END OF FILE – Ebook Pascal

fórum eof: http://endoffile.umforum.net
face eof: https://www.facebook.com/ForumEof
site kodo: http://fts315.xp3.biz
autor: kõdo no kami (www.facebook.com/hacker.fts315) - skype: hackefts315
greetz: mmxm, sir.rafiki, suspeit0@virtual, entre outros
sites recomendados: hc0der, injectionsec, brutal security, made in brazil
17/04/2014 – 16/07/2014 (sem revisão)

Ola membros do forum EOF e um salve para os parceiros do EOF também ^^ , eu sou kõdo no kami (コードの神) antigo hfts315 em mais um ebook de programação, nesse ebook vou ensinar a programar em pascal porém não vou me aprofundar muito igual no ebook anterior de c/c++ esse será mais superficial porém não tanto a ponto de limitar o conhecimento do leitor ao mesmo, nesse ebook vou ensinar a lógica básica da linguagem pascal usando a IDE como dev pascal, vamos aprender estruturas lógicas e condicional, de repetição e aritmética, sobre variáveis e constantes, ponteiros e alocação, orientação a objeto entre outras coisas, esse ebook não pode ser comercializado a distribuição dele é livre sem fins financeiros ^^

Indice

1.0 – Introdução da linguagem pascal

A linguagem pascal é uma linguagem compilada e bem antiga muito usada para ensinar logica de programação embora ela nao se limita apenas nisso, essa linguagem foi criada pelo Niklaus Wirth e teve o nome escolhido como homenagem ao matematico Blanze Pascal na decada de 70 e é usada ate hoje para criação de software comerciais graças a facilidade de algumas IDE como o delphi, existem muitos compiladores e IDE tanto freeware quanto comerciais e também para varias plataformas e sistemas operacionais (windows, linux, mac, android entre outros), o pascal teve uma grande melhoria pela apple se tornando orientado a objeto assim nasceu a linguagem object pascal que é mas usadas nas atuais IDE

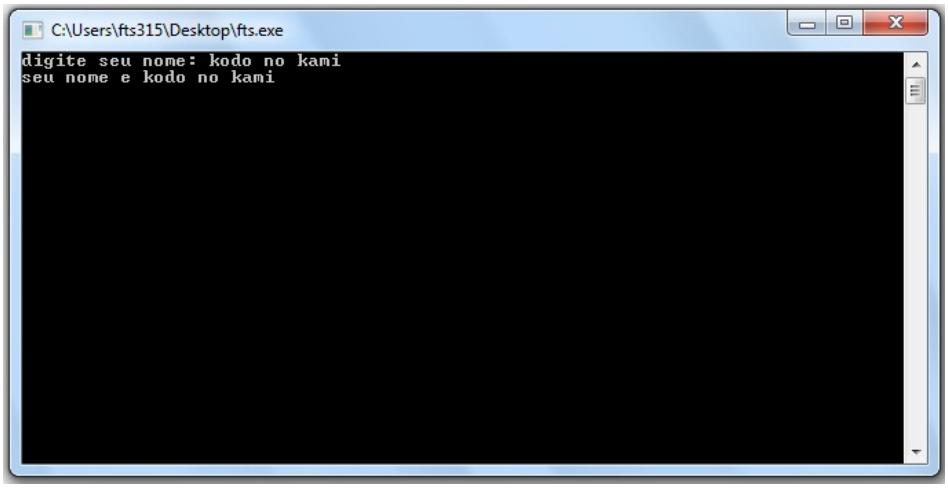
1.1 – Compiladores e IDE

para um programa ser feito o programador escreve o codigo fonte (source) dele, esse codigo fonte é convertido para linguagem da maquina essa conversao é feita pelo compilador, o programador so escreve as funçoes especificas que ele deseja que o programa faça e o compilador dessa linguagem vai converte ela para o executavel deixando ele portavel para o mesmo sistema ou plataforma, para facilitar as coisas para os programadores existem as IDE que são ambiente de programação elas contem um compilador e um editor de texto proprio que pode destacar a sintaxe da linguagem, um debugger que checa erros, algumas tem ate um editor de interface grafica para criar programas grafico (GUI), entre outras opçoes, alguns compiladores usados para o pascal são o turbo pascal, free pascal e gnu pascal, algumas IDE dessa linguagem são o dev pascal, pascalzinho, delphi, kylix, lazarus entre outros, abaixo tem alguns links onde pode encontrar eles

http://www.bloodshed.net/devpascal.html (dev pascal) http://www.lazarus.freepascal.org/ (lazarus) http://www.embarcadero.com/ (embarcadeiro) http://www.freepascal.org/ (free pascal) http://www.gnu-pascal.de/ (gnu pascal)

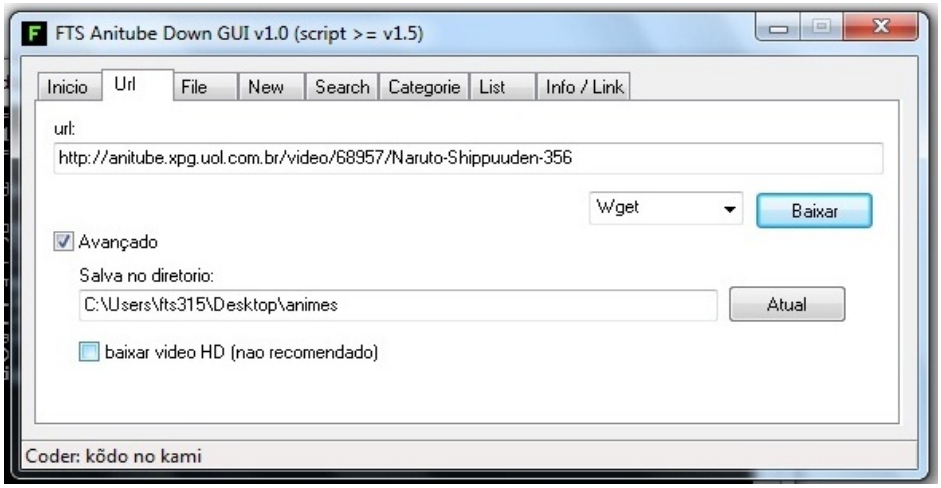
1.2 – Interfaces CLI e GUI

quando aprendemos alguma linguagem de programação os primeiros programas que vamos criar vai ser para o terminal do sistema (console, prompt, cmd, sh, ou qualquer outro nome XD), pelo motivo de ser mais facil desenvolver um programa para o terminal do que criar uma janela grafica (tirando alguns casos de IDE especifica como o delphi que o foco é mais a parte grafica, em alguns cursos de delphi eles nem aborda parte de terminal so a parte grafica mesmo =/), os programas que sao criados para o console ou terminal são chamados programas CLI, esse programas funciona com base em comandos digitados no terminal, existe duas formas de usar esses programas uma delas é passar o comando junto ao executavel, um exemplo é ping seguindo do ip da maquina alvo outra forma é executar o programa e ele pedir os comandos, os programas CLI são mais rapidos gasta menos memoria e processamento porem o uso deles é mais para usuarios avançados devido dificuldade de um usuario leigo em um terminal, abaixo tem um programa em interface CLI feito no dev pascal



os programas com interface grafica são chamados programas GUI, esse programas são mas recomendados para os usarios finais devido a facilidade de uso, para criar esses programas graficos é necessario conhecimento em

determinadas API graficas seja do sistema operacional ou outras como GTK, algumas IDEs faz a parte grafica automaticamente sem precisar de nenhum esforço do programador ou conhecimento em API grafica como por exemplo o delphi e lazarus, um programa GUI tem mais facilidade para vender do que um pograma CLI, um programa GUI é mais complicado e trabalhoso de ser fazer (teoricamente falando é claro), um programa GUI é mais limitado que alguns programas CLIs (um programa CLI que passamos os comandos como argumento por exemplo o ping pode ser criado scripts em outras linguagens para manipular a saida do terminal ou criar programas para automatizar ele, um exemplo disso é um script em perl que eu fiz "fts anitube down" a parte grafica dele foi feito em delphi), abaixo tem um exemplo de uma imagem de um programa GUI o "fts anitube down gui" (http://fts315.xp3.biz/ftsanitubedown/) feito em delphi



2.0 – Sintaxe Pascal

quase todas as linguagens de programação são iguais na logica (semática), o que muda nelas sao as sintaxe que pode variar de linguagem pra linguagem, a sintaxe da linguagem pascal deve sempre começar com o nome do programa, para definir ele usamos a palavra program seguido do nome do programa que pode ser qualquer nome menos palavras ja usadas pelo compilador (program, begin, end entre outras), o nome do programa tambem nao pode ter espaço (coloque underline para substituir o espaço, exemplo kodo_no_kami), e por fim um ponto e virgula indicando o final da sintaxe

```
program endoffile ;
```

caso alguma sintaxe esteja incorreta o compilador acusara erro e nao vai compilar, o compilador pascal nao é case sensitive ou seja nao faz diferença escrever maisculo ou minusculo (caixa alto ou caixa baixa), entao tambem estaria correto fazer assim

```
ProGraM EndoFfile ;
```

o compilador tambem nao interpreta espaço e nem quebra de linha entao o proximo exemplo seria a mesma coisa que os dois anteriores (escrever programa assim é feio, então nunca faça isso ^^)

```
program

                endoffile
  ;
```

alem do program é necessario um bloco tambem chamado de escopo, nesse bloco que vamos escrever todas as funçoes para o nosso programa, esse bloco começa com a palavra begin e termina com a palavra end e um ponto no final (esse ponto indica o final do escopo principal, no programa vai ter outros escopos porem apenas esse tera um ponto no final os demais vai ter um ponto e virgula, vamos ver isso mais pra frente)

```
program endoffile;
```

```
begin end.
```

como dito antes se fizer o programa na mesma linha ainda estaria certo, porem ficaria meio ilegivel, uma forma de deixa o programa com uma boa legibilidade é sempre dar um espaço ou tabulação a cada novo escopo e uma quebra de linha a cada final isso é chamado identação ou recuo

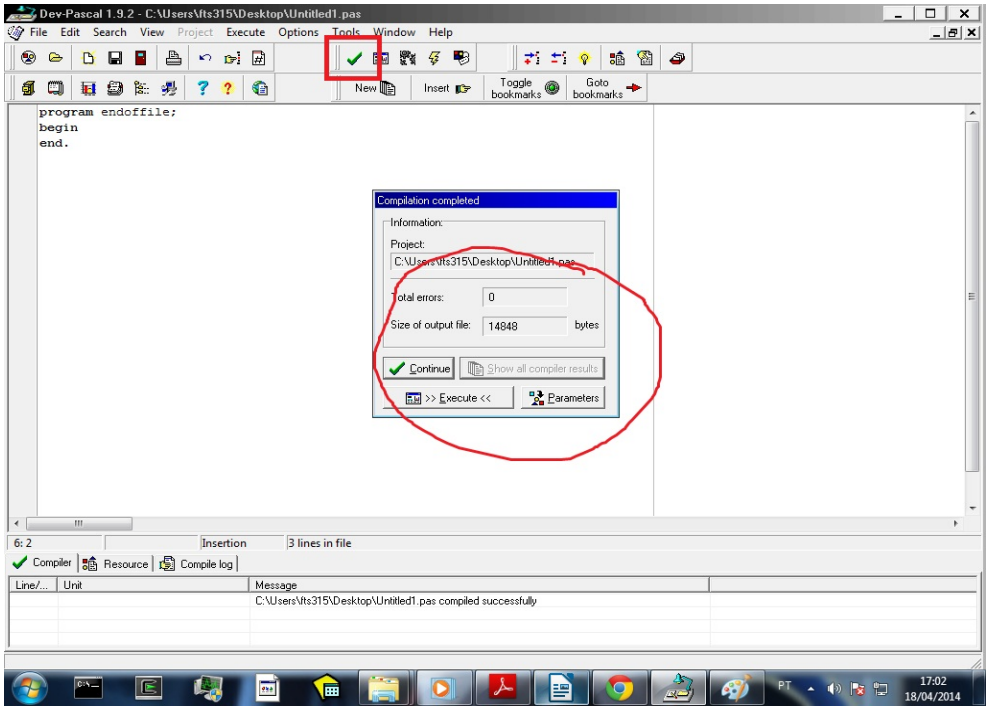
```
program endoffile;
begin
end.
```

2.1 – Compilando no Dev Pascal

vamos começar brincando com o dev pascal, caso você nao tenha ele vai no site oficial dele e baixe ele depois instale (http://www.bloodshed.net/devpascal.html), depois de instalar abra ele e coloque o codigo que aprendemos ate agora

```
program endoffile;
begin
end.
```

para compilar aperte no botao verde, ou aperte no menu "execute" e depois "compile" (ou f9), caso o arquivo não esteja salvo em nenhum lugar ele vai pedir para voce salvar em um local (escolha o local onde sera salvo e ao mesmo tempo gerado o executavel), caso de algum erro aperte contine e veja se voce nao erro nenhuma sintaxe, caso não de erro vai aparecer habilitado o botao de execute embaixo do continue, ele serve para executar o nosso programa ja compilado (outra forma de executar ele seria ir na pasta que salvamos, dentro dela ta o executavel depois de compilar), se voce executo e o programa fecho nao se preocupe esse programa ainda nao faz nada por isso que abriu e fecho



depois de ter gerado o executavel basta copiar ele e mandar para seus amigos (esse nao porque ele nao faz nada e sim os futuros), ou codigo fonte (source) para os demais programadores (quando voce libera o codigo fonte junto ao programa seu programa é considerado open source ou seja voce esta permitindo a modificação dele)

