Técnicas

de

Programação JJ

Osmar de Oliveira Braz Júnior - 1998.

Tubarão / Santa Catarina / BRAZIL
E-Mail : osmarjr@unisul.rct-sc.br
http://tec1.unisul.rct-sc.br/osmarjr/

Índice

1. Tipo Record

Os tipos de dados que são mais comumente usados, e que foram vistos com maior ênfase em programação I, são :

1. WORD
2. INTEGER
3. REAL
4. BYTE
5. STRING
6. CHAR
7. ARRAY
8. BOOLEAN

Uma outra forma de definir uma variável em Pascal, é através do tipo RECORD. Esse tipo é diferente das demais formas de definir variáveis, porque permite que uma variável armazene valores de diversos tipos diferentes.

Exemplo: Imagine que fosse desejado armazenar informações de uma pessoa, tais como: Nome, Idade, Altura, Sexo, Número de Dependentes, Profissão.
Na forma tradicional, seria necessário definir uma variável para cada tipo de informação, ou seja:

```
VAR
        Nome        : STRING;
        Idade       : BYTE;
        Altura      : REAL;
        Sexo        : CHAR;
        NumDep      : BYTE;
        Profissão   : STRING;
```

Utilizando o tipo RECORD, a definição seria a seguinte:

```
VAR
        Pessoa      : RECORD
                        Nome        : STRING;
                        Idade       : BYTE;
                        Altura      : REAL;
                        Sexo        : CHAR;
                        NumDep      : BYTE;
                        Profissão   : STRING;
                      END;
```

Ao definir uma variável como sendo do tipo RECORD, devemos definir, também quais serão as partes componentes desta variável(Nome, Idade, Altura , Sexo, NumDep e Profissão), junto com o seu tipo. Quando estamos trabalhando com RECORD, as partes componentes do mesmo recebem um Nome próprio, o qual é conhecido como "campo ". Desta forma, uma variável RECORD pode ter campos de qualquer tipo válido do Pascal, sendo permitido inclusive que um RECORD seja definido dentro do outro, ou como parte de um ARRAY.

Continuando o Exemplo, caso desejarmos atribuir um valor a variável Pessoa, devemos fazê-lo da seguinte forma:

```
Pessoa.idade : = 45
```

O uso do "." indica que esta variável possui campos, e que "Idade" é um deles. É importante lembrar que as operações realizadas sobre uma variável RECORD, são as mesmas de uma variável "comum", a única diferença que devemos indicar o Nome da variável, seguido de um ponto(.), seguido do Nome do campo.

É possível atribuir o conteúdo de uma variável RECORD para outra variável, de mesmo tipo, da mesma forma que é feito como as outras variáveis do Pascal.

Exemplo: Caso duas variáveis, digamos A e B sejam definidas como sendo RECORDs, e caso seja desejado passar o conteúdo, isto é os valores existentes nos campos, a variável A para a variável B, bastará realizar a seguinte atribuição:

A: =B

1.1.1 Exercícios:

1. Definir um RECORD tendo os seguintes campos: Nome, Semestre, Sala, Curso, Notas(total de seis)
2. Faça um programa para ler as informações, descritas acima
3. Ampliar a definição anterior, acrescentando a definição de um outro campo(Endereço) que será também um RECORD, o qual terá os seguintes campos: Rua, Bairro, Cidade, Estado, CEP
4. Faça um programa para ler as informações de um aluno, junto com o endereço descrito acima
5. Defina um ARRAY de alunos, os campos serão os mesmos descritos nos itens anteriores
6. Faça um programa para ler as informações de n alunos
7. Ordene crescentemente pelo Nome, os alunos

2. Tipos Definidos Pelo Usuário e Constantes

2.1 Tipos Definidos Pelo Usuário

O Pascal possui vários tipos pré-definidos, como INTEGER, WORD, REAL etc, mas além destes tipos básicos, existe a possibilidade de o usuário definir seus próprios tipos de dados. Para isto, é necessário o uso da palavra reservada TYPE, a qual indica que um novo tipo será criado.

