 até 10 separados por espaços.

Há também a possibilidade de fazer o inverso com uso do comando “downto” ao invés do “to” do “for”. Veja o exemplo abaixo:

```
program Exemplo;
uses crt;
var i : integer;
begin
      clrscr;
      for i:=10 downto 1 do
          write(i,' ');
      readln;
end.
```

O programa acima apenas imprime os números de 1 até 10 em ordem decrescente.

11.2 O comando WHILE

A sintaxe básica de uma instrução while é:

```
while (CONDICAO) do
  begin
    <INST 1>
    <INST 2>
      …
    <INST N>
 end;
```

Veja o exemplo abaixo:

```
program Exemplo;
uses crt;
var i : integer;
begin
      clrscr;
      i:=1;
      while (i<=10) do
        begin
          write(i,' ');
          i:=i+1;
        end
      readln;
end.
```

O programa acima faz exatamente a mesma coisa que o anterior. Logo, assim como os comandos IF e CASE, o que se pode fazer com FOR também se pode fazer com o WHILE como alternativa. Já o inverso não é verdade: Nem tudo que se pode fazer com WHILE se pode fazer com FOR.

11.3 O comando REPEAT

A sintaxe básica de uma instrução while é:

```
repeat
    <INST 1>
    <INST 2>
      …
    <INST N>
until (CONDICAO);
```

Perceba que o repeat repete as instruções de seu bloco de código até que a condição do until seja satisfeita. Veja o programa abaixo:

```
program Exemplo;
uses crt;
var i : integer;
begin
      clrscr;
      i:=1;
      repeat
          write(i,' ');
          i:=i+1;
      until (i>10);
      readln;
end.
```

O programa acima também, apenas imprime os números de 1 até 10.

11.4 Loop infinito

Quanto utilizamos repetições, devemos levar em conta que o programa precisa saber quando parar de repetir. Ele sabe isso verificando a condição de parada do comando de repetição, os comandos de repetição são conhecidos por formar um **laço**, ou **loop**. Mas, você poderia se perguntar: O que aconteceria se a condição nunca fosse satisfeita? A resposta é, o programa travaria executando infinitamente o bloco de código do laço. Veja o exemplo abaixo:

```
program Exemplo;
uses crt;
var i, num, fat:integer;
begin
      clrscr;
      while(2<4)do
          write('A');
      readln;
end.
```

O programa acima executará o write('A') infinitas vezes, isto é, até você conseguir interromper a execução do programa de outro modo. Isso porque, 2 sempre será menor que 4!

11.5 Exemplos de uso

11.5.1 Prog. Fatorial

O fatorial de um número é calculado da seguinte forma:

Fatorial de 4 = 4! = 4*3*2*1 = 24

Fatorial de 5 = 5! = 5*4*3*2*1 = 120

Veja abaixo como fazer um programa que imprime o fatorial de um número com uso do WHILE:

```
program Exemplo;
uses crt;
var i, num, fat:integer;
begin
      clrscr;
      write('Informe um numero: ');
      readln(num);
      fat:=1;
      i:=num;
      while ( i > 0 ) do
          begin
              fat:=fat*i;
              i:=i-1;
          end;
      writeln('O fatorial de ',num,' eh: ',fat);
      readln;
end.
```

No exemplo acima, a variável "fat" começa valendo o número informado e o contador "i" vai sendo decrementado até valer zero e, então, a execução do laço while ser finalizada.

Veja abaixo como seria com o for:

```
program Exemplo;
uses crt;
var i, num, fat:integer;
begin
      clrscr;
      write('Informe um numero: ');
      readln(num);
      fat:=1;
      i:=num;
      for i:=num downto 1 do
          fat:=fat*i;
      writeln('O fatorial de ',num,' eh: ',fat);
      readln;
end.
```

Outros modos de fazer:

```
fat:=1;
i:=1;
while (i<=num) do
  begin
      fat:=fat*i;
      i:=i+1;
  end;
```

Ou,

```
fat:=1;
i:=1;
for i:=1 to num do
   fat:=fat*i;
```

Também é possível fazer o exemplo do fatorial com REPEAT/UNTIL. **"Fica como exercício!"**

11.5.2 Prog. Série Fibbonacci

A série fibbonacci é um conjunto de números sequenciais cujo valor de cada elemento é a soma dos dois elementos anteriores.

