

Linguagem de Programação I e II
Linguagem de Programação I e II
Pascal Object

CERTIFICADO DE REGISTRO

Nº REGISTRO: 179.755 LIVRO : 304 FOLHA : 407

PROCEDIMENTO E FUNÇÃO	3
ESCOPO DE VARIÁVEIS.....	3
PROCEDIMENTOS	4
FUNÇÃO	9
FUNÇÕES RECURSIVAS	10
<i>Exercícios:</i>	11
INTRODUÇÃO.....	17
CONCEITOS BÁSICOS DA POO	18
CLASSE	18
OBJETO	18
MENSAGEM	18
MÉTODO.....	18
HERANÇA	18
ENCAPSULAMENTO	21
COMPONENTES REUTILIZÁVEIS (BIBLIOTECAS UNIT)	21
COMO CRIAR UMA UNIDADE.....	21
INTERFACE	21
IMPLEMENTATION	21
COMO UTILIZAR	21
AS CLASSES.....	22
<u>DEFINIÇÕES DA POO DO TURBO PASCAL</u>	23
<i>Exemplo</i>	39
INICIALIZANDO OBJETOS	43
<i>Exercício</i>	46
OBJETOS E MÉTODOS DINÂMICOS	47
<i>Exercícios</i>	49
VINCULAÇÃO TARDIA E POLIMORFISMO	50
OBJETOS DINÂMICOS.....	53
COMANDOS DE MANIPULAÇÃO DE ARQUIVOS	59
UTILIZANDO ARQUIVOS COM OBJETOS	61
REFERÊNCIAS BIBLIOGRÁFICAS.....	64

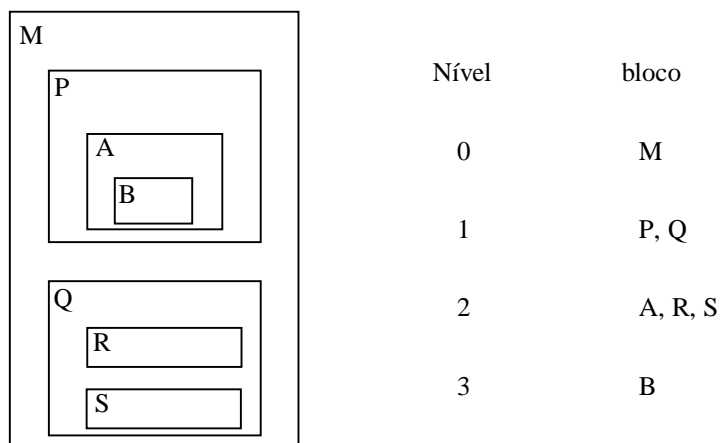
PROCEDIMENTO E FUNÇÃO

Escopo de Variáveis

Diz-se que um bloco é externo a outro, quando o segundo faz parte do primeiro.

Neste sentido, uma variável declarada em um bloco é *global* para todos os blocos internos e *local* para o próprio bloco.

Podemos ter diversos bloco aninhados, conforme mostra a figura a seguir:



P e Q internos a M

M é externos a P e Q

A é interno a P

P é externo a A

B é interno a A

A é externo a B

R e S são internos a Q

Q é externo a R e S

Conforme foi mostrado, uma variável declarada dentro de um bloco só é conhecida dentro deste bloco.

Se uma variável *A* declarada em um bloco já foi declarada com mesmo nome num bloco mais externo, a variável ativa no bloco é aquela declarada localmente. A variável *A* deixa de ser global para aquele bloco.

Procedimentos

Um procedimento é um bloco precedido de um cabeçalho. Com isto será possível fazer referência ao bloco de qualquer ponto do algoritmo. A sintaxe da declaração será:

Procedimento <nome do procedimento>;

início

```
|           <declarações>  
|  
|           C1;  
|  
|           C2;  
|  
|           C3;  
|           .  
|           .  
|           .  
|           Cn;
```

fim; {<nome do procedimento>}

Exemplo:

Procedimento TROCA;

início

```
|           inteiro: AUX;  
|  
|           AUX ← X;  
|  
|           X ← Y;  
|  
|           Y ← AUX;
```

fim; {TROCA}

A declaração de um procedimento deve vir sempre no início do bloco em que estiver sendo declarado (antes de qualquer comando executável). Um procedimento só é executado sob a *chamada*. A chamada no procedimento é feita por um comando que se resume no nome do procedimento. O exemplo a seguir esclarece a utilização do procedimento TROCA declarado anteriormente. O que será impresso no algoritmo abaixo?

início

inteiro: X, Y, A, B, C, D;

Procedimento TROCA;

início

inteiro: AUX, X;

AUX ← X;

X ← Y;

Y ← AUX;

Serão impressos:

5 3

fim; {TROCA} 3 5 e

A ← 5; 4 9

B ← 3;

escreva(A, B);

X ← A;

Y ← B;

TROCA;

A ← X;

B ← Y;

escreva (A, B);

C ← 4;

D ← 9;

escreva (C, D);

X ← C;

Y ← D;

TROCA;

C ← X;

Y ← D;

escreva (C, D);

fim.

No exemplo anterior o procedimento TROCA foi utilizado duas vezes com o objetivo de trocar o valor das variáveis *A* e *B* e depois *C* e *D*. O mesmo resultado poderia ser obtido de uma forma mais compacta introduzindo-se parâmetros no procedimento.

A sintaxe mais geral para procedimentos será:

Procedimento <nome do procedimento> (<lista de parâmetros>);

<especificação dos parâmetros>

início

<declarações de variáveis locais>

C₁;

C₂;

C₃;

.

.

.

C_n;

fim; {nome do procedimento}

A <especificação dos parâmetros> consiste na declaração dos tipos das variáveis que compõem a lista de parâmetros. No exemplo do procedimento TROCA, X e Y deixariam de ser variáveis globais e passariam a fazer parte da lista de parâmetros:

Procedimento TROCA(X, Y);

procedure TROCA (X, Y: integer);

inteiro: X, Y;

var AUX : integer;

início

begin

inteiro: AUX;

AUX := X;

AUX ← X;

X := Y;

X ← Y;

Y := AUX;

Y ← AUX;

end;

fim; {TROCA}

Modifique, agora, o exemplo anterior adaptando-o ao novo procedimento.

Tudo o que foi falado vale para o Pascal, incluindo nesta definição o conceito de passagem por valor e por referência, que abordaremos mais adiante.

Sintaxe em Pascal:

```
procedure NOME (<lista de parâmetros>);
```

```
    <lista de variáveis locais>
```

```
begin
```

```
end;
```

Função

Quando se necessitar atribuir o resultado da chamada de um procedimento a uma variável, utilizar este resultado numa expressão aritmética ou imprimir este resultado, é necessário que exista um parâmetro de retorno na chamada do procedimento e este parâmetro é que será utilizado.

Por exemplo:

```
ABS(-3,Y);
```

```
X ← Y * 2;
```

Seria mais conveniente se pudéssemos escrever:

```
X ← ABS(-3) * 2;
```

Isto é possível utilizando um tipo especial de procedimento, denominado função cuja sintaxe é a seguinte:

Função <nome da função> (<lista de parâmetros formais>): <tipo básico>;

<especificação dos parâmetros>;

início

<declaração de variáveis locais>;

C₁;

C₂;

C₃;

.

.

.

<nome da função> ← <expressão>;

C_n;

fim;{<nome da função>}

O procedimento ABS(X, Y) visto anteriormente poderia ser transformado na função ABS(X), da seguinte maneira:

Função ABS(X): real;

real: X;

início

se X >= 0

então ABS ← X;

senão ABS ← -X;

fim se;

fim; {ABS}

function ABS(X: real):real;

begin

 if X >= 0 then ABS := X

 else ABS := X * -1;

end;

A chamada da função é, portanto, uma pseudovariável, isto é, depois de executada a chamada, o valor calculado é retornado no nome da função, que passa a ser uma variável da expressão.

Funções Recursivas

Existem casos em que um procedimento ou função chama a si próprio. Diz-se então que o procedimento ou função é recursivo. Por exemplo, o fatorial de um número n pode ser definido recursivamente, ou seja:

$$n! = \begin{cases} n(n-1)! & \text{se } n \geq 1 \\ 1 & \text{se } n = 0 \end{cases}$$

Escreva aqui a função FATORIAL recursiva em algoritmo.

Exercícios:

- 1) Escrever um procedimento para imprimir o cabeçalho:

UNESA – SISTEMA DE PROGRAMAÇÃO

XX/XX/XX

PÁG. 9999

fornecer a DATA e o número da página como parâmetros.

- 2) Escrever uma função que receba dois números inteiros, positivos, e determine o produto dos mesmos, utilizando o seguinte método de multiplicação:

- dividir sucessivamente o primeiro número por 2 até que se obtenha 1 como quociente;
- paralelamente, dobrar, sucessivamente, o segundo número;
- somar os números da segunda coluna que tenham como correspondente na primeira coluna um número ímpar. O total é o produto procurado.

Exemplo:

```

                                9 x 6
          9      6      →   6
          4      12
          2      24
          1      48      → + 48
                                -----
                                54
```

A seguir escrever um algoritmo que leia 10 pares de números e calcule os respectivos produtos, usando a função acima.

- 3) Escrever uma função que, ao analisar se um número recebido via parâmetro está no intervalo $5 < \text{NÚMERO} < 20$, retorne o valor *verdadeiro* e *falso* em caso contrário.

- 4) Preparar um procedimento VOGAL para aceitar como parâmetro uma cadeia de caracteres arbitrária e retornar uma cadeia contendo somente os caracteres alfabéticos da cadeia original. Todos os brancos, sinais de pontuação, números e caracteres especiais devem ser removidos.
- 5) Números palíndromos são aqueles que escritos da direita para esquerda, têm o mesmo valor. Exemplo: 545; 97379; etc.

Escrever uma função que, recebendo como parâmetro um número inteiro, retorne este número escrito ao contrário.

A seguir, escrever um algoritmo que determine e imprima, usando a função acima, todos os números palíndromos entre 1 e 1000.

- 6) Faça um programa para imprimir uma tabela de capital acumulado, dado o valor do capital inicial, da taxa de juros e do número de períodos. A fórmula para o cálculo do capital acumulado é a seguinte:

$$M = C \times (1 + i/100)^n$$

Em que C = capital inicial

i = taxa de juros

n = número de períodos

M = capital acumulado

Por exemplo, se o capital inicial é R\$100,00, a taxa é de 22% ao mês e o número de meses da aplicação é igual a 6, teremos que calcular os valores:

$$M_1 = 100 \times (1 + 22/100)^1$$

$$M_2 = 100 \times (1 + 22/100)^2$$

•

•

•

$$M_6 = 100 \times (1 + 22/100)^6$$

- 7) Faça um programa que contenha um procedimento (procedure) para verificar se um ano é bissexto ou não. Utilize a seguinte regra:

Um ano é bissexto se é divisível por 4, mas não por 100, ou então se é divisível por 400.