Exemplo: Imagine que seja desejado criar um tipo matriz 4X4, sendo que logo em seguida este novo tipo será usado para definir uma variável como sendo deste tipo. P ara isto deverá ser usada a seguinte definição:

```
TYPE
        Matriz  = ARRAY[1..4,1..4] OF INTEGER
VAR
        Mat     : Matriz
```

O Pascal permite a definição de tipos usando qualquer um dos tipos pré-definidos, ou até mesmo utilizando tipos definidos pelo usuário

2.1.1 Exercícios:

1. Usando a definição para aluno apresentada no exercício da secção anterior, crie um tipo de dado para alunos e em seguida defina uma variável como sendo um ARRAY deste tipo. A título de ilustração, defina o RECORD do campo endereço, como sendo também um tipo
2. Defina um tipo de dado chamado funcionário, o qual deverá ter o seguinte layout:

```
Nome
Endereço:
        Rua
        Número
        Bairro
        Cidade
        Estado
        CEP
Profissão:
        Cargo:
                Departamento
                Função
        Salário:
                Bruto
                Desconto(percentual)
                Salário Família(Somente para filhos de 18 anos)
Dependentes:
        Número
        Descrição(Uma para cada dependente):
                Nome
                Idade
                Se é filho ou não
```

3. Dado a definição acima, faça um programa para:
   - Ler as informações de n funcionários
   - Ordenar crescentemente os nomes dos Funcionários

- Emitir um relatório com o salário líquido de cada funcionário, onde deverá ser impresso somente o Nome do funcionário e seu salário.
- Emitir um relatório dos funcionários que trabalhem na contabilidade e que tenham mais de dois filhos menores de 18 anos
- Emitir um relatório com o Nome do funcionário e o seu salário bruto

4. Uma empresa compra uma série de produtos de diversos fabricantes, e precisa que sejam emitidos os seguintes relatórios:

- Qual o produto que possui a maior quantidade em estoque, e qual o que tem a menor quantidade
- Qual o produto mais caro e o mais barato
- Quais são os produtos pertencentes ao fabricante XYZ
- Quais os produtos que são de cor Azul
- Listagens de todos os produtos em estoque com todas as informações existentes sobre cada um dos produtos.

2.2 Constantes

Uma constante é uma posição de memória que possui um valor fixo, constante, durante toda a existência do programa. A sua utilização possibilita uma maior clareza do código, tornando a tarefa de manutenção ou entendimento do programa muito mais simples.

Exemplo:

```
IF Tecla = CHR(24) THEN
BEGIN
        <executa comandos>;
END;
```

O pedaço de código mostrado acima seria mais legível se , ao invés da utilização da *Função* CHR(24), fosse utilizado uma constante. Desta forma , o programa alterado ficaria como é mostrado abaixo:

```
IF Tecla = SetaParaBaixo THEN
BEGIN
        <executa comandos>;
END;
```

A forma de se declarar uma constante é através do uso da palavra reservada CONST.

Exemplo: Declarar uma constante que representa o valor da seta para baixo, do teclado do PC.

```
CONST
        SetaParaBaixo=CHR(24)
BEGIN
        <Comandos>;
END
```

Um outro uso muito útil de constantes é o de definir o tamanho de um ARRAY e o escopo dos laços de repetição, como FOR DO, WHILE DO e REPEAT UNTIL.

```
PROGRAM Teste;
CONST
        TotalLinhas = 10;
        TotalColunas = 20;
TYPE
        matriz = ARRAY[ 1..totallinhas, 1 ..totalcolunas]  OF INTEGER;

VAR
        Mat      :  matriz;
        lin,col  : BYTE;
BEGIN
        FOR lin : = 1 TO totallinhas DO;
        BEGIN
                FOR col: = 1 TO totalcolunas DO
                BEGIN
                        READ(mat[lin,col]);
                END;
        END;
END.
```