Abaixo, os 8 primeiros elementos da série fibbonacci:

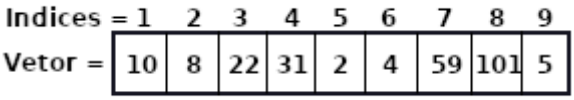
1 1 2 3 5 8 13 21

Agora vamos fazer um programa que lê o número de primeiros elementos a serem mostrados da série fibbonacci:

```
program Fibbonacci;
uses crt;
var i, atual, anterior, aux, num : integer;
begin
      clrscr;
      write('Informe o número de elementos: ');
      readln(num);
      anterior:=0;
      atual:=1;
      for i := 1 to num do
          begin
              aux:=atual;
              atual:=atual+anterior;
              anterior:=aux;
              write(atual,'  ');
          end;
      readln;
end.
```

12 Arrays unidimensionais

Arrays são listas de variáveis. Isto é, trata-se de uma variável que tem outras variáveis como parte. Claro, para essas listas terem utilidade é necessário que se possa acessar cada variável da lista e isso acontece de modo que cada variável da lista tem um número sequencial associado. Veja abaixo uma representação de um vetor de variáveis inteiras:



O vetor acima representado ocupa 9 posições de memória alocadas sequencialmente e com capacidade de armazenar inteiros.

Mas, você deve lembrar que já foi dito por aqui que um string é um conjunto de caracteres, certo? Então, o conjunto, neste caso, nada mais é do que um vetor de chars. Por exemplo, como mostrar para o usuário a terceira letra de um texto informado por ele se tendo a certeza que o texto digitado tem 3 ou mais caracteres? Simples, basta imprimir o que está na posição 3 da string. Será abordado novamente sobre esse assunto mais adiante.

Para utilizar as posições de memória do vetor como se fossem variáveis, é necessário, utilizar o índice da posição que se quer ler ou escrever um novo valor. Veja o programa abaixo:

```
program ArrayDeInteiros;
uses crt;
var
      lista:array[1..4] of integer;
      num:integer;
begin
      clrscr;

      lista[1]:=100;
      lista[2]:=4;
      lista[3]:=1930;
      lista[4]:=-549;

      num:=lista[3];

      writeln('Primeiro elemento: ',lista[1]);
      writeln('Segundo elemento: ',lista[2]);
      writeln('Terceiro elemento: ',lista[3]);
      writeln('Quarto elemento: ',lista[4]);
      writeln;
      writeln('Terceiro elemento: ',num);
      readln;
end.
```

Muito bem, vamos para um exemplo mais útil. Que tal solicitar ao usuário cinco números e armazená-los nas posições de um array de tamanho 5 e, após isto, imprimir todo o array. Vamos lá, veja o programa abaixo:

```
program Exemplo;
uses crt;
var
      lista:array[1..5] of integer;
      i:integer;
begin
      clrscr;
      for i:=1 to 5 do
          begin
              write('Informe um numero: ');
              readln(lista[i]);
          end;
      writeln;
      write('Numeros lidos: ');
      for i:=1 to 5 do
          write(lista[i], ' ');
      readln;
end.
```

O programa acima solicita que o usuário informe cinco números e, a cada iteração do for utilizado na leitura, o número lido é armazenado na devida posição do vetor. Perceba o uso da variável "i". Após a leitura e o vetor carregado com os números informados, é feita a exibição dos elementos do vetor com outro for. Perceba também que lá na declaração do vetor, entre colchetes, fica a definição do intervalo de índices do vetor. Logo, no exemplo acima, o menor índice é 1 e o maior é 5. Poderia ser de 10 à 20, -5 à 5 ou, ainda, de 'a' à 'z'.

E se quisermos alterar o programa anterior para armazenar os resultados da divisão de cada elemento lido por 2. Isto é, se o usuário informar os números: 2 4 6 15, o vetor resultado terá: 1 2 3 7.5. Perceba que uma divisão por dois pode gerar um número real (com vírgula). Logo, o vetor resultado necessita ser do tipo real. Veja o programa alterado abaixo:

```
program Exemplo;
uses crt;
var
      lista:array[1..5] of integer;
      resultados:array[1..5] of real;
      i:integer;
begin
      clrscr;
      for i:=1 to 5 do
          begin
              write('Informe um numero: ');
              readln(lista[i]);
          end;
      for i:=1 to 5 do
          resultados[i]:=lista[i]/2;
      writeln;
      write('Numeros lidos: ');
      for i:=1 to 5 do
          write(lista[i], ' ');
      writeln;
      write('Resultados= ');
      for i:=1 to 5 do
          write(resultados[i]:0:2,' ');
      readln;
end.
```