Exemplo: 1988 é bissexto porque 1988 é divisível por 4 mas não por 100

2000 é bissexto porque é divisível por 400.

1900 não bissexto porque é divisível por 4 e por 100, mas não é divisível por 400

- 8) Em relação à tabela abaixo, deseja-se calcular a somatória dos valores de x . Faça um programa para armazenar essa tabela numa matriz e determinar a soma desejada, servindo-se de uma função para o cálculo da soma. O programa deve também imprimir a tabela e a soma.

	1	2	3	4	5	6	7	8	9	10	A
X	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.1	15,6

Observe que: $A = \sum x = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} = 15.6$

- 9) Em relação a uma tabela dada, deseja-se calcular a somatória dos quadrados dos valores dados. Faça um programa que contenha uma função para executar essa tarefa. A função deve ter dois parâmetros: a matriz e o número de elementos da matriz como em

Function SomaXquadrado(X: matriz; N: integer): real;

	1	2	3	4	5	6	7	8	9	10
X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.1

Deseja-se obter o valor de B:

	1	2	3	4	5	6	7	8	9	10
X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.1

Programação Orientada à Objetos com Pascal

x^2	0.01	0.04	0.09	0.16	0.25	0.36	0.49	0.64	0.81	1.21
-------	------	------	------	------	------	------	------	------	------	------

$$B = \sum x^2 = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 + x_7^2 + x_8^2 + x_9^2 + x_{10}^2 = 4.06$$

Acumule a soma de x^2 em B. Observe que os valores de x^2 não precisam ser armazenados

- 10) Faça um programa que contenha um subprograma para imprimir o signo do zodíaco correspondente a uma data (dia/mês) qualquer.

A tabela abaixo mostra o último dia de cada mês e o signo correspondente.

Mês	Último dia	Signo
Jan	20	CAPRICÓRNIO
Fev	19	AQUÁRIO
Mar	20	PEIXES
Abr	20	ÁRIES
Mai	20	TOURO
Jun	20	GÊMEOS
Jul	21	CÂNCER
Ago	22	LEÃO
Set	22	VIRGEM
Out	22	LIBRA
Nov	21	ESCORPIÃO
Dez	21	SAGITÁRIO
De 22/dez a 31/dez		CAPRICÓRNIO

- 11) O dia da Páscoa é o primeiro domingo depois da lua cheia que ocorre a partir do vigésimo-primeiro dia do mês de março. Se a lua cheia ocorre num domingo, então a Páscoa é no domingo seguinte. Ao cálculo do dia da Páscoa deu-se o nome de Computus. Em 1800, Carl Friedrich Gauss, um dos maiores matemáticos de todos os tempos, resumiu o cálculo do dia da Páscoa num fórmula bastante simples. O processo de Gauss para determinar em que dia se comemora a Páscoa no ano X é o seguinte:

(1) Determine duas constantes M e N, pela tabela:

ANO	1582-1699	1700-1799	1800-1899	1900-2099	2100-2229
M	22	23	23	24	25
N	2	3	4	5	6

(2) Determine $A = \text{resto da divisão de Ano por } 4$

(3) Determine $B = \text{resto da divisão de ano por } 7$

(4) Determine $C = \text{resto da divisão de Ano por } 19$

(5) Determine $D = \text{resto da divisão de } (19 \times C + M) \text{ por } 30$

(6) Determine $E = \text{resto da divisão de } (2 \times A + 4 \times B + 6 \times D + N) \text{ por } 7$

(7) O dia da Páscoa ocorre no dia $(22 + D + E)$ de março ou no dia $(D + E - 9)$ de abril.

(8) Existem duas exceções:

Se $D = 28$ ou $D = 29$ e $E = 6$ então o dia da Páscoa deve ser celebrado uma semana antes do dia calculado pela fórmula.

Faça um programa para determinar em que dia se comemora a Páscoa num ano qualquer.

Programação Orientada à Objeto

INTRODUÇÃO

A programação orientada para objeto é uma nova disciplina do Desenvolvimento de Software. Ela apresenta uma estratégia diferenciada de organizar, planejar, conceitualizar, escrever, atualizar e fazer manutenção do software.

As linguagens de alto nível representam uma conexão inteligente entre o programador e o computador. Sua evolução é paralela às técnicas e métodos do Desenvolvimento do Software, sendo inspirada pelo anseio de criar componentes de softwares robustos e reutilizáveis que reduzem os ciclos e o tempo da produção do software. Esta diminuição no tempo de produção é ainda mais relevante quando uma equipe de programadores está trabalhando em um projeto de software.

A história das linguagens de programação reflete a evolução de linguagens não-estruturadas para linguagens estruturadas e destas para as orientadas ao objeto. A linguagem Pascal desempenha um papel de destaque na promoção da programação estruturada, tanto na área profissional quanto na área acadêmica – pena que a borland tenha cancelado a distribuição deste programa – e das disciplinas de programação que a acompanham. O Pascal e outras linguagens estruturadas chamaram a atenção do programador e professores universitários por serem linguagens organizadas e didáticas, que além de economizar tempo, permitem uma melhor programação, com código robusto e reutilizável, de fácil legibilidade, manutenibilidade, atualização e acesso a bibliotecas de software.

As linguagens estruturadas são linguagens procedurais, onde um programa é particionado em um conjunto de procedimentos e funções. A seqüência de rotinas descreve como os dados são manipulados. Especificamente, o programador controla a interação entre o código e os dados. O foco principal das linguagens procedurais é a rotina, enquanto o foco secundário são os dados que estão sendo manipulados.

Neste capítulo, abordaremos uma introdução à programação orientada para objeto. Procuro ser o mais simples e didático possível, espero que o conteúdo aqui apresentado seja de grande valia para o estudante ou pesquisador.

CONCEITOS BÁSICOS DA POO

Classe

Uma **classe** detalha os campos de dados de um objeto e os métodos que atuam sobre os dados. As classes são parecidas com os tipos de dados em Pascal.

Objeto

Um **objeto** é um membro de uma classe, similar à tradicional variável em Pascal.

Mensagem

Uma **mensagem** é uma solicitação enviada a um objeto para chamar um de seus métodos.

Método

Um **método** é o procedimento ou função chamado para atuar num objeto. Um método define como uma mensagem deve ser executada.

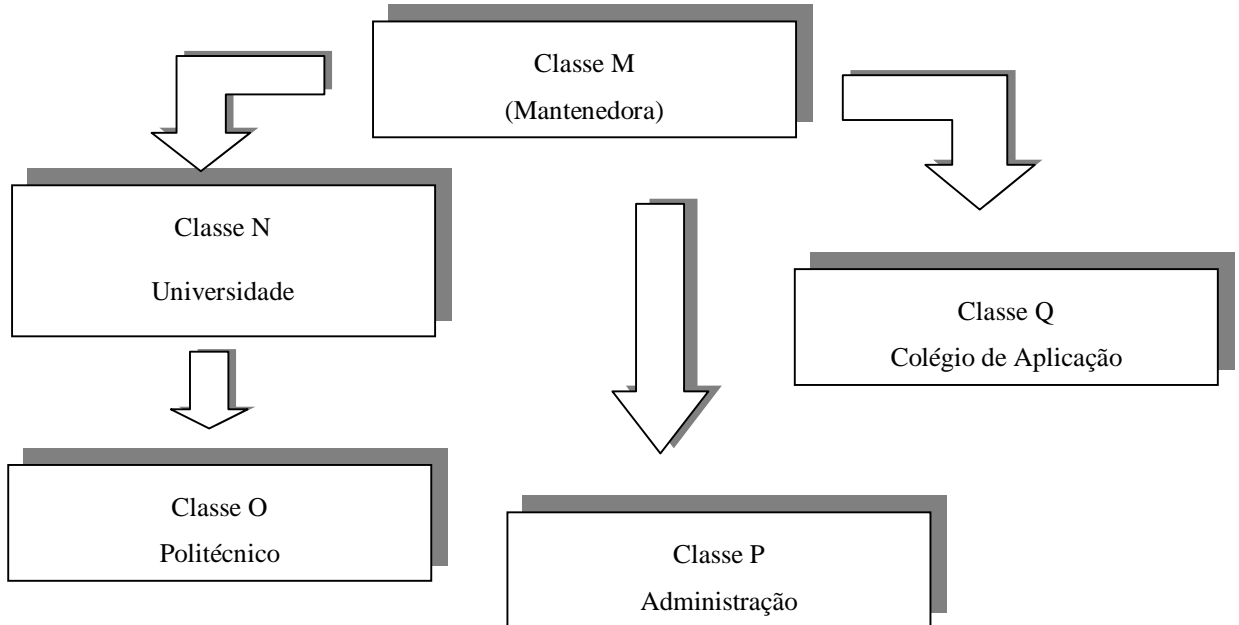
A POO (Programação Orientada a Objeto) permite uma abordagem mais simples para a utilização de dados usando objetos que respondem a mensagens. Esta abordagem demonstra uma noção de que objetos em um programa são similares a alunos em uma sala de aula – você passa um exercício ou um trabalho para um aluno ou à um grupo de alunos (o objeto, neste caso) solicitando que lhe seja apresentado uma solução para um determinado problema. Supondo-se que o trabalho seja passado ao aluno ou grupo certo, ele responderá usando sua experiência e recursos. O método pelo qual o trabalho é executado depende do objeto receptor (aluno). Os detalhes da resposta são transparentes ao professor (emissor da mensagem).

Herança

Uma outra característica da POO é a **herança**, um recurso que produz um grande efeito, que lhe permite criar subclasses. Cada subclasse recebe as características de sua classe-mãe (da mesma forma que um criança traz consigo características peculiares de seus pais). Uma subclasse acrescenta novos atributos às classes herdadas e pode também substituir qualquer atributo herdado. O Turbo Pascal, que implementa o Object Pascal, permite apenas herança simples (também conhecida como linear). Este é um esquema de herança simples onde cada subclasse tem apenas uma classe-mãe.

A figura demonstra um exemplo de classe com suas subclasses. A subclasse O é filha da subclasse N, que é filha da classe M. As subclasses P e Q são filhas da classe M. As classes N, P e Q herdam os vários atributos da classe M. Os atributos de M não precisam ser redefinidos quando se declara as subclasses N, P e Q – apenas os novos atributos e os atributos anulados precisam ser declarados. Do

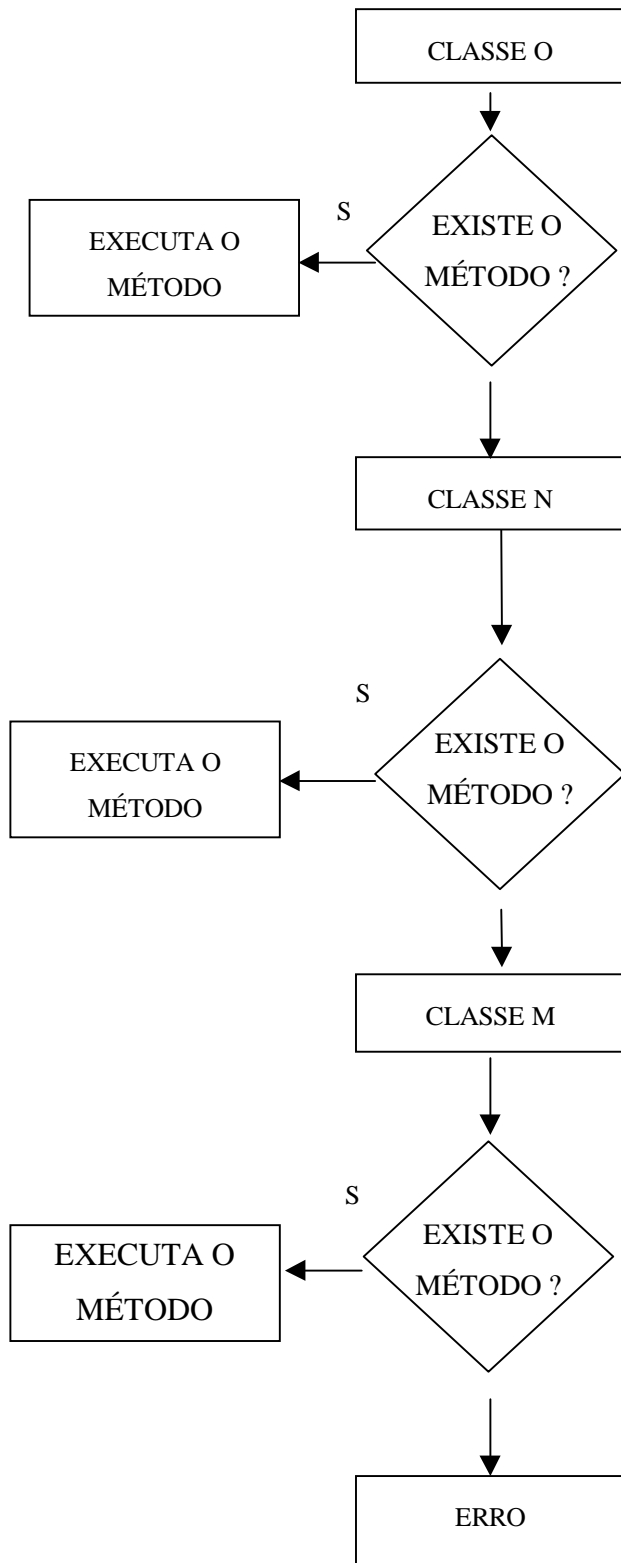
mesmo modo, a subclasse O herda os atributos da subclasse N e da Classe M, e precisa declarar apenas os novos atributos e os atributos cancelados. Isto será melhor abordado mais adiante.



Exemplo de uma Instituição de ensino e suas subclasses

A **herança** influi muito significativamente na diminuição do código do programa. Ela permite evitar codificação repetidas e diminui o tempo de desenvolvimento do software.

O gráfico representa o caminho percorrido quando da chamada de um método na Classe O.



A procura de um método que combine com a chamada nas classes hierarquicamente superior demonstra a hereditariedade.

Encapsulamento

Uma das principais características da POO é o encapsulamento, pois evita que o programador não precise acessar diretamente os campos de dados de um objeto. Para isto é necessário criar métodos que serão os responsáveis pelo tratamento de dados.

Encapsulamento na verdade é unir, numa única estrutura, campos de dados e códigos que agirão sobre eles (métodos). O grande benefício do encapsulamento é que o programador trabalha voltado única e exclusivamente ao campo de dados que deseja atingir com seu código.

COMPONENTES REUTILIZÁVEIS (BIBLIOTECAS UNIT)

Uma das ferramentas que possibilitam o encapsulamento de objetos é a criação de Unidades (UNITS). Elas permitem que o programador empacote um conjunto de códigos de procedimentos ou funções que executem uma determinada tarefa.

Com esta possibilidade é possível compartimentalizar um determinado programa e reutilizá-lo posteriormente em seus futuros projetos.

Como Criar Uma Unidade

Uma unidade é composta por duas partes distintas: IMPLEMENTATION E INTERFACE.

INTERFACE

Este bloco conterá os cabeçalhos dos procedimentos que farão parte da biblioteca (unidade) e que serão definidos na INTERFACE.

IMPLEMENTATION

É o local onde serão codificados os procedimentos ou funções que foram declarados no bloco IMPLEMENTATION.

Como Utilizar

Após ter sido criada, a sua utilização é bem simples, basta mencionar o nome da unidade no bloco USES e chamar um dos procedimentos ou funções no corpo principal do programa onde a unidade fora inserida.

Para melhorar o entendimento do leitor, demonstro um código muito importante para os programadores em Pascal. Este código permite que o programador defina uma ou várias molduras de tamanhos diferentes em seus programas.

Para facilitar, criei uma unidade onde defino o código para a impressão, na tela, de uma moldura.

Na listagem a seguir ilustra a criação da unidade MOLDURA, repararem que no bloco interface foi declarado duas linhas *uses crt* e *procedure mold(C1,L1,L2,C2: integer)*. No bloco implementation fora definido o código criado para a impressão da moldura na tela.

```
unit moldura;  
  
interface  
uses crt;  
procedure mold(C1,L1,L2,C2:integer);  
  
implementation  
procedure mold(C1,L1,L2,C2:integer);  
var X : byte;  
begin  
  gotoxy(C1,L1);write(chr(201));  
  for X := (C1+1) to C2 do  
    begin  
      gotoxy(X, L1);write(chr(205));  
      gotoxy(X,L2);write(chr(205));  
    end;  
  gotoxy(C1,L2);write(chr(200));  
  for X := (L1+1) to (L2-1) do  
    begin  
      gotoxy(C1, X);write(chr(186));  
      gotoxy(C2,X);write(chr(186));  
    end;  
  gotoxy(C2,L1);write(chr(187));  
  gotoxy(C2,L2);write(chr(188));  
end;  
end.
```

A chamada do Unit Crt se dá devido ao uso do comando *gotoxy*, que é um procedimento exclusivo desta unidade.

Para demonstrar a utilidade desta unidade apresentarei um programa mais adiante onde utilizei a unidade Moldura.

As Classes

Definir uma classe hierárquica significa estabelecer a classe principal e as subclasses que definem aperfeiçoamentos praticados na classe principal. A escolha da classe principal e das subclasses não é uma tarefa das mais simples. A regra prática a seguir é a relação *ÉUm* (*ÉUma*). Por exemplo, entre as seguintes classes: cadeira, cadeira de balanço, cadeira universitária, cadeira de diretor e cadeira de secretária, é fácil identificar a classe base cadeira. Uma cadeira de balanço **é uma** cadeira. Da mesma

forma, uma cadeira de diretor também é **uma** cadeira; portanto, cadeira de diretor é **uma** subclasse de cadeira.

Definições da POO do Turbo Pascal

O Turbo Pascal implementa de forma muito cuidadosa o Object Pascal. As extensões das linguagens orientadas ao objeto são poucas, mas muito poderosas. A sintaxe geral para declarar uma classe é mostrada aqui:

Type

```
<nome-classe> = OBJECT | OBJECT (<classe mãe>)  
  
<lista de campos de dados>  
  
<lista de cabeçalhos de funções e procedimentos - métodos>  
  
End;
```

Para se definir uma nova classe, utiliza-se a palavra-chave OBJECT. Na definição de uma subclasse, a palavra-chave OBJECT deve ser seguida do nome da classe-mãe entre parênteses. A lista de campos é projetada da mesma maneira que os registros.

Os códigos dos procedimentos e funções não são definidos juntamente com seu cabeçalho na declaração da classe e sim mais adiante de uma maneira bem peculiar.

Os conceitos básicos da POO estão ilustrados na Listagem 1. Este programa define uma classe de objetos que são conjuntos de caracteres. O Pascal aceita o tipo de dados *sets* (conjuntos) e implementa as operações de união, diferença e interseção de conjuntos. Além disso, um programador consegue criar rotinas personalizadas para manipular conjuntos como lhe for mais conveniente. Usando as técnicas de programação estruturada, todas as manipulações de conjunto são implementadas de forma dividida. O programa da Listagem 1 apresenta uma classe de conjuntos de caracteres que encapsulam um campo do tipo conjunto e os métodos que atuam sobre ele.

Listagem 1. Programa CONJOB.PAS para ilustrar os conceitos básicos da POO aplicada ao objeto de conjunto de caracteres.

```
Program Conjunto_De_Caracteres;  
Uses Crt;  
Type CarConj = Set of Char;  
Type  
ClcarConj = OBJECT  
    CDConj : CarConj;  
    CDMembro : Byte;  
    Procedure Limpa;
```

```
Procedure EscreveConj;
Procedure InsereMembro(Achar:char);
Procedure EliminaMembro (Achar: char);
Function  MembroNoConj (Achar:char):boolean;

End;

Procedure CLCarConj.limpa;
Begin
    CDConj := [];
    CDMembro := 0;

end;

Procedure CLCarConj.EscreveConj;
const Aspas = ```;
var I : byte;
begin
    write('[');
    for I := 0 to 255 do
        if chr(I) in CDConj then
            write(Aspas, chr(I), aspas,',');
        write(#8); {retrocesso}
    writeln(')');
end;

Procedure CLCarConj.InsereMembro (achar:char);
begin
    if not(achar in CDConj) then begin
        CDConj := CDConj +[Achar];
        inc(CDMembro)
    end;
end;

procedure CLCarConj.EliminaMembro (Achar: char);
begin
    if Achar in CDConj then begin
        CDConj := CDConj - [Achar];
        Dec(CDMembro)
    end;
end;

function CLCarConj.MembroNoConj(Achar: char): Boolean;
begin
    MembroNoConj := Achar in CDConj;
end;

procedure PressioneUmaTecla;
```

```
var Sai : char;
begin
    writeln;
    write('Pressione qualquer tecla para continuar...');
    sai := readkey;
    writeln;
    wtriteln;
end;
var CONJUNTO : CLCarConj;
begin
    Clrscr;
    writeln('Testando objeto de conjunto genérico');
    CONJUNTO.LIMPA;
    CONJUNTO.InsereMembro('X');
    CONJUNTO.InsereMembro('y');
    CONJUNTO.InsereMembro('z');
    CONJUNTO.InsereMembro('+');
    writeln('O conjunto é');
    CONJUNTO.EscreveConj;writeln;
    CONJUNTO.EliminaMembro('X');
    CONJUNTO.EliminaMembro('y');
    CONJUNTO.EliminaMembro('Z');
    Writeln('Depois de deletar X, y e Z, o conjunto é ');
    CONJUNTO.EscreveConj;writeln;
    writeln('O      caractere      P      está      no
conjunto?',Conjunto.MembroNoConj('P'));

    writeln('O      caractere      z      está      no      conjunto?      ',
Conjunto.MembroNoConj('z'));
    PressioneUmaTecla;
end.
```

O programa define a classe CLCarConj da seguinte maneira:

```
CLCarConj = OBJECT
    CDConj : CarConj;
    CDMembro : Byte;
    Procedure Limpa;
    Procedure EscreveConj;
    Procedure InsereMembro(Achar:char);
    Procedure EliminaMembro (Achar: char);
    Function MembroNoConj (Achar:char):boolean;
End;
```

Existem dois campos de dados, a saber, CDConj e CDMembro. O campo CDConj toma nota dos caracteres que são membros do objeto de conjunto. O campo CDMembro armazena o tamanho do conjunto (número de membros). A classe CLCarConj define cinco métodos para alterar o estado dos objetos que são instâncias da classe CLCarConj. O método **Limpa** é usado para inicializar e reinicializar o objeto. O método **EscreveConj** exibe os membros do conjunto. O método **InsereMembro** insere um membro não-existente e atualiza o tamanho do conjunto. O método **EliminaMembro** remove um membro existente e diminui o tamanho do conjunto e retorna **TRUE** se o caracter Achar for membro, ou **FALSE** se este não for o caso.

Ao analisar o código relativo aos métodos, observe o seguinte:

- A. Que o nome do método é precedido pelo nome da classe e pelo ponto que é o operador de acesso. Esta é uma exigência para informar ao compilador sobre a propriedade dos métodos.
- B. O programa cria o objeto **CONJUNTO** na seção de declaração VAR, exatamente como qualquer outra variável.
- C. As mensagens são enviadas ao objeto **CONJUNTO** usando o formato <objeto>.<mensagem>. Por exemplo, a instrução CONJUNTO.Limpa é interpretada como “enviar a mensagem Limpa ao objeto CONJUNTO”. O objeto Conjunto responde executando o método CLCarConj.Limpa e limpando o conjunto de caracteres. Da mesma forma, a instrução CLCarConj.InsereMembro('A'); é interpretada como “enviar a mensagem InsereMembro('A') ao objeto”, à qual o objeto CONJUNTO responde executando o método CLCarConj.InsereMembro e inserindo o caractere 'A' no conjunto.

O resultado do programa anterior é o seguinte:

Testando objeto de conjunto genérico

O conjunto é

```
['X', 'Y', 'Z', '+']
```

Depois de deletar X, Y e Z, o conjunto é

```
['+', 'y', 'z']
```

O caractere P está no conjunto ? FALSE

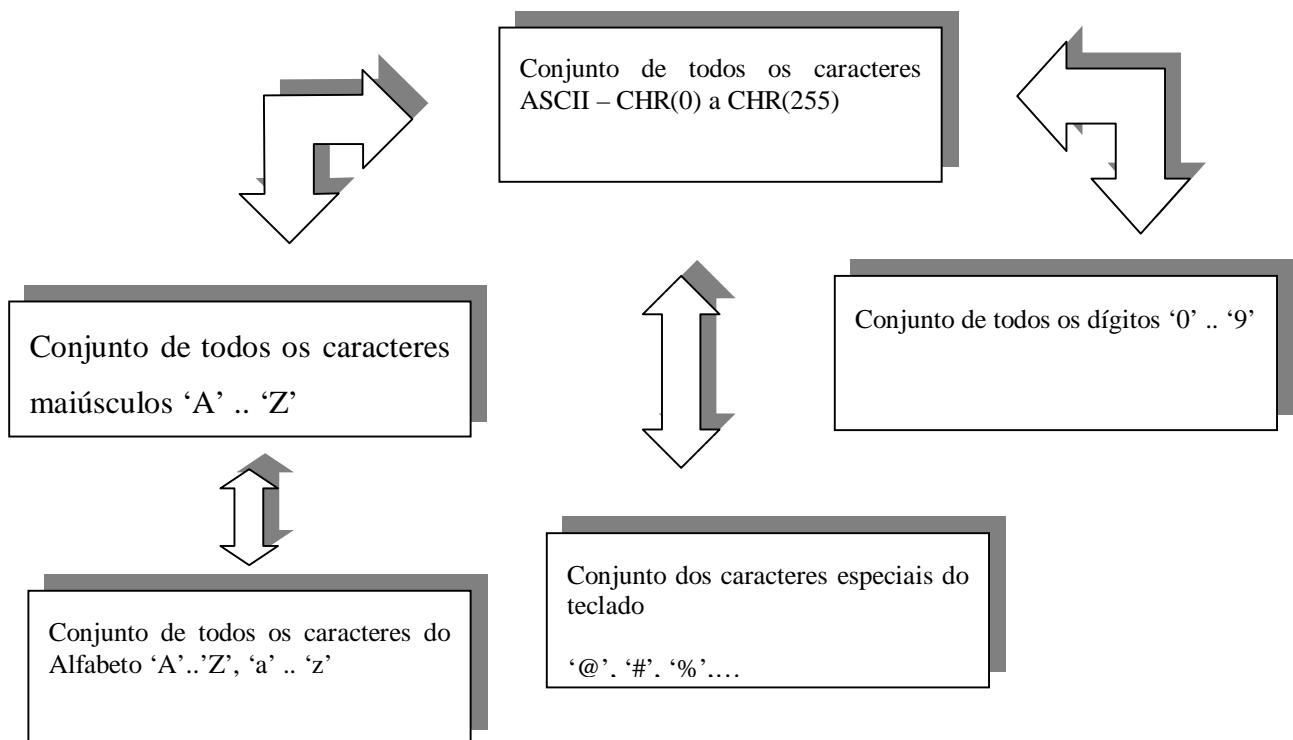
O caractere z está no conjunto ? TRUE

Pressione qualquer tecla para continuar...

O programa da listagem 1 lidou com uma única classe e um único objeto. A força e a versatilidade da POO vêm com a criação de uma hierarquia de classes. Para ilustrar este aspecto da POO, apresento um programa que insere subclasses de objetos representando maiores aprimoramentos da classe base CLCarConj:

- A. Uma subclasse de letras maiúsculas;
- B. Uma subclasse de dígitos;
- C. Uma subclasse de letras maiúsculas e minúsculas;
- D. Uma subclasse dos caracteres especiais do teclado.

A figura abaixo ilustra a hierarquia das várias classes de conjuntos de caracteres.



Vamos analisar como cada subclasse é criada. Lembre-se de que a classe base CLCarConj é definida da seguinte maneira:

```
CLCarConj = OBJECT
    CDConj : CarConj;
    CDMembro : Byte;
    Procedure Limpa;
    Procedure EscreveConj;
    Procedure InsereMembro(Achar:char);
    Procedure EliminaMembro (Achar: char);
    Function MembroNoConj (Achar:char):boolean;
End;
```

A primeira subclasse de CLCarConj é o conjunto de letras maiúsculas e podemos chamá-lo SubLeMai. A diferença entre a subclasse SubLeMai e a sua classe base CLCarConj, se inicia nas finalidades de seus campos de dados, no campo CDConj os elementos que serão armazenados, apesar de serem os mesmos, não tem os mesmos conceitos, pois são somente as letras maiúsculas que serão armazenadas. Já o campo CDMembro tem função idêntica a de sua classe base, logo é perfeitamente possível a herança dos campos de dados. Verificamos, também que não é necessário a criação de novos campos de dados.

Os métodos herdados satisfazem as suas necessidades, exceto o método CLCarConj.InsereMembro e uma nova versão precisa ser definida, isto cancelará a antiga.

Com base nesta análise, a definição da classe SubLeMai é a seguinte:

```
SubLeMai = object (CLCarConj)
    Procedure InsereMembro (Achar : Char); {cancelar}
End;
```

Observe que Object(CLCarConj) é usada para informar ao compilador que a classe SubLeMai é uma subclasse de CLCarConj e não uma nova classe base. As instâncias de SubLeMai têm acesso direto a todos os campos de dados e métodos da classe base. A sobrescrição do método InsereMembro é a nova versão do método herdado.

A segunda subclasse de CLCarConj é o conjunto de dígitos 0 a 9; chame-a Digitos. Do mesmo modo que em SubLetMai, os campos herdados, bem como, os seus métodos, satisfazem à subclasse SubDigito, menos CLCarConj.InsereMembro que deve ser redefinida. A declaração da subclasse SubDigito é a seguinte:

```
SubDigito = object (CLCarConj)
```

```
Procedure InereMembro (Achar : char); {cancelar}  
End;
```

O poder da herança fica evidente nesta declaração, que inclui um único método novo – todo o mais é herdado da classe-mãe CLCarConj!

A subclasse SubLeMai será usada como base para a criação da subclasse SubAlfa, que armazenará as letras maiúsculas e minúsculas. Apesar dos campos de dados herdados da classe-mãe CLCarConj serem adequados, iremos inserir mais dois campos de dados CDLetMai, CDLetMin, que servirão para armazenar as quantidades de letras maiúsculas e minúsculas, respectivamente.

Como nas outras subclasses, devemos sobrescrever o método InereMembro. Como inserimos dois campos de dados, precisamos de um novo código para o método Limpa.

A declaração da nova subclasse é a que vemos a seguir:

```
CAlfabeto = object(SubLeMai)  
  
    FSubLeMa, fLetMin : byte;  
  
    Procedure Limpa;  
  
    Procedure InereMembro (Achar : char); {cancelar}  
  
End;
```

A listagem a seguir demonstra a eficiência da herança e a sobrescrição de métodos, teste este programa e analise sua saída e você poderá comprovar esta eficiência.

Procure entender o porque da sobrescrição de métodos, faça-se as seguintes perguntas: O Pascal não se confunde com nomes de métodos parecidos?; De que maneira são tratados os campos de dados das classes?.

Listagem 2. Programa SCONJOB.PAS, que demonstra a hierarquia de classes, a herança e a sobrescrição de métodos herdados

```
Program Testa_Conjunto;  
uses Crt;  
Type CarConj = set of Char;  
    CLCarConj = object  
        CDConj : CarConj;  
        CDMembro : Byte;  
        Procedure Limpa;  
        Procedure EscreveConj;
```

```
        Procedure InereMembro(Achar:char);
        Procedure EliminaMembro(Achar:char);
        Function MembroNoConj (Achar:char): boolean;
end;
SubLeMa = object(CLCarConj)
    Procedure InereMembro(Achar:char); {Sobrescrever}
end;

SubDigito = object(CLCarConj)
    Procedure InereMembro(Achar:Char); {Sobrescrever}
end;

SubAlfa = object(SubLeMa)
    CDSubLeMa, CDLetMin : byte;
    Procedure Limpa; {Sobrescreve}
    Procedure InereMembro(Achar:char); {sobrescreve}
end;

SubCarEsp = object(CLCarConj)
    Procedure InereMembro (achar:char); {sobrescreve}
end;

Const
    ALFA : CarConj = ['A'..'Z','a'..'z'];
    MAIUS: CarConj = ['A'..'Z'];
    CDIGITOS : CarConj = ['0'..'9'];

procedure CLCarConj.Limpa;
begin
    CDConj := [];
    CDMembro := 0;
end;

Procedure SubAlfa.Limpa;
begin
    SubLeMa.limpa;
    CDSubLeMa := 0;
    CDLetMin := 0;
end;
```

```
Procedure CLCarConj.EscreveConj;
const Aspas = ' " ';
var I : byte;
begin
    write('[ ');
    for I := 0 to 255 do
        if chr(I) in CDConj then
            write(Aspas, chr(I), Aspas, ',');
    write(#8);
    writeln(']');
    writeln('O conjunto tem ',CDMembro, ' elemento(s)');
end;
```

```
Procedure CLCarConj.InsereMembro(Achar:Char);
begin
    if not (Achar in CDConj) then begin
        CDConj := CDConj + [Achar];
        inc(CDMembro);
    end;
end;
```

```
Procedure SubAlfa.InsereMembro(Achar:Char);
begin
    if not(Achar in CDConj) and (Achar in Alfa) then begin
        CDConj := CDConj + [Achar];
        inc(CDMembro);
        if Achar in Maius then
            inc(CDSubLeMa)
        else
            inc(CDLetMin);
    end;
end;
```

```
Procedure SubLeMa.InsereMembro(Achar:char);
begin
    if not(Achar in CDConj) and (Achar in Alfa) and (Achar in
    Maius) then begin
        CDConj := CDConj + [Achar];
        inc(CDMembro);
    end;
end;
```

```
Procedure SubDigito.InsereMembro(Achar:char);
begin
    if not(Achar in CDConj) and (Achar in CDigitos) then begin
        CDConj := CDConj + [Achar];
        inc(CDMembro);
    end;
end;

Procedure SubCarEsp.InsereMembro(Achar:char);
begin
    if not(Achar in CDConj) and not(Achar in Alfa) and not(Achar
    in Maius) and not(Achar in CDigitos) then begin
        CDConj := CDConj + [Achar];
        inc(CDMembro);
    end;
end;

Procedure CLCarConj.EliminaMembro(Achar:char);
begin
    if Achar in CDConj then begin
        CDConj := CDConj - [Achar];
        Dec(CDMembro);
    end;
end;

Function CLCarConj.MembroNoConj(Achar:char):boolean;
begin
    MembroNoConj := Achar in CDConj;
end;

Procedure PressioneUmaTecla;
var Sai : char;
begin
    writeln;
    write('Pressione uma tecla para dar nova entrada ');
    sai := readkey;
    writeln;
    writeln;
end;
```

```
var
  OBJCONJ   : CLCarConj;
  OBJLetras : SubAlfa;
  OBJMai    : SubLeMa;
  OBJDigito : SubDigito;
  OBJSimbolo : SubCarEsp;
  op, op1   : byte;
  let       : char;

begin
  OBJConj.Limpa;
  ObjMai.Limpa;
  ObjLetras.limpa;
  OBJDigito.Limpa;
  OBJSimbolo.Limpa;
  repeat
    clrscr;
    gotoxy(6,5);writeln('para sair escolha digite 5');
    gotoxy(6,10);writeln('ENTRE COM UMA DAS OPCOES ABAIXO');
    gotoxy(7,15);write('1 - Inclusao de caracteres');
    gotoxy(7,16);write('2 - Eliminacao de caracteres');
    gotoxy(7,17);write('3 - Consulta de caracteres');
    gotoxy(7,18);write('4 - Mostrar Caracteres');
    gotoxy(7,19);read(op);
    clrscr;
    case op of
      1:begin
        op := 0;
        gotoxy(7,15);write('1 - Inclusao no conjunto Global');
        gotoxy(7,16);write('2 - Inclusao no conjunto dos
Mausculos');
        gotoxy(7,17);write('3 - Inclusao no conjunto de letras
Mausculas e Minusculas');
        gotoxy(7,18);write('4 - Inclusao no conjunto de
Caracteres especiais');
        gotoxy(7,19);write('5 - Inclusao no conjunto de
Digitos');
        gotoxy(7,20);readln(op1);
        case op1 of
          1: begin
            clrscr;
            write;
            Gotoxy(5,4);write('Qual o Caracter? ');
```

```
Gotoxy(5,5);read(let);
OBJConj.InsereMembro(Let);
end;

2: begin
  clrscr;
  Gotoxy(5,4);write('Qual o Character? ');
  gotoxy(5,5);read(let);
  OBJMai.InsereMembro(Let);
  OBJLetras.InsereMembro(Let);
end;

3: begin
  clrscr;
  Gotoxy(5,4);write('Qual o Character? ');
  gotoxy(5,5);read(let);
  OBJLetras.InsereMembro(Let);
  OBJMai.InsereMembro(Let);
end;

4: begin
  clrscr;
  Gotoxy(5,4);write('Qual o Character? ');
  gotoxy(5,5);read(let);
  OBJSimbolo.InsereMembro(Let);
end;

5: begin
  clrscr;
  Gotoxy(5,4);write('Qual o Character? ');
  gotoxy(5,5);read(let);
  OBJDigito.InsereMembro(Let);
end;
end;

2 : begin
  gotoxy(7,15);write('1 - Eliminacao no conjunto
Global');
  gotoxy(7,16);write('2 - Eliminacao no conjunto dos
Maiusculos');
  gotoxy(7,17);write('3 - Eliminacao no conjunto de
letras Maiusculas e Minusculas');
  gotoxy(7,18);write('4 - Eliminacao no conjunto de
Caracteres especiais');
```

```
        gotoxy(7,19);write('5 - Eliminacao no conjunto de
Digitos');
        gotoxy(7,20);readln(op1);
        case op1 of
            1: begin
                clrscr;
                Gotoxy(5,4);write('Qual o Character? ');
                gotoxy(5,5);read(let);
                OBJConj.EliminaMembro(Let);
            end;
            2: begin
                clrscr;
                Gotoxy(5,4);write('Qual o Character? ');
                gotoxy(5,5);read(let);
                OBJMai.EliminaMembro(Let);
            end;
            3: begin
                clrscr;
                Gotoxy(5,4);write('Qual o Character? ');
                gotoxy(5,5);read(let);
                OBJLetras.EliminaMembro(Let);
            end;
            4: begin
                clrscr;
                Gotoxy(5,4);write('Qual o Character? ');
                gotoxy(5,5);read(let);
                OBJSimbolo.EliminaMembro(Let);
            end;
            5: begin
                clrscr;
                Gotoxy(5,4);write('Qual o Character? ');
                gotoxy(5,5);read(let);
                OBJDigito.EliminaMembro(Let);
            end;
        end;
    end;
3: begin
    gotoxy(7,15);write('1 - Consulta no conjunto Global');
    gotoxy(7,16);write('2 - Consulta no conjunto dos
Maiusculos');
```

```
    gotoxy(7,17);write('3 - Consulta no conjunto de letras
Maiusculas e Minusculas');

    gotoxy(7,18);write('4 - Consulta no conjunto de
Caracteres especiais');

    gotoxy(7,19);write('5 - Consulta no conjunto de
Digitos');

gotoxy(7,20);readln(op1);
case op1 of
  1: begin
      clrscr;
      Gotoxy(5,4);write('Qual o Character? ');
      read(let);
      Gotoxy(5,7);write('O caracter ',let,' esta no
conjunto ? ', OBJConj.MembroNoConj(Let));
      readln;readln;
    end;
  2: begin
      clrscr;
      Gotoxy(5,4);write('Qual o Character? ');
      read(let);
      Gotoxy(5,7);write('O caracter ',let,' esta no
conjunto ? ', OBJMai.MembroNoConj(Let));
      readln;readln;
    end;
  3: begin
      clrscr;
      Gotoxy(5,4);write('Qual o Character? ');
      read(let);
      Gotoxy(5,7);write('O caracter ',let,' esta no
conjunto ? ', OBJletras.MembroNoConj(Let));
      readln;readln;
    end;
  4: begin
      clrscr;
      Gotoxy(5,4);write('Qual o Character? ');
      read(let);
      Gotoxy(5,7);write('O caracter ',let,' esta no
conjunto ? ', OBJSimbolo.MembroNoConj(Let));
      readln;readln;
    end;
  5: begin
      clrscr;
      Gotoxy(5,4);write('Qual o Character? ');
```

```
        read(let);
        Gotoxy(5,7);write('O caracter ',let,' esta no
conjunto ? ', OBJDigito.MembroNoConj(Let));
        readln;readln;
    end;
end;
end;
4: begin
    gotoxy(7,15);write('1 - Mostra o conjunto Global');
    gotoxy(7,16);write('2 - Mostra o conjunto dos
Maiusculos');
    gotoxy(7,17);write('3 - Mostra o no conjunto de letras
Maiusculas e Minusculas');
    gotoxy(7,18);write('4 - Mostra o conjunto de Caracteres
especiais');
    gotoxy(7,19);write('5 - Mostra o conjunto de Digitos');
    gotoxy(7,20);readln(op1);
    case op1 of
        1: begin
            clrscr;
            Gotoxy(5,4);write('O conjunto e ');
            OBJConj.EscreveConj;
            readln;readln;
        end;
        2: begin
            clrscr;
            Gotoxy(5,4);write('O conjunto e ');
            OBJMai.EscreveConj;
            readln;readln;
        end;
        3: begin
            clrscr;
            Gotoxy(5,4);write('O Conjunto e');
            OBJLetras.EscreveConj;
            write('A quantidade de letras maiúsculas é ',
ObjLetras.CDLetMai);
            write('A quantidade de Letras minúsculas é ',
ObjLetras.CDLetMin);
            readln;readln;
        end;
        4: begin
            clrscr;
```

```
        Gotoxy(5,4);write('O conjunto e ');
        OBJSimbolo.EscreveConj;
        readln;readln;
    end;
5: begin
    clrscr;
    Gotoxy(5,4);write('O conjunto e ');
    OBJDigito.EscreveConj;
    readln;readln;
    end;
end;
end;
until op = 5;
end.
```

Exemplo

O programa a seguir contém um vetor de string com 30 posições e executa os seguintes métodos:

1. lê um nome para cada posição do vetor;
2. o nome digitado será convertido para maiúsculo;
3. para cada nome digitado, será ser mostrado na tela a relação dos nomes já incluídos juntamente com o número da posição que ele ocupa no vetor;
4. a remoção de um nome do vetor é feito a partir do número da posição que ele ocupa no vetor.
5. Após a remoção, o vetor deverá ser reorganizado, não podendo ficar nenhum elemento em branco entre duas posições do vetor;
6. após a reorganização deverá ser atualizada a lista na tela.

```
program exemplo;
uses crt,
    moldura;
var
    PegNom  : string;
    Op, P   : byte;
type v = array [1..30] of string;
    vetnomObj = object
        Nome : v;
        I    : integer;
        procedure inicio;
        procedure PegaNome;
        procedure ConverteNome(Nom:string);
        procedure IncluiNome (Nom : string);
        procedure Mostra;
        function InibeRemocao : boolean;
        procedure Remocao (posi : byte);
        procedure Arranjo (PosiSai : byte);
    end;
procedure vetnomObj.Inicio;
var ind : byte;
begin
    for ind := 1 to 30 do
        nome[ind] := ' ';
    i := 1;
end;
```

```

procedure vetnomObj.PegaNome;
begin
    window(5,20,40,24);
    gotoxy(1,1);
    write('Entre com um nome --> ');
    gotoxy(2,2);
    read(PegNom);
    window(1,1,80,25);
end;
Procedure vetnomObj.ConverteNome(nom:string);
var is : byte;
begin
    for is := 1 to length(nom) do
        pegnom[is] := upcase(nom[is]);
    end;

Procedure vetnomObj.IncluiNome (nom:string);
var il : byte;
begin
    if (i = 1) and (nome[1] = ' ') then
        nome[1] := nom
    else
        for il := 2 to 30 do
            if nome[il] = ' ' then begin
                nome[il] := nom;
                exit;
            end;
        end;
    end;
procedure vetnomObj.Mostra;
var ind:byte; col, ino:byte;
begin
    lowvideo;
    clrscr;
    mold(1,1,17,40);
    mold(42,1,17,79);
    mold(1,19,24,79);
    ind:=1; col:=3; ino:= 2;
    while (ind <= 30) and (nome[ind] <> ' ') do begin
        gotoxy(col,ino);
        ino := ino + 1;
        write(ind,' ', nome[ind]);
        if ind mod 15 = 0 then
            begin
                col := col + 20;
                ino := 2;
            end;
        ind := ind + 1;
    end;
end;
function vetnomObj.inibeRemocao:boolean;
begin

```

```

        if nome[1] = ' ' then InibeRemocao := true
            else InibeRemocao := false;
end;

procedure vetnomObj.remocao(posi:byte);
begin
    nome[posi] := ' ';
end;
procedure vetnomObj.Arranjo(posisai:byte);
var ind : byte;
begin
    ind := posisai + 1;
    while (ind <= 30) and (nome[ind] <> ' ') do begin
        nome[posisai] := nome[ind];
        posisai := ind;
        ind:=posisai + 1;
    end;
    nome[ind-1] := ' ';
end;

var vetornome : vetnomObj;
begin
    textbackground(1);
    clrscr;
    mold(1,1,17,40);
    mold(42,1,17,79);
    mold(1,19,24,79);
    vetornome.inicio;
    repeat
        window(50,8,70,30);
        gotoxy(50,8);
        writeln('= Incluir Nomes ==> (1) ');
        gotoxy(50,9);
        writeln('= Excluir Nomes ==> (2)');
        gotoxy(50,10);
        writeln('= Sair do programa ==> (0)');
        gotoxy(45,5);
        write('Escolha a opcao -->      ');
        gotoxy(65,5);
        readln(op);
        window(1,1,80,25);
        case op of
            1: begin
                vetornome.Peganome;
                vetornome.ConverteNOME(pegnom);
                vetornome.IncluiNOME(pegnom);
            end;
            2: begin
                window(5,20,40,24);
                if not(vetornome.InibeRemocao) then

```

```
        write('Entre com a Posicao ');
        read(p);
        vetornome.remocao(p);
        vetornome.arranjo(p);
    end
else begin
        textcolor(4);
        highvideo;
        gotoxy(5,21);
        write('Nao existem nomes para
remocao');

        write(chr(7));
        delay(1500);
        lowvideo;
        textcolor(15);
        end;
        window(1,1,80,25);
    end;
end;
vetornome.mostra;
until op=0;
end.
```

INICIALIZANDO OBJETOS

O Turbo Pascal e outras variações de Pascal Objeto não aceitam a inicialização automática de objetos. Em Turbo Pascal, os objetos estáticos são automaticamente criados e removidos, exatamente como as variáveis estáticas. No entanto, o processo de inicialização de campos de dados é exigido na maioria das aplicações, para evitar que valores inadequados sejam inseridos nos campos de dados de uma classe. É fácil diagnosticar e remediar um erro de omissão de um único método de inicialização. O remédio não é tão simples quando vários métodos contribuem para a inicialização de um objeto.

No caso de um único método de inicialização, pode-se empregar um campo do tipo string ao qual se atribui a constante "INICIALIZADA" (ou qualquer outra string que possa ser diferenciada dos dados não válidos) quando o método de inicialização é evocado. Todos os outros métodos podem procurar pela string INICIALIZADA no campo de inicialização CDSituação. Geralmente, o método é encerrado quando a string não for INICIALIZADA. A listagem a seguir contém um programa que cria um vetor dinâmico. A definição de classe inclui o campo de dados CDSituação para reportar o *status* da inicialização. O método Início é usado para alocar o tamanho do vetor dinâmico, uma etapa necessária. O método Enche preenche o vetor com um valor. Este método examina o campo CDSituação. Se CDSituação for NÃO INICIADA, o vetor dinâmico será criado usando um tamanho default (supondo-se que esta ação seja aceitável na aplicação) e depois será preenchido com números.

Listagem – Programa DEVEINIC.PAS para testar a inicialização obrigatória.

```
Program Teste_de_Inicialização;
Uses Crt;
{$R-}
CONST TAM_INICIAL = 100;
TYPE
  UmElemVetor = array [1..1] of real;
  UmElemPtr = ^UmElemVetor;
  VetorDin = Object
    CDSituação : string;
    CDTamMax : Word;
    CDVetPtr : UmElemPtr;
    Procedure Início (PTamMax : word);
    Function PegaTam : word;
    Procedure Enche (X : Real);
    Procedure Remove;
  End;
Procedure VetorDin.Início(PTamMax : Word)
Begin
  CDMaxTam := PTamMax;
  If CDMaxTam < 1 then
    CDTamMax := TAM_INICIAL;
  GetMem (CDVetPtr, CDTamMax * SizeOf(real));
  CDSituação := 'INICIALIZADA';
End;
Function VetorDin.PegaTam:Word;
```

```
Begin
    PegaTam := CDTamMax;
End;
Procedure VetorDin.Enche (X : Real);
Begin
    If CDSituação <> 'INICIALIZADA' then
        Início(TAM_INICIAL);
    For I := 1 to CDTamMax do
        CDVetPtr^[I] := X;
End;
Procedure VetorDin.Remove;
Begin
    If CDSituação = 'INICIALIZADA' then begin
        FreeMem (CDVetPtr, TamMax * SizeOf(Real));
        CDSituação := 'NÃO INICIADA';
    End;
End;
Var  A : VetorDin;
     X : Real;
     Sai : char;
Begin
    ClrScr;
    X := 100.0;
    A.Enche(X); {inicializa no tamanho default}
    Writeln ('Vetor dinâmico preenchida com ', X:4:0,
's');
    Writeln('O Tamanho do vetor dinâmico é ',
A.PegaTam);
    A.Remove;
    Writeln;
    Write('Pressione qualquer tecla para encerrar o
programa...');
    Sai := readkey;
End.
```

No caso de vários métodos de inicialização, uma array de flags boolean é usado para certificar que cada método seja chamado para preparar o objeto.

Sobre o Programa

Diretivas de compilação

Erros durante a execução – Verificação da Faixa {\$R} – Quando esta opção é ativada, é gerado um código que verifica subscritos de cadeia e de array e condições fora da faixa referentes a tipos escalares de dados. Quando esta opção é desativada, esse código não é gerado e o seu programa se torna menor. Esses tipos de erro são difíceis de detectar, de forma que você deve manter essa diretiva no seu código até que ele esteja depurado.

Function **SizeOf(Variável Var):word**

Retorna o número de bytes exigido por uma variável ou um tipo de dados.

Procedure GetMem (Var P : pointer; I: Integer);

Reserva a quantidade de bytes especificada em **I** na pilha e armazena o endereço inicial na variável **P**.

Procedure FreeMem(Var P: pointer; I : integer);

Libera a quantidade de bytes especificada em **I**, corresponde à área de memória de pilha, associada à variável **P**, que deve ter sido previamente alocada por **GetMem**.

Exercício

1. Construa um programa em Pascal que calcule o valor futuro de um investimento com base no valor presente, a taxa de juros e o número de períodos do investimento. O programa deve conter uma classe que empregue um método separado de inicialização. A fórmula para o cálculo do valor futuro é a seguinte:

$$M = C \times (1 + I / 100)^N$$

em que C = Valor Presente

 I = Taxa de Juros

 n = números de períodos

 M = Valor Futuro

OBJETOS E MÉTODOS DINÂMICOS

Os objetos e os métodos que vimos até aqui são estáticos. O Turbo Pascal também incrementa objetos dinâmicos. Os objetos estáticos são compilados em código de máquina que roda mais rápido do que o dos objetos dinâmicos. Por outro lado, os objetos dinâmicos oferecem mais flexibilidade no momento da execução e suportam o polimorfismo.

Os métodos estáticos são caracterizados quando de sua chamada, pois estas são combinadas durante a sua compilação. Já nos métodos virtuais, estas chamadas são combinadas durante a execução no momento que é efetuada a chamada.

Os principais veículos dos objetos dinâmicos são os métodos virtuais. A palavra-chave VIRTUAL é colocada em uma instrução separada depois de cada declaração de método virtual. Os métodos virtuais precisam ser acompanhados de **construtores**, que são métodos dedicados à inicialização de objetos com métodos virtuais. A palavra-chave CONSTRUCTOR substitui a palavra PROCEDURE quando da declaração de um construtor. O nome sugerido para os construtores é INÍCIO. O Turbo Pascal também utiliza **destrutores** que anulam os efeitos dos construtores. Sua tarefa é executar uma limpeza nos campos de dados. O nome sugerido para os destrutores é LIMPA.

É necessário seguir algumas regras para declaração de métodos virtuais:

1. Se um método é declarado virtual em uma classe, ele precisa ser declarado virtual em todas as classes descendentes.

```
Type r = object
    A, B : byte;
    constructor INICIO;
    procedure SOMA(L, S : byte); VIRTUAL;
    destructor LIMPA;
end;
r1 = object (r)
    procedure SOMA (L, S : byte); VIRTUAL;
end;
```

2. A declaração de um método virtual em uma subclasse pode sobrescrever um método não-virtual na(s) classe(s) hierarquicamente superior.

```
Type r = object
    A, B : byte;
    constructor INICIO;
    procedure SOMA(L, S : byte);
    destructor LIMPA;
end;
r1 = object (r)
    procedure SOMA (L, S : byte); VIRTUAL;
end;
```

3. Os métodos virtuais *precisam ter a mesma lista de parâmetros em cada classe onde ela for usada.*

Errado:

```
Type r = object
    A, B : byte;
    constructor INICIO;
    procedure SOMA(L, S : byte); VIRTUAL;
    destructor LIMPA;
end;
r1 = object (r)
    procedure SOMA (L : byte); VIRTUAL;
end;
```

Certo:

```
Type r = object
    A, B : byte;
    constructor INICIO;
    procedure SOMA(L, S : byte); VIRTUAL;
    destructor LIMPA;
end;
r1 = object (r)
    procedure SOMA (L, S : byte); VIRTUAL;
end;
```

4. Os métodos virtuais podem ser herdados.

```
Type r = object
    A, B : byte;
    constructor INICIO;
    procedure SOMA(L, S : byte); VIRTUAL;
    destructor LIMPA;
```

```
end;  
  r1 = object (r)  
end;
```

As regras para utilização de construtores são:

1. É imprescindível chamar um construtor antes de que se envie uma mensagem virtual à um objeto. Sob pena de cancelamento da execução do programa.
2. Múltiplos construtores permitem várias maneiras de inicializar um objeto.
3. Os construtores podem ser herdados.

As regras para utilização de destrutores são:

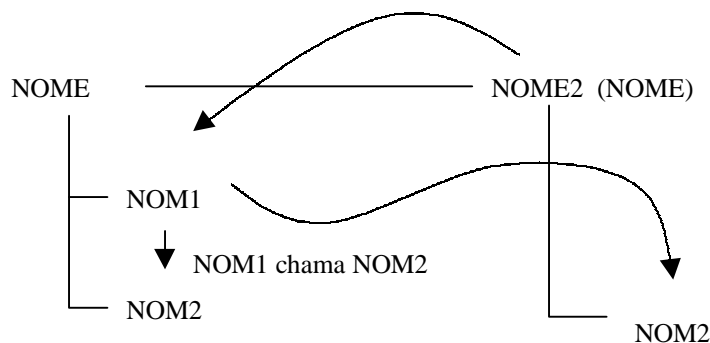
1. Um destrutor deve ser chamado para que haja limpeza adequada dos campos dados.
2. Múltiplos destrutores permitem várias maneiras de remover um objeto.
3. Os destrutores podem ser herdados.
4. Destrutores virtuais são permitidos.

Exercícios

1. Existe diferença entre objetos estáticos e objetos dinâmicos?
2. Se a resposta anterior for sim, quais?
3. Qual a palavra-chave colocada para indicar que um método é virtual?
4. O que é um construtor?
5. O que é um destrutor?

VINCULAÇÃO TARDIA E POLIMORFISMO

Polimorfismo significa a capacidade que um objeto tem de responder a um comando ou mensagem de maneira peculiar a ele próprio. A mensagem em si não é propriedade de nenhuma classe específica. O melhor exemplo do polimorfismo é a vinculação tardia de métodos no momento da execução. Para entender polimorfismo e vinculação tardia, considere a hierarquia de classes mostrada na figura abaixo. A figura mostra uma classe NOME com dois métodos, NOM1 e NOM2, tal que NOM1 chama NOM2. Uma mensagem NOM1 enviada a um objeto da classe NOME, chamemo-lo ObNomA, executa os métodos NOME.NOM1 e NOME.NOM2. A figura também mostra a classe NOME2 como subclasse de NOME com o método NOM2 sobrescrito. Quando uma mensagem NOM1 é enviada a um objeto da classe NOME2, chamemo-la ObNomB, a seqüência de métodos executados será a seguinte:



1. O sistema em execução pega a mensagem ObNomB.NOM1 e tenta encontrar um método coincidente entre aqueles da classe NOME2.
2. A ação acima não obtém sucesso e o sistema recorre ao exame dos métodos da classe-mãe NOME. A busca é eficaz desta vez e o código do método NOME.NOM1 é usado.
3. Uma vez que o método NOME.NOM1 chama o método NOME.NOM2, o sistema em execução precisa resolver a chamada. Ele primeiro volta aos métodos da subclasse NOME2 (a classe cuja instância recebeu a mensagem original) e tenta localizar um método NOM2.
4. A busca obtém êxito e o sistema emprega o código de NOME2.NOM2 para resolver a chamada de ObNomB.NOM1.

Na ausência da vinculação tardia, o sistema em execução teria inserido incorretamente (do ponto de vista da lógica do programa) o código do método NOME.NOM2 para resolver a chamada feita pelo método NOME.NOM1.

O programa-exemplo ilustra como funciona a vinculação tardia. A classe CLNOME declara um construtor Início e dois métodos, BoasVindas e PegNome. A classe SUBNOME, uma classe-filha de CLNOME, emprega um método PegNome virtual para pedir que você forneça o nome utilizando uma

frase diferente. A subclasse SUBNOME também utiliza o construtor herdado Início. A parte principal do programa envia a mensagem BoasVindas às instâncias de ambas as classes A e B. Por causa da vinculação tardia, as mensagens enviadas exibem dois prompts diferentes.

O programa VINCTARD.PAS ilustra o efeito da vinculação tardia.

```
Program Teste_de_Vinculação_Tardia;

Uses Crt;

Type
  CLNOME = OBJECT
    CDNome : STRING;
    CONSTRUCTOR Início;
    PROCEDURE BoasVindas;
    PROCEDURE PegNome; VIRTUAL;
  End;

  SubNome = OBJECT(CLNOME)
    PROCEDURE PegNome; VIRTUAL;
  End;

CONSTRUCTOR CLNOME.Início;

Begin
  cdNome := '';{Inicializar o campo Nome em uma string nula}
End;

PROCEDURE CLNOME.PegNome;

Begin
  WRITE('Seu nome, por favor? ');
  READLN(cdNome); WRITELN;

End;

PROCEDURE CLNOME.BoasVindas;

Begin
  PegNome;

  WRITELN('Olá ', cdNome, ', como vai?');
  WRITELN;

End;

PROCEDURE SubNome.PegNome;

Begin
```

```
WRITE('Digite seu nome -> ');  
READLN(cdNome); WRITELN;  
  
End;  
  
Var  
  
    A: CLNOME;  
    B: SubNome;  
    Sai: char;  
  
Begin  
  
    Clrscr;  
    {inicializa objetos}  
    A.Início;  
    B.Início;  
    {Envia a mensagem BoasVindas aos objetos A e B}  
    A.BoasVindas;  
    B.BoasVindas;  
    Writeln;  
    Write('Pressione qualquer tecla para encerrar o programa');  
    SAI := READKEY;  
  
End.
```

Modifique o programa retirando a palavra virtual dos procedimentos PegNome e veja o que acontece.

Após ter visto a nova execução retire a palavra Virtual somente do primeiro procedimento PegNome, execute o programa e veja a resposta obtida.

OBJETOS DINÂMICOS

New

Sintaxe: New(Var P: Pointer);

Aloca memória na pilha para o ponteiro P. Depois de alocar memória, a variável é tratada como P.

Dispose

Sintaxe: Dispose(P: Pointer);

Libera memória da pilha alocada para uma variável de ponteiro. A função Dispose é usada junto com o comando New.

Para utilizá-las com objetos as funções citadas sofrem uma pequena modificação.

A instrução NEW pode incluir o nome de um construtor (juntamente com a lista de argumentos do construtor) como segundo parâmetro, conforme mostra a sintaxe geral a seguir:

New(<ponteiro>, <construtor>(<lista de parâmetros>));

Da mesma forma, o procedimento DISPOSE pode incluir o nome de um destrutor, conforme mostra a sintaxe geral a seguir:

Dispose(<ponteiro>, <destrutor>(<lista de parâmetros>));

A listagem a seguir contém um programa simples que ilustra um objeto dinâmico. O programa implementa uma pilha usando uma vetor dinâmico. Os valores randômicos das coordenadas de tela são colocados e tirados da pilha. O tamanho da vetor é determinado no momento da execução. Uma vez estabelecido, permanece fixo. Consequentemente, a pilha que é baseada na vetor dinâmico, também tem tamanho fixo. O programa define a classe Cursor da seguinte maneira:

```
Cursor = Object
  Posicao : VetorPtr;
  CDTam, CDIndice : word;
  CDChamIni : Boolean;
  Constructor Inicio (dTam : word);
  Function InsereXY (Xloc, Yloc : byte):
boolean;
  Function RemoveXY : Boolean;
  Destructor Retira;
End;
```

O campo de dados Posição, é um ponteiro dinâmico para um vetor de registros do tipo Ponteiro que armazenam as coordenadas do cursor. O campo de dados CDTam armazena o tamanho do vetor dinâmico usado para implementar a pilha. O campo de dados CDIndice toma nota da altura da pilha. O campo de dados CDChamIni é usado para sinalizar se o construtor Inicio foi ou não chamado.

O programa define um construtor e um destrutor para alocar e desalocar os dados dinâmicos, respectivamente. Os métodos InsereXY e RemoveXY são usados para inserir e remover as coordenadas do cursor. Estes métodos retornam TRUE quando obtém êxito, e FALSE em caso de falha.

O programa emprega as seguintes instruções para criar e remover o ponteiro para o objeto e seu vetor dinâmico:

```
New (A, Inicio(Teste_Tam)); {para criar}
```

```
Dispose(A, Retira);{para remover}
```

Observe que um circunflexo é usado depois do identificador A quando do envio das mensagens InsereXY e RemoveXY, já que A é um ponteiro formal do Pascal. O programa exibe a string “Aqui” em cinco locais aleatórios da tela.

```
Program Objetos_Dinâmicos;
Uses Crt;
{-$R}
const Tamanho = 5;
type
Ponteiro = record
    x, y : byte;
end;
UmElemPont = array[1..1] of Ponteiro;
VetorPtr = ^UmElemPont;
Cursor = Object
    Posicao : VetorPtr;
    CDTam, CDIndice : word;
```

```
    CDChamIni : Boolean;
    Constructor Inicio (dTam : word);
    Function InsereXY (Xloc, Yloc : byte): boolean;
    Function RemoveXY : Boolean;
    Destructor Retira;
End;

Posi = ^Cursor;

CONSTRUCTOR Cursor.Inicio (dTam:word);
begin
    GetMem(Posicao, dTam);
    CDTam := dTam;
    CDIndice := 0;
    CDChamIni := True;
end;
Destructor Cursor.Retira;
begin
    FreeMem(Posicao, CDTam);
    CDTam := 0;
    CDIndice := 0;
    CDChamIni := False;
end;
Function Cursor.InsereXY(Xloc, Yloc : byte): boolean;
begin
    if (not CDChamIni) or (CDIndice = CDTam) then
begin
        InsereXY := False;
        Exit
    end
    else
        Inc(CDIndice);
        if Not(Xloc in [1..80]) then Xloc := 1;
        if not(Yloc in [1..25]) then Yloc := 1;
        Posicao^[CDIndice].X := Xloc;
        Posicao^[CDIndice].Y := Yloc;
        InsereXY := True;
    end;
Function Cursor.RemoveXY:Boolean;
begin
```

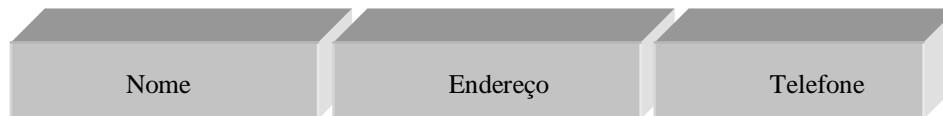
```
        If CDChamIni and (CDIndice > 0 ) then begin
            GotoXY(Posicao^[CDIndice].X,
Posicao^[CDIndice].Y);
                Dec(CDIndice)
            end
        else
            RemoveXY := False;
    end;
Var
    A : Posi;
    I : Byte;
    Sai : Char;
Begin
    New(A,Inicio(Tamanho)); {Alocar Espaço}
    ClrScr;
    Randomize;
    While A^.InsereXY(Random(80), Random(25)) do;
    While A^.RemoveXY do Begin
        write('Aqui!!!');
        delay(1000);
    end;
    Dispose(A, Retira); {Desaloca espaço}
    GotoXY(1,24);
    ClrEol;
    Writeln;
    Write('Pressione qualquer tecla para encerrar...');
    sai := Readkey;
end.
```

ARQUIVOS COM OBJETOS

Arquivo ou File é um conjunto de informações que podem ser conservadas permanentemente em memória secundária, como disco ou disquete, sem que seu conteúdo seja extinto no momento de desligar o computador. É integrado por vários elementos do tipo RECORD, e seu grande proveito está associado ao fato de o usuário poder acessar qualquer dado nele cadastrado, através de programas delineados com este intuito.

O Turbo Pascal oferece três tipos básicos de arquivos em disco: arquivos de texto; arquivos com tipo definido; e arquivo sem tipo definido.

Para facilitar nosso entendimento irei fixar-me em Arquivos com tipo definido.



Para gerar o arquivo contendo vários registros do tipo ilustrado no esquema anterior (RECORD Registro) pode-se aventar o seguinte tipo de cabeçalho no programa:

```
Program ArquivoObjeto;  
Uses Crt;  
Type  
    Registro = RECORD  
        Nome : string[30];  
        Endereco : string[40];  
        Telefone: string[15];  
    end;  
Arquivo = file of Registro;
```

Após termos criado nosso registro, este passará a ser o “objeto” de nosso estudo, podemos, agora definir uma classe que encapsule este objeto, juntamente com seus métodos.

```
ClassReg = Object  
  
    DadosPessoais : array [1..100] of Registro;  
  
    procedure CriaArquivo;
```

```
procedure MostraArquivo;
```

```
procedure IncluiRegistro;
```

```
end;
```

Precisamos associar um nome para que o Turbo reconheça o nosso registro, o arquivo e a classe.

```
Var  
    A          : ClassReg;  
    RegA, RegB : Arquivo;  
    Sai        : char;  
    UltReg     : integer;
```

Os nomes dos campos do registro-objeto DadosPessoais do arquivo Arquivo são formados por uma combinação do identificador-objeto DadosPessoais do registro-objeto com cada campo desejado, separados por um ponto, ou se desejar, utilizando o comando with ... do, como já vimos em aulas anteriores.

DadosPessoais[1]•nome, representando o campo nome

DadosPessoais[1]•Endereco, representando o campo endereço

DadosPessoais[1]•Telefone, representando o campo telefone

ou

```
with DadosPessoais[1] do  
    begin  
        Nome  
        Endereco  
        Telefone  
    end;
```

COMANDOS DE MANIPULAÇÃO DE ARQUIVOS

O Turbo Pascal dispõe de uma série de funções e procedimentos que permitem gerar, atualizar e consultar arquivos.

ASSIGN

é utilizado para associar o nome dado ao arquivo no programa Turbo com o designado para representá-lo no disco ou disquete. Ambos são escolhidos pelo desenvolvedor.

No exemplo: `Assign (RegA, 'Teste.dat');`

O nome do arquivo é:

`RegA` → no programa Turbo

`Teste.dat` → na unidade de disco ou disquete.

Após o **ASSIGN**, qualquer alusão ao arquivo externo é feita através de seu nome no programa Turbo, como é o caso de `RegA` para designar a entrada de `Teste.dat`.

RESET

Deixa o arquivo externo pronto para ser lido, a partir de seu registro inicial.

Exemplo: `Reset (RegA);`

REWRITE

O procedimento **REWRITE** prepara um arquivo externo para ser gravado. Após a sua execução, o primeiro registro do arquivo, que é sempre o de número zero, fica disponível para ser escrito. Se já houver algum cadastro com o mesmo nome do que aparece no **Rewrite**, o antigo é destruído.

Exemplo: `Rewrite (RegA);`

EOF

End Of File é uma função incumbida de situar o último registro de um arquivo. Ela tem meios de verificar se o usuário mandou ler alguma coisa situada após o fim do cadastro.

Exemplo: `while not eof(RegA) do`

CLOSE

Serve para encerrar a manipulação de um arquivo e levar ao sistema operacional referências atualizadas sobre ele.

Exemplo: `Close(RegA);`

READ

Lê arquivos em disco ou disquetes, cuja forma geral é:

`read(nome do arquivo, nome do registro);`

Exemplo: `read(RegA, DadosPessoais[I]);`

WRITE

Quando se tratar de gravação de informações em disco ou disquetes sua sintaxe é a seguinte:

`Write (nome do arquivo, nome do registro);`

Exemplo: `write (RegA, DadosPessoais[I]);`

FILESIZE

Função que gera um número inteiro, que simboliza a quantidade de registros gravados em um arquivo.

Exemplo: `Filesize(RegA);`

SEEK

Procedimento que permite localizar determinado registro num arquivo em disco ou disquete.

`seek (nome do arquivo, NUM);`

onde Num é o número do registro desejado.

Supondo como exemplo:

`K := Filesize(RegA) - 1;`

tem-se que:

```
Seek (RegA, K);
```

situa o último registro do cadastro RegA, pois o primeiro deles é o de número zero.

UTILIZANDO ARQUIVOS COM OBJETOS

A seguir apresento um pequeno programa, cuja finalidade é criar um arquivo de registro, sendo este registro objeto em uma classe. O programa permite, incluir novos registros no arquivo. Atente para o método de inclusão de registro e repare que neste método abri dois arquivos, um com reset e o outro com rewrite.

```
program ArquivoObjeto;
uses crt;
type
  Registro = record
    Nome : string[30];
    Endereco : string[40];
    Telefone : string[15];
  end;
  arquivo = file of Registro;
  ClassReg = object
    DadosPessoais : array [1..100] of Registro;
    procedure CriaArquivo;
    procedure MostraArquivo;
    procedure IncluiRegistro;
  end;
var
  A : ClassReg;
  RegA, RegB : arquivo;
  sai : char;
  ultReg : integer;
procedure ClassReg.CriaArquivo;
var
  i : integer;
begin
  assign(RegA, 'Teste.dat');
  rewrite(RegA);
  with DadosPessoais[1] do
    begin
      nome := 'Rosane Santos' ;
      Endereco := 'Barra';
      Telefone := '99992111';
    end;
  write(RegA, DadosPessoais[1]);
  with DadosPessoais[2] do
    begin
      nome := 'Sergio Santos';
      Endereco := 'Estacio de Sa - Barra';
      Telefone := '96789061';
    end;
end;
```

```
        write(RegA,DadosPessoais[2]);
with DadosPessoais[3] do
begin
    nome := 'Portela';
    Endereco := ` Centro';
    Telefone :='55167000';
end;
write(RegA,DadosPessoais[3]);
close(RegA);
end;

{*****}

procedure ClassReg.MostraArquivo;
var i,j : integer;
begin
    clrscr;
    i := 1;
    Assign(RegA,'Teste.dat');
    Reset(RegA);
    while not eof(RegA) do
    begin
        read(RegA,DadosPessoais[i]);
        with DadosPessoais[i] do
        begin
            writeln(Nome,' ',Endereco,' ',Telefone);
        end;
        inc(i);
    end;
    ultReg := i;
    close(RegA);
end;

{*****}

procedure ClassReg.IncluiRegistro;
var IncNome : String[30];
    IncEnd   : String[40];
    IncTel   : String[15];
    i,j      : integer;
begin
    assign(RegA,'Teste.dat');
    assign(RegB,'TesteB.dat');
    reset(RegA);
    rewrite(RegB);
    j:=1;
    i :=ultreg;
    read(RegA,DadosPessoais[j]);
    while not eof(RegA) do
    begin
        write(RegB,DadosPessoais[j]);
        inc(j);
        read(RegA,DadosPessoais[j]);
    end;
    write(RegB,dadosPessoais[j]);
    Close(RegA);
```

```
Write('Entre com um nome ... ');
readln(IncNome);
write('Entre com um endereço ... ');
readln(IncEnd);
write('Entre com um Telefone ... ');
readln(IncTel);

with DadosPessoais[i] do
begin
    nome := IncNome;
    Endereco := IncEnd;
    Telefone := IncTel;
end;
write(RegB,DadosPessoais[i]);
close(RegB);
reset(RegB);
rewrite(RegA);
j := 1;
read(RegB,DadosPessoais[j]);
while not eof(RegB) do
begin
    write(RegA,DadosPessoais[j]);
    inc(j);
    read(RegB,DadosPessoais[j]);
end;
write(RegA,DadosPessoais[j]);
Close(RegB);
close(RegA);
end;

{*****}

Begin
clrscr;
sai := ' ';
a.CriaArquivo;
a.MostraArquivo;
repeat
    if sai <> #13
        then a.IncluiRegistro;
    a.MostraArquivo;
    writeln;
    writeln('Pressione QQ tecla para continuar ou Enter para
sair...');
    sai := readkey;
until sai = #13;
end.
```

REFERÊNCIAS BIBLIOGRÁFICAS

- FORBELLONE, André Luiz Villar, EBERSPÄCHER, Henri Frederico. *Lógica de Programação: A Construção de Algoritmos e Estruturas de Dados*. São Paulo: Makron Books, 1993.
- TERADA, Routo, SETZER, Valdemar W. *Introdução à Computação e à Construção de Algoritmos*. São Paulo: Makron Books, 1992.
- WIRTH, Niklaus. *Algoritmos e Estrutura de Dados*. Rio de Janeiro: PHB, 1989.
- GUIMARÃES, Ângelo de Moura, LAGES, Newton Alberto de Castilho. *Algoritmos e Estruturas de Dados*. Rio de Janeiro: LTC, 1994.
- O'BRIEN, Stephen. *Turbo Pascal 6 – Completo e Total*. São Paulo: Makron Books, 1993.
- BARBOSA, Lisbete Madsen. *Pascal II*. São Paulo: McGraw Hill, 1990
- TREMBLAY, Jean-Paul, BUNT, Richard B. *Ciência dos Computadores: uma abordagem algorítmica*. Trad. Moacir de Souza Prado; ver. téc. João Pedro Perotti. São Paulo: McGraw-Hill, 1983.
- JAMSA, Kris. *Turbo Pascal 4: Guia de referência básica*. trad. Lars Erik Gustav Unonius. ver. tec. Flávio Silva Gianini. São Paulo: McGraw-Hill, 1988.
- TERADA, Routo. *Desenvolvimento de algoritmos e estruturas de dados*. São Paulo: McGraw-Hill, Makron, 1991.