2.2.1 Exercícios:

1. Faça um programa para definir constantes representando os códigos das teclas como HOME, END, ESC etc do teclado do PC.
2. Faça um programa para declarar constantes que representem as seqüências de caracteres necessários para programar uma impressora de modo a imprimir diversas qualidades como expandido, qualidade carta , condensado, etc. Para isto será necessário o uso do manual de sua impressora , na parte relacionada a programação de impressora.
3. Faça um programa para definir constantes que representem as diversas cores/tonalidades que o vídeo do PC possa operar em modo texto, tanto para cor de fundo(bakcground) , como para cor das letras(foreground).
4. Faça um PROGRAMA que defina constantes para construção de molduras. Uma moldura é uma área retangular cercada por caracteres específicos da tabela ASCII. As molduras podem ser por Exemplo, simples , duplas, sombreadas, etc. A tabela ASCII tem uma boa variedade de caracteres específicos para este fim, só depende da imaginação de cada um.

3. Sub-Rotinas

Um matemático uma vez disse que um grande problema se resolve dividindo-o em pequenas partes e resolvendo tais partes em separado. Estes dizeres servem também para a construção de programas. Os profissionais de informática quando necessitam construir um grande sistema, o fazem, dividindo tal programa em partes, sendo então desenvolvido cada parte em separado, mais tarde, tais partes serão acopladas para formar o sistema. Estas partes são conhecidas por vários nomes. Nós adotaremos uma destas nomenclaturas: sub-Rotinas.

Podemos dar um conceito simples de sub-Rotina dizendo ser um pedaço de código computacional que executa uma Função bem definida, sendo que esta sub-Rotina pode ser utilizadas várias vezes no programa.

Neste curso iremos tratar de dois tipos de sub-Rotinas: PROCEDURE e FUNCTION.

3.1 Procedure

Sintaxe:

```
PROCEDURE <Nome> [(parâmetros)]
        <definições>
BEGIN
        <comandos>;
END;
```

Uma "PROCEDURE", é um tipo de sub-Rotina que é ativada através da colocação de seu Nome em alguma parte do programa. Desta forma, assim que o Nome de uma "PROCEDURE"é encontrado, ocorre um desvio no programa, para que os comandos da sub-Rotina sejam executados. Ao término da sub-Rotina, a execução retornará ao ponto subsequente a chamada da "Pocedure".

Exemplo:

```
PROGRAM Teste;
VAR
        Número, N :  BYTE;
PROCEDURE  EscreveNoVideo;
BEGIN
        FOR Número : = 1 TO n DO
        BEGIN
                WRITE(Número);
        END;
END;

BEGIN
        READ(N);
        EscreveNoVideo;
        WRITE('fim');
END.
```

3.1.1 Exercícios:

1. Construa uma sub-Rotina para ler uma matriz NXM DO tipo INTEGER. Os valores N e M deverão ser lidos.
2. Faça uma sub-Rotina para ler um vetor A de N elementos, e um vetor B de M elementos. Os valores M e N deverão ser lidos.
3. Faça um programa para ler as informações de N alunos, tais como: Nome, idade e sexo. Após construa sub-Rotina para: a - Emitir um relatório ordenado crescentemente pelo Nome; b - Emitir um relatório ordenado decrescentemente pela idade; c - Informar qual o percentual de alunos do sexo feminino.
4. Faça uma PROCEDURE para desenhar uma moldura no vídeo.

3.2 Variáveis Globais e Locais

Damos o Nome de variáveis globais para aquelas variáveis que são definidas logo após o comando VAR do programa principal, sendo desta forma visíveis em qualquer parte do programa.

Exemplo:

```
PROGRAM Teste;
VAR
        Nome :  STRING[80];      (variável global)
PROCEDURE Setanome;
BEGIN
        READ(Nome);
END;

BEGIN
        Setanome;
        WRITE(Nome);
END
```

No Exemplo acima, a variável "Nome" , por ser definida como global, pode ser manipulada dentro de qualquer ponto do programa, sendo que qualquer mudança no seu conteúdo, será visível nas demais partes da Rotina.