13 Arrays bidimensionais

O array bidimensional pode ser entendido como um array de arrays. Ou mesmo, uma matriz de dados. Veja uma representação de uma matriz ilustrada abaixo:

$$A=\begin{bmatrix} a1,1 & a1,2 & a1,3 \\ a2,1 & a2,2 & a2,3 \\ a3,1 & a3,2 & a3,3 \end{bmatrix}$$

Perceba que a matriz A tem seus elementos acessados por índices. A diferença entre os vetores unidimensionais, já vistos, e os vetores bidimensionais (matrizes) é que seus elementos têm associados, não apenas um, mas, dois índices. Conforme ilustrado acima. Vamos a um exemplo prático:

```
program ProgMatriz;
uses crt;
var mat:array[1..3,1..3] of integer;
begin
      clrscr;
      mat[1,1]:=1;
      mat[2,3]:=2;
      write('Informe o numero do meio: ');
      readln(mat[2,2]);

      writeln('O primeiro numero da matriz eh: ',mat[1,1]);
      writeln('O numero do meio da matriz eh: ',mat[2,2]);
      readln;
end.
```

Perceba que o array agora tem 2 intervalos. No exemplo, os dois são definidos como de 1..3. Ou seja, os índices das linhas da matriz correspondem aos índices do primeiro intervalo e os índices das colunas da matriz correspondem aos índices do segundo intervalo. Neste caso, há três linhas e três colunas. Isso pode variar conforme o que se deseje fazer. Pode ser: [1..4,2..6], [0..5,0..3], etc. Agora veja outro exemplo:

```
program ProgMat;
uses crt;
var   mat:array[1..3,1..3] of real;
      i,j:integer;
begin
      clrscr;
      for i:=1 to 3 do
        begin
          for j:=1 to 3 do
            begin
              write('Informe o elemento (',i,',',j,'): ');
              readln(mat[i,j]);
            end;
        end;
      writeln;
      writeln('Veja abaixo a matriz lida');
      writeln;
      for i:=1 to 3 do
        begin
          for j:=1 to 3 do
            write(mat[i,j]:8:2);
          writeln;
        end;
```

```
end.
```

Muito bem, perceba agora que há um aninhamento de “fors”, onde, cada for tem suas variáveis contadores variando de 1 a 3, que são os intervalos de linha e coluna da matriz. Perceba também que, na hora de imprimir os elementos da matriz, é utilizado, claro, outro for aninhado para percorrer todo os elementos da matriz. Perceba que, após o mat[i,j] do write, tem :8:2. Lembra do deslocamento ao qual o primeiro número, isto é, neste caso, o 8 representa? Pois é, esse oito indica o seguinte: é calculada 8 posições após a atual e subtraída do tamanho (em caracteres) do número ou outro tipo de variável. Em resumo, se o deslocamento é 8 e o número de caracteres do elemento é 5, então, desloca o número em (8-5), ou seja, pula 3 espaços para imprimir o número. Rode o exemplo acima e verifique o modo que a matriz é impressa. Tente alterar o programa para ler e imprimir, no mesmo formato espaçado, uma matriz de inteiros!

O deslocamento também pode ser feito para valores não do tipo real. Por exemplo:

```
writeln(‘abc’:5);
writeln(x:4);
```

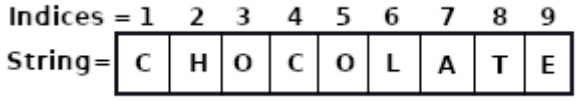
No primeiro writeln, o programa desloca a string ‘abc’ em 5 posições à direita e, desloca em 3 posições a esquerda. O que corresponde ao mesmo que deslocar 2 posições à direita. Logo, dois espaços são impressos, a partir do cursor, e antes de ser impressa a string ‘abc’.

No segundo writeln, supondo que x seja uma variável inteira de valor 10, então, seguindo o mesmo processo de impressão da string, o 10 também é deslocado em duas posições à direita.

Caso o deslocamento seja menor que o tamanho de caracteres a ser impresso, nenhum deslocamento é feito e o valor é impresso após a posição corrente do cursor.

14 Strings

Como já foi dito, uma string é um conjunto (array) de caracteres (chars). Logo, cada caractere da string tem um índice associado, iniciando de 1 até o número de correspondente ao tamanho da string. Veja abaixo:



Agora, analise o programa abaixo:

```
program StringProg;
uses crt;
var str:string;
begin
      clrscr;
      str:=’CHOCOLATE’;
      writeln(str[1],’ ‘,str[2],’ ‘,str[3],’ ‘,str[4],’ ‘,str[5]);
      str[4]:=‘0’;
      str[9]:=‘0’;
      writeln(‘String= ‘,str);
      readln;
end.
```

Você consegue entender a saída do programa acima? Primeiro, a variável string é inicializada com o valor correspondente a palavra CHOCOLATE. Após isto, assim como se pode fazer com array de inteiros, se pode acessar as letras da string pelos devidos índices. Isto é feito no primeiro writeln. São impressas as cinco primeiras letras da palavra. Após isto os caracteres das posições 4 e 9 são substituídos pelo valor ‘0’. Entenda que, neste caso, o zero não corresponde a um valor numérico, mas, ao caractere 0 que é do tipo char (perceba que o zero está entre aspas simples). Depois da alteração a nova string é impressa, isto é, CHO0OLAT0.

14.1 A função “length”

Uma função muito importante é a função length que recebe como argumento a variável string e retorna o seu tamanho. Veja o exemplo abaixo:

```
program Exemplo;
uses crt;
var frase:string;
    tamanho:integer;
begin
      clrscr;
      write(‘Informe uma frase: ‘);
      readln(frase);
      writeln(‘Frase: ‘,frase);

      tamanho:=length(frase);
      writeln(‘Tamanho da frase: ‘,tamanho);
      readln;
end.
```

Perceba que o programa acima apenas lé uma string, imprime ela e após isto atribui seu número de caractéres à variável “tamanho” e, então, imprime o tamanho da string. Rode este programa e experimente fazer alterações e rodar novamente.

Você saberia fazer o programa acima funcionar do mesmo jeito, só que sem o uso da variável “tamanho”? Dica, o writeln pode receber no lugar de uma variável, qualquer constante, expressão ou função que retorne um valor.

15 Funções matemáticas

Assim como a função length que retorna o tamanho de uma string passada como parâmetro (argumento), a linguagem pascal também suporta algumas outras funções. Inclusive, funções matemáticas. Veja abaixo algumas funções matemáticas suportadas:

15.1 Função SQR

Retorna o quadrado de um número. Veja os exemplos abaixo:

```
sqr(4) retorna 16
sqr(2.2) retorna 2.2 * 2.2 = 4.84
sqr(x) corresponde a “x” vezes “x” = (x²)
```

15.2 Função SQRT

Retorna a raiz quadrada de um número. Veja os exemplos abaixo:

```
sqrt(4) retorna 2
sqrt(x) retorna a raiz quadrada do valor armazenado em “x”
```

15.3 Função ABS

Retorna o número positivo ou convertido para positivo, em caso de ser negativo. Veja os exemplos abaixo:

```
abs(1) retorna 1
abs(-1) retorna 1
abs(0) retorna 0
abs(-2.5) retorna 2.5
```

15.4 Função EXP

Retorna o número de Euler (Constante E) elevado ao argumento passado como parâmetro para a função. Veja abaixo a função EXP relacionada a potenciação:

$$\exp(X) = E^X$$

Agora veja abaixo alguns exemplos:

```
exp(10) retorna (E elevado a 10)
exp(1) retorna (E elevado a 1), corresponde ao valor de E
exp(48.6) retorna (E elevado a 48.6)
```

15.5 Função LN

A função LN ou Logaritmo neperiano retorna o logaritmo de base correspondente ao número de Euler (Constante E) e logaritmando correspondente ao valor passado como parâmetro para a função. Veja abaixo a função LN relacionada a notação de logaritmos:

$$\ln(X) = \log_E X$$

Entenda que a constante E que é a base no logaritmo acima é o número de Euler. Agora, veja os exemplos abaixo:

ln(E) retorna 1
ln(4) retorna (o número tal que E elevado a ele seja igual a 4)

15.6 Cálculo de potência

A linguagem pascal não dispõe de uma função específica para calcular potências. No entanto, dispõe das funções exp e ln que podem ser utilizadas com a finalidade de calcular potência.

Veja as duas propriedades abaixo que podem ser aplicadas na resolução de expressões com logaritmos:

$$\log_a{}^{\log_a b} = b$$

$$\log_a X^Y = Y \log_a X$$

Com base nessas propriedades podemos levar em conta que:

```
exp(Y*ln(X)) = X elevado a Y
```

Exemplo:

```
4³ = exp(3*ln(4))=64
```

Mais na frente será abordado como você pode criar suas próprias funções, sejam elas matemáticas ou de outro tipo e, então, você poderá criar sua própria função potência conforme a informação mostrada nesta seção!

15.7 Função INT

Retorna a parte inteira de um número real, também no formato real. Veja os exemplos abaixo:

```
int(3) retorna 3.00
int(4.5) retorna 4.00
int(-0.4) retorna -0.00
```

15.8 Função FRAC

Retorna a parte fracionária de um número real. Isto é, a parte após o ponto flutuante (vírgula). Veja os exemplos abaixo:

```
frac(3) retorna 0.00
frac(4.5) retorna 0.50
frac(-0.4) returna -0.40
```

16 Procedures e functions

Subprogramas são programas encapsulados em uma estrutura de subprograma. Por exemplo, há dois tipos de subprogramas em pascal: procedures e functions. O comando writeln é um exemplo de procedure pré-definida pela linguagem pascal, assim como o procedimento readln.

16.1 Parâmetros

Os subprogramas geralmente recebem parâmetros em sua chamada. Por exemplo, veja os procedimentos writeln e readln:

```
a:=1;
b:=2;
c:=3;
d:=4;
writeln(a,b,c,d);
writeln('A=',a,'; B=',b,'; C=',c,'; D',d);

readln(a,b);
writeln(a+b,' ',d-c);
```

Qualquer chamada a writeln recebe um número qualquer de parâmetros que podem ser variáveis, constantes, expressões, chamadas a funções, etc. No caso dela, se pode passar um número variável de parâmetros, diferentemente dos subprogramas que podemos definir e nomear como quisermos.

Para entender a passagem de parâmetros em linguagens de programação assim como a linguagem pascal, é necessário entender dois conceitos chaves. Me refiro a **passagem de parâmetros** que pode ser **por valor** ou **por referência**.

Até aqui entendemos que os parâmetros podem ser: valores constantes, variáveis, expressões ou funções que retornam valores. Mas, é necessário levar em conta também que, em caso de se passar uma variável para a função ou procedimento, esta variável pode ter o seu valor alterado dentro do subprograma. E, assim, após a chamada do subprograma, essa variável pode ser manipulada com o novo valor, após o subprograma ter alterado o valor dela.

Então, é ai que entra a passagem de parâmetros por referência. É o seguinte: perceba que o procedimento readln altera o valor das variáveis passadas como parâmetro para ela. Por exemplo, após uma chamada do procedimento readln e passada uma variável qualquer, o valor informado pelo usuário é colocado na posição de memória correspondente a variável. Logo, internamente, a função readln define que a passagem de parâmetros para o procedimento deve ser por referência.

Ok, e a passagem por valor? Então, se é por referência ou por valor, isso é explicitado na declaração do procedimento ou função, conforme será visto mais adiante. Quando se passa uma constante, expressão, ou função que retorna um valor como parâmetro, essa passagem de parâmetro, certamente, é por valor. Agora, se for uma variável, se no subprograma estiver explicitado que a passagem é por valor, há a garantia que, após a execução do subprograma, a variável terá o mesmo valor de antes da chamada do subprograma. Isto é, há a garantia que o subprograma não alterará o valor da variável. Ou, ainda, caso altere, a alteração só tem efeito local. Isto é, dentro do subprograma.

16.2 Procedimentos e funções

Os procedimentos são subprogramas que não retornam valor algum em sua chamada. Seja no programa principal ou dentro de outro subprograma. Veja o exemplo abaixo:

```
program Exemplo1;
uses crt;
var a,b,c:integer;
begin
      clrscr;
      write('Informe o valor de A: ');
      readln(a);
      write('Informe o valor de B: ');
      readln(b);
      c:=(a+b)*(a-b);
      writeln('O resultado eh: ',c);
      readln;
end.
```

O programa acima não traz nenhuma novidade. Agora, vamos modularizá-lo. Isto é, quebrá-lo em subprogramas. Veja abaixo:

```
program Subprogramas;
uses crt;

procedure faz_leitura(var n1:integer; var n2:integer);
begin
      write('Informe o valor de A: ');
      readln(n1);
      write('Informe o valor de B: ');
      readln(n2);
end;

function calcula(n1,n2:integer):integer;
begin
      resultado:=(n1+n2)*(n1-n2);
end;

procedure imprime_resultado(c:integer);
begin
      writeln('O resultado eh: ',c);
end;

{VARIÁVEIS DO PROGRAMA PRINCIPAL}

var
      a,b,c:integer;

begin
      clrscr;

      faz_leitura(a,b);

      c:=calcula(a,b);

      imprime_resultado(c);
end.
```

Observe que o procedimento de nome “faz_leitura” necessita de dois parâmetros. Perceba que a declaração de parâmetros é muito similar a declaração de variáveis. Com exceção da palavra reservada “var” que indica que a passagem do parâmetro que a sucede para tal procedimento deve ser por referência, uma vez que os valores dos parâmetros são alterados dentro do subprograma com

as chamadas ao procedimento readln. Nos parâmetros sem “var”, significa que a passagem de parâmetros é por valor.

Perceba que o begin end é obrigatório e, entre a declaração do protótipo da procedure (ou function) e o begin, podem haver constantes, variáveis e definição de tipos (visto mais adiante).

Uma coisa importante de se considerar é que, os parâmetros do procedimento “faz_leitura” são, respectivamente, “n1” e “n2”. E no programa principal, os parâmetros passados como parâmetro na chamada ao procedimento “faz_leitura” são: “a” e “b”. Isso porque, os nomes não têm nada ha ver um com o outro. Apesar de os parâmetros estarem definidos como “n1” e “n2”, os valores de “a” e “b” são alterados com a execução da chamada ao procedimento. Inclusive, perceba que, intencionalmente, coloquei para o procedimento “imprime_resultado” o parâmetro com o mesmo nome da variável passada como parâmetro na chamada do procedimento. Isto é, tanto o parâmetro quanto a variável passada têm o mesmo nome: “c”. Mas, poderiam ter nomes diferentes e o comportamento seria o mesmo.

Na definição da função “calcula”, perceba que, por se tratar de uma função, há um tipo de retorno explicitado após os dois pontos do fecha parênteses. Ou seja, a função retorna um inteiro. Outra coisa a mais em se tratando de funções é que o nome da função deve ser utilizado como variável de retorno. Isto é, se sua função executa uma expressão numérica, você precisa atribuir o valor ou a própria expressão numérica a uma variável que tem o nome da função para que o valor seja retornado em uma chamada a função.

Mais uma coisa importante de se levar em conta: em alguns casos, é mais adequado colocar as leituras e escritas no programa principal, e em outros não. Por isso, as procedures “faz_leitura” e “imprime_resultado” poderiam não terem sido declaradas e seu conteúdo colocado no programa principal mesmo. Mas, para exemplificar procedimentos, functions e passagem por parâmetros por valor e por referência, foi conveniente deixar como está.

Agora vamos a outro exemplo, uma calculadora de áreas:

```
program CalcAreas;
uses crt;

function calc_area_retangulo(base,altura:real):real;
begin
      calc_area_retangulo:=base*altura;
end;

function calc_area_triangulo(base,altura:real):real;
begin
      calc_area_triangulo:=(base*altura)/2;
end;

function calc_area_circulo(raio:real):real;
begin
      calc_area_circulo:=PI*raio*raio;
end;

procedure menu;
begin
      clrscr;
      writeln('(******** MENU ********)');
      writeln;
      writeln('    (1) Quadrado');
      writeln('    (2) Retangulo');
```

```
      writeln('    (3) Triangulo');
      writeln('    (4) Circulo');
      writeln('    (5) Sair');
      writeln;
end;

var lado,base,altura,raio,result:real;
    op:char;

begin
      menu;
      write('Informe a opcao: ');
      readln(op);

      writeln;

      case (op) of
        '1': begin
               write('Informe o lado: ');
               readln(lado);

               result:=calc_area_retangulo(lado,lado);

               writeln;
               writeln('A area do quadrado eh: ',result:0:2);
             end;
        '2': begin
               write('Informe a base: ');
               readln(base);
               write('Informe a altura: ');
               readln(altura);

               result:=calc_area_retangulo(base,altura);

               writeln;
               writeln('A area do retangulo eh: ',result:0:2);
             end;
        '3': begin
               write('Informe a base: ');
               readln(base);
               write('Informe a altura: ');
               readln(altura);

               result:=calc_area_triangulo(base,altura);

               writeln;
               writeln('A area do retangulo eh: ',result:0:2);
             end;
        '4': begin
               write('Informe o raio: ');
               readln(raio);

               result:=calc_area_circulo(raio);

               writeln;
               writeln('A area do circulo eh: ',result:0:2);
             end;
        else
             begin
               writeln;
               write('Opcao invalida!');
             end;
      end;
```

```
      readln;
end.
```

O programa acima é o que já havia sido feito anteriormente. É um programa de cálculo de áreas. Perceba que foram definidas functions para os cálculos das áreas e a procedure menu não recebe nenhum parâmetro. Logo, não necessita nem dos parênteses. Seu nome antecede o ponto e vírgula. Todas as funções definidas no programa acima, recebem parâmetros por valor.

16.3 Recursividade

Agora que você já aprendeu sobre subprogramas que podem ser procedures ou functions, você compreende que é possível um subprograma chamar outro. Não é sempre que se chama apenas no programa principal. Mas, e se o subprograma chamar ele no corpo dele mesmo? Veja o exemplo abaixo:

```
program Recursividade;

procedure proc;
begin
      proc;
end;

begin
      proc;
end.
```

Você pode perceber no programa acima que a procedure de nome “proc”, está sendo chamada, tanto dentro do programa principal, quanto dentro do begin end dela mesma. A esta chamada de procedimento ou função dentro de seu bloco de código dar-se o nome de **chamada recursiva**. Inclusive, não teste o programa acima, porque, o mesmo gera um **loop infinito**. Sim, as chamadas recursivas têm um comportamento semelhante ao dos laços, embora a chamada recursiva seja bem mais complexa.

Vamos para um exemplo útil. Veja o programa abaixo:

```
program ProgFatorial;
uses crt;

function fatorial(n:integer):integer;
begin
      if (n=1) then
          fatorial:=1
      else
          fatorial:=n*fatorial(n-1);
end;

var num, fat:integer;

begin
      clrscr;
      write(‘Informe um numero: ‘);
      readln(num);

      fat:=fatorial(num);

      writeln;
      writeln(‘O fatorial de ‘,num,’ eh: ‘,fat);
```

```
      readln;
end.
```

No programa acima, a função fatorial é chamada dentro dela mesma, sempre com o valor de (n-1). Ou seja, na primeira chamada, no programa principal, é passado o número “num” que é associado ao parâmetro “n” e, então, se o “n” não for igual a 1, então, é retornado o valor de “n” vezes o resultado retornado da própria função fatorial, sendo que, com o valor de (n-1). Assim, quando essa segunda chamada for executada, então, o “n” já vale “n-1”, então, ela retona “n-1” vezes o valor retornado da chamada a fatorial(n-1) que, neste caso, como n já vale “n-1”, e passado para ela “n-1-1”, isto é, n-2. Assim, o n vai sendo decrementado até a função retornar 1. Quando o n vale 1, então, apenas a primeira instrução, após o then é executada. Isto é, a função retorna 1.

Em se tratando de recursividade, assim como nos laços, é preciso definir uma condição de parada para a execução não entrar em loop infinito. No caso da função fatorial, a condição de parada é “n=1”. Isto é, perceba que, a cada chamada recursiva, o n é decrementado, até valer 1.

Veja abaixo outro jeito de explicar a função fatorial recursiva:

Suponha N=4, então:

```
fatorial(4) = 4 * fatorial(3)
fatorial(3) = 3 * fatorial(2)
fatorial(2) = 2 * fatorial(1)
fatorial(1) = 1
```

Perceba que a fórmula do fatorial é: “n!=n*(n-1)!”. Agora, se sairmos substituindo do fim para o começo, então:

```
fatorial(1) = 1
fatorial(2) = 2 * fatorial(1) = 2 * 1 = 2;
fatorial(3) = 3 * fatorial(2) = 3 * 2 = 6;
fatorial(4) = 4 * fatorial(3) = 4 * 6 = 24
```

17 Definição de tipos

Outro recurso importante da linguagem pascal é a possibilidade de você personalizar seus próprios tipos de dados. Claro, isso, a partir dos tipos já existentes. Veja o exemplo abaixo:

```
program Exemplo;
uses crt;

type
      vetor=array[1..10] of integer;

procedure le_vetor(var v:vetor; tam:integer);
var i:integer;
begin
      for i:= 1 to tam do
          begin
              write('Informe o elemento ',i,': ');
              readln(v[i]);
          end;
end;

procedure exibe_vetor(v:vetor; tam:integer);
var i:integer;
begin
      write('Vetor= ');
      for i:= 1 to tam do
           write(v[i],'  ');
end;

var v:vetor;
begin
      clrscr;
      le_vetor(v,10);
      exibe_vetor(v,10);
      readln;
end.
```

No programa acima, assim como as declarações de variáveis, há a definição de um tipo personalizado, um tipo nomeado de vetor. Perceba que há dois procedimentos declarados. Um para leitura, que recebe uma variável do tipo vetor (tipo definido no type), e o tamanho do vetor como segundo argumento. Repare no var antes da declaração da variável "v" do tipo "vetor". O var indica que a passagem é por referência, dado que os elementos da variável "v" são acessados e alterados neste procedimento com uso do procedimento readln.

O segundo procedimento é mais simples. Apenas recebe a variável "v" (lembrando que você pode dar o nome que quiser para o parâmetro. Eu escolhi "v") e exibe seus elementos. Perceba que o type, assim como variáveis e constantes podem ser declarados em qualquer parte do programa, após o uses crt. Como o type está no início do programa, os tipos declarados nele têm **visibilidade global**, isto é, visibilidade no programa principal e em todo subprograma declarado após a declaração do type.

18 Escopo global ou local

Preste atenção ao seguinte: Se você declarar uma variável, constante ou tipo dentro de um procedure, isto é, entre a definição “procedure” e o “begin” do procedimento. Tais declarações têm escopo local. Isto é, apenas dentro do procedimento. O mesmo vale para as functions. Não obstante, caso você declare o tipo, variável ou constante após todas as procedures e functions, então, as declarações terão efeito, apenas, no programa principal. Caso você declare antes de todas as procedures e functions, então terá visibilidade global. Veja o exemplo abaixo:

```
program Escopo;
uses crt;

type inteiro=integer;   { Tipo com escopo global }
var soma,sub:inteiro;   { Variáveis com escopo global, pratica não recomendada }

procedure processa(var n1:inteiro; var n2:inteiro);
var aux:inteiro;        { Variavel de escopo local }
begin
      aux:=n1;
      n1:=n2;
      n2:=aux;

      soma:=n1+n2;       { Atribuição a variavel global }
      sub:=n1-n2;        { Atribuição a variavel global }
end;

var aux, n1, n2:inteiro;  {Variáveis com escopo local, visibilidade apenas no
                            programa principal}

                          {entenda que as variaveis aux, n1, e n2 do procedure
                           “processa” não tem relacao com estas, apesar dos
                           nomes serem os mesmos}
begin
      clrscr;
      n1:=10;
      n2:=20;
      processa(n1,n2);
      aux:=soma-sub;

      writeln(‘N1=’,n1);
      writeln(‘N2=’,n2);
      writeln(‘Soma=’,soma);
      writeln(‘Subtracao=’,sub);
      writeln(‘Aux=’,aux);
      readln;
end.
```

19 O tipo registro

Pascal permite a definição de um tipo registro, que pode agrupar variáveis ou, até mesmo, subregistros declarados internamente a ele. Cada variável que compõe o registro é chamada de campo do registro. Vamos a um exemplo:

```
program Exemplo;
uses crt;
type registro=record
       nome:string;
       peso:real;
       altura:real;
       telefone:longint;
       email:string;
     end;

var reg:registro;
begin
      clrscr;

      reg.nome:='Maria';
      reg.peso:=62.5;
      reg.altura:=1.72;
      reg.telefone:=999066483;
      reg.email:='maria@hotmail.com';

      writeln('Nome=',reg.nome);
      writeln('Peso=',reg.peso:0:2);
      writeln('Altura=',reg.altura:0:2);
      writeln('Telefone=',reg.telefone);
      writeln('E-Mail=',reg.email);
      readln;
end.
```

Perceba que o tipo registro é um record que tem 5 campos: nome, peso, altura, telefone, e-mail. O telefone é do tipo longint (inteiro longo de 32bits=4bytes). Perceba no programa principal que cada campo do registro é acessado como uma variável, através do nome do registro seguido de um ponto e o nome do campo.