2.2 – Bibliotecas da linguagem pascal

um programador nao precisa criar tudo do zero existem bibliotecas com varias funçoes prontas (no caso do pascal essas bibliotecas são chamadas units), essas bibliotecas tem varias funções dentro algumas são funçoes do proprio

sistema operacional (API), outras são padrao da propria linguagem, algumas criado por terceiros, e algumas serao criadas por voce mesmo para nao precisar reescrever determinado trecho de codigo em outros programas futuros, tambem existe um arsenal de bibliotecas na internet onde voce pode baixar e incrementar mais seu codigo, para usar uma biblioteca no pascal basta usar a palavra uses seguindo da biblioteca fora do escopo bem no começo do codigo depois do program (as units deve fica junto com o codigo fonte do programa ou em uma variavel de ambiente compilador), as bibliotecas padroes que costuma vim com o pascal são crt essa biblioteca tem algumas funçoes para manipular o terminal como mudar a cor, ler uma tecla, mudar a posição do cursor, limpar o terminal, entre outras (alguns compiladores nao tem o crt), existe a biblioteca sysutils que tem varias funçoes como conversão de dados e tipos, formatação de string, a biblioteca math tem alguma funçoes matematica, a biblioteca windows vem varias API dos sistema windows entre elas manipular processo e memoria, manipular janelas, manipular o registro do windows entre outras milhares de funçoes, a biblioteca winsock permite manipular rede, conexao com outras maquinas e protocolos, existem outras bibliotecas padroes alem das citadas

```
program endoffile;

uses crt;

begin
end.
```

para usar mais de uma biblioteca basta separar por virgula

```
program endoffile;

uses crt, sysutils;

begin
end.
```

2.3 – Comentarios

é sempre bom comentar seu codigo para evitar esquecer para que determinada funçao funciona ou por que uso determinada função, ou para descrever o codigo colocando as informação neles, para comentar uma linha toda usamos barra barra (o que estiver depois dele sera comentario)

```
//autor: kõdo no kami
//forum: endoffile.umforum.net
program endoffile; //nome do programa
//escopo principal
begin
    //sempre use identação a cada novo escopo
end.
```

para comentar mais de uma linha usamos abre chaves e fecha chaves o que estiver entre elas sera comentario

```
{
  autor: kõdo no kami
  forum: endoffile.umforum.net
}
program endoffile;
begin
end.
```

tambem é valido comentario entre uma sintaxe com o comentario de varias linhas deis de que nao deforme ela

```
program { kkkkk } endoffile;
begin
```

```
end.
```

em pascal mais antigo o comentarios de varias linhas era parenteses seguido do asterisco (evite usar esse pode ser que algum compilador nao reconheça ele)

```
(* comentario antigo *)
program endoffile;
begin
end.
```

2.4 – Pausando o terminal

na parte anterior da compilação o programa abriu e fecho isso aconteceu devido nao existir uma função nele que pause o terminal, para evitar isso podemos usar a função readkey da biblioteca crt no final do nosso codigo para ele pausar antes de finalizar

```
program endoffile;

uses crt;

begin
    readkey;
end.
```

agora podemos notar que o terminal fico parado esperando apertar uma tecla para continuar (finalizar), isso permite que o programa CLI nao finalize (outra forma seria executar o programa diretamente no terminal e nao abrindo o executavel direto), tambem é possivel criar um loop infinito ate determinado condição (vamos ver isso mais para frente)

3.0 – Saida de dados (output)

saida de dados ou output seria enviar determinado fluxo de dados de um ponto A para o ponto B, um exemplo seria enviar determinado texto para o monitor, e entrada dados ou input é inverso seria alguma coisa entrando para o programa exemplo escrever alguma coisa no teclado, a saida de dados mais usada no pascal para o terminal seria o write que escreve um texto (os texto na logica de programação são chamados de strings, e toda string fica entre aspas para o compilador saber que é uma string e nao uma função ou variavel, no caso do pascal ela fica entre aspas simples por exemplo 'isso e uma string'), para usar a função write basta escrever o nome da funçao e passar como argumento a strings (os argumentos no pascal fica entre parenteses)

```
program endoffile;

uses crt;

begin
    write('aperte enter para sair');
    readkey;
end.
```

podemos usar o write quantas vezes a gente quiser

```
program endoffile;

begin
    write('by kodo no kami');
    write('segundo write');
end.
```

o write e tambem permite mostrar varias strings separados por virgula

```
program endoffile;

begin
    write('end of file','kodo no kami');
end.
```

nos exemplos anteriores ele mostro tudo na mesma linha, para dar uma quebra de linha podemos usar o writeln que mostra a string e da uma quebra de linha

```
program endoffile;

begin
    writeln('linha 1');
    writeln('linha 2');
    write('linha 3');
end.
```

ou usar apenas o writeln para da a quebra de linha sem nehum argumento

```
program endoffile;

begin
    write('linha 1');
    writeln;
    write('linha 2');
end.
```

3.1 – Variaveis e Constantes

as variaveis são alocação de memoria de um tipo especifico de dados, a utilidade das variaveis que podemos armazena determinados tipos de dados na memoria e acessar esses dados quando precisar (esses dados so fica armazenado enquanto o programa estiver em execução, quando o programa é finalizado o processo do mesmo e a alocação dele é liberada para ser usada por novos processos), existem varios tipos de dados os principais sao integer que armazena numeros inteiros tanto positivo quanto negativo (1, 50, 315, -1 , -6 , -1000), tipo real que sao numeros quebrado (5.5 , 850.356 , -10.3), tipo boolean que armazena verdadeiro ou falso (true ou false), tipo char que armazena um unico caracter ('a', 'f', '3'), tipo string que armazena um conjunto de caracter no caso é um texto ('kodo no kami' , 'hfts315'), para criar uma variavel usamos a palavra var antes do begin do escopo seguido do nome da variavel (pode ser qualquer um menos palavra com espaço, ja existentes ou que comece com numeros ou simbolos especiais) depois dois pontos seguido do tipo

```
program endoffile;

var numero: integer;
begin
end.
```

para criar mais de uma variavel nao precisamos usar a palavra var

```
program endoffile;

var numero: integer;
     numero2: integer;
begin
end.
```

quando a variavel é do mesmo tipo podemos declarar ela junto separando por virgula

```
program endoffile;

var numero, numero2 : integer;
begin
end.
```

veja outro exemplo com varios tipos

```
program endoffile;

var numero_int, numero_int2: integer;
    numero_real: real;
    letra: char;
    texto: string;
begin
end.
```

para a gente atribuir um valor para a variavel basta digitar o nome dela dentro do escopo coloca dois ponto e igual seguido do valor para ser atribuido

```
program endoffile;

var numero: integer;
begin
   numero := 315;
end.
```

depois de atribuir o valor podemos usar ela como a gente desejar

```
program endoffile;

var numero: integer;
begin
  numero := 315;
  write(numero);
end.
```

as variaveis pode mudar o valor ao decorrer do programa

```
program endoffile;

var texto: string;
begin
  texto := 'kodo no kami';
  writeln(texto);
  texto := 'end of file';
  write(texto);
end.
```

tambem é possivel atribuir o valor de outra variavel em uma variavel

```
program endoffile;

var numero, numero2: real;
begin
```

```
  numero := 3.15;
  numero2 := numero;
  write(numero2);
end.
```

algumas vezes para nao precisar criar duas variaveis voce pode atribuir uma variavel a ela mesma (isso é muito usado em contadores para incrementar ou decrementar determinados valores)

```
program endoffile;

var numero: real;
begin
  numero := 100;
  numero := numero;
  write(numero2);
end.
```

alem das variaveis existem as constantes essas nao muda o valor ficando com um valor fixo, para usar uma constante usamos a palavra const no lugar de var, depois o nome da constante, no lugar do dois pontos usamos igual e o valor

```
program endoffile;

const kodo = 315;
begin
end.
```

as constantes pode ser usadas iguais as variaveis so nao pode ser mudado o valor delas

```
program endoffile;

const kodo = 315;
begin
  write(kodo);
end.
```

3.2 – Vetores e Matrizes

as vezes precisamos criar um numero muito grande de variaveis e isso poderia dar muito trabalho para o programador ter que criar uma a uma, para evitar esse trabalho todo existem as arrays que são variaveis com varios segmentos, enquanto que uma variavel armazena um unico valor uma array armazena varios dependendo do tamanho que ela criada, existem dois tipos de arrays os vetores que armazena um unico segmento e as matrizes que armazena outros segmentos (as matrizes podem contem varios segmentos um dentro do outro), para criar uma array do tipo vetor na linguagem pascal é parecido com criação de uma variavel mudado apenas a parte onde fica o tipo, nele temos que definir a palavra array seguido de abre e fecha colchete dentro dele o numero incial seguido de dois ponto o tamanho da nossa array, e por fim a palavra of seguido do tipo

```
program endoffile;

var kodo: array[1 .. 5] of integer;
begin
end.
```

a array anterior seria equivalente a criar 5 variaveis do tipo inteiro (ja imagino se voce tiver que criar 1000 variaveis, uma unica array com 1000 posiçoes seria mais logica de se fazer ne?)

```
program endoffile;
```

```
var kodo1, kodo2, kodo3, kodo4, kodo5: integer;
begin
end.
```

para acessar uma array, basta digitar o nome dela seguido de colchetes e dentro dele a posição que vai ser acessada (no caso do exemplo da array anterior vai de 1 ate 5, totalizando 5 posiçoes)

```
program endoffile;

var kodo: array[1 .. 5] of integer;
begin
    write(kodo[1]);
    write(kodo[2]);
    write(kodo[3]);
    write(kodo[4]);
    write(kodo[5]);
end.
```

para atribuir um valor a array basta fazer da mesma forma que faria com uma variavel so que atribuindo em cada posição da array (nao precisamos usar todos os segmentos da array, porem criar uma array e nao usar é um desperdicil de memoria)

```
program endoffile;

var kodo: array[1 .. 5] of integer;
begin
    kodo[1] := 100;
    kodo[2] := 315;
    write(kodo[1]);
    write(kodo[2]);
end.
```

uma matriz é uma array que cada posição dela aponta para uma nova posição, para criar uma matriz é a mesma coisa que criar um vetor, a unica diferença que temos que coloca a nova posição depois da outra separando por virgula

```
program endoffile;

var kodo: array[1 .. 2, 1 .. 3] of integer;
begin
end.
```

o exemplo anterior tem duas posições e dentro dessas duas posiçoes existem 6 posiçoes cada (3 x 2 = 6), para acessar ela basta coloca as duas posições separado por virgula

```
program endoffile;

var kodo: array[1 .. 2, 1 .. 3] of integer;
begin
   kodo[1,1] := 315;
   kodo[1,2] := 100;
   kodo [1,3] := 50;
   kodo[2,1] := 1500;
   kodo[2,2] := 31500;
   kodo[2,3] := 600;
end.
```

podemos criar quantos segmentos a gente quiser

```
program endoffile;

var kodo: array[1 .. 2, 1 .. 2, 1 .. 2] of integer;
begin
   kodo[1,1,1] := 315;
   kodo[1,1,2] := 100;
   kodo[1,2,1] := 50;
   kodo[1,2,2] := 1500;
   kodo[2,1,1] := 31500;
end.
```

3.3 – Ponteiros e alocação dinamica

os ponteiros são uma especie de variaveis, porém elas nao aloca um espaço na memoria e sim aponta para um, o ponteiro serve para ter mais de uma maneira de acessar a mesma varivel por varios pontos diferentes, para usar os ponteiros precisamos aponta para um endereço valido ou seja uma variavel existente, o tipo usado no ponteiro é o tipo pointer e sua declaração é a mesma que as variaveis

```
program endoffile;

var ponteiro: pointer;
begin
end.
```

depois de criar o ponteiro temos que apontar ele para o endereço valido, para isso basta atribuir o endereço para ele ou retornar o endereço de uma variavel, para retornar o endereço de uma variavel basta digitar o nome dela com um arroba antes

```
program endoffile;

var variavel: integer;
     ponteiro: pointer;
begin
    ponteiro := @variavel;
end.
```

tambem é possivel usar a função addr para retornar o endereço de uma variavel

```
program endoffile;

var variavel: integer;
     ponteiro: pointer;
begin
    ponteiro := addr(variavel);
end.
```

os ponteiros do tipos pointer não sao recomendado para manipular o valor naquele endereço ele serve apenas para transportar determinado endereço de memoria isso porque ele nao sabe definir qual é o tipo de variavel que esta naquele endereço (cada tipo de variavel tem um tamanho especifico e como o compilador deve acessar), entao para manipular um valor onde por um ponteiro temos que criar pointeiros para tipos especificos (esse tipo deve ser o mesmo para onde ele está apontando), para criar um ponteiro de um tipo especifico basta criar uma variavel normal daquele tipo e colocar antes um acento circunflexo (^)

```
program endoffile;
```

```
var ponteiro: ^integer;
begin
end.
```

da mesma maneira que atribuimos um endereço de memoria de uma variavel para o tipo pointer podemos atribuir para um ponteiro de tipo especifico

```
program endoffile;

var variavel: integer;
     ponteiro: ^integer;
begin
    ponteiro := @variavel;
end.
```

quando passamos um ponteiro do tipo pointer para um ponteiro do tipo especifico nao precisamos usar o arroba para pegar o endereço isso porque os ponteiros ja tem atribuido o endereço

```
program endoffile;

var ponteiro: pointer;
     ponteiro2: ^integer;
begin
    ponteiro2 := ponteiro;
end.
```

para fazer um ponteiro apontar para um endereço de memoria basta atribuir o endereço nele, para manipular o endereço onde ele esta apontando temos que colocar acento circunflexo depois do nome, no exemplo abaixo apontamos um ponteiro para variavel e depois atribuímos o valor 315 para onde o ponteiro esta apontando entao tanto a variavel quanto o ponteiro vai modificar ou ler o mesmo endereço de memoria e o mesmo valor

```
program endoffile;

var variavel: integer;
     ponteiro: ^integer;
begin
    ponteiro := @variavel;
    ponteiro^ := 315;
    writeln(ponteiro^);
    write(variavel);
end.
```

como ja sabemos um ponteiro apenas aponta para determinado endereço, porem podemos alocar um espaço na memoria e apontar o ponteiro para ela (em teoria é isso que as variaveis faz), para alocar um espaço usamos a função new e passamos como argumento pra ela um ponteiro de um tipo especifico, essa alocação vai ser dinamica ou seja a gente pode criar e liberar ela a qualquer momento

```
program endoffile;

var ponteiro: ^integer;
begin
   new(ponteiro);
   ponteiro^ := 315;
   write(ponteiro^);
end.
```

para librar o espaço de uma alocação dinamica usamos a função dispose e passamos como argumento o ponteiro, esse espaço que foi liberado nao fica mais em uso e pode ser futuramente usado por outras variaveis essa é uma das vantagens das variaveis dinamicas elas so precisa ficar em uso enquanto precisamos dela enquanto as outras variaveis vai ficar ate o programa ser finalizado ou a funçao ser destruida)

```
program endoffile;

var ponteiro: ^integer;
begin
   new(ponteiro);
   ponteiro^ := 315;
   write(ponteiro^);
   dispose(ponteiro);
end.
```

mesmo depois da gente liberar o espaço alocado ainda existe o ponteiro naquele endereço, é sempre bom e recomendado atribuir a constante nil para o ponteiro depois que liberar a alocação (isso evitaria do programa tentar ler ou escrever em um endereço memoria ja liberado)

```
program endoffile;

var ponteiro: ^integer;
begin
   new(ponteiro);
   ponteiro^ := 315;
   write(ponteiro^);
   dispose(ponteiro);
   ponteiro := nil;
end.
```

3.4 – Entrada de dados (input)

ja vimos sobre saida de dados para o monitor agora vamos ver sobre entrada de dados ou input do teclado ou seja escrever algo no teclado para dentro do programa, a função mais usada de input em pascal é o read que ler alguma coisa do teclado e armazena em uma variavel, para usar o read basta passar como argumento uma variavel (podendo ser qualquer tipo de variavel deis de string a numerica), assim que computador achar a função read no programa ele pausa e espera o usuario escrever alguma coisa e apertar enter para continuar

```
program endoffile;

var numero: integer;
begin
    read(numero);
end.
```

para o read ficar mais perfeito usamos um write antes para dizer para o usuario o que deve ser digitado

```
program endoffile;

var nome: string;
begin
    write('digite seu nome: ');
    read(nome);
    write('seu nome e ',nome);
end.
```

tambem existe o readln que da uma quebra de linha depois que digitar o read (alguns compiladores ele da a quebra de linha automaticamente a cada read)

```
program endoffile;

var nome: string;
begin
    write('digite seu nome: ');
    readln(nome);
    write('seu nome e ',nome);
end.
```

alem do read existe o readkey que ja vimos no começa para da uma pausa no terminal, diferente do read o readkey so pega uma unica tecla e deve ser atribuido a uma variavel do tipo char ao inves de passar como argumento ela (tambem deve declarar a biblioteca crt)

```
program endoffile;

uses crt;

var letra: char;
begin
    write('digite uma letra: ');
    letra := readkey;
    write('sua letra foi  ',letra);
end.
```

o readkey tem varias utilidades uma delas era pausar o terminal como ja vimos, outra é capturar apenas uma tecla sem precisar apertar enter, e outra é que ele nao mostrado no terminal a tecla digitada assim voce poderia fazer um programa de senha que mostre aqueles asteriscos quando digitada

```
program endoffile;

uses crt;

var letra1, letra2, letra3: char;
begin
    write('digite uma senha de 3 digitos: ');
    letra1 := readkey;
    write('*');
    letra2 := readkey;
    write('*');
    letra3 := readkey;
    writeln('*');
    write('sua senha foi ',letra1,letra2,letra3);
end.
```

3.5 – Manipulação de string

as string é um conjunto de caracteres ou uma array do tipo char, entao podemos criar uma array do tipo char para armazenar uma string, cada caracter da string sera atribuido a uma posição da array

```
program endoffile;

var texto: array[1..1000] of char;
begin
    write('digite o nome da sua cidade: ');
    read(texto);
    write('voce mora na cidade ', texto);
end.
```

o exemplo anterior é equivalente ao proximo, porem existe uma diferença de criar uma array do tipo char para armazenar uma string ou um tipo string, essa diferença é no tamanho de armazenamento se voce criar uma array ela vai ficar com um tamanho fixo (esse tamanho fixo pode causar grandes problemas em compiladores antigos causado buffer overflow), se voce criar uma variavel do tipo string ela vai realocar o espaço sempre que armazenar um novo valor ja na array ela sempre vai ter o mesmo tamanho

```
program endoffile;

var texto: string;
begin
    write('digite o nome da sua cidade: ');
    read(texto);
    write('voce mora na cidade ', texto);
end.
```

tambem podemos acessar qualquer posição de uma string retornando o caracter correspondente a ela

```
program endoffile;

var texto: string;
begin
    texto := 'fts315';
    writeln('primeiro caracter ',texto[1]);
    writeln('segundo caracter ',texto[2]);
    writeln('terceiro caracter ',texto[3]);
end.
```

podemos modificar uma string caracter por caracter direto na posição

```
program endoffile;

var texto: string;
begin
    texto := 'end of file';
    texto[1] := 'k';
    texto[2] := 'o';
    texto[3] := 'd';
    texto[4] := 'o';
    write(texto);
end.
```

temos que ter cuidado quando modificarmos uma string caracter por caracter por dois motivos, primeiro o pascal ele realoca o tamanho de uma string automaticamente quando for atribuido toda a string, quando atribuimos caracter por caracter a historia é outra, se tiver o espaço correspondente ao tamanho de todos os caracteres atribuido como no exemplo anterior que atribuimos a palavra kodo caracter por caracter ja existindo a palavra end of file ele vai atribuir nesse espaço normalmente porem se nao houver o espaço suficiente para ele atribuir possivelmente ou compilador vai da erro ou seu programa vai ficar com baita falha segurança como a buffer overflow, outra coisa que temos que ter cuidado é o ultimo caracter de uma string ele sempre deve ser o caracter nulo indicando o fim da string se nao o compilador vai continuar lendo ate achar o caracter nulo, o proximo exemplo seria uma falha no programa devido nao ter realocado o tamanho da string para adicionar os caracteres

```
program endoffile;

var texto: string;
begin
    texto[1] := 'k';
    texto[2] := 'o';
```

```
    texto[3] := 'd';
    texto[4] := 'o';
    write(texto);
end.
```

uma das formas de evitar esse problema anterior seria concatenar o caracter ao invese de atribuir ele diretamente na posição, para concatenar de uma forma facil basta atribuir a variavel nela mesmo e usar o operador + seguido do caracter ou string (fazendo dessa forma a propria concatenação ja colocaria o caracter nulo no final)

```
program endoffile;

var texto: string;
begin
    texto := 'k';
    texto := texto + 'o';
    texto := texto + 'd';
    texto := texto + 'o';
    write(texto);
end.
```

outra forma seria criar uma array do tipo char com um tamanho maior que a string que vamos criar (fazendo dessa forma nao é atribuido o caracter nulo, para atribuirmos ele podemos usar a função chr para converter um tipo para o tipo char e passar como argumento o numero 0 ou podemos atribuir direto usando sharps seguido do codigo #0)

```
program endoffile;

var texto: array[1..10] of char;
begin
    texto[1] := 'k';
    texto[2] := 'o';
    texto[3] := 'd';
    texto[4] := 'o';
    texto[5] := chr(0);
    write(texto);
end.
```

tambem podemos criar um ponteiro do tipo char para atribuir uma string, esse ponteiro sera apontado para uma alocação valida quando receber uma string

```
program endoffile;

var texto: ^char;
begin
    texto := 'isso vai ser alocado ';
    write(texto);
end.
```

existe o tipo pchar que seria equivalente a um ponteiro do tipo char

```
program endoffile;

var texto: pchar;
begin
    texto := 'declarar pchar e mesmoa coisa que ^char';
    write(texto);
end.
```

quando manipulamos um ponteiro do tipo char (pchar) podemos alocar um espaço com determinado tamanho com a função stralloc da biblioteca sysutils, para usar essa função passamos como argumento o tamanho da alocação e depois basta atribuir essa função ao ponteiro do tipo char (a alocação do tipo char ocupa 1byte, entao uma string com o tamanho de 10 caracteres deve ter pelo menos 11bytes somando com o caracter nulo), so lembrando que nesse caso a variavel seria dinamica ou seja podemos liberar aquele espaço

```
program endoffile;

uses sysutils;

var texto: pchar;
begin
    texto := stralloc(50);
    texto := 'kodo no kami';
    write(texto);
end.
```

com a função sizeof podemos retornar o tamanho exato que seria alocado de tipo de dados bastando passar como argumento o tipo, a função anterior ficaria mais correta e segura dessa forma especificando o tamanho de uma alocação do tipo char e multiplicando ela pela quantidade de tamanho alocado

```
program endoffile;

uses sysutils;

var texto: pchar;
begin
    texto := stralloc(sizeof(char) * 50);
    texto := 'kodo no kami';
    write(texto);
end.
```

como ja foi alocado o espaço podemos atribuir caracter por caracter, porem quando usamos o stralloc a posição começa no numero 0

```
program endoffile;

uses sysutils;

var texto: pchar;
begin
    texto := stralloc(sizeof(char) * 50);
    texto[0] := 'e';
    texto[1] := 'n';
    texto[2] := 'd';
    texto[3] := #0;
    write(texto);
end.
```

a função sizeof apenas retonar o tamano que seria alocado de um tipo especifico, para a gente saber o tamanho de uma alocação podemos usar a função strbufsize da biblioteca sysutils, para usar ela basta passar como argumento o ponteiro do tipo char e ela vai retornar o tamanho

```
program endoffile;

uses sysutils;

var texto: pchar;
```

```
begin
    texto := stralloc(sizeof(char) * 50);
    write(strbufsize(texto));
end.
```

tambem podemos liberar aquela alocação com a função strdispose da biblioteca sysutils, para usar ela basta passar como argumento o ponteiro do tipo char

```
program endoffile;

uses sysutils;

var texto: pchar;
begin
    texto := stralloc(sizeof(char) * 50);
    strdispose(texto);
end.
```

podemos concatenar uma string com um ponteiro do tipo char usado a função strcat da biblioteca sysutils, para usar essa função basta passar como argumento o ponteiro onde vamos atribuir e a string que sera atribuida (no lugar da string tambem poderia ser outra variavel)

```
program endoffile;

uses sysutils;

var texto: pchar;
begin
    texto := 'forum ';
    strcat(texto,'end of file');
    write(texto);
end.
```

o exemplo anterior seria equivalente ao proximo

```
program endoffile;

uses sysutils;

var texto: pchar;
begin
    texto := 'forum ';
    texto := texto + 'end of file';
    write(texto);
end.
```

com a função strlcat podemos concatenar uma string em uma variavel do tipo pchar porem uma quantidade especifica de caracteres

```
program endoffile;

uses sysutils;

var texto: pchar;
begin
    texto := 'forum ';
    strlcat(texto,'end of file',10);
```

```
    write(texto);
end.
```

alem de concatenar podemos copiar uma string para uma alocação usado a função strcopy da biblioteca sysutils, a diferença que sera apagado caso tenha uma string dentro da alocação

```
program endoffile;

uses sysutils;

var texto: pchar;
begin
    texto := 'forum ';
    strcopy(texto,'end of file');
    write(texto);
end.
```

a função strcopy seria equivalente a atribuiçao da string na variavel

```
program endoffile;

uses sysutils;

var texto: pchar;
begin
    texto := 'forum ';
    texto := 'end of file';
    write(texto);
end.
```

tambem existe a função strlcpy da biblioteca sysutils que é parecida com strcpy porem copia apenas uma quantidade especifica de caracteres

```
program endoffile;

uses sysutils;

var texto: pchar;
begin
    texto := 'forum ';
    strcopy(texto,'end of file',3);
    write(texto);
end.
```

podemos retornar o tamanho de um string usado variaveis pchar com a função strlen da biblioteca sysutils

```
program endoffile;

uses sysutils;

var texto: pchar;
     tamanho : integer;
begin
    texto := 'kodo no kami';
    tamanho := strlen(texto);
    write(tamanho);
end.
```

ou a função length para os tipos string

```
program endoffile;

uses sysutils;

var texto: string;
     tamanho : integer;
begin
    texto := 'kodo no kami';
    tamanho := length(texto);
    write(tamanho);
end.
```

podemos comparar duas variaveis pchar usado a função strcomp da biblioteca sysutils, para usar essa função basta passar como argumento as duas variaveis e a função retorna o numero 0 se as duas variaveis for igual

```
program endoffile;

uses sysutils;

var texto, texto2: pchar;
     comp: integer;
begin
    texto := 'kodo no kami';
    texto2 := 'kodo no kami';
    comp := strcomp(texto,texto2);
    write(comp);
end.
```

também existe o strlcomp que compara apenas uma quantidade especifica de caracteres

```
program endoffile;

uses sysutils;

var texto, texto2: pchar;
    comp: integer;
begin
    texto := 'kodo no kami';
    texto2 := 'kodo no akuma';
    comp := strlcomp(texto,texto2,7);
    write(comp);
end.
```

também podemos comparar com o operador igual, porem usando esse operador ele retorna um tipo booleano verdadeiro ou falso

```
program endoffile;

uses sysutils;

var texto, texto2: pchar;
    comp: boolean;
begin
    texto := 'kodo no kami';
    texto2 := 'kodo no kami';
```

```
    comp := texto = texto2;
    write(comp);
end.
```

para deixar todos os caracter da string maiúscula podemos usar a função strupper da biblioteca sysutils

```
program endoffile;

uses sysutils;

var texto: pchar;
begin
    texto := 'kodo no kami';
    strupper(texto);
    write(texto);
end.
```

para deixa minusculo podemos usar strlower da biblioteca sysutils

```
program endoffile;

uses sysutils;

var texto: pchar;
begin
    texto := 'KODO NO KAMI';
    strlower(texto);
    write(texto);
end.
```

4.0 – Logica aritmética

podemos fazer qualquer conta aritmética usando a programação seja a adição, subtração, multiplicação e divisão entre outros tipos de operações, para a operação de adição podemos usar o operador de mais (+)

```
program endoffile;

var numero: integer;
begin
    numero := 300 + 15;
    write(numero);
end.
```

seria valido fazer as operaçoes diretamente na função write ou em qualquer outra função

```
program endoffile;

begin
    write(300 + 15);
end.
```

também podemos usar as variaveis (só lembrando que as variáveis aponta para um valor na memória então usar um numero direto ou uma variável dá no mesmo)

```
program endoffile;

var numero1, numero2, resultado: integer;
```

```
begin
    numero1 := 100;
    numero2 := 200;
    resultado := numero1 + numero2;
    write(resultado);
end.
```

para subtração usamos o operador subtrair (-)

```
program endoffile;

begin
    write(300 - 15);
end.
```

para multiplicação usamos o asterisco (*)

```
program endoffile;

begin
    write(315 * 2);
end.
```

para divisão usamos a barra (/), quando usamos a barra para divisão será mostrado um numero bem grande com vários 0 (embora esse numero seja dificil de leitura ele é uma forma facil de manipular números muitos grandes)

```
program endoffile;

begin
    write(100 / 2);
end.
```

ou podemos usar o operador div para dividir em numeros normais

```
program endoffile;

begin
    write(100 div 2);
end.
```

na divisão costuma sobrar numeros essa sobra é chamada de resto ou modulo, para fazer o modulo usamos o operador modificar

```
program endoffile;

begin
    writeln('10 divido por 8: ');
    writeln('resultado = ', 10 div 8);
    write('resto = ',10 mod 8);
end.
```

quando mexemos com valores quebrados (números não inteiros ou variaveis do tipo real), a função write mostra um valor bem grande depois do ponto (caso o numero seja 3.15 será mostrado 3.150000000000000E+000), ou quando o numero antes do ponto esta acima da casa da dezena tambem é mostrado (31.5 seria mostrado 3.015000000000000E+001)

```
program endoffile;

var quebrado: real;
begin
    quebrado := 3.15;
    write(quebrado);
end.
```

para formatar a saida de um valor quebrado evitando dele mostrar um monte de 0 podemos usar dois pontos seguido do valor antes do ponto e dois pontos seguido do valor depois do ponto (um exemplo seria 6:2 que sera mostrado um numero ate a 6º casa antes do ponto e depois do ponto sera mostado só ate 3º casas)

```
program endoffile;

var quebrado: real;
begin
    quebrado := 3.15;
    write(quebrado:6:3);
end.
```

podemos fazer várias operações ao mesmo tempo

```
program endoffile;

var numero: integer;
begin
    numero := 190 + 10 + 100;
    write(numero);
end.
```

ou com vários operadores

```
program endoffile;

var numero: integer;
begin
    numero := 100 + 50 -  30;
    write(numero);
end.
```

porem quando tem operadores de multiplicação e divisão junto com operadores de adição e subtração o compilador segue a mesma regra na matematica fazendo os operadores de multiplicação e divisão primeiro, no caso se fosse 10 + 5 * 2 seria igual a 20 e não 30 porque o compilador faz primeiro 5 * 2 depois soma com o 10

```
program endoffile;

var numero: integer;
begin
    numero := 10 + 5 * 2;
    write(numero);
end.
```

para fazer primeiro a adição e subtração basta colocar ela entre parênteses, assim o compilador vai fazer as que tiver dentro do parênteses primeiros

```
program endoffile;
```

```
var numero: integer;
begin
    numero := (10 + 5) * 2;
    write(numero);
end.
```

a gente tambem pode usar parênteses dentro de outro para ter o mesmo efeito

```
program endoffile;

var numero: integer;
begin
    numero := ((10 + 5) - 8) * 2;
    write(numero);
end.
```

não existe limite para o uso de parênteses só lembrando ele sempre vai fazer a operação dentro para fora

```
program endoffile;

var numero: integer;
begin
    numero := ((((((10 + 5))) * 2)));
    write(numero);
end.
```

existe a biblioteca math que permite algumas operações e funções, uma delas é a ceil que pega um numero quebrado e arredonda ele

```
program endoffile;

uses math;

var numero: real;
begin
    numero := ceil(3.5);
    write(numero:6:2);
end.
```

a função floor da biblioteca math permite zerar o numero quebrado

```
program endoffile;

uses math;

var numero: real;
begin
    numero := floor(3.5);
    write(numero:6:2);
end.
```

para fazer a raiz quadrada usamos a função sqrt da biblioteca math

```
program endoffile;

uses math;
```

```
var numero: real;
begin
    numero := sqrt(9);
    write(numero:6:2);
end.
```

para o quadrado usamos a função sqr da biblioteca math (teoricamente é só multiplicar o numero por ele mesmo)

```
program endoffile;

uses math;

var numero: real;
begin
    numero := sqr(9);
    write(numero:6:2);
end.
```

para potencia usamos duas vezes o operador de multiplicação que é o asterisco (**)

```
program endoffile;

var numero: integer;
begin
    numero := 5 ** 3;
    write(numero);
end.
```

ou podemos usar a função power da biblioteca math para potencia tambem

```
program endoffile;

uses math;

var numero: real;
begin
    numero := power(5,3);
    write(numero:6:2);
end.
```

para retornar o numero absoluto podemos usar a função abs

```
program endoffile;

uses math;

var numero: integer;
begin
    numero := abs(-100);
    write(numero);
end.
```

existem varias bases numéricas além da base decimal (base 10) que só tem 10 digitos antes de incrementar um novo valor assim formando outra casa (0,1,2,3,4,5,6,7,8,9), a base decimal vai do numero 0 ate o numero 9 o próximo numero depois do 9 seria uma concatenação do numero um (1) mais o numero zero (0) formando o

numero dez (0,1,2,3,4,5,6,7,8,9,10), a base decimal é a mais usada pelo ser humano porem não é a única, para usar base decimal no pascal basta digitar o numero normalmente como estamos fazendo nos exemplos anteriores

```
program endoffile;

var numero: integer;
begin
    numero := 315;
    write(numero);
end.
```

além da base decimal existe a base hexadecimal ou base 16, essa base tem 16 digitos que vai de 0 ate 9 e existe a letra A ate F que seria equivalente ao novos numeros (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,10), diferente da base decimal que o numero 10 vem depois do numero 9 na base hexadecimal o numero 10 vem depois do numero F, em outras palavras 9+1 em decimal é igual a 10 já em hexadecimal 9+1 é igual a A (o 10 em hexadecimal seria F+1=10), para manipular numeros hexadecimal em pascal usamos o caracter cifrão antes do numero

```
program endoffile;

var numero: integer;
begin
    numero := $b;
end.
```

embora um numeros em uma base diferente da decimal gera numeros diferentes em decimal, é possível converter uma base para outra então 13b em hexadecimal seria equivalente ao numero 315 em decimal, embora pareça que os números seja diferente é o mesmo valor (13b em hex = 315 em dec), o programa por padrao vai sempre imprimir em base decimal para facilitar para o ser humano mesmo que a gente tenha atribuido uma base diferente

```
program endoffile;

var numero: integer;
begin
    numero := $13b;
    write(numero);
end.
```

outra base conhecida conhecida e usada em programação é base octal que tem apenas 8 digitos (0,1,2,3,4,5,6,7), na base octal o numero 10 seria o numero 8 na base decimal e o numero 10 em decimal seria equivalente ao numero 12 em octal, para usar base octal em pascal usamos o caracter E comercial (&) antes do numero

```
program endoffile;

var numero: integer;
begin
    numero := &473;
    write(numero);
end.
```

embora a base decimal (base 10) seja a base usada pelo ser humano a maquina usa outra base que é a binaria (base 2), o binario tem apenas dois digitos assim sendo 0 e 1 antes de concatenar um novo valor (0,1,10), para usar base binaria em pascal colocamos o caracter porcentagem antes do numero

```
program endoffile;

var numero: integer;
begin
```

```
    numero := %100111011;
    write(numero);
end.
```

outra base que não é padrao na linguagem pascal mais é usada muito em informatica é base64 que é usada para codificar determinadas saidas para ficar facil leitura sem trucar codigos (embora a leitura dela não seja tao facil para ser humano kkkk)

4.1 – Logica booleana

logica booleana diz se determinada ação é verdadeira ou falsa, podemos definir qualquer coisa com logica booleana mesmo aquelas que não tem nada a ver com o computador, a logica booleana só existe dois estados que poderia ser verdadeiro ou falso, sim e não, high e low, andando ou parado, vivo ou morto, ligado ou desligado, positivo ou negativo, bem ou mal (yin yang), qualquer coisa que tenha dois estados que seja oposto do outro pode ser definido como uma logica booleana, na computação qualquer numero igual e maior que 1 é positivo sendo assim verdadeiro (true) e qualquer numero igual e menor que 0 é negativo então falso (false), no pascal existe a variavel boolean que permite armazenar verdadeiro ou falso, alem desse tipo de dados existem duas constantes que são TRUE para o valor verdadeiro e FALSE para o valor falso

```
program endoffile;

var yinyang: boolean;
begin
    yinyang := true;
    write(yinyang);
end.
```

na linguagem pascal talvez não seja possível atribuir um numero a uma variavel do tipo boolean diretamente sem converte-lo devido a maneira que o compilador aloca o espaço para cada tipo, mesmo que essa logica em teoria seja valida não quer dizer que o compilador vai converter os tipos automaticamente e nem interpretar uma logica valida, no caso uma variavel do tipo integer aloca 4bytes na memoria já uma do tipo booleano apenas 1bytes e isso pode da incompatibilidade então sempre use as constante TRUE e FALSE para manipular o tipo boolean

```
program endoffile;

begin
    writeln('integer tamanho: ',sizeof(1));
    write('booleano tamano: ',sizeof(true));
end.
```

tambem podemos fazer uma conversao para o tipo boolean usando ele como se fosse uma função (porem é bem limitado essa conversao)

```
program endoffile;

var yinyang: boolean;
begin
    yinyang := boolean(0);
    write(yinyang);
end.
```

4.2 – Operação bit a bit

quando escrevemos um caracter por exemplo a letra 'a' na verdade o computador processa essa informação em binario (base 2), a letra 'a' é um conjunto de vários bits formando um único byte (um byte é um conjunto de 8bits), o caracter 'a' em binario seria esse conjunto de bits 01100001, se a gente mudar um único bit formaria outro caracter sem ser o caracter 'a', um exemplo seria o codigo 01100010 que seria a letra 'b', uma forma facil de manipular os bits seria converte para outra base então 01100001 seria equivalente ao numero 97 em decimal ou 61

em hexadecimal, então digitar o numero decimal 97 seria o mesmo que digitar o codigo binario 01100001, tambem existem operações chamada bit a bit que manipula cada bit em um byte, as operaçoes mais usadas na informatica são OR (ou), AND (e), XOR (ou Exclusivo) e NOT (negação), a operação OR pega dois conjuntos de bits e compara cada bit gerando um novo conjunto de bits, se um dos dois bits for igual a 1 ele retorna verdadeiro (1) se os dois for 0 ele falso (0)

```
01100001 (97)
01100010 (98)
–----------------
01100011 (99)
```

para fazer a operação bit a bit OR no pascal usamos o operador OR

```
program endoffile;

var num, num2, resu: integer;
begin
    num := 97;
    num2 := 98;
    resu := num or num2;
    write(resu);
end.
```

na operação AND ele retorna verdadeiro apenas se os bits for igual a 1 se um dos dois ou os dois for igual a 0 ele retorna falso

```
01100001 (97)
01100010 (98)
–----------------
01100000 (96)
```

Para fazer a operação and no pascal basta usar o operador and

```
program endoffile;

var num, num2, resu: integer;
begin
    num := 97;
    num2 := 98;
    resu := num and num2;
    write(resu);
end.
```

na operação XOR ele retorna verdadeiro apenas se os dois bits for diferente um do outro

```
01100001 (97)
01100010 (98)
–----------------
00000011 (3)
```

A operação xor no pascal é parecida com as anteriores só mudando para o operador

```
program endoffile;

var num, num2, resu: integer;
begin
    num := 97;
```

```
    num2 := 98;
    resu := num xor num2;
    write(resu);
end.
```

a operação NOT é diferente das outras, ela apenas manipula um único conjunto invertendo todos os bits

```
01100001 (97)
–----------------
10011110 (? - depende do tipo da variavel podendo ser -98 ou outro valor caso não tenha sinal)
```

para fazer a operação not bit a bit, basta coloca o operador not antes do numero

```
program endoffile;

var num, resu: integer;
begin
    num := 97;
    resu := not num;
    write(resu);
end.
```

4.3 – Operadores de comparação e lógicos

as operações logicas diz a maquina como ela deve ser comportar em determinda situação ou evento, e os de comparação que compara dois valores e retorna verdadeiro ou falso dependendo da logica, um dos operadores comparação é o operador de igualdade que compara dois valores e retorna verdadeiro caso eles seja iguais, para usar o operador de igualdade basta usar o simbolo de igual (=)

```
program endoffile;

var num, num2: integer;
     logico : boolean;
begin
    num := 100;
    num2 := 100;
    logico := num = num2;
    write(logico);
end.
```

tambem existe o operador de diferença que retorna verdadeiro caso os dois valores seja diferentes, para usar o operador de diferença usamos o simbolo de menor que seguido do simbolo de maior que

```
program endoffile;

var num, num2: integer;
     logico : boolean;
begin
    num := 100;
    num2 := 300;
    logico := num <> num2;
    write(logico);
end.
```

podemos usar o operador de maior para retornar verdeiro caso o primeiro valor seja maior que o segundo

```
program endoffile;
```

```
var num, num2: integer;
     logico : boolean;
begin
    num := 300;
    num2 := 100;
    logico := num > num2;
    write(logico);
end.
```

ou usar o operador maior ou igual que retorna verdadeiro se o primeiro valor for tanto maior ou igual ao segundo, para usar esse operador basta colocar o simbolo de maior que seguido do igualdade

```
program endoffile;

var num, num2: integer;
     logico : boolean;
begin
    num := 300;
    num2 := 300;
    logico := num >= num2;
    write(logico);
end.
```

alem do operador maior que existe o menor que, que seria o inverso do maior que retornando verdadeiro caso o primeiro valor seja menor que o segundo

```
program endoffile;

var num, num2: integer;
     logico : boolean;
begin
    num := 100;
    num2 := 300;
    logico := num < num2;
    write(logico);
end.
```

tambem existe o operador menor ou igual

```
program endoffile;

var num, num2: integer;
     logico : boolean;
begin
    num := 100;
    num2 := 100;
    logico := num <= num2;
    write(logico);
end.
```

os operadores logicos tambem é possível usar parênteses como os operadores aritméticos para retornar apenas uma única condição de várias

```
program endoffile;

var num, num2: integer;
```

```
    logico : boolean;
begin
    num := 300;
    num2 := 100;
    logico := ((num = num2) <> true);
    write(logico);
end.
```

além dos operadores de comparação existe os operadores logicos AND, OR e NOT que permite manipular os operadores de comparação (não confunda eles com os operadores bit a bit, os operadores bit a bit manipula cada bit em um byte e gera um novo byte, os operadores logico manipula apenas a condição logica e retorna verdadeiro ou falso), o operador logico OR compara duas ou mais condições logicas e retornar verdadeiro caso uma das delas seja verdadeira

```
program endoffile;

var num, num2: integer;
    logico : boolean;
begin
    num := 300;
    num2 := 100;
    logico := (num > 100) or (num2 = 315);
    write(logico);
end.
```

o operador logica and só retorna verdadeiro caso todas as condições seja verdadeira

```
program endoffile;

var num, num2: integer;
    logico : boolean;
begin
    num := 300;
    num2 := 100;
    logico := (num > 100) and (num2 = 315);
    write(logico);
end.
```

o operador not inverte a condição se era verdadeiro vira falso, se é falso vira verdadeiro

```
program endoffile;

var num: integer;
    logico : boolean;
begin
    num := 315;
    logico := not num = 100;
    write(logico);
end.
```

4.4 – Estrutura condicional

a estrutura condicional permite executar determinado trecho do codigo apenas sob determinada condição, caso essa condição não seja verdadeira o programa não executa aquele trecho, a estrutura condicional mais usada é a estrutura if e passar como argumento para ela a condição logica depois usamos a palavra then seguido de um escopo, dentro do escopo colocamos os codigos que sera executado caso a condição seja verdadeira

```
program endoffile;

begin
    if(315 = 315) then
    begin
    end;
end.
```

um outro exemplo que o usuario entra com uma senha ele mostra a mensagem caso a senha seja igual

```
program endoffile;

var senha: string;
begin
    write('digite a senha: ');
    read(senha);
    if(senha = 'eof') then
    begin
        write('senha esta correta');
    end;
end.
```

podemos usar a estruturas if quantas vezes a gente quiser

```
program endoffile;

var numero: integer;
begin
    numero := 315;
    if(numero > 100) then
    begin
        write('numero e maior que 100');
    end;
    if(numero < 100) then
    begin
        write('numero e menor que 100');
    end;
end.
```

tambem podemos usar uma dentro da outra para criar varias condiçoes

```
program endoffile;

var login, senha: string;
begin
    write('digite o login: ');
    readln(login);
    write('digite a senha: ');
    readln(senha);
    if(login = 'fts') then
    begin
        if(senha = '315') then
        begin
            write('login e senha esta correta');
        end;
    end;
end.
```

o código acima tambem seria equivalente ao próximo onde usamos a mesma estrura if com varias condiçoes

```
program endoffile;

var login, senha: string;
begin
    write('digite o login: ');
    readln(login);
    write('digite a senha: ');
    readln(senha);
    if((login = 'fts') and (senha = '315')) then
    begin
        write('login e senha esta correta');
    end;
end.
```

existe o else que deve ser usado sempre depois do if, o else serve para executar um trecho de codigo caso o if não seja executado, o ponto e virgula deve ser usado apenas no ultimo end quando tem if e um else junto

```
program endoffile;

var senha: string;
begin
    write('digite a senha: ');
    read(senha);
    if(senha = 'eof') then
    begin
        write('senha esta correta');
    end
    else
    begin
        write('senha invalida');
    end;
end.
```

alem do if e else existe o else-if que checa outra codinção caso o if não seja executado

```
program endoffile;

var senha: string;
begin
    write('digite a senha: ');
    read(senha);
    if(senha = 'eof') then
    begin
        write('senha esta correta');
    end
    else if(senha = 'fts') then
    begin
        write('senha esta correta');
    end
    else
    begin
        write('senha invalida');
    end;
end.
```

podemos usar quantos else-if a gente quiser

```
program endoffile;

var numero: integer;
begin
    write('digite um numero: ');
    read(numero);
    if(numero = 100) then
    begin
        write('numero e igual a 100');
    end
    else if(numero = 315) then
    begin
        write('numero e 315');
    end
    else if(numero = 1000) then
    begin
        write('numero e 1000');
    end
    else
    begin
        write('numero desconhecido');
    end;
end.
```

quando tem mais de um else if com condição verdadeira o programa vai executar apenas o primeiro, então quando precisamos executar varias condições usamos vários if e não else-if (tambem podemos fazer um if dentro do outro nesses casos ou colocar varias condições em um único if)

```
program endoffile;

var numero: integer;
begin
    numero := 150;
    if(numero > 100) then
    begin
        write('numero e maior que 100');
    end
    else if(numero < 1000) then
    begin
        write('numero e menor que 1000');
    end
    else
    begin
        write('numero desconhecido');
    end;
end.
```

para não precisar ficar usando muitos else-if existe a estrutura case que passamos para ela determinado valor e ela executa determinada parte, para usar ela basta passar o valor e palavra case seguido das condiçoes com um simbolo de dos pontos e o trecho do codigo que sera executado e terminamos ela com um end

```
program endoffile;

var numero : integer;
begin
    numero := 3;
    case numero of
```

```
        1: write('seu numero foi 1');
        2: write('seu numero foi 2');
        3: write('seu numero foi 3');
        315: write('seu numero foi 4');
    end;
end.
```

podemos colocar vários codigos em cada trecho porem temos que usar um escopo begin e end a cada trecho

```
program endoffile;

var numero, idade : integer;
     nome: string;
begin
    numero := 2;
    case numero of
        1: begin
              write('digite seu nome: ');
              read(nome);
              write('seu nome e ', nome);
           end;
        2: begin
              write('digite sua idade: ');
              read(idade);
              write('sua idade e ', idade);
           end;
    end;
end.
```

o else também pode ser usado para definir um trecho a ser executado caso nenhum outro seja executado

```
program endoffile;

var numero : integer;
begin
    numero := 3;
    case numero of
        1: write('seu numero foi 1');
        2: write('seu numero foi 2');
        3: write('seu numero foi 3');
        else write('nenhuma opcao')
    end;
end.
```

tambem podemos colocar vários numeros seperado por virgula para executar um único trecho

```
program endoffile;

var numero : integer;
begin
    numero := 3;
    case numero of
        1,2,3,4,5,: write('seu numero foi 1,2,3,4,5');
        10,15,50: write('seu numero foi 10,15 e 50');
        else write('nenhuma opcao')
    end;
end.
```

ou usar duas vezes o ponto para definir uma range de um numero ao outro

```
program endoffile;

var numero : integer;
begin
    numero := 3;
    case numero of
        1 .. 100 ,: write('seu numero esta entre 1 a 100');
        101 .. 1000: write('seu numero esta entre 101 e 1000');
        else write('nenhuma opcao')
    end;
end.
```

4.5 – Estrutura de repetição

a estrutura de repetição diferente da estrutura condicional repete determinado trecho de codigo essa repetição é com base em determinada logica condicional, uma das estruturas de repetição é a while que repete enquanto a condição for verdadeira para ser mais especifico a cada loop ele checa a codição se ela for verdadeira ele executa o trecho do codigo se for falso ele para o loop e pula a estrutura, para usar o while basta passar como argumento a codição seguido da palavra do e depois criar um escopo para codigo, dentro desse escopo temos que manipular a condição com uma logica que pare o loop se não o loop sera infinito e o programa não vai passar daquela estrutura, abaixo tem um programa preso em um loop infinito

```
program endoffile;

begin
    while true do
    begin
        writeln('isso faz parte do loop');
    end;
    write('nao faz parte do loop');
end.
```

por outro lado se a condição for falsa o programa não executa o trecho do loop

```
program endoffile;

begin
    while false do
    begin
        writeln('isso faz parte do loop');
    end;
    write('nao faz parte do loop');
end.
```

e sempre recomendado usar uma variavel para manipular o loop, assim podemos modificar o valor da variavel dentro do loop

```
program endoffile;

var loop: boolean;
begin
    loop:= true;
    while loop do
    begin
```

```
        writeln('isso faz parte do loop');
    end;
    write('nao faz parte do loop');
end.
```

para ficar mais facil manipular a quantidade loops podemos criar um contador e fazer uma condição enquanto esse contador for menor ou maior que determinado numero a condição retorne verdadeira, e a cada novo loop ele incrementa ou decrementa assim ate chega naquele numero invertendo a logica pra dar falso e sessar o loop

```
program endoffile;

var contador: integer;
begin
    contador := 1;
    while contador < 10 do
    begin
        writeln('isso faz parte do loop');
        contador := contador + 1;
    end;
    write('nao faz parte do loop');
end.
```

para incrementar tambem podemos usar a função inc passar como argumento a variavel e o valor do incremento

```
program endoffile;

var contador: integer;
begin
    contador := 1;
    while contador < 10 do
    begin
        writeln('isso faz parte do loop');
        inc(contador,1);
    end;
    write('nao faz parte do loop');
end.
```

para decrementar usamos dec no lugar de inc

```
program endoffile;

var contador: integer;
begin
    contador := 10;
    while contador > 1 do
    begin
        writeln('isso faz parte do loop');
        dec(contador,1);
    end;
    write('nao faz parte do loop');
end.
```

veja um exemplo de uma simples tabuada

```
program endoffile;
```

```
var contador, numero: integer;
begin
    contador := 1;
    write('digite um numero: ');
    read(numero);
    while contador < 10 do
    begin
        writeln(numero * contador);
        inc(contador,1);
    end;
end.
```

outra estrutura de repetição é a estrutura for que permite escolher a quantidade loops, diferente da while que repete enquanto for verdadeiro, para usar a estrutura for basta passar uma variavel que sera o contador, depois atribuir um valor inicial nela seguido da palavra to e um numero final seguido da palavra do (a estrutura for vai repetir do numero inicial ate o numero final)

```
program endoffile;

var contador: integer;
begin
    for contador := 1 to 10 do
    begin
         writeln(contador);
    end;
end.
```

tambem podemos faz o contador regressivo colocando dowto no lugar do to

```
program endoffile;
var contador: integer;
begin
    for contador := 10 downto 1 do
    begin
         writeln(contador);
    end;
end.
```

outra estrutura de repetição é a repeat, diferente do for e do while essa estrutura vai repetir pelo menos uma vez, para usar a estrutura repeat basta declarar ele sem escopo seguido do codigo e no final a palavra until seguido da condição logica, a condição logica do repeat é diferente da while e for ela inverte a logica ou seja para continuar no loop a logica deve ser falsa

```
program endoffile;

var numero : integer;
begin
    numero := 1;
    repeat
        writeln(numero);
        inc(numero,1);
    until numero = 10;
end.
```

tanto a estrutura for quanto a while ou outro pode ser usado mais de uma estrutura dentro embora o programa vai executar todas as de dentro primeiro ou a cada loop de uma de fora executa o loop de dentro, abaixo tem uma tabuada melhorada que mostra a tabuada de 1 ate 10

```
program endoffile;

var contador, contador2, numero: integer;
begin
    contador := 1;
    contador2 := 1;
    numero := 1;
    while contador <= 10 do
    begin
        while contador2 <= 10 do
        begin
            write(contador * contador2,' ');
            inc(contador2,1);
        end;
        writeln;
        contador2 := 1;
        inc(contador,1);
    end;
end.
```

existe um pulo incondicional que pode ser usado como laço que é o goto, para usar o goto temos que criar um label para isso basta usar a palavra label seguido de um nome fora do escopo, depois basta usar o label dentro do escopo seguido de dois pontos

```
program endoffile;

label kodo;
begin
    kodo:
end.
```

agora usamos o goto depois ou antes do label para pular ou voltar para label, dependendo da forma que você usa o label e goto seu programa pode ficar preso em um loop infinito como ele não faz checagem sendo apenas um pulo incondicional podemos usar a condição if para evitar isso

```
program endoffile;

label kodo, fts;
var cont: integer;
begin
    cont := 0;
    kodo:
    if cont > 10 then
    begin
        goto fts;
    end;
    writeln(cont);
    inc(cont,1);
    goto kodo;
    fts:
end.
```

5.0 – Funções

as funções são trechos de procedimentos que o programa vai executar toda a vez que a função for chamada como por exemplo a função write que exibe um argumento na tela, as funções serve para facilitar para o programador não precisar criar o mesmo trecho de codigo varias vezes seguidas criando um único trecho e chamando ele

quantas vezes desejar, quando o programa acha uma chamada de função no codigo ele automaticamente pula para o endereço da chamada da função e executa ela e quando termina o trecho da função ele destrói todas as alocações e depois retorna para o endereço anterior da chamada dela e continua o codigo, no pascal existem dois tipos de funções são elas as procedures e os function, as function são funções que tem um retorno e as procedures são funçoes sem retorno, para criar uma função procedure basta colocar a palavra procedure fora do escopo seguido de um nome para função esse nome pode ser qualquer um porem não pode começar com numeros e nem simbolo e não pode ser nomes de funçoes já usadas, depois um escopo para o trecho de codigos

```
program endoffile;

procedure kodo;
begin
end;

begin
end.
```

podemos colocar qualquer função ou procedimento dentro da nossa função

```
program endoffile;

procedure kodo;
begin
    write('minha funçao');
end;

begin
end.
```

para chamar executar a nossa função basta chamar ela no escopo principal

```
program endoffile;

procedure kodo;
begin
    write('minha funçao');
end;

begin
    kodo();
end.
```

podemos chamar a nossa função quantas vezes a gente quiser

```
program endoffile;

procedure kodo;
begin
    writeln('minha funçao');
end;

begin
    kodo();
    kodo();
    kodo();
end.
```

tambem podemos criar variáveis dentro da nossa função bastando criar elas antes do begin

```
program endoffile;

procedure kodo;
var texto: string;
begin
    texto := 'minha funçao';
    writeln(texto);
end;

begin
    kodo();
    kodo();
    kodo();
end.
```

para criar um argumento na nossa função basta colocar parênteses depois do nome e depois criar uma variavel dentro dela (não precisa colocar a palavra var apenas colocar o nome e o tipo da variavel), quando chamar a nossa função temos que passar um argumento do mesmo tipo da variavel que criamos

```
program endoffile;

procedure kodo(texto: string);
begin
    writeln(texto);
end;

begin
    kodo('forum end of file');
    kodo('forum eof');
end.
```

podemos criar quantos argumentos a gente quiser bastando separar eles com ponto e virgula

```
program endoffile;

procedure kodo(nome: string; ano: integer; idade: integer);
begin
    writeln('seu nome: ',nome);
    writeln('sua idade: ', idade);
    writeln('ano atual: ', ano);
end;

begin
    kodo('flavio',2014,21);
end.
```

as variaveis criadas para determinadas funções não podem ser acessadas por outras funções isso porque elas não pertence ao mesmo escopo, essas variaveis que pertence ao escopo sempre fica antes do begin daquele escopo e são chamadas de variáveis locais, além das variáveis locais existem as variáveis globais que são criadas antes de qualquer função e assim não pertencendo a nenhum escopo e ao mesmo tempo pertencendo a todos, essa variaveis globais pode ser acessada por qualquer função e existe uma diferença entre as variaveis locais e globais, as variaveis locais são destruidas quando a função e finalizada já as variaveis globais apenas quando o programa e finalizado

```
program endoffile;

var texto: string;
```

```
procedure kodo();
begin
    texto := texto + ' show';
end;

begin
    texto := 'anime e';
    kodo();
    write(texto);
end.
```

uma forma de acessar uma variavel local fora do escopo é usar ponteiro, se você passar o endereço de memoria da variavel pode modicar ela pelo ponteiro especifico, para evitar problemas sempre use ponteiros do tipo pointer para transporta um endereço de memoria, depois basta atribuir o endereço para um ponteiro especifico e manipular ele

```
program endoffile;

procedure kodo(endereco: pointer);
var ponteiro: ^integer;
begin
    ponteiro := endereco;
    ponteiro^ := 21;
end;

var idade: integer;
begin
    kodo(@idade);
    write(idade);
end.
```

outro tipo de função é a function que permite retornar um valor, o funcionamento desse tipo de função e parecido com o procedure porem a gente deve especificar o tipo de retorno

```
program endoffile;

function kodo: integer;
begin
end;

begin
end.
```

para retornar um valor basta atribuir determinado valor para o nome da função, quando determinado valor e retornado para função ela automaticamente finaliza então sempre retorne determinado valor no final função e nunca no começo

```
program endoffile;

function kodo: integer;
begin
    kodo := 315;
end;

var numero: integer;
begin
    numero := kodo();
```

```
    write(numero);
end.
```

podemos fazer qualquer coisa no function que a gente tava fazendo com o procedure passar como argumento, variveis locais e globais entre outras coisas

```
program endoffile;

function kodo(x : integer; y: integer): integer;
var resu : integer;
begin
    resu :=  x + y;
    kodo := resu;
end;

var numero: integer;
begin
    numero := kodo(300,15);
    write(numero);
end.
```

tambem é possível chamar a própria função dentro dela mesma o nome disso é função recursiva, tome cuidado ao usar funçoes recursiva isso pode consumir toda a memoria dependendo da forma que ela for chamada caso gere um loop infinito

```
program endoffile;

procedure kodo(contador: integer);
begin
    if(contador > 0) then
    begin
        writeln(contador);
        kodo(contador - 1);
    end;
end;

begin
    kodo(10);
end.
```

5.1 – Diretivas de pré-processamento

as diretivas de pre-processamento permite manipular a forma que o compilador vai compilar o codigo, removendo determinada parte ou só compilando determinada parte caso acontece determinada checagem ou condição, essas diretivas não interfere no programa final apenas no compilador antes de gerar o programa, as diretivas na linguagem pascal é usado o simbolo de chaves seguido do cifrao e o codigo da diretiva depois fecha chaves

```
{$ }
```

uma diretiva muito usada é o $define que permite definir uma diretiva ou atribuir determinado valor a uma, para usar a diretiva $define basta colocar um nome para ela

```
program endoffile;

{$define kodo}
```

```
begin
end.
```

tambem podemos destruir a diretiva $define com o $undef

```
program endoffile;

{$define kodo}
{$undef kodo}

begin
end.
```

com a diretiva $ifdef podemos checar para ver se existe determinada diretiva, caso exista ele executa os codigos abaixo dela ate a diretiva fim que seria $endif

```
program endoffile;

{$define kodo}

begin
    {$ifdef kodo}
     write('existe a diretiva');
    {$endif}
end.
```

tambem podemos colocar uma diretiva $else para executar caso o $ifdef não seja executado

```
program endoffile;

{$define kodo}
{$undef kodo}

begin
    {$ifdef kodo}
     write('existe a diretiva');
    {$else}
     write('nao existe essa diretiva');
    {$endif}
end.
```

o compilador todas as diretivas e gera um novo codigo excluindo dinamicamente as que não foram executadas antes da compilação, então o codigo anterior para o compilador seria equivalente a essa na compilação

```
program endoffile;

begin
    write('nao existe essa diretiva');
end.
```

alem da diretiva $ifdef existe a diretiva $ifndef que seria parecida só que com a logica invertida

```
program endoffile;

{$define kodo}

begin
```

```
    {$ifndef kodo}
     write('existe a diretiva');
    {$else}
     write('nao existe essa diretiva');
    {$endif}
end.
```

5.2 – Criando biblioteca unit

as bibliotecas mais usadas na linguagem pascal são as units que permite armazenar funções prontas para reaproveitar determinados codigo em novos projetos, uma biblioteca unit se dividi em 3 partes que são unit, interface e implementation, a primeira parte a unit usamos para renomear nossa unit ela tem a mesma função que o program (isso não é regra mais tente sempre usar o nome na unit com o mesmo nome do arquivo no caso meu arquivo se chama kodo.pas)

```
unit kodo;
```

a segunda parte interface serve para adicionar a assinatura (prototipo) da função que sera usada fora da unit, se a gente não colocar a assinatura dela as source que declarar nossa unit não vai enxergar a função, as vezes criamos função que devera ser chamada apenas dentro da unit então não precisamos declarar esse prototipo, o prototipo ou assinatura é apenas o nome da função e seu argumento sem o escopo

```
unit kodo;
interface
```

a ultima parte é a implementation que vai ser onde vamos escrever nossas funções, tambem devemos criar um escopo pricipal no final do codigo

```
unit kodo;
interface
implementation
begin
end.
```

O arquivo biblioteca deve ter a extensao .pas e ele deve ficar junto com o codigo fonte ou em uma variavel de ambiente do sistema ou do compilador, para declarar a nossa biblioteca basta usar o uses seguido do nome do arquivo (sem a extensao)

```
program endoffile;

uses kodo;

begin
end.
```

para criar a função basta criar ela dentro do implementation e colocar o prototipo dela dentro do interface

```
unit kodo;

interface
function fts_quadrado(x : integer): integer;

implementation
function fts_quadrado(x : integer): integer;
begin
   fts_quadrado := x * x;
end;
```

```
begin
end.
```

depois podemos usar a nossa função dentro da unit em qualquer codigo fonte que a gente quiser bastando ter a unit junto e declarar ela no codigo

```
program endoffile;

uses kodo;

var resu : integer;
begin
    resu := fts_quadrado(5);
    write(resu);
end.
```

as variaveis globais deve ser criada dentro da interface

```
unit kodo;

interface
var x: integer;

implementation

begin
end.
```

podemos atribuir determinado valor a uma variavel ou chamar determinada função quando a unit é declarada bastando atribuir o valor ou chamar a função dentro do escopo principal (isso é chamado construtor)

```
unit kodo;

interface
var x: integer;

implementation

begin
    x := 315;
end.
```

as variaveis que são declaradas dentro do implementation ou as função que não tem prototipos no interface só podem ser usadas pelo escopo principal da unit, porem é possível atribuir o valor para uma variavel que esteja dentro da interface ou chamar uma função por outra que não tenha prototipo

```
unit kodo;

interface
var x: integer;

implementation

var y, k: integer;

begin
```

```
    y := 100;
    k := 300;
    x := y + k;
end.
```

5.3 – Biblioteca dinamica (DLL)

outra biblioteca usada não só pela linguagem pascal mais por qualquer outra linguagem de programação são as dll, as dll são bibliotecas carregadas em tempo de execução (runtime), essa biblioteca é muito usada pelo sistema operacional windows para armazenar as API do sistema, a vantagem da dll que a função é criada apenas quando precisamos dela e pode ser liberada a qualquer momento economizando memoria, outra vantagem que uma dll criada em uma linguagem pode ser usada em outra linguagem, a desvantagem que se a dll faltar o programa vai ficar faltando a função e certamente vai da erro, para criar uma dll precisamos usar a palavra library seguido do nome da dll

```
library kodo;
```

criamos a função com a palavra export no final, e depois usamos um exports com o nome da função, para terminar criamos um escopo

```
library kodo;

function fts_quadrado(x : integer): integer ; export;
begin
   fts_quadrado := x * x;
end;

exports fts_quadrado;

begin
end.
```

quando existe mais de uma função devemos colocar virgula para separar os nomes no exports

```
library kodo;

function fts_quadrado(x : integer): integer ; export;
begin
   fts_quadrado := x * x;
end;

function fts_cubo(x : integer): integer ; export;
begin
   fts_cubo := x * x * x;
end;

exports fts_quadrado, fts_cubo;

begin
end.
```

com a nossa dll criada agora temos que aprender como usar a função que está dentro dela, existem varias formas de acessar a função uma delas e a mais facil de todas é usar o external do pascal outra seria abrir a dll com api do sistema e apontar um ponteiro direto para função no endereço da dll, para carregar a dll e instanciar a função com o external podemos criar o prototipo da função com a palavra external no final seguido da dll, e depois só usar função normalmente muito facil (quando mexemos com dll temos que saber como é os argumentos e retorno da mesma se não a gente não vai conseguir usar)

```
program endoffile;

function fts_quadrado(x : integer): integer ; external 'kodo.dll';

var resu: integer;
begin
    resu := fts_quadrado(5);
    write(resu);
end.
```

outra forma de usar uma função dentro de uma dll seria carregar ela com a API LoadLibrary da biblioteca windows, nessa função basta passar como argumento a dll e atribuir o retorno para uma variavel do tipo HANDLE, tambem podemos liberar essa dll a qualquer momento com a API FreeLibrary e passar como argumento pra ela a variavel do tipo HANDLE

```
program endoffile;

uses windows;

var dll: HANDLE;
begin
    dll := LoadLibrary('kodo.dll');
    FreeLibrary(dll);
end.
```

O exemplo anterior a gente apenas carrego e libero a dll, para usar a função temos que criar um prototipo para a função no caso o prototipo deve ter os mesmos tipos de argumentos que a função dentro da dll e tambem o mesmo tipo de retorno, para criar o prototipo usamos type seguido do nome que pode ser qualquer um, colocamos igual seguindo da palavra function e os mesmos argumentos da função que esta dentro da dll

```
program endoffile;

uses windows;

type funcao = function(x: integer):integer;

var dll: HANDLE;
begin
    dll := LoadLibrary('kodo.dll');
    FreeLibrary(dll);
end.
```

agora basta declarar uma variavel daquele tipo, usar API GetProcAddess para pegar o endereço da função dentro da dll, nessa api passamos como argumento a variavel do tipo HANDLE seguido de uma string que seria o nome da função, atribuimos essa api para a variavel (tambem devemos usar typecast para converter para aquele tipo)

```
program endoffile;

uses windows;

type funcao = function(x: integer):integer;

var dll: HANDLE;
     f : funcao;
begin
    dll := LoadLibrary('kodo.dll');
    f := funcao(GetProcAddress(dll,'fts_quad'));
```

```
    FreeLibrary(dll);
end.
```

agora podemos usar aquela variavel como se fosse a própria função dentro da dll

```
program endoffile;

uses windows;

type funcao = function(x: integer):integer;

var dll: HANDLE;
     f : funcao;
     numero: integer;
begin
    dll := LoadLibrary('kodo.dll');
    f := funcao(GetProcAddress(dll,'fts_quad'));
    numero := f(5);
    write(numero);
    FreeLibrary(dll);
end.
```

6.0 – Conversão de tipo

um tipo de conversão muito usado no pascal é a typecast que converte um tipo de dados em outro tipo por exemplo um caracter em um tipo inteiro, para usar o typecast basta usar o tipo como se fosse uma função e passar como argumento o valor que a gente quer converte (esses dois tipos deve ser compativel)

```
program endoffile;

var letra: char;
     numero: integer;
begin
    letra := 'a';
    numero := integer(letra);
    write(numero);
end.
```

o typecast é perfeito para fazer sistemas criptograficos mudando os caracter de uma string para numeros inteiros e depois fazendo operações aritimeticas nele, veja um exemplo simples de uma cifra de ceasar

```
program endoffile;

var texto: string;
     numero, cont: integer;
begin
    texto := 'kodo no kami';
    cont := 0;
    while cont < (length(texto) - 2) do
    begin
        numero := integer(texto[cont]);
        numero := numero + 3;
        texto[cont] := char(numero);
        inc(cont,1);
    end;
    write(texto);
end.
```

podemos converter os dados inteiros para string com a função inttostr da biblioteca sysutils, essa conversão gera uma string com os mesmos caracteres dos números, por exemplo o numero 315 vai gerar a string '315', para usar o inttostr basta passar como argumento o valor numero e atribuir o retorno para uma variável string

```
program endoffile;

uses sysutils;

var numero: integer;
     texto: string;
begin
    numero := 315;
    texto := inttostr(numero);
    write(texto);
end.
```

tambem é possível converter uma string que tenha caracteres numérico em um tipo inteiro usando a função strtoint da biblioteca sysutils

```
program endoffile;

uses sysutils;

var numero: integer;
     texto: string;
begin
    texto := '315';
    numero := strtoint(texto);
    write(numero);
end.
```

podemos converter um numero real para string usando a função floattostr da biblioteca sysutils

```
program endoffile;

uses sysutils;

var quebrado: real;
     texto: string;
begin
    quebrado := 31.5;
    texto := floattostr(quebrado);
    write(texto);
end.
```

com a função strtofloat da para converter uma string com números reais para um número real

```
program endoffile;

uses sysutils;

var quebrado: real;
     texto: string;
begin
    texto := '31.5';
    quebrado := strtofloat(texto);
    write(quebrado);
end.
```

podemos formatar vários tipos em uma única string usando a função format da biblioteca sysutils, nessa função passamos como argumento a string que seria a formatação e as variaveis dentro de um colchete separado por virgula, tambem devemos atribui ela para uma variavel do tipo string, para diferenciar os tipos na string de formatação usamos %c para os tipos caracter, %s para as strings, %d para os inteiros, %x para hexadecimais, %o para os octais, %f para os reis

```
program endoffile;

uses sysutils;

var numero: integer;
     texto: string;
begin
    numero := 315;
    texto := format('%x',[numero]);
    write(texto);
end.
```

podemos escrever qualquer coisa na string de formatação e tambem incluir quantas variaveis a gente quiser porem sempre seguindo a ordem na inclusão

```
program endoffile;

uses sysutils;

var numero: integer;
     texto, nome: string;
begin
    numero := 315;
    nome := 'kodo no kami';
    texto := format('seu nome e %s seu numero da sorte e %d',[nome,numero]);
    write(texto);
end.
```

6.1 – Manipulando o tempo

podemos manipular o tempo com a linguagem pascal usando várias funções, uma delas é o delay da biblioteca crt que pausa o programa por um determinado tempo, para usar a função delay basta passar como argumento o tempo em milésimos de segundos ou seja 1000 é igual a 1 segundo

```
program endoffile;

uses crt;

begin
    writeln('espere 1 segundo');
    delay(1000);
    write('obrigado por esperar');
end.
```

a função delay costuma não existir em determinados compiladores por causa da biblioteca crt então é recomendado usar a função sleep da biblioteca windows no lugar dele

```
program endoffile;

uses windows;
```

```
begin
    writeln('espere 2 segundo');
    sleep(2000);
    write('obrigado por esperar');
end.
```

podemos usar o delay ou sleep para fazer um contador regressivo (se eu usar o tempo do contador direto no sleep tambem vai funciona porem ele vai pausar e não vai mostrar os segundos restantes)

```
program endoffile;

uses windows;

var segundos: integer;
begin
    write('digite os segundos: ');
    read(segundos);
    while segundos > 0 do
    begin
        writeln('contador: ',segundos);
        dec(segundos,1);
        sleep(1000);
    end;
    write('buuuuuuum');
end.
```

o exemplo anterior tambem poderia ser feito assim

```
Program endoffile;

uses windows;

var segundos: integer;
begin
    write('digite os segundos: ');
    read(segundos);
    sleep(segundos * 1000);
    write('buuuuuuum');
end.
```

para a gente pegar a hora e a data atual usamos a função now da biblioteca sysutils, a função now pega a hora e data de forma não formatada em numeros

```
program endoffile;

uses sysutils;

begin
    write(now());
end.
```

tambem podemos atribuir esse tempo em uma variavel do tipo TDateTime para facil manipulação

```
program endoffile;

uses sysutils;
```

```
var tempo: TDateTime;
begin
    tempo := now();
    write(tempo);
end.
```

podemos converter aquele tempo em uma string com a hora formatada usando a função timetostr da biblioteca sysutils

```
program endoffile;

uses sysutils;

var tempo: TDateTime;
     hora: string;
begin
    tempo := now();
    hora := timetostr(tempo);
    write(hora);
end.
```

podemos fazer o mesmo com a data usando a função datetostr da biblioteca sysutils

```
program endoffile;

uses sysutils;

var tempo: TDateTime;
     data: string;
begin
    tempo := now();
    data := datetostr(tempo);
    write(data);
end.
```

é possível transformar em uma string que tenha tanto a data quanto a hora com a função datetimetostr da biblioteca sysutils

```
program endoffile;

uses sysutils;

var tempo: TDateTime;
     data: string;
begin
    tempo := now();
    data := datetimetostr(tempo);
    write(data);
end.
```

podemos formatar a data ou a hora da maneira que a gente quiser usando a função formatdatetime da biblioteca sysutils, nessa função passamos como argumento uma string que sera o formato a tempo atual, a string de formatação pode ter os caracteres representativo h para hora, n para os minutos, s para os segundos, d para o dia, m para o mês e y para o ano, se a gente combinar o mesmo caracter representativo varias vezes ele retorna uma representação da data ou hora de forma diferente, por exemplo yy ele retorna o ano 14 e yyyy ele retorna o ano 2014

```
program endoffile;

uses sysutils;

var tempo: TDateTime;
     data: string;
begin
    tempo := now();
    data := formatdatetime('hh:nn:ss (dd/mm/yy)',tempo);
    write(data);
end.
```

para escrever um texto dentro da string de formatação usamos aspas duplas com o texto dentro

```
program endoffile;

uses sysutils;

var tempo: TDateTime;
     data: string;
begin
    tempo := now();
    data := formatdatetime('"hora: "hh:nn:ss - "data: "dddd/mmmm/yyyy',tempo);
    write(data);
end.
```

podemos calcular a diferença entre dois tempos subtraindo um pelo outro

```
Program endoffile;

uses sysutils, crt;

var tempo1, tempo2, tempo3: TdateTime;
     dif: string;
begin
    tempo1 := now();
    writeln('aperte ente depois de alguns segundos');
    readkey;
    tempo2 := now();
    tempo3 := tempo2 - tempo1;
    dif := formatdatetime('hh:nn:ss',tempo3);
    write('diferenca de tempo e de ',dif);
end.
```

6.2 – Manipulando arquivo

quando o programa está em execução ele fica na memoria o que armazenamos em variaveis se perde assim que o programa é finalizado, porém as vezes temos dados que queremos armazenar para acessar mais para frente em novas execuções ou informação que deve ser salva então temos que armazená-las em arquivos, para manipular um arquivo temos que abrir ele com determinada permissão, as 3 permissões padrões de arquivos são escrita (rewrite), leitura (reset) e concatenação (append), a permissão de escrita (rewrite) serve para escrever em arquivos sem ela a gente não consegue escrever nesse arquivo, ela tambem cria um arquivo caso ele não exista e apaga o conteúdo dele caso ele exista, a permissão de leitura (reset) permite abrir um arquivo para ler o seu conteúdo mais sem modifica-lo, a permissão de concatenação (append) permite abrir um arquivo para escrita porem se houver conteúdo nele sera escrito depois desse conteudo sem apaga-lo, para abrir um arquivo em pascal usamos a função assign passamos como argumento para ela uma variavel do tipo text, e uma string que seria a localização do arquivo que vamos abrir, tambem podemos fechar o arquivo com a função close e passar como argumento para ela a mesma variavel

```
program endoffile;

var arquivo: text;
begin
    assign(arquivo,'c:\users\fts315\desktop\kodo.txt');
    close(arquivo);
end.
```

ate agora apenas abrimos o arquivo sem nenhuma permissão, para da permissão de escrita usamos a função rewrite e passamos como argumento para ela a variavel que abrimos, para a gente escrever no arquivo usamos a função write passamos como argumento para ela o variavel depois a string que queremos escrever no arquivos

```
program endoffile;

var arquivo: text;
begin
    assign(arquivo,'c:\users\fts315\desktop\kodo.txt');
    rewrite(arquivo);
    write(arquivo,'esperando a terceira temporada de high school dxd =/ ');
    close(arquivo);
end.
```

só armazenado dentro do arquivo depois que o arquivo for fechado por isso sempre use a função close para fechar o arquivo depois que escrever nele, tambem podemos escrever quantas vezes a gente quiser

```
program endoffile;

var arquivo: text;
begin
    assign(arquivo,'c:\users\fts315\desktop\kodo.txt');
    rewrite(arquivo);
    writeln(arquivo,'esperando a terceira temporada de high school dxd =/ ');
    write(arquivo,'outro anime massa e o high school of dead');
    close(arquivo);
end.
```

porem se a gente fechar o arquivo e reabrir ele com permissão de escrita ele vai apagar o conteúdo que estava escrito para escrever o novo

```
program endoffile;

var arquivo: text;
begin
    assign(arquivo,'c:\users\fts315\desktop\kodo.txt');
    rewrite(arquivo);
    writeln(arquivo,'esperando a terceira temporada de high school dxd =/ ');
    writeln(arquivo,'outro anime massa e o high school of dead');
    close(arquivo);
    rewrite(arquivo);
    write(arquivo,'ultimo anime que eu assistir foi o no game no life ^^');
    close(arquivo);
end.
```

então usamos a função append no lugar da função rewrite para concatenar o conteudo que já estava no arquivo ao invés de sobrescrever o arquivo com o novo

```
program endoffile;

var arquivo: text;
begin
    assign(arquivo,'c:\users\fts315\desktop\kodo.txt');
    rewrite(arquivo);
    writeln(arquivo,'esperando a terceira temporada de high school dxd =/ ');
    writeln(arquivo,'outro anime massa e o high school of dead');
    close(arquivo);
    append(arquivo);
    write(arquivo,'ultimo anime que eu assisti foi o no game no life ^^');
    close(arquivo);
end.
```

agora que a gente já sabe como escrever em um arquivo vamos aprender como ler o arquivo, para leitura é o mesmo esquema usamos a função assign e depois usamos a função reset no lugar do rewrite e append para permissão de leitura, criamos uma variavel do tipo string e usamos a função read para ler o arquivo, passamos como argumento para a função read a variavel do tipo text que abrimos o arquivo e a variavel do tipo string onde sera armazenado o texto dentro do arquivo, porem isso vai uma linha do arquivo

```
program endoffile;

var arquivo: text;
     texto: string;
begin
    assign(arquivo,'c:\users\fts315\desktop\kodo.txt');
    reset(arquivo);
    read(arquivo,texto);
    close(arquivo);
    write(texto);
end.
```

para ler todas as linhas no arquivo usamos o laço while passamos como argumento para ele a função eof com negação e a variavel do arquivo como argumento, dentro usamos o readln isso diz para o programa ler linha por linha ate o final do arquivo

```
program endoffile;

var arquivo: text;
     texto: string;
begin
    assign(arquivo,'c:\users\fts315\desktop\kodo.txt');
    reset(arquivo);
    while not eof(arquivo) do
    begin
        readln(arquivo,texto);
        writeln(texto);
    end;
    close(arquivo);
end.
```

podemos deletar um arquivo com função DeleteFile da biblioteca windows, nessa função basta passar como argumento o local do arquivo

```
program endoffile;

uses windows;
```

```
begin
    DeleteFile('c:\users\fts315\desktop\kodo.txt');
end.
```

é possível mover um arquivo com a função MoveFile da biblioteca windows, passamos como argumento para essa função o local do arquivo, e o local para onde vamos mover ele

```
program endoffile;

uses windows;

begin
    MoveFile('c:\users\fts315\desktop\kodo.txt','c:\users\fts315\desktop\kodo\kodo.txt');
end.
```

para copiar o arquivo usamos CopyFile da biblioteca windows, passamos como argumento o local do arquivo e o local para onde vamos mover, e um terceiro argumento que seria verdadeiro ou falso para sobreescrever um arquivo caso ele já exista

```
program endoffile;

uses windows;

begin
    CopyFile('c:\users\fts315\desktop\kodo.txt','c:\users\fts315\desktop\kodo\kodo.txt',true);
end.
```

7.0 – Orientação a objeto

a orientação a objeto ou POO (programação orientada a objeto) é uma forma de manipular parte do codigo como um único objeto ou uma estrutura, com a orientação a objeto podemos abstrair parte do codigo para ser acessada de formas diferentes, podemos reaproveitar aquela parte do codigo para construir novas, ou podemos usar aquela parte de forma polimorfica, qualquer objeto gerado não interfere em um outro igual que foi gerado da mesma estrutura, a orientação a objeto trata qualquer variavel ou função de um objeto como um único objeto ou seja faz parte de um todo, um exemplo bem basico seria um carro onde sabemos que abrir uma porta e girar o volante gera efeitos diferentes embora ambos faça parte do mesmo carro, tambem podemos usar aquela arquitetura do carro para criar um novo modelo de carro com base no anterior na orientação a objeto isso é possível e é chamado de herença, ou então abstrair partes no caso não temos que saber como o motor do carro funciona para aprender a dirigir ele, é dessa forma que os codigos orientado a objeto funciona

7.1 – Orientação a objeto: Class, Object, Record e With

a estrutura usada para orientação a objeto é a class, essa estrutura deve ser criada fora do escopo principal, nela temos que usar a palavra type seguido do nome da nossa class que pode ser qualquer um, em seguida o simbolo de igual com a palavra class para definir o inicio do escopo e um end para definir o final do escopo

```
program endoffile;

type kodo = class end;

begin
end.
```

para deixar o codigo mais legivel e organizado, eu costumo fazer dessa forma em linhas separadas

```
program endoffile;
```

```
type kodo = class
end;

begin
end.
```

no nosso codigo podemos criar quantas class a gente quiser

```
program endoffile;

type kodo = class
end;

type fts = class
end;

begin
end.
```

no caso se a gente for criar varias class uma embaixo da outro não precisamos usar a palavra type em todas elas apenas na primeira (igual as variaveis com o var)

```
program endoffile;

type kodo = class
end;

fts = class
end;

begin
end.
```

para a gente instanciar um objeto ou seja gerar um objeto da nossa classe basta criar uma variavel do mesmo tipo dela e depois atribuir o metodo create da nossa class para a variavel criada (para usar qualquer metodo ou atributo do objeto basta colocar o nome do objeto seguido de ponto e o metodo ou atributo que sera usado)

```
program endoffile;

type kodo = class
end;

var fts : kodo;
begin
    fts := kodo.create();
end.
```

para liberar da memoria podemos usar o metodo free

```
program endoffile;

type kodo = class
end;

var fts: kodo;
begin
    fts := kodo.create();
```

```
    fts.free();
end.
```

podemos instanciar quantos objetos a gente quiser da mesma class, embora cada objeto instaciado não afetara o outro sendo eles iguais com os mesmos atributos e metodos porem não é o mesmo objeto

```
program endoffile;

type kodo = class
end;

var fts, hack: kodo;
begin
    fts := kodo.create();
    hack := kodo.create();
    fts.free();
    hack.free();
end.
```

para criar um atributo em uma class basta declarar ele seguido do tipo (quando não especificamos o tipo de acesso o compilador pascal vai tratar ele como tipo publico, não recomendo declarar um objeto sem tipo de acesso isso pode gerar incompatibilidade de um compilador para outro)

```
program endoffile;

type kodo = class
    x: integer;
end;

begin
end.
```

uma vez que o atributo foi criado dentro da class podemos usar ele no objeto

```
program endoffile;

type kodo = class
    x: integer;
end;

var fts: kodo;
begin
    fts := kodo.create();
    fts.x := 315;
    write(fts.x);
    fts.free();
end.
```

os metodos que seria os procedimentos ou funçoes devemos declarar o prototipo completo deles dentro da class e criar eles fora dela, porem no nome temos que especificar a class que esta os prototipos

```
program endoffile;

type kodo = class
    function quad(x: integer): integer;
end;
```

```
function kodo.quad(x: integer): integer;
begin
   quad := x * x;
end;

var fts: kodo;
     resu: integer;
begin
    fts := kodo.create();
    resu := fts.quad(5);
    write(resu);
    fts.free();
end.
```

uma forma interessante para o exemplo anterior seria criar um atributo dentro do objeto ao inves de retornar um valor da função podemos atribuir ao atributo, e depois ler o atributo

```
program endoffile;

type kodo = class
    procedure quad(x: integer);
    resu: integer;
end;

procedure kodo.quad(x: integer);
begin
   resu := x * x;
end;

var fts: kodo;
begin
    fts := kodo.create();
    fts.quad(5);
    write(fts.resu);
    fts.free();
end.
```

outra estrutura que seria igual a estrutura class é a estrutura object que não precisamos atribuir ela para uma variavel ou usar o metodo create para instaciar ela, ela é instanciada automaticamente assim que criarmos uma variavel do tipo dela, o uso dessa estrutura é o mesmo que a estrutura class mudado apenas a maneira que é instanciada

```
program endoffile;

type kodo = object
    x: integer;
end;

var fts: kodo;
begin
    fts.x := 315;
    write(fts.resu);
end.
```

outro tipo de estrutura é o record, a diferença entre a class ou object e o record que na estrutura class ou object podemos definir os tipos de acesso já no record todos são tipos publicos (a estrutura class e record da linguagem pascal é bem parecido com a diferença entre as estruturas class e struct da linguagem c++), para criar uma estrutura record é parecido com a estrutura class só mudando a palavra class para record

```
program endoffile;

type kodo = record
end;

begin
end.
```

as estruturas records são um pouco diferentes das class, nas estruturas records não aceita funçoes e procedimentos apenas variaveis, e não é possível alocar dinamicamente igual a estrutura class

```
program endoffile;

type kodo = record
    x: integer;
    y: integer;
    r: integer;
end;

var fts: kodo;
begin
    fts.x := 315;
    fts.y := 100;
    fts.r := fts.x + fts.y;
    write(fts.r);
end.
```

uma forma de acessar os atributos e metodos de um determinado objeto de forma facil é usar a estrutura with que permite acessar ele apenas pelo nome do atributo ou metodo sem precisar colocar o nome do objeto antes, para usar a estrutura with basta colocar o nome do objeto que vamos acessar seguido da palavra do, por fim um escopo onde podemos colocar apenas o nome do atributo ou metodo sem precisar do nome do objeto

```
program endoffile;

type kodo = class
    procedure quad(x: integer);
    resu: integer;
end;

procedure kodo.quad(x: integer);
begin
   resu := x * x;
end;

var fts: kodo;
begin
    fts := kodo.create();
    with fts do
    begin
        quad(5);
        write(resu);
        free();
    end;
end.
```

7.2 – Orientação a objeto: Encapsulamento e Property

encapsulamento é o ato de você restringir acesso a determinado metodo ou atributo, ou apenas conceder acesso indiretamente naquela parte, na linguagem pascal podemos definir o tipo de acesso dentro das classes, esses tipos de acessos pode ser publico (public), protegido (protected) ou privado (private), o tipo de acesso publico permite a utilização do metodo ou atributo em qualquer parte que ele for instanciado ou herdado, para usar o tipo publico colocamos a palavra public antes dos metodos e atributos

```
program endoffile;

type kodo = class
    public x: integer;
end;

begin
end.
```

o tipo de acesso privado permite a utilização apenas dentro da própria class, assim esse metodo ou atributo não sera herdado ou poderar ser chamado quando instanciado, para usar o tipo de acesso privado basta colocar a palavra private antes dos metodos ou atributos

```
program endoffile;

type kodo = class
    private x: integer;
end;

begin
end.
```

o tipo de acesso protegido apenas pode ser acessado dentro da própria classe e das classes que herdar ela, para usar o tipo de acesso protegido basta colocar a palavra protected antes

```
program endoffile;

type kodo = class
    protected x: integer;
end;

begin
end.
```

em uma class podemos usar mais de um tipo para metodos ou atributos diferentes

```
program endoffile;

type kodo = class
    public x: integer;
    private y: integer;
end;

begin
end.
```

quando existir metodos e atributos com tipos de acesso iguais não precisamos especificar o tipo para todos eles basta colocar um embaixo do outro para os tipos iguais, então o compilador vai definir o tipo de acesso do anterior

```
program endoffile;
```

```
type kodo = class
    public x: integer;
           y: integer;
    private w: integer
end;

begin
end.
```

uma das utilidades do encapsulamento é definir acesso indireto para determinado atributo ou metodo, um exemplo basico seria o famoso metodo get e set para escrever em um atributo privado usando metodos publicos

```
program endoffile;

type kodo = class
    private idade: integer;
    public procedure setidade(entrada: integer);
           function getidade: integer;
end;

procedure kodo.setidade(entrada: integer);
begin
    idade := entrada;
end;

function kodo.getidade: integer;
begin
    getidade := idade;
end;

var fts: kodo;
begin
    fts := kodo.create();
    fts.setidade(21);
    write(fts.getidade());
    fts.free();
end.
```

uma forma de facilitar ate agilizar o set e get seria usar os property que são atalhos para escrita e leitura, para usar um property temos que declarar ele dentro da class, a declaração dele é o nome seguido do tipo da onde ele vai ler ou escrever depois o tipo se vai ser leitura ou escrita e por fim o nome do atributo ou metodo que ele vai ler

```
program endoffile;

type kodo = class
    private idade: integer;
    property setidade: integer write idade;
    property getidade: integer read idade;
end;

var fts: kodo;
begin
    fts := kodo.create();
    fts.setidade := 21;
    write(fts.getidade);
    fts.free();
end.
```

7.3 – Orientação a objeto: Herança

podemos aproveitar os atributos e metodos de determinadas classe em outras classe herdando ela, a herança só é possível para os atributos e metodos publicos ou protegidos (public ou protected), os atributos e metodos privados (private) não são enxergado pela classe que herdo, é comum chamar a class herdada de classe pai e a classe que herdo de classe filho quando se referenciar a elas, para herda uma classe basta declarar a classe que deseja herdar junto com a palavra class quando criar a estrutura da class filho

```
program endoffile;

type kodo = class
    public idade: integer;
end;

filho = class(kodo)
end;

var fts: filho;
begin
    fts := filho.create();
    fts.idade := 21;
    write(fts.idade);
    fts.free();
end.
```

se determinada classe filho for herdada então a class filho dela tera os atributos e metodos da classe pai e da classe avo

```
program endoffile;

type kodo = class
    public idade: integer;
end;

filho = class(kodo)
end;

neto = class(filho)
end;

var fts: neto;
begin
    fts := neto.create();
    fts.idade := 21;
    write(fts.idade);
    fts.free();
end.
```

todas as classes filhos pode ter seus próprios atributos e metodos melhorando os atributos e metodos herdado da classe pai, ou podemos criar classe pai apenas para servir como base para futuras estruturas filhos

```
program endoffile;

type kodo = class
    protected x: integer;
    protected y: integer;
end;
```

```
fts = class(kodo)
    public procedure setxy(entradax: integer; entraday: integer);
            procedure somar;
            resultado: integer;
end;

procedure fts.setxy(entradax: integer; entraday: integer);
begin
    x := entradax;
    y := entraday;
end;

procedure fts.somar;
begin
     resultado := x + y;
end;

var f: fts;
begin
    f := fts.create();
    f.setxy(300,15);
    f.somar();
    write(f.resultado);
    f.free();
end.
```

tambem é possível usar herança nas estruturas do tipo object

```
program endoffile;

type kodo = object
    public idade: integer;
end;

filho = object(kodo)
end;

var fts: filho;
begin
    fts.idade := 21;
    write(fts.idade);
end.
```

tambem podemos atribuir dois objetos do mesmo tipo, esse objeto que recebe o outro vai servir com um ponteiro para outro objeto

```
program endoffile;

type kodo = class
    public idade: integer;
end;

var obj1, obj2: kodo;
begin
    obj1 := kodo.create();
    obj2 := obj1;
    obj2.idade := 22;
    write(obj1.idade);
```

```
end.
```

podemos atribuir um objeto de uma classe pai para uma classe filho para acessar apenas os atributos e metodos da classe pai dentro da class filho

```
program endoffile;

type kodo = class
    public idade: integer;
end;

type kodofilho = class(kodo)
end;

var objfilho: kodofilho;
     objpai : kodo;
begin
    objfilho := kodofilho.create();
    objpai := objfilho;
    objpai.idade := 22;
    write(objfilho.idade);
end.
```

7.4 – Orientação a objeto: Construtores

em uma classe não podemos iniciar um valor junto com ela apenas atribuindo o valor no atributo ou metodo, para a gente iniciar um valor temos que criar um construtor para essa classe, um construtor como o nome já diz sera uma forma de passar um valor para determinado atributo ou metodo de todos objetos gerados dessa classe quando eles forem instanciados ou chamar um metodo automaticamente quando determinado objeto for instanciado, para criar um construtor basta criar um metodo construtctor com o nome create dentro da nossa classe que é parecido com uma procedure

```
program endoffile;

type kodo = class
    constructor create;
end;

constructor kodo.create;
begin
end;

begin
end.
```

dentro do nosso metodo construtor podemos atribuir os valores que queremos que se inicialize quando instanciarmos determinado objeto

```
program endoffile;

type kodo = class
    public x: integer;
    constructor create;
end;

constructor kodo.create;
begin
    x := 315;
```

```
end;

var fts: kodo;
begin
   fts := kodo.create();
   write(fts.x);
   fts.free();
end.
```

podemos usar um construtor para chamar qualquer metodo especifico ou função

```
program endoffile;

type kodo = class
    constructor create;
end;

constructor kodo.create;
begin
    write('eu fui instanciado');
end;

var fts: kodo;
begin
   fts := kodo.create();
   fts.free();
end.
```

outra vantagem dos construtores é que podemos passar valores para ser atribuido assim que for instanciado o objeto, para fazer isso basta criar argumentos para o construtor e passar eles quando instanciarmos o nosso objeto

```
program endoffile;

type kodo = class
    public r: integer;
    constructor create(x: integer; y: integer);
end;

constructor kodo.create(x: integer; y: integer);
begin
    r:= x + y;
end;

var fts: kodo;
begin
   fts := kodo.create(300,15);
   write(fts.r);
   fts.free();
end.
```

podemos criar mais de um construtor ou um construtor com qualquer nome, tambem podemos escolher qual construtor usar para instanciar o objeto

```
program endoffile;

type kodo = class
    public r: integer;
    constructor create_somar(x: integer; y: integer);
```

```
    constructor create_subtrair(x: integer; y: integer);
end;

constructor kodo.create_somar(x: integer; y: integer);
begin
    r:= x + y;
end;

constructor kodo.create_subtrair(x: integer; y: integer);
begin
    r:= x - y;
end;

var fts: kodo;
begin
   fts := kodo.create_subtrair(300,15);
   write(fts.r);
   fts.free();
end.
```

alem dos construtores existe os destrutores que é ativado quando o objeto for liberado ou destruido, para criar um destrutor basta criar um metodo destructor e o funcionamento dele é parecido com os construtores

```
program endoffile;

type kodo = class
    public
        constructor create;
        destructor free;
end;

constructor kodo.create;
begin
    writeln('fui instanciado ^^');
end;

destructor kodo.free;
begin
    writeln('fui destruido T.T');
end;

var fts: kodo;
begin
   fts := kodo.create();
   fts.free();
end.
```

tambem podemos criar destruidores com outros nomes

```
program endoffile;

type kodo = class
    public
        constructor create;
        destructor delete;
end;

constructor kodo.create;
```

```
begin
    writeln('fui instanciado ^^');
end;

destructor kodo.delete;
begin
    writeln('fui destruido T.T');
end;

var fts: kodo;
begin
   fts := kodo.create();
   fts.delete();
end.
```

7.5 – Orientação a objeto: Polimorfismo

polimorfismo é quando determinado objeto assume mais de uma forma, ou assume varias formas diferente a cada novo objeto, um exemplo seria os animais que emiti sons porem nem todos os animais emite o mesmo som se o metodo para emitir o som for o mesmo porem gerando som diferente a cada objeto isso é um metodo polimorfico, um dos tipos de polimorfismo é abstração que diz que um metodo só sera criando em uma class herdeira porem seu prototipo vai estar na class pai, como prototipo vai esta na class pai é um metodo polimorfico porque é class filho que vai criar seus procedimentos porem ela já foi declarada na class antecessor, com isso qualquer classe filho poderia gerar procedimentos diferentes, para criar um metodo abstrato é necessario declarar o prototipo dele na classe pai com as palavras virtual e abstract, depois declarar o prototipo na class filho com a palavra override indicando que ela pertence a class pai

```
program endoffile;

type kodo = class
    public procedure mensagem; virtual; abstract;
end;

fts = class(kodo)
   public procedure mensagem; override;
end;

procedure fts.mensagem;
begin
  write('end of file');
end;

var obj: fts;
begin
   obj := fts.create();
   obj.mensagem();
   obj.free();
end.
```

outro exemplo com mais de um objeto gerando mensagem diferente a cada herança

```
program endoffile;

type kodo = class
    public procedure mensagem; virtual; abstract;
end;

fts = class(kodo)
```

```
   public procedure mensagem; override;
end;

hack = class(kodo)
   public procedure mensagem; override;
end;

procedure fts.mensagem;
begin
  writeln('end of file');
end;

procedure hack.mensagem;
begin
  writeln('kodo no kami');
end;

var obj: fts;
     obj2: hack;
begin
   obj := fts.create();
   obj2 := hack.create();
   obj.mensagem();
   obj2.mensagem();
   obj.free();
   obj2.free();
end.
```

8.0 – Fim

bom galera a linguagem pascal não se limita só o conteudo desse ebook, existe milhares funçoes bibliotecas e componentes para diversas IDE como o delphi ou lazarus, talvez futuramente eu modifique esse ebook melhorando ele com novos conteudos, sugestões e duvidas basta entrar em contato nos fóruns ou redes sociais ^^

http://www.facebook.com/hacker.fts315
http://endoffile.umforum.net
http://fts315.xp3.biz
http://kodonokami.blogspot.com

outros sites e forums que eu recomendo são:

http://hc0der.blogspot.com.br
http://injectionsec.com/
http://brutalsecurity.com.br
http://madeinbrazil.umforum.net