Damos o Nome de variáveis locais às variáveis que são declaradas dentro de uma sub-Rotina, sendo que as mesmas só podem ser manipuladas dentro da sub-Rotina que as declarou, não sendo visíveis em nenhuma outra parte do programa.

Exemplo:

```
PROGRAM Teste;
PROCEDURE EscreveNoVídeo;
VAR
        Número, N  : INTEGER;
BEGIN
        READ(N);
        FOR número : = 1 TO N DO
        BEGIN
                WRITE(Número);
        END;
END;

BEGIN
        EscreveNoVídeo;
END;
```

Obs: É possível definir variáveis globais e locais com o mesmo Nome, sendo qualquer mudança no conteúdo da variável local não afetará o conteúdo da variável global.

Exemplo:

```
PROGRAM Teste;
VAR
        Nome :  STRING;
PROCEDURE Setanome;
BEGIN
        READ(Nome);
END;
PROCEDURE Mudança;
VAR
        Nome :  STRING;
BEGIN
        READ(Nome);
END;

BEGIN
        Setanome;
        WRITE(Nome);
```

```
        mudança;
        WRITE(Nome);
END;
```

No Exemplo acima, a variável global “Nome” e a variável local “Nome” representam posições de memória totalmente diferentes, logo, qualquer mudança no conteúdo da variável local, não afetará o conteúdo da variável global.

3.2.1 Exercícios:

1 - Faça uma “PROCEDURE” para calcular A elevado a um expoente B.
2 - Faça uma “PROCEDURE” para calcular o fatorial de um número X qualquer.
3 - Faça um PROGRAMA para calcular a seguinte expressão matemática:

$$Y = 1 + \frac{X^2}{2!} + \frac{2X^3}{3!} + \frac{3X^4}{4!} + \frac{4X^5}{5!} + \frac{5X^6}{6!} + \ldots + \frac{nX^{(n+1)}}{(n+1)!}$$

4 - Faça uma PROCEDURE que informe se uma STRING qualquer é palíndrome.
5 - Faça um PROGRAMA que leia um vetor de números inteiros. Após, emita um relatório com cada número diferente, e o número de vezes que o mesmo apareceu repetido no vetor.
6 - Faça um PROGRAMA para:

- Ler as informações de n pessoas : Nome, Idade, sexo, altura, peso e endereço(Rua, Número, Bairro, Cidade, Estado), armazenando-as em um vetor. O valor n deverá ser lido.
- Alterar o vetor de tal forma que na parte superior, sejam colocados, em ordem crescente, as pessoas cujas idades sejam pares e na parte inferior, sejam colocadas, em ordem decrescente, as pessoas cujas idades sejam ímpares.

Obs: O PROGRAMA deve prever a possibilidade de no vetor , não existirem números pares ou então, não existirem números ímpares.

4. Passagem de Parâmetros

Até agora vimos que para ativar uma sub-Rotina bastaria colocar o seu Nome em alguma parte do programa. Mas isto nem sempre significa que o trabalho de escrever o programa irá diminuir. Com o que vimos até agora , dependendo da tarefa a ser realizada pela sub-Rotina, o trabalho de um programador pode até ser bem complicado. Por Exemplo, como faríamos para ler 5 vetores, todos com tamanhos diferentes? Poderíamos , por Exemplo, criar 5 sub-Rotinas, uma para cada vetor a ser lido. Isto sem dúvida resolveria esta situação, mas, e se fossem 100 vetores?, ou 1000? Seria realmente uma tarefa muito trabalhosa ter de escrever 100, ou 1000 sub-Rotinas, isto só para ler os vetores, imagine se tivéssemos também que ordená-los, ou realizar outro processo qualquer. Com toda esta dificuldade, o uso das sub-Rotinas deveria ser considerado. Como já foi dito, as sub-Rotinas foram criadas para serem genéricas o bastante para se adaptarem a qualquer situação, visando justamente a possibilidade de reutilização do código. Para realizar esta "mágica", foi criado o conceito de passagem de parâmetros, ou seja, passar informações para serem tratadas dentro da Sub-Rotina.

Sintaxe:

```
PROCEDURE <Nome> (<Variável> :  <Tipo>);
        <Definições>;
BEGIN
        <comandos>;
END;
```

Obs: Variável do mesmo tipo são separadas por vírgulas (,). Variáveis de tipos diferentes, são separadas por ponto e vírgula (;).

Exemplo:

```
PROGRAM Teste;
VAR
        Número        : INTEGER;
        Funcionário   : STRING;
PROCEDURE EscreveNome(N : INTEGER; Nome : STRING);
VAR
        I : INTEGER;
BEGIN
        FOR i : = 1 TO n DO
        BEGIN
                WRITE(Nome);
        END;
END;

BEGIN
        READ(Número, Funcionário);
        EscreveNome(Número, Funcionário);
END.
```

Obs: Os números dados aos parâmetros não necessitam serem iguais as variáveis passadas para sub-Rotina. No Exemplo acima, o valor contida em "Número" será passado para o parâmetro "N", da mesma forma que o valor contido na variável "Funcionário" será passada para o parâmetro "Nome". Note que os nomes são diferentes.

4.1.1 Exercícios:

1 - Faça um PROGRAMA para calcular N!
2 - Faça um PROGRAMA para calcular A^b
3 - Faça um PROGRAMA para calcular a seguinte expressão até o n-ésimo:

$$Y = X - X^2 + X^3 - X^4 + X^5 - \ldots$$

a) Passagem de Parâmetros por Valor

Qualquer alteração no conteúdo de um parâmetro, dentro de uma sub-Rotina, não será refletido no programa chamado.

Exemplo:

```
PROGRAM  Teste;
VAR
        X       : INTEGER;
PROCEDURE PorValor(A :  INTEGER);
BEGIN
        A : = 5;
END;

BEGIN
        X : = 10;
        PorValor(X);
        WRITE(X);
END.
```

No Exemplo acima, o conteúdo da variável "X" não será alterado após o retorno ao programa principal.

b) Passagem do Parâmetros por Referência

Quando a alteração no conteúdo de um parâmetro, dentro de uma sub-Rotina, se reflete no programa chamador. Os parâmetros a serem passados por referência deverão ter, na definição da sub-Rotina, colocado na frente do Nome do parâmetro, a palavra "VAR".

Exemplo:

```
PROGRAM Teste;
VAR
        X :  INTEGER;
PROCEDURE PorReferência(VAR A:  INTEGER);
BEGIN
        A : = 5;
END;

BEGIN
        X : = 10;
        PorReferência(X);
        WRITE(X);
END.
```

No Exemplo acima , o conteúdo da variável "X" será alterado após o retorno ao programa principal

c) O problema dos tipos na definição de parâmetros

O Pascal, a princípio, aceita somente que sejam definidos parâmetros com os seguintes tipos : INTEGER, REAL, BYTE, WORD, BOOLEAN, CHAR, STRING e os outros tipos ditos simples . Desta forma, tipos como ARRAY, RECORD e STRING com tamanho definido pelo usuário, não são aceitos. Acontece que existe uma forma de fazer o Pascal aceitar qualquer tipo de dados na definição de parâmetros, através da definição de tipos pelo usuário, ou seja, criar tipos através do comando TYPE.

Exemplo:

```
PROGRAM Teste;
CONST
        Máximo = 50
```

```
TYPE
        Vetor = ARRAY[1..Máximo] OF INTEGER;
        Registro =RECORD
                                descrição       : STRING;
                                cor             : STRING;
                                Quant           : BYTE;
                        END;
VAR
        Vet :  vetor;
        Reg :  registro;

PROCEDURE LeInfo(VAR V:  Vetor; VAR R :  Registro);
VAR
        i :  INTEGER;
BEGIN
        WRITE('Digite os elementos DO vetor: ');
        FOR i : = 1 TO máximo DO
                READ(v[i]);
        WRITE('Digite os elementos DO Registro');
        READ(r.descrição, r.cor, r.quant);
END;

BEGIN
        LeInfo(Vetor, Reg);
END.
```

4.1.2 Exercícios:

1. Faça um Programa para ler 5 vetores do tipo REAL, todos com tamanhos diferentes.
2. Faça um programa para :
    - Ler um vetor A com N elementos e um vetor B com M elementos(os valores N e M podem ou não serem iguais).
    - Formar um terceiro vetor (C) com os elementos dos vetores A e B intercalados.

Exemplo:

```
C[1] : = A[1];
C[2] : = B[1];
C[3] : = A[2];
C[4] : = B[2];
```

Obs.: Nenhum tipo de ARRAY poderá ser utilizado além dos ARRAY's A, B e C.

3. Faça um programa para :
    - Ler um vetor A com N elementos e um vetor B com M elementos(os valores M e N podem ou não serem iguais).
    - Ordenar crescentemente estes vetores
    - Formar um terceiro vetor (C) , com os elementos dos vetores A e B intercalados, de forma que ao final do processamento (intercalação), o vetor C continue ordenado. A ordenação será obtida somente através do processo de intercalação.

Obs: Nenhum outro tipo de ARRAY poderá ser utilizados além dos tipos A, B e C. Caso os elementos de um dos vetores(A ou B) termine um antes do outro, as posições restantes do vetor C, deverão ser preenchidas com os elementos restantes do Vetor (A ou B) que ainda possui elementos.

4. Simule um arquivo de clientes na memória e crie um pequeno sistema para envio de mala direta. O sistema deverá ter as seguintes funções:
    - Inclusão, alteração e exclusão dos clientes
    - Listagem dos clientes em ordem alfabéticas, dentro de um intervalo de letras especificado (A..Z)
    - Listagem dos Clientes por código, dentro de um intervalo especificado (código inicial..código final)

5. Function

Sintaxe:

```
FUNCTION <Nome> [(Parâmetros)] :  < Tipo DO valor retornado>;
        <Definições>;
BEGIN
        <Comandos>;
END;
```

Uma sub-Rotina do tipo "FUNCTION" possui as mesmas características de uma "PROCEDURE" no que se refere a passagem de parâmetros, variáveis globais e locais, mas possui uma importante diferença, que é o retorno de um valor ao término de sua execução, ou seja, uma FUNCTION sempre deverá retornar um valor ao chamador.

Na definição de uma "FUNCTION" , deverá ser informado qual o tipo do valor retornado, sendo que poderá ser usado, nesta definição, tanto tipos pré-definidos da linguagem, como tipos definidos pelo usuário. Somente não poderão ser retornados tipos ARRAY e RECORD, justamente por serem tipos que definem variáveis que armazenam mais de um valor.

Para informar qual o valor deve ser retornado deve ser colocado, em algum ponto do código da "FUNCTION" uma linha com a seguinte Sintaxe:

<Nome da FUNCTION > : = < o valor a ser retornado>

Exemplo:

```
PROGRAM Teste;
VAR
        K          : BYTE;
FUNCTION Soma(V1, V2  : BYTE) : BYTE;
BEGIN
        Soma : = V1 + V2;
END;

BEGIN
        K : = Soma(2,3);
        WRITE(K);
END.
```

5.1.1 Exercícios:

1. Construa "Functions" para :
   a) Calcular N!
   b) Calcular A^B
   c) Calcular:

$$\sum_{n-0}^{50} \frac{1}{n!}$$

   d) Calcular:

$$\sum_{n-0}^{50} \frac{1}{2^n}$$

   f) Retornar TRUE caso um número seja par, FALSE caso contrário
   g) Retornar TRUE caso um número seja ímpar, FALSE caso contrário
2. Faça uma FUNCTION que codifique uma mensagem, da seguinte forma:
   A por Z
   B por Y

C por X
.
.
X por C
Y por B
Z por A

Obs.: a Rotina deverá fazer o mesmo para letras minúsculas.

3. Faça uma FUNCTION para transformar as letras de uma STRING de minúsculas para maiúsculas
4. Faça uma FUNCTION para transformar as letras de uma STRING de maiúsculas para minúsculas
5. Dado um vetor com n elementos numéricos, faça uma FUNCTION que verifique se um dado valor existe neste vetor
6. Faça uma FUNCTION para acrescentar N espaços em branco a esquerda de uma STRING qualquer
7. Faça uma FUNCTION para acrescentar N espaços em branco a direita de uma STRING qualquer
8. Dado uma STRING qualquer e um valor N , faça uma FUNCTION para gerar uma nova STRING que tenha este tamanho N . Caso a STRING original possua um tamanho menor que o valor N informado, deverão ser acrescentados espaços em branco a esquerda da STRING, até que o tamanho N seja alcançado.
9. Dado uma STRING qualquer e um valor N , faça uma FUNCTION para gerar uma nova STRING que tenha este tamanho N. Caso a STRING original possua um tamanho menor que o valor N informado, deverão ser acrescentados espaços em branco a direita da STRING, até que o tamanho N seja alcançado.

6. Recursividade

Diz-se que uma FUNCTION ou uma PROCEDURE é recursiva, quando ela chama a si própria, esta característica pode , a princípio parecer estranha, ou até mesmo desnecessária devido ao nível de programas o qual estamos trabalhando, mas o uso da recursividade muitas vezes , é a única forma de resolver problemas complexos. No nível que será dado este curso, bastará saber o conceito e o funcionamento de uma sub-Rotina recursiva.

Abaixo seguem exemplos de sub-Rotinas recursivas:

a)
```
PROCEDURE Recursão(A :  BYTE);
BEGIN
        IF a > 0 THEN
        BEGIN
                WRITE(A);
                Recursão (A - 1);
        END;
END;
```

b)
```
PROCEDURE Recursão( A :  BYTE);
BEGIN
        IF a > 0 THEN
        BEGIN
                Recursão ( A -1 );
                WRITE( A ); { Esta linha será executada ao final de cada execução da
                                Rotina recursiva }
        END;
END;
```

No primeiro Exemplo, a saída gerada será a seguinte seqüência de números: 5 4 3 2 1.
No segundo Exemplo, a saída gerada será a seguinte seqüência de números: 1 2 3 4 5 .

c)
```
PROCEDURE Recursão(A :  BYTE) ;
VAR
        Valor :  BYTE;
BEGIN
        Valor : = A DIV 2;
        IF valor > 0 THEN
        BEGIN
                Recursão(Valor);
        END;
        WRITE(valor);
END;
```

Para um valor inicial igual a 80, a seqüência gerada será a seguinte: 0 1 2 5 10 20 40

No Exemplo acima será criado, a chamada da Rotina Recursão, uma variável diferente de Nome “Valor”, a qual assumirá valores diferentes, dependendo do valor do parâmetro “A”.

Uma característica importante das Rotinas recursivas diz respeito a forma de tratamento das variáveis e parâmetros. Usando como Exemplo o item acima, vemos que existe um parâmetro chamado “A” e uma variável local a sub-Rotina, chamado Valor. É importante notar que a cada ativação da Rotina recursiva, todos os parâmetros e variáveis locais, são tratados como sendo posições de memória totalmente diferentes e independentes, apesar de terem o mesmo Nome.

Segue abaixo uma representação das variáveis e seus conteúdos em cada uma das chamadas:

1ª Chamada	A=80	Valor=40
2ª Chamada	A=40	Valor=20
3ª Chamada	A=20	Valor=10
4ª Chamada	A=10	Valor=5
5ª Chamