



UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
PROGRAMA DE EDUCAÇÃO TUTORIAL

II CURSO DE ENGENHARIA

APOSTILA DE PROGRAMAÇÃO

Realização:



Fortaleza, Fevereiro/2010

Sumário

1.	Introdução.....	3
2.	Conceitos Básicos	3
3.	Tipos de Linguagens de Programação	4
4.	Compiladores e compilação.....	5
5.	Exemplos de Linguagens de Programação	5
6.	IDE's	7
7.	Introdução ao Pascalzim.....	10
7.1.	<i>A barra de Tarefas</i>	11
7.2.	<i>A Barra de Ajuste</i>	12
7.3.	<i>O Menu de Comandos</i>	12
7.4.	<i>A Barra de Status</i>	13
8.	Estrutura de um programa em Pascal	14
8.1.	<i>Cabeçalho do Programa</i>	14
8.2.	<i>Área de Declarações</i>	15
8.3.	<i>Corpo do Programa</i>	15
9.	Variáveis.....	15
9.1.	<i>Tipos de Variáveis</i>	16
9.2.	<i>Atribuição de Valores</i>	16
10.	Entrada e Saída de dados	17
10.1.	<i>Saída de dados</i>	17
10.2.	<i>Entrada de dados</i>	18
11.	Operadores	19
11.1.	<i>Operadores Aritméticos</i>	19
11.2.	<i>Operadores Lógicos</i>	20
11.3.	<i>Operadores de Comparação</i>	20
12.	Estruturas de Controle de Fluxo	21
12.1.	<i>Comando if ... then</i>	21
12.2.	<i>Comando If ... then ... else</i>	22
12.3.	<i>Ifs aninhados</i>	23
13.	Estruturas de Repetição	24
13.1.	<i>Comando for</i>	24
13.2.	<i>Comando While</i>	25
13.3.	<i>Comando Break</i>	26
14.	Exercícios Propostos.....	27

1. Introdução

O computador pode ser dividido em duas partes: hardware e software. O hardware engloba a estrutura física do computador, como os componentes eletrônicos e as placas. Já o software é o conjunto de todos os programas armazenados nele, a parte lógica.

Os programas são os responsáveis por permitir o computador a fazer inúmeras tarefas, como o controle de processos industriais, a execução remota de complicadas cirurgias e o gerenciamento das contas dos clientes de um banco.

Um **programa** nada mais é do que uma seqüência de instruções que possui significado para o computador.

O nosso foco será entender como criar um programa.

2. Conceitos Básicos

Uma etapa da criação do programa é a descrição deste através de ferramentas como a descrição narrativa, o fluxograma e o pseudocódigo. Essa etapa é um momento onde o **programador** vai poder desenvolver seus pensamentos de como resolver os problemas propostos.

Essa descrição dos passos e etapas que serão feitos no programa é chamada de **algoritmo** e podemos escrevê-lo através destas formas:

A descrição narrativa: escreveremos aquilo que queremos fazer assim como em uma receita de bolo.

O fluxograma: utilizaremos figuras pra descrever o programa.

O pseudocódigo: escreveremos (em português) o programa utilizando algumas regras.

Exemplo de algoritmo para mostrar a multiplicação de dois números (escrito nas três formas apresentadas):

Algoritmo em descrição narrativa:

Passo 1 – Receber os dois números que serão multiplicados

Passo 2 – Multiplicar os números

Passo 3 – Mostrar o resultado obtido na multiplicação

Algoritmo em fluxograma:

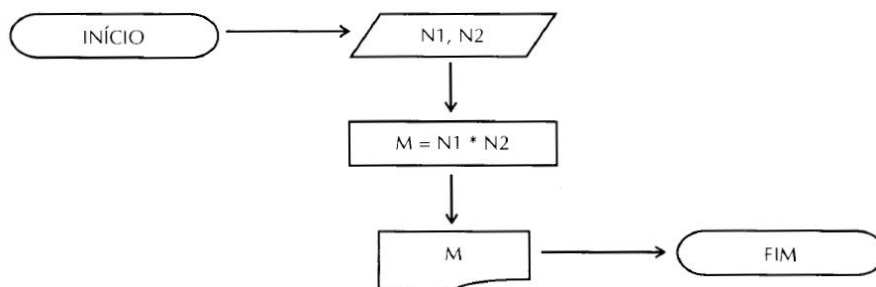


Figura 1

Algoritmo em pseudocódigo:

```

ALGORITMO
DECLARE N1, N2, M NUMÉRICO
ESCREVA "Digite dois números"
LEIA N1, N2
M ← N1 * N2
ESCREVA "Multiplicação = ", M
FIM_ALGORITMO.

```

Exemplo do Algoritmo de Euclides

Às vezes, quando lidamos com números grandes, torna-se difícil encontrar o máximo divisor comum entre os dados números. O algoritmo de Euclides ajuda-nos a encontrar o máximo divisor comum entre dois números inteiros diferentes de zero de uma forma simples e eficiente. Veja:

	3			
1128	336	120		
120				

	3	2	1	4
1128	336	120	96	24
120	96	24	0	

Figura 2

Calculando o mdc entre 1128 e 336. Divide-se 1128 por 336, escreve-se o quociente acima do 336, e o resto embaixo do 1128. Depois se repete este valor ao lado do 336, e assim por diante. Quando o resto for zero, o mdc entre os números será o número mais à direita na linha central do algoritmo, nesse caso o 24.

Quando queremos escrever (criar, desenvolver) um programa para realizar uma determinada tarefa precisamos utilizar uma linguagem que tanto o computador quanto o desenvolvedor do programa (programador) entendam. Essa linguagem é chamada de **linguagem de programação**.

Quando "traduzimos" o algoritmo para alguma linguagem de programação, estamos codificando esse algoritmo, pois a linguagem de programação possui sintaxe e semântica definidas assim como o nosso código, o Português.

O código escrito pelo programador em uma determinada linguagem é denominado **código-fonte** (source code).

3. Tipos de Linguagens de Programação

As linguagens de programação podem ser classificadas em:

- Linguagens de alto nível:** onde as instruções se assemelham ao vocabulário humano (read, print, if, then, else, etc...). Exemplo: Basic, Java, Pascal, etc.
- Linguagens de baixo nível:** onde as instruções se assemelham mais à linguagem de máquina. A linguagem de máquina é a linguagem binária. Por serem dispositivos eletrônicos, apenas

trabalham dados representados na forma de alto e baixo nível de tensão. São úteis para programar hardware. Exemplo: Assembly.

Vale ressaltar que há linguagens, como no caso da linguagem C, que se enquadram em um nível intermediário, pois apresentam sintaxe parecida com a linguagem humana mas que trabalham também com instruções de baixo nível.

As linguagens podem ser classificadas pelo paradigma que suportam (usam). Um paradigma é a maneira (modelo, jeito) que o programador vai desenvolver o seu programa. A maioria das linguagens suporta apenas um tipo de paradigma. O paradigma do Pascal, linguagem que estudaremos, é procedural, isto é, o programador irá desenvolver um programa através de blocos de comando.

Paradigma procedural: Os conjuntos de instruções são organizados em blocos.

4. Compiladores e compilação

Os computadores utilizam internamente o sistema binário. Através deste sistema, todas as quantidades e todos os valores de quaisquer variáveis poderão ser expressos através de uma determinada combinação de dígitos binários, ou seja, usando apenas os algarismos 1 e 0. O computador necessita que alguém ou algo traduza as informações colocadas no código fonte (aquele escrito pelo programador em uma determinada linguagem) para um código escrito apenas com 1 e 0. Este código escrito com o sistema binário é chamado de **código executável**.

O programa responsável por converter um código-fonte em programa executável (binário) é o **compilador**. Ao processo de conversão denominamos de **compilação**.

O tempo em que o código é transformado de código fonte escrito em uma linguagem de programação para o código em linguagem de máquina (código objeto) é denominado **tempo de compilação**. O tempo em que o programa está sendo executado é denominado **tempo de execução**.

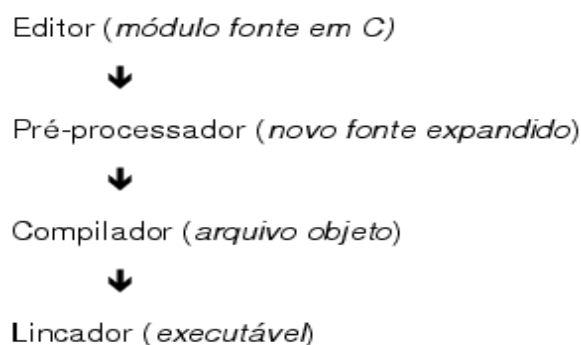


Figura 3

5. Exemplos de Linguagens de Programação

Assembly: Trata-se de uma linguagem de baixo-nível e, conseqüentemente, não estruturada. Sua vantagem está na possibilidade de controlar todos os recursos de hardware existentes (programação do processador) e no fato de gerar códigos pequenos e velozes, sendo possível utiliza o código em microcontroladores (onde a memória está na ordem de Kbytes). A desvantagem reside na complexidade do código, sendo necessária a digitação de várias linhas de código para a realização de tarefas simples. Há

uma linguagem Assembly para cada arquitetura computacional. O código é baseado em mnemônicos. Exemplo de código em Assembly:

```
section .text
    global _start

_start:

    mov     edx,len
    mov     ecx,msg
    mov     ebx,1
    int     0x80

section .data
msg     db     'Hello, world!' ,0xa
len     equ   $ - msg
```

BASIC: Linguagem de programação de alto nível, não estruturada e interpretada. Sua principal característica reside na simplicidade, daí o nome: Beginner All-purpose Symbolic Instruction Code. Originou a plataforma de desenvolvimento Microsoft Visual Basic.

```
print 'Hello, world!'
```

C: Trata-se de uma das linguagens de programação mais conhecidas do mundo. Desenvolvida por Brian Kernighan e Dennis Ritchie, é uma linguagem de médio nível e estruturada. É uma linguagem versátil, sendo utilizada para construção dos mais diversos tipos de programas, como Sistemas Operacionais. Vale ressaltar que no desenvolvimento de Sistemas Operacionais há trechos de código em Assembly.

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf ("Hello, World\n");
    return (0);
}
```

C++: Evolução da linguagem C. Sua principal diferença em relação ao C é o suporte à orientação a Objetos. Sistemas Operacionais há trechos de código em Assembly.

```
#include <iostream>

int main(void)
{
    cout << "Hello, World\n";
    return (0);
}
```

C#: Linguagem da plataforma .NET. Trata-se de uma tentativa de fazer concorrência à linguagem Java.

```
using System;
namespace HelloWorldApplication
{
    class HelloWorldApp
    {
        public static void Main()
        {
            Console.WriteLine("Olá Mundo!");
        }
    }
}
```

Fortran: A linguagem Fortran é principalmente suada em Ciência da Computação e análise numérica. Apesar de ter sido inicialmente uma linguagem de programação procedural, versões recentes de Fortran possuem características que permitem suportar programação orientada a objetos.

```
PROGRAMHELLO
PRINT*, 'Hello World!'
END
```

Java: Trata-se de uma das mais utilizadas linguagem de programação da atualidade. Trata-se de uma linguagem com suporte à orientação a objetos, de alto nível, estruturada e híbrida. Traz consigo a JVM (Java Virtual Machine), que permite que os programas desenvolvidos em Java sejam portáteis, permitindo inclusive a criação de softwares para celulares.

```
public class hello
{
public static void main (String[] args)
    {
        System.out.println("Hello, World");
    }
}
```

Python: Trata-se de uma linguagem interpretada, de alto nível, orientada a objetos e relativamente fácil de aprender. É possível, também, desenvolver aplicações para celulares.

```
print "Hello, World!"
```

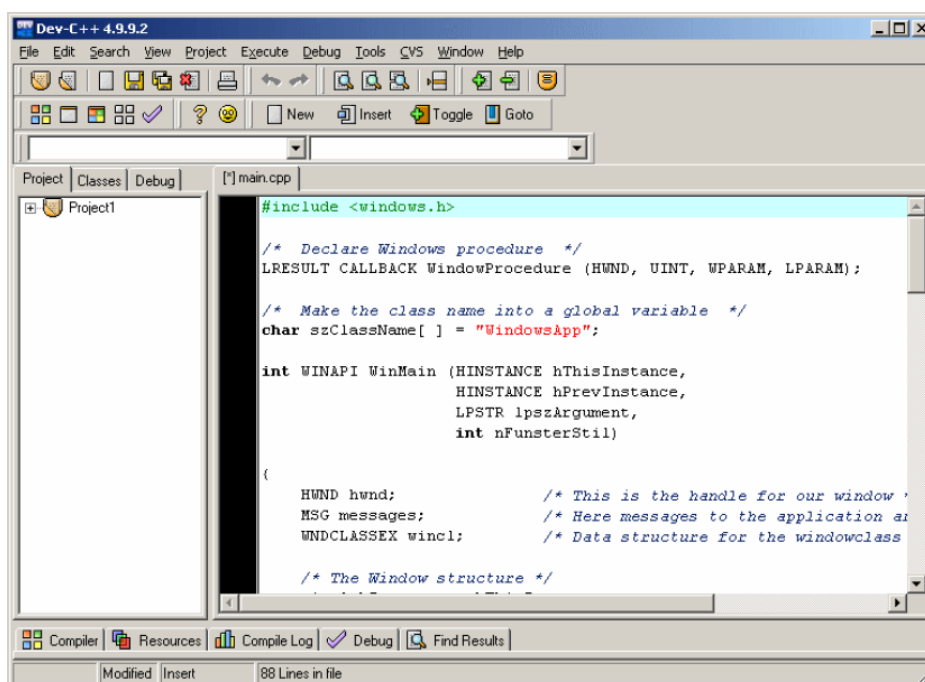
Tabela 1 - Quadro resumo

	Tipo	Nível	Paradigma
Assembly	Compilado	Baixo	Procedural
BASIC	Interpretado	Alto	Procedural
C	Compilado	Médio	Procedural
C++	Compilado	Alto	Orientado a Objetos
Java	Híbrido	Alto	Orientado a Objetos
Object Pascal	Compilado	Alto	Orientado a Objetos
Pascal	Compilado	Alto	Procedural
Python	Interpretado	Alto	Orientado a Objetos
Visual Basic	Híbrido	Alto	Orientado a Objetos

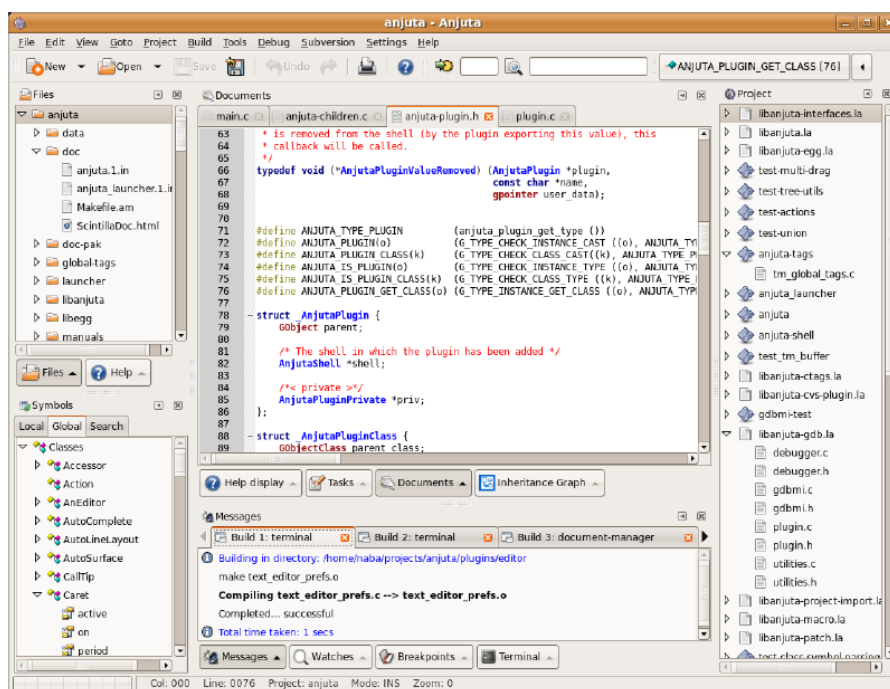
6. IDE's

IDE's (Integrated Development Environment – Ambiente de Desenvolvimento Integrado) são softwares ou pacotes de softwares que facilitam a tarefa de programação. Geralmente contam com um editor de texto (com recursos de ressaltar a sintaxe por meio de cores, identificação de erros, identificação automática, autocompletar, etc.), depurador compilador e linker. O uso de IDE's permite implementação do modelo Rapid Application Development (RAD) ou Desenvolvimento Rápido de Aplicação (em português), que é um modelo de processo de desenvolvimento de software interativo e incremental que enfatiza um ciclo de desenvolvimento extremamente curto (entre 60 e 90 dias). Exemplos:

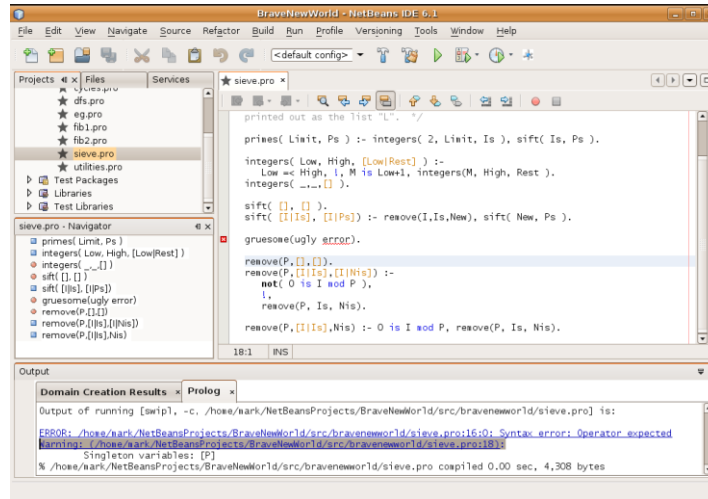
DEV C++: IDE livre voltado para a linguagem C/C++ para a plataforma Microsoft Windows.



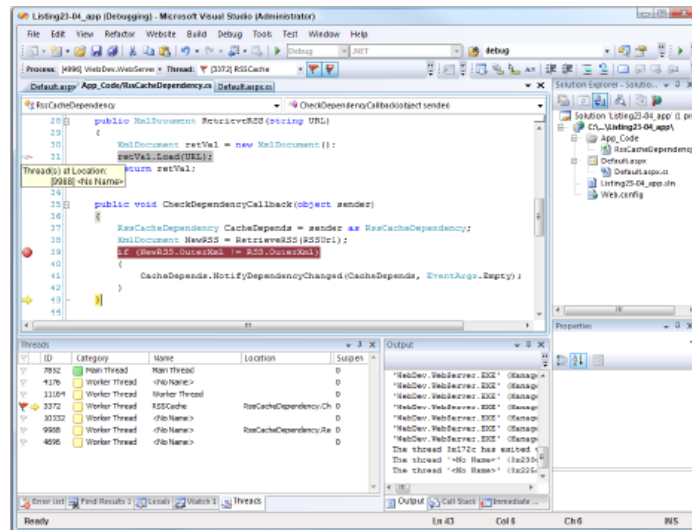
Anjuta: Semelhante ao Dev C+, mas para a plataforma GNU/Linux.



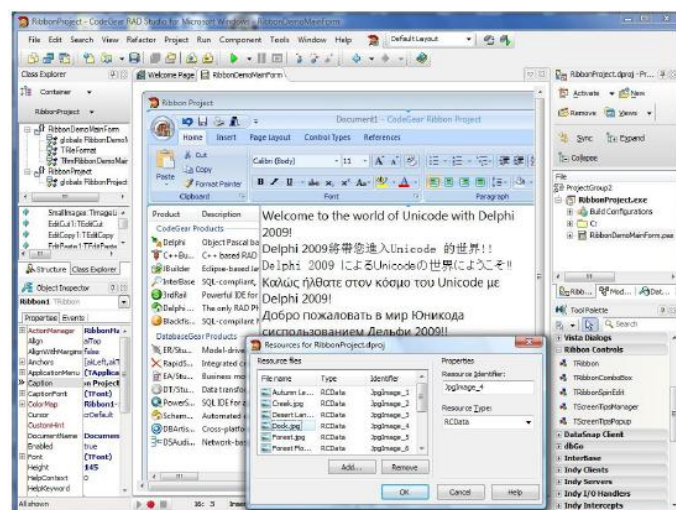
NetBeans: Atualmente é uma das melhores IDE's existentes. Além de ser livre, contém diversos recursos e embora seja muito difundida entre programadores Java, tem suporte para as linguagens C, C++, Assembly, Python, além de suporte para UML, PHP, XML e para desenvolvimento SOA. Há versões tanto para GNU/Linux como para Microsoft Windows.



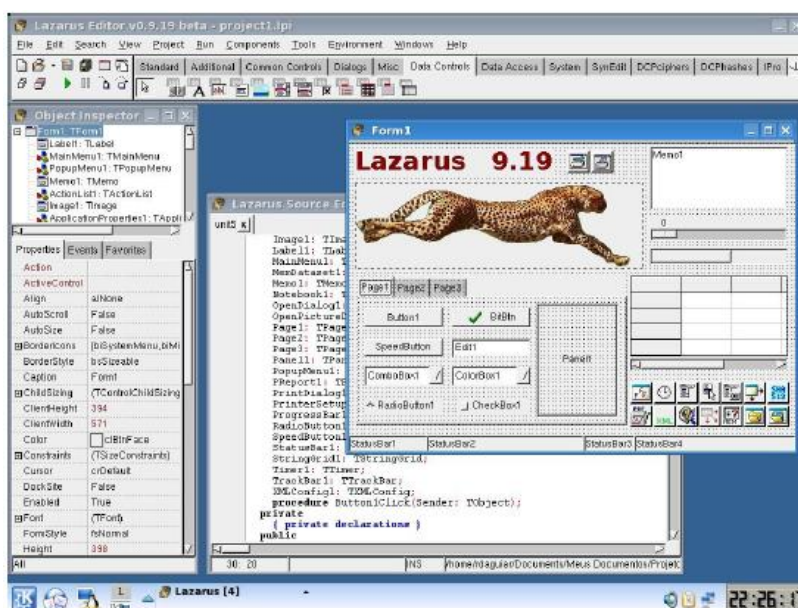
Visual Studio: Pacote proprietário da Microsoft voltado para a plataforma .NET. Contêm os programas: Visual Basic, Visual C++ e Visual C#.



Delphi: IDE proprietária da Borland para a linguagem Delphi (Object Pascal). Plataforma Microsoft Windows.



Lazarus: IDE livre de Linguagem Delphi (Object Pascal). Plataforma Microsoft Windows e GNU/Linux.



7. Introdução ao Pascalzim

O Pascalzim é um compilador gratuito para a linguagem Pascal que foi desenvolvido na Universidade de Brasília (UnB) e é utilizado em várias disciplinas de introdução à programação na UnB e outras universidades.

O compilador Pascalzim compõe-se de duas telas, uma apresenta a barra de tarefas, o editor de textos e da barra de status. Quando o programa é carregado. Já apresenta no editor uma inicialização do programa identificando o nome do programa e o seu início (BEGIN) e término (END) e na segunda tela, é mostrada a saída do programa no formato (.EXE).

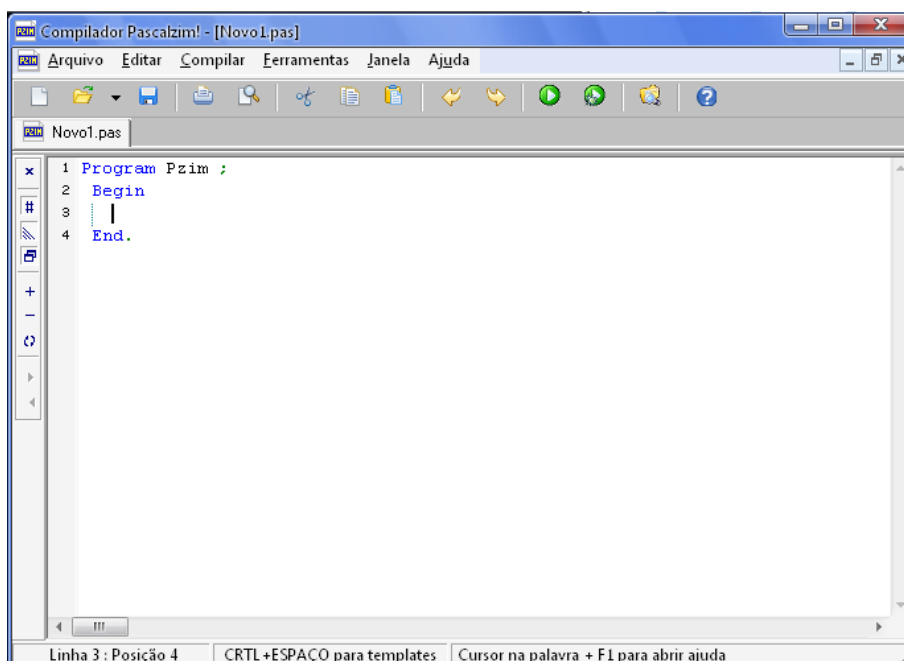


Figura 4

7.1. A barra de Tarefas

Contém os comandos mais utilizados no Pascalzim (estes comandos também podem ser acessados pelo menu ou por atalhos no teclado).

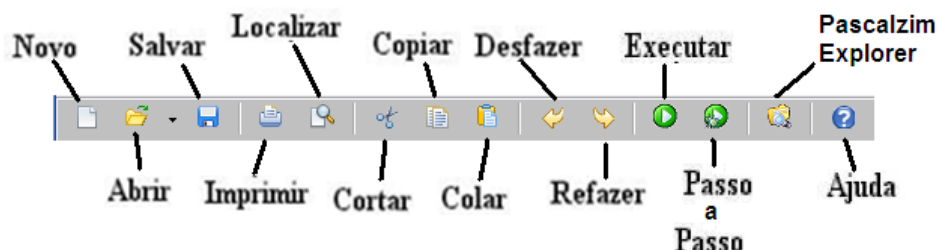


Figura 5

Abrir (Ctrl+A): Abre um arquivo anteriormente gravado, substituindo o texto presente. Se este tiver sido modificado, o Pascalzim pedirá para salvá-lo antes que seja sobreposto.

Novo (Ctrl+O): Cria uma nova tela inicializando o código, texto presente no editor. Se este tiver sido modificado, o Pascalzim pedirá sua confirmação para salvá-lo antes que seja sobreposto.

Salvar (Ctrl+S): Grava imediatamente o texto presente no editor. Na primeira vez que um novo texto é gravado, o Pascalzim pede seu nome e sua localização.

Imprimir (Ctrl+P): Imprime imediatamente na impressora padrão o texto presente no editor. Para configurar a impressão, use o comando Imprimir do menu Arquivo.

Localizar (Ctrl+L): Localiza no texto presente no editor determinada palavra especificada.

Cortar (Ctrl+X): Apaga o texto selecionado, armazenando-o em uma área de transferência.

Copiar (Ctrl+C): Copia o texto selecionado para a área de transferência.

Colar (Ctrl+V): Copia o texto selecionado para a área de transferência.

Desfazer (Ctrl+Z): Desfaz o último comando efetuado.

Refazer (Ctrl+R): Refaz o último comando desfeito.

Executar (F9): Inicia (ou continua) a execução automática do código apresentado no editor.

Passo a Passo: Inicia (ou continua) a execução linha por linha do código, dando ao usuário a oportunidade de acompanhar o fluxo de execução, os valores das variáveis e a pilha de ativação dos subprogramas.

Pascalzim Explorer: Procura nas unidades de disco os programas existentes e permite executá-los e/ou modificá-los.

Ajuda (F1): Possibilita acesso às páginas de ajuda e às informações sobre o Pascalzim.

7.2. A Barra de Ajuste

Localizada no lado esquerdo da tela contém comandos que permitem habilitar ou desabilitar algumas funções que afetam diretamente o editor.

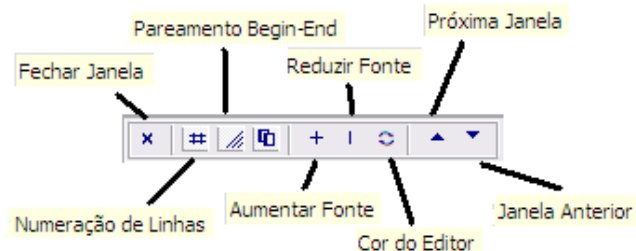


Figura 6

Fechar Janela (Ctrl+F4): Fecha a janela ativa.

Numeração de Linhas: Habilita/Desabilita a numeração de linhas.

Pareamento Begin-End: Habilita/Desabilita o pareamento Begin-End.

Aumentar Fonte: Aumentar a fonte na janela ativa.

Reduzir Fonte: Reduz a fonte na janela ativa.

Cor do Editor: Troca a cor do editor.

Próxima Janela (Ctrl+PageUp): Exibe a próxima janela da lista.

Janela Anterior (Ctrl + PageDown): Exibe a janela anterior da lista.

7.3. O Menu de Comandos

O Pascalzim possui um menu de comandos com seis opções que possibilitem executar diversas tarefas operacionais. Você poderá ter acesso a esse menu de três formas diferentes:

A primeira pode ser conseguida com o pressionamento da tecla de função <F10> e em seguida usando as teclas setas para movimentar o cursor sobre as opções desejadas.

A segunda forma pode ser conseguida com o pressionamento da tecla <ALT> + a letra que estiver grifada em maiúsculo, que é a primeira letra de cada opção do menu.

A terceira forma poderá ser conseguida com a utilização de um mouse, cujo ponteiro deverá se posicionado sobre a opção desejada e em seguida ser dado um clique.

Para sair do menu ou de qualquer caixa de diálogo que venha a se acionada basta pressionar a tecla <ESC>. O menu do Pascalzim apresenta os seguintes comandos a saber:

- **Arquivo**

Esta opção possibilita executar operações de controle com os arquivos. Desta forma é possível: Criar uma janela de trabalho (Novo), abrir um programa existente (Abrir), salvar um programa em disco (Salvar), salvar um programa em disco com outro nome (Salvar Como), fechar a janela ativa (Fechar), fechar todas as janelas ativas (Fechar Todos), imprimir o arquivo da janela ativa (Imprimir) e Sair do comando Arquivo (Sair).

- **Editar**

Esta opção possibilita executar operações de editor do programa, sendo possível remover, movimentar e copiar vários textos que estejam selecionados. Desta forma, é possível: Desfazer (Desfazer) e refazer (Refazer) operações efetuadas com a edição, Remover o texto previamente selecionado (Cortar), copiar um texto selecionado do editor para uma área de transferência (Copiar), copiar um texto da área de transferência para o editor (Colar), selecionar todo o texto pertencente ao editor (Selecionar Tudo), localizar uma sequência de caracteres por outra (Localizar), substituir uma sequência de caracteres por outra (Substituir) e mover o cursor para uma linha previamente selecionada (Ir para Linha).

- **Compilar**

Esta opção possibilita compilar o programa. Desta forma é possível: Compilar o programa da janela ativa (Executar!), compilar o programa passo a passo inclusive as suas sub-rotinas existentes e, também, gerar um arquivo executável do Ms-Dos (Gerar Executável).

- **Ferramentas**

Esta opção possibilita a utilização de ferramentas configuradas pelo usuário. Desta forma é possível: Acessar o banco de dados de programas existentes em disco (Pascalzim Explorer), acessar o bloco de notas do Windows (Bloco de Notas), acessar a calculadora (Calculadora), acessar a internet através do navegador Internet Explorer (Internet Explorer), acessar o Prompt de comandos do Ms-Dos (Prompt do MS-DOS) e também permite o acesso ao Windows Explorer (Windows Explorer).

- **Janela**

Este comando possibilita o controle das janelas que estejam abertas. Desta forma é possível: Configurar a cor do Editor para o estilo Clássico azul (Estilo Clássico), estilo escuro (Estilo Dark do Pascal), estilo branco (Estilo Moderno), além disso, é possível ordenar as janelas em cascata (Cascata), dividir as janelas horizontalmente (Dividir Horizontalmente) ou em lado a lado (Lado a Lado).

- **Ajuda**

Este comando permite executar o modo ajuda do Pascalzim. O modo de ajuda poderá ser executado de qualquer parte do programa com a tecla de função <F1> ou <Ctrl + F1> para visualizar aplicações de instruções que estejam marcadas com o posicionamento do cursor sobre elas.

7.4. A Barra de Status

Situada na parte inferior da tela, esta barra contém três painéis: o primeiro mostra a linha e a coluna onde o cursor está, o segundo mostra a palavra (Ctrl+Espaço para templates), onde indica as

palavras-chaves da linguagem Pascal e o terceiro painel disponível indica a função de cada ícone pertencente tanto ao menu quanto as barras de tarefas e de ajuste.

8. Estrutura de um programa em Pascal

Um programa em Pascal é composto, basicamente, de três partes. São elas:

- Cabeçalho do Programa;
- Área de Declarações;
- Corpo do Programa.

Vejamos, na figura a seguir como essas partes são distribuídas em um programa:

```

1 PROGRAM Multiplosde3e5; } Cabeçalho do programa
2
3
4 VAR c,n,mult3,mult5:INTEGER;
5     car:CHAR; } Area de Declarações
6
7
8 BEGIN
9     c:=0;
10    WRITELN('NUM      MULT.3      MULT.5');
11    FOR n:=10 TO 1000 DO
12    BEGIN
13        mult3:=n MOD 3;
14        mult5:=n MOD 5;
15        IF (mult3=0) AND (mult5<>0) THEN WRITELN(n,'      X');
16        IF (mult5=0) AND (mult3<>0) THEN WRITELN(n,'      X');
17        IF (mult3=0) AND (mult5=0) THEN WRITELN(n,'      X      X');
18        IF ((mult3=0) AND (mult5=0)) OR (mult3=0) OR (mult5=0) THEN c:=c+1;
19        IF c=23 THEN BEGIN
20            WRITE('c para continuar ');
21            REPEAT
22                READLN(car)
23            UNTIL car='c';
24            c:=0;
25            WRITELN('NUM      MULT.3      MULT.5')
26        END;
27    END;
28    READLN
29 END.
```

Figura 7 – Um programa escrito em Pascal

Obs: Ao escrever um programa em Pascal, devemos sempre fazê-lo nessa ordem (Cabeçalho – Declarações – Corpo), caso contrário o compilador mostrará uma mensagem de erro e o programa não será construído.

8.1. Cabeçalho do Programa

Esta área é utilizada para se fazer a identificação do programa com um nome. O cabeçalho de um programa é constituído pela instrução **PROGRAM** seguida de um nome. Ao final do nome deve-se colocar o símbolo ponto-e-vírgula (;).

Em nosso exemplo, temos:

```
PROGRAM Multiplosde3e5;
```

8.2. Área de Declarações

Esta área é utilizada para validar o uso de qualquer tipo de identificador que não seja predefinido, estando dividida em sete sub-áreas: **uses**, **label**, **const**, **type**, **var**, **procedure** e **function**.

No II Curso Pré-engenharia, vamos estudar apenas a sub-área **var**. Esta é utilizada na declaração das variáveis que serão utilizadas durante a execução de um programa, assim como o seu tipo. Estudaremos declaração de variáveis no capítulo seguinte.

8.3. Corpo do Programa

Nessa área escreveremos nossos algoritmos utilizando as funções da linguagem Pascal e as declarações feitas na área anterior. Aqui está o programa propriamente dito, isto é, a seqüência de instruções que daremos à máquina para que ela crie um programa que execute as ações que desejamos.

Iniciamos o corpo do programa com a instrução **BEGIN** e finalizamos com a instrução **END** seguida de um ponto (.). O uso dessas instruções caracteriza o que chamamos de **bloco**.

A estrutura do corpo do programa pode ser exemplificada pela figura abaixo.

```
1 BEGIN
2
3  instruções
4
5  (...)
6
7  END.
```

Figura 8 – corpo do programa

9. Variáveis

Em programação, variáveis são regiões da memória do computador previamente identificadas, que têm por finalidade armazenar informações (dados) de um programa temporariamente. Podem ser vistas como um papel, inicialmente em branco, onde se podem escrever valores (sejam numéricos, de texto ou lógicos) no decorrer da execução do programa.

Uma variável pode armazenar apenas um valor por vez. Sendo considerado como valor o conteúdo de uma variável, este valor está associado ao tipo de dado da variável.

Para declarar variáveis na linguagem Pascal, devemos obedecer à seguinte sintaxe:

```
1 VAR
2
3 Variável1, Variável2 ... : tipo-de-dado1;
4
5 VariávelN, VariávelN+1 ... : tipo-de-dado2;
6
7 (...)
```

Figura 9 - Declaração de Variáveis

Por exemplo:

```
1 VAR
2
3 Soma, Total, Salario : REAL;
4
5 Idade, Contador : INTEGER;
6
7 Nome : STRING;
```

Figura 10 - Exemplo de declaração de variáveis

Nesse exemplo estamos declarando as variáveis de nome Soma, Total e Salário como sendo do tipo **REAL**; Idade e Contador como sendo do tipo **INTEGER**; e Nome como sendo do tipo **STRING**.

9.1. Tipos de Variáveis

Veremos aqui os principais tipos de variáveis que são utilizados na linguagem Pascal.

Tabela 2 - Principais tipos de variáveis em Pascal

Tipo	Descrição	Tamanho
Boolean	Variáveis desse tipo podem armazenar apenas dois valores distintos: True ou False (Verdadeiro ou Falso).	8 bits (1 byte)
Char	Variáveis desse tipo podem armazenar apenas um caractere .	8 bits (1 byte)
Byte	Armazena valores inteiros positivos entre 0 e 255, inclusive.	8bits (1 byte)
Integer	Armazena valores inteiros positivos ou negativos entre -32768 e 32767.	16 bits (2 bytes)
Longint	Armazena valores inteiros positivos ou negativos entre -2147483648 e 2147483647.	32 bits (4 bytes)
Shortint	Armazena valores inteiros positivos ou negativos entre -128 e 127.	8 bits (1 byte)
Word	Armazena valores inteiros positivos entre 0 e 65535.	16 bits (2 bytes)
Double	Armazena valores reais. Possui precisão de 15 dígitos.	8 bytes
Real	Armazena valores reais. Possui precisão de 11 dígitos.	6 bytes
String	Armazena cadeias de caracteres (palavras ou frases).	Depende do valor da variável

9.2. Atribuição de Valores

Para atribuir valores a uma variável, utilizamos a seguinte construção:

Nome-da-Variável := Valor;

Vejamos este exemplo:

```
1 PROGRAM AtribuiValores;
2
3 VAR
4 Nome : String;
5 Caractere : Char;
6 NumeroReal : Real;
7 NumeroInteiro : Integer;
8
9 BEGIN
10
11     Nome := 'Pré Engenharia';
12     Caractere := 'c';
13     NumeroReal := 3.14159;
14     NumeroInteiro := 2010;
15
16 END.
```

Figura 11 - Atribuindo valores a variáveis

Esse programa declara algumas variáveis e atribui valores a elas. Observe que quando trabalhamos com caracteres ou string, o valor atribuído deve estar entre aspas simples (' '). Caso contrário o resultado não sairá como o esperado.

Observe também que esse programa não exibe nenhuma mensagem na tela do computador. Ele apenas atribui valores às variáveis e finaliza sem qualquer mensagem. No capítulo seguinte aprenderemos como fazer isso.

10. Entrada e Saída de dados

Aqui começaremos a apresentar os primeiros comandos da linguagem Pascal. É muito importante prestar atenção na sintaxe dos comandos, pois pequenos erros farão com que o compilador exiba uma mensagem de erro e o programa não será criado.

Entrada e saída de dados são fundamentais em todos os programas criados, pois estabelecem uma comunicação entre a máquina e o usuário.

10.1. Saída de dados

Em Pascal, a saída de dados é feita utilizando o comando `writeln()`. Esse comando imprime alguma mensagem na tela do computador. Vejamos sua sintaxe:

```
Writeln('Mensagem 1', variável 1, ...);
```

Vejamos abaixo um programa que apenas exibe uma mensagem na tela:

```
1 PROGRAM OlaMundo;  
2  
3 BEGIN  
4  
5 writeln('Olá Mundo!');  
6  
7 END.
```

Figura 12 - Olá Mundo em Pascal

Compilando esse programa obtemos:

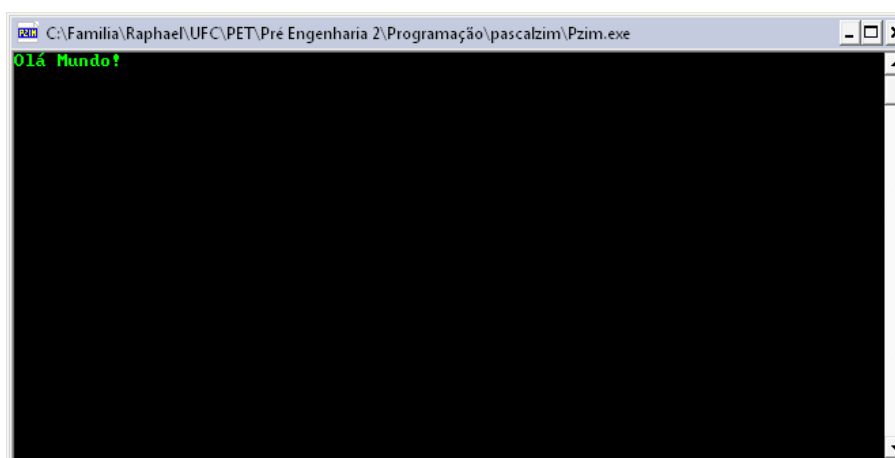


Figura 13 - Programa rodando

Como vimos, é muito simples exibir mensagens simples na tela do computador. Agora, se quisermos exibir mensagens juntamente com valores de variáveis?

Vejamos esse exemplo:

```
1 PROGRAM MostrandoValores;
2
3 VAR
4 Nome1 : String;
5 Caractere : Char;
6 Ano : Integer;
7
8 BEGIN
9
10     Nome1 := 'Hexa';
11     Caractere := 'é';
12     Ano := 2010;
13
14     writeln('O Brasil ', Caractere, ' penta! Rumo ao ', Nome1, ' em ', Ano, '!');
15
16 END.
```

Figura 14 - Exibindo mensagens com variáveis

Aqui estamos misturando textos com valores armazenados em variáveis. Quando compilamos esse programa ele gera o seguinte resultado:

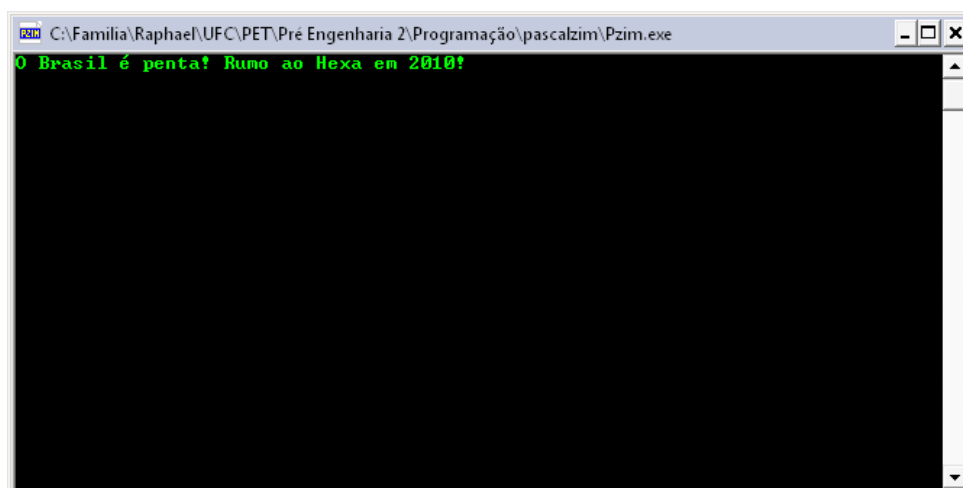


Figura 15 - Programa rodando

Repare: onde não se usou aspas simples, dentro do *writeln*, o que foi mostrado na tela foi o valor contido na variável, e não o seu nome. Exemplo: onde seria exibido “Nome1”, exibiu-se “Hexa”, que era o valor contido na variável.

10.2. Entrada de dados

Em Pascal, a entrada de dados é feita utilizando o comando *readln()*. A sintaxe desse comando é a seguinte:

readln(Variável1, Variável2, ...);

Nesse caso, o programa vai parar sua execução até que o usuário digite algum valor. Ao digitar o valor e pressionar a tecla ENTER, o valor digitado será armazenado na *Variável1* e o programa esperará

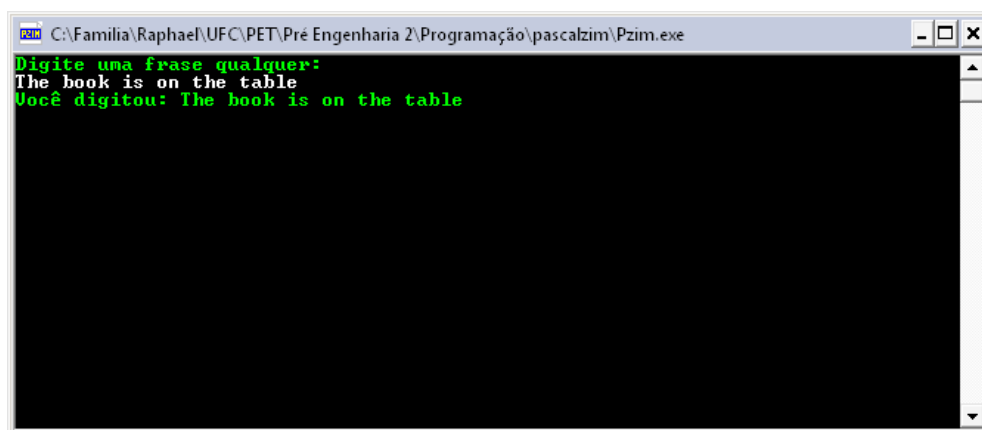
novamente por um valor, mas o armazenará na *Variável2*. Isso se repete até que todas as variáveis listadas estejam preenchidas com algum valor.

Vejamos um programa que utiliza entrada de dados pelo usuário:

```
1 PROGRAM Entrada;  
2  
3 VAR  
4 frase : String;  
5  
6 BEGIN  
7  
8     writeln('Digite uma frase qualquer: ');  
9     readln(frase);  
10    writeln('Você digitou: ', frase);  
11  
12 END.
```

Figura 16 - Programa que trabalha com entrada de dados

Que gera o seguinte resultado:



```
C:\Familia\Raphael\UFC\PET\Pré Engenharia 2\Programação\pascalzim\Pzim.exe  
Digite uma frase qualquer:  
The book is on the table  
Você digitou: The book is on the table
```

Figura 17 - Programa funcionando

11. Operadores

Os operadores, como o nome sugere, nos permitem realizar operações entre variáveis numéricas. Essas operações podem ser algébricas, lógicas ou de comparação. Dessa forma, existem operadores diferentes para cada tipo. Veremos a seguir.

11.1. Operadores Aritméticos

Resumiremos os operadores aritméticos na seguinte tabela:

Tabela 3 - Operadores aritméticos

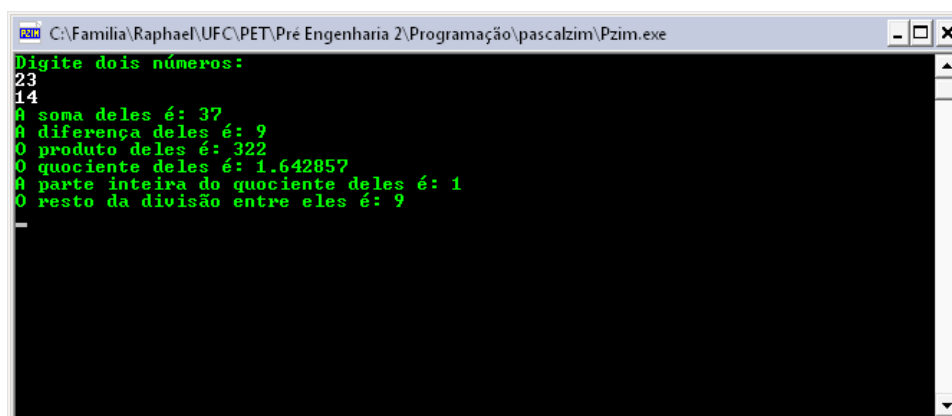
Operador	Função
+	Soma
-	Subtração
*	Multiplicação
/	Divisão simples
div	Divisão inteira
mod	Resto da divisão inteira

Um exemplo da utilização desses operadores pode ser observado no programa abaixo.

```
1 PROGRAM Operacoes;
2
3 VAR
4 Num1, Num2 : Integer;
5
6 BEGIN
7
8     writeln('Digite dois números: ');
9     readln(Num1,Num2);
10
11     writeln('A soma deles é: ', Num1+Num2);
12     writeln('A diferença deles é: ', Num1-Num2);
13     writeln('O produto deles é: ', Num1*Num2);
14     writeln('O quociente deles é: ', Num1/Num2);
15     writeln('A parte inteira do quociente deles é: ', Num1 div Num2);
16     writeln('O resto da divisão entre eles é: ', Num1 mod Num2);
17
18
19 END.
```

Figura 18 - Operações aritméticas

Gerando o seguinte resultado:



```
C:\Familia\Raphael\UFC\PET\Pré Engenharia 2\Programação\pascalzim\Pzim.exe
Digite dois números:
23
14
A soma deles é: 37
A diferença deles é: 9
O produto deles é: 322
O quociente deles é: 1.642857
A parte inteira do quociente deles é: 1
O resto da divisão entre eles é: 9
```

Figura 19 - Programa funcionando

11.2. Operadores Lógicos

Os operadores lógicos estão resumidos na tabela a seguir.

Tabela 4 - Operadores Lógicos

Operador	Função
NOT	Negação
AND	Conjunção (e)
OR	Disjunção (ou)

A utilização desses operadores ficará mais evidente no estudo de Estruturas de controle de fluxo.

11.3. Operadores de Comparação

Os operadores de comparação estão resumidos na tabela a seguir.

Tabela 5 - Operadores de Comparação

Operador	Função
=	Igualdade
<>	Diferença
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual

Assim como nos operadores lógicos, a utilização dos operadores de comparação ficará mais evidente no estudo das Estruturas de controle de fluxo.

12. Estruturas de Controle de Fluxo

Essas estruturas, também chamadas de Estruturas de Decisão, garantem que alguns comandos só sejam executados se uma ou mais condições sejam satisfeitas.

Exemplo: só podemos calcular a área de um triângulo de lados de comprimento x , y e z se $x < y + z$, $y < x + z$ e $z < y + x$. Caso contrário, é aconselhável que o programa avise ao usuário de que não existe um triângulo para aqueles valores de comprimentos dos lados.

12.1. Comando *if ... then*

Esta estrutura define se um comando ou bloco de comandos (de ações) será realizado pelo programa, a partir da verificação de uma ou mais condições. Sua sintaxe é a seguinte:

```
If condição then
  begin
  ..... bloco de comandos;
  end;
```

A condição é determinada utilizando-se operadores lógicos, matemáticos e de comparação. O mecanismo desta estrutura é simples: se o valor da condição for verdadeiro (*true*), o bloco de comandos será executado. Caso contrário, o programa ignorará o bloco de comandos contido no interior do *if ... then*.

Exemplo de programa em que se utiliza da estrutura de decisão *if ... then*:

```
1 Program ComparacaoEntreNumeros;
2
3 var  num1,num2 : integer;
4
5 Begin
6     write('Digite dois números inteiros: ');
7     readln(num1,num2);
8
9     If (num1 > num2) then
10        begin
11            ..... writeln('Sem dúvidas ', num1, ' é maior que ', num2);
12        end;
13 End.
```

Figura 14 – Uma aplicação básica do comando *if ... then*.

Compilando, obtemos o seguinte:



Figura 15 – Programa em execução.

Repare: se tivéssemos digitado os mesmo números, mas em ordem inversa, o programa não teria feito mais nada (Questione-se: por quê?).

Observação: num mesmo comando *if ... then* pode-se ter várias condições a serem satisfeitas (pense no exemplo dos lados do triângulo).

12.2. Comando *If ... then ... else*

Na maioria dos casos, quando verificamos uma condição, queremos que uma ação seja tomada no caso da condição ser satisfeita, mas também queremos que outra ação seja tomada no caso da condição não ser verificada. Para isso existe a estrutura supracitada, cuja sintaxe é a que segue (atenção à ausência de ponto-e-vírgula antes do comando *else*):

```
If condição then
begin
    comando (ou bloco de comandos);
end
Else
begin
    comando (ou bloco de comandos);
end;
```

Exemplo: se fazemos um programa para verificar se um número é múltiplo de outro, queremos que o programa diga que é, caso o resto da divisão de um pelo outro seja zero, mas também queremos que o programa diga que não é, caso o resto da divisão seja diferente de zero. Veja:

```
1 Program Multiplos;
2
3 var num1, num2: integer;
4
5 Begin
6     write('Escreva um número inteiro: ');
7     readln(num1);
8
9     write('Escreva outro. Diremos se o primeiro é multiplo do segundo: ');
10    readln(num2);
11
12    If (num1 mod num2 = 0) then
13        begin
14            writeln(num1, ' é multiplo de ', num2);
15        end
16    Else
17        begin
18            writeln(num1, ' não é multiplo de ', num2);
19        end;
20 End.
```

Figura 16 – Aplicação simples do comando *if ... then ... else*.

Então, temos o seguinte:

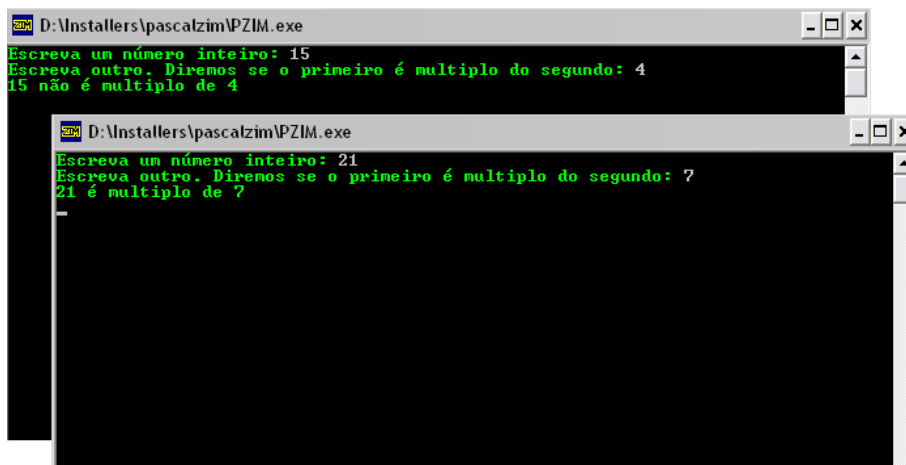


Figura 17 – Dois exemplos do programa em execução.

12.3. *Ifs aninhados*

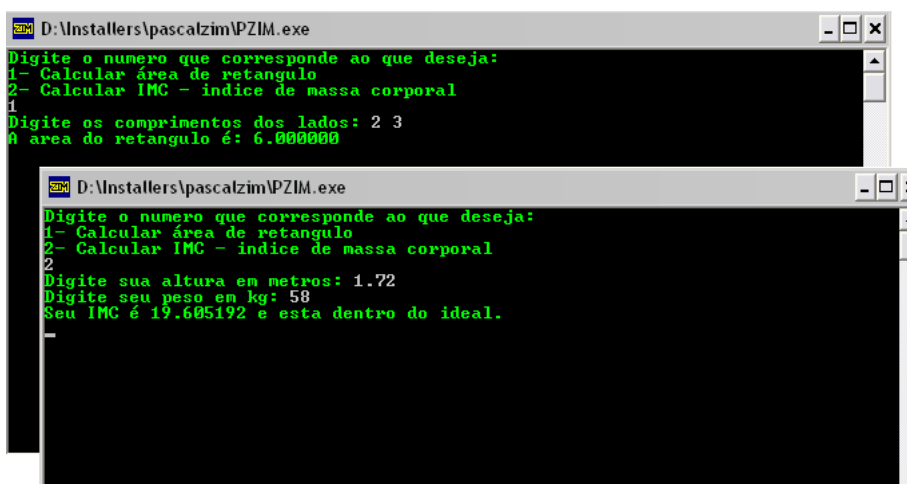
Dependendo de sua complexidade, um programa pode utilizar o comando *if ... then* inúmeras vezes no seu código-fonte, inclusive um dentro de outro, o que chamamos de *ifs aninhados*. Ao utilizar, deve-se apenas atentar para não cometer erros de lógica, uma vez que a sintaxe permanece a mesma. Vejamos um exemplo (não se assuste com a extensão do programa):

Daqui pra frente é bastante útil se observar a linha tracejada que une *begin* e *end* correspondentes, de modo a saber a que condição cada bloco de comandos está sujeito. A partir do próximo capítulo a atenção a isto será ainda mais importante.

```
1 Program CalculosEscolhidos;
2
3 var x : integer;
4     a, b, area, IMC, h, peso : real;
5
6 Begin
7     writeln('Digite o numero que corresponde ao que deseja:');
8     writeln('1- Calcular área de retangulo');
9     writeln('2- Calcular IMC - indice de massa corporal');
10    readln(x);
11
12    if x=1 then
13    begin
14        write('Digite os comprimentos dos lados: ');
15        readln(a,b);
16        area := a*b;
17        write('A area do retangulo é: ',area);
18    end
19 else
20 begin
21     write('Digite sua altura em metros: ');
22     readln(h);
23     write('Digite seu peso em kg: ');
24     readln(peso);
25     IMC := peso/(h*h);
26     if (IMC>19) and (IMC<28) then
27     begin
28         writeln('Seu IMC é ',IMC,' e esta dentro do ideal.');
```

Figura 18 – Exemplo de aplicação de *Ifs aninhados*

Quando compilamos, obtemos o seguinte:



```
D:\Installers\pascalzim\VPZIM.exe
Digite o numero que corresponde ao que deseja:
1- Calcular área de retangulo
2- Calcular IMC - indice de massa corporal
1
Digite os comprimentos dos lados: 2 3
A area do retangulo é: 6.000000

D:\Installers\pascalzim\VPZIM.exe
Digite o numero que corresponde ao que deseja:
1- Calcular área de retangulo
2- Calcular IMC - indice de massa corporal
2
Digite sua altura em metros: 1.72
Digite seu peso em kg: 50
Seu IMC é 17.605172 e esta dentro do ideal.
```

Figura 19 – Dois exemplos do programa em execução.

13. Estruturas de Repetição

Por vezes, precisamos que o programa realize determinada ação repetidamente, seja por um número determinado de vezes, ou enquanto uma condição está sendo satisfeita, ou mesmo até que uma condição seja verificada. Para isto utilizam-se as Estruturas de Repetição, para evitar que, por exemplo, tenhamos que escrever 30 vezes o comando `readln()` quando quisermos ler do teclado as notas dos 30 alunos de uma turma do Pré-engenharia.

13.1. Comando *for*

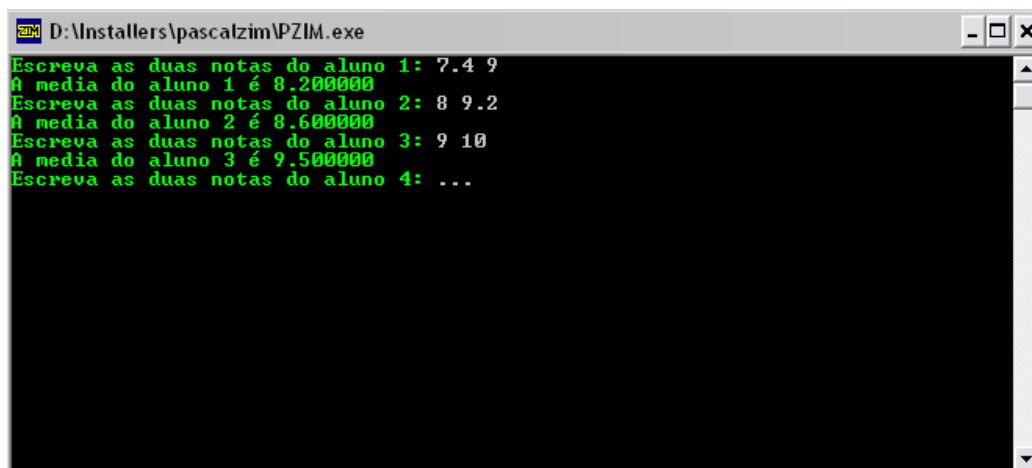
Repete a execução de um bloco de comandos por um número de vezes que depende dos objetivos do programa. Sua sintaxe é a seguinte:

```
for variável := ValorInicial to ValorFinal do
begin
    bloco de comandos;
end;
```

Determinados os valores iniciais e finais, a variável, que deve ser do tipo inteiro, é incrementada de 1 a cada laço. Isto é, o bloco de comandos é executado, a variável torna-se uma unidade maior, então o bloco de comandos é executado novamente, a variável é incrementada novamente, e assim por diante, até que ela ultrapasse o valor final. Veja:

```
1 Program MediaDaTurma;
2
3 var i : integer;
4     n1, n2, media : real;
5
6 Begin
7     for i:=1 to 10 do
8     begin
9         write('Escreva as duas notas do aluno ',i,':');
10        readln(n1,n2);
11        media := (n1+n2)/2;
12        writeln('A media do aluno ',i,'é:',media);
13    end;
14
15 End.
```

Figura 20 – Programa que exhibe médias de alunos logo após receber as duas notas.



```
D:\Installers\pascalzim\VPZIM.exe
Escreva as duas notas do aluno 1: 7.4 9
A media do aluno 1 é 8.200000
Escreva as duas notas do aluno 2: 8 9.2
A media do aluno 2 é 8.600000
Escreva as duas notas do aluno 3: 9 10
A media do aluno 3 é 9.500000
Escreva as duas notas do aluno 4: ...
```

Figura 21 – Parte da execução do programa (o programa vai até o aluno 10).

Repare: as variáveis *n1* e *n2* inicialmente recebem as notas do aluno 1, a média é calculada e, no laço seguinte, as mesmas variáveis recebem os valores das notas do aluno 2, e assim por diante. Esta propriedade é importante: uma variável carregará em cada momento o último valor que lhe foi atribuído.

13.2. Comando *While*

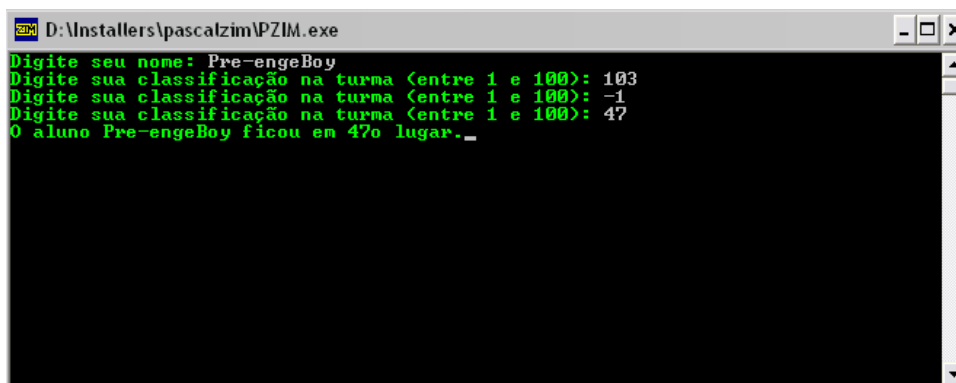
Difere do comando *for* quanto à sintaxe. A maioria dos programas em que se usa o *while* podem ser feitos usando-se o *for*, e vice-versa. Porém, em alguns casos a escolha de um deles simplifica significativamente a lógica a ser utilizada. Vejamos a sintaxe do comando *while*:

```
while condicao do
begin
    ..... bloco de comandos;
end;
```

Exemplo de programa:

```
1 Program Classificacao;
2
3 var  posicao : integer;
4     nome  : string;
5
6 Begin
7     write('Digite seu nome: ');
8     readln(nome);
9
10    while (posicao>100) or (posicao<1) do
11    begin
12        write('Digite sua classificação na turma (entre 1 e 100): ');
13        readln(posicao);
14    end;
15
16    write('O aluno ',nome,' ficou em ',posicao,'o lugar.');
```

Figura 22 – Aplicação do *while* para validar a classificação fornecida.



```
D:\Installers\pascalzim\pzim.exe
Digite seu nome: Pre-engeBoy
Digite sua classificação na turma (entre 1 e 100): 103
Digite sua classificação na turma (entre 1 e 100): -1
Digite sua classificação na turma (entre 1 e 100): 47
O aluno Pre-engeBoy ficou em 47o lugar._
```

Figura 23 – O programa pede a classificação até que ela tenha um valor válido.

13.3. Comando Break

Algumas vezes é interessante parar o programa antes que um laço seja realizado por completo. Nestes casos, utiliza-se o comando *break*, cuja sintaxe será mostrada no exemplo a seguir:

```
1 Program AcerteIdade;
2
3 var i, n, idade : integer;
4     nome : string;
5
6 Begin
7     write('Quantos alunos há na turma? ');
8     readln(n);
9
10    for i := 1 to n do
11        begin
12            write('Diga seu nome: ');
13            readln(nome);
14            write('Tente acertar minha idade: ');
15            readln(idade);
16            if idade = 43 then
17                begin
18                    break;
19                end;
20        end;
21
22    writeln('O aluno ', nome, ' acertou minha idade.');
```

Figura 24 – Aplicação do comando *break* para encerrar o laço *for*.

```
D:\Installers\pascalzim\pzim.exe
Quantos alunos há na turma? 20
Diga seu nome: Joao
Tente acertar minha idade: 21
Diga seu nome: Maria
Tente acertar minha idade: 45
Diga seu nome: Pre-engeBoy
Tente acertar minha idade: 43
O aluno Pre-engeBoy acertou minha idade.
```

Figura 25 – Caso o terceiro aluno não acertasse, o laço iria até que alguém acertasse ou até o 20º aluno.

14.Exercícios Propostos

1. Fazer um programa que peça um número inteiro x e um valor de potência n e calcule x^n .
2. Dado um número inteiro positivo n , calcular a soma dos n primeiros números naturais.
3. Criar um programa que receba uma senha numérica e teste-a, verificando se está correta ou não.
4. Escreva um programa em PASCAL que leia a razão de uma PA, o seu primeiro termo e a quantidade de termos e com isso calcule a soma dessa PA.
5. Escreva um programa que leia as notas das duas avaliações normais e a nota da avaliação optativa. Caso o aluno não tenha feito a optativa deve ser fornecido o valor "0". Calcular a média do semestre considerando que a prova optativa substitui a nota mais baixa entre as duas primeiras avaliações. Escrever a média e mensagens que indiquem se o aluno foi aprovado, reprovado ou está de AF, de acordo com as informações abaixo:
Aprovado: media ≥ 7.0
Reprovado: media < 4.0
AF: media ≥ 4.0 e < 7.0
6. Faça um programa que leia um número indeterminado de idades. A última idade lida, que não entrará nos cálculos, deverá ser igual a zero. Ao final o programa deverá escrever quantas idades foram lidas, calcular e escrever a média de idade desse grupo de idades.
7. Faça um programa que leia 10 números inteiros positivos, calcule e imprima os que são números perfeitos. Sendo que, um número perfeito é aquele cuja soma de seus divisores, exceto ele próprio, é igual ao número. Exemplo: 6 é perfeito porque $1 + 2 + 3 = 6$.
8. Escreva um algoritmo no qual o usuário digite o valor de um ângulo em graus e o programa retorne na tela o valor do mesmo em radianos.
9. Escreva um programa que retorne na tela os 30 primeiros termos da seqüência de Fibonacci. Tal seqüência inicia-se com 0 e 1, e os próximos termos da seqüência são adquiridos a partir da soma dos 2 termos anteriores.
10. Elabore um programa que calcule as raízes de equações do 2º grau. O algoritmo deverá avisar caso o valor de delta seja negativo.
11. Elabore um programa que leia dois números e calcule a combinação simples (do estudo de probabilidade e combinatória) entre eles.
12. Implemente um programa de conversão de temperaturas em Kelvin para Celsius.
13. Considere uma certa cultura de bactérias na qual cada bactéria se divida em duas a cada 1s e assim por diante. Idealize um programa que, dado um valor qualquer de segundos, forneça a quantidade de

bactérias originadas de apenas uma na cultura. Mas cuidado! Para valores de tempo um pouco grandes o cálculo se tornará inviável, pois excederá os limites numéricos da linguagem.

14. Dadas as duas notas e percentagem de freqüências de um aluno, construir um programa que calcule e exiba a média aritmética do aluno e sua situação na disciplina:
 - Média ≥ 7 e freqüência ≥ 0.75 aluno aprovado
 - Média ≥ 7 e freqüência ≤ 0.75 aluno reprovado por falta
 - 4 \leq Média < 7 e freqüência < 0.75 aluno reprovado por falta
 - 4 \leq Média < 7 e freqüência > 0.75 aluno deverá fazer exame final
 - Média < 4 e freqüência > 0.75 aluno reprovado por média
 - Média < 4 e freqüência < 0.75 aluno reprovado por média e por falta
15. Construa um programa que leia uma coordenada (x,y) e verifique qual o quadrante que ela está, ou se está em um dos eixos, ou na origem. Neste mesmo programa entre com uma segunda coordenada e verifique qual dos 2 pontos no Plano Cartesiano está mais próximo da origem dos eixos (0,0). Não é necessário fazer dois programas.
16. Faça um programa que leia três números (a, b, c) e indique se esses números podem ser os tamanhos dos lados de um triângulo. Caso a resposta seja afirmativa, indique se o triângulo é equilátero (3 lados iguais), isósceles (2 lados iguais) ou escaleno (3 lados diferentes). OBS: três números NÃO representam os lados de um triângulo quando algum deles é maior que a soma dos outros dois.
17. Escrever um algoritmo que escreva a soma dos números pares entre 0 e 100.
18. Escrever um algoritmo que lê 5 valores, um de cada vez, e conta quantos destes valores são negativos, escrevendo esta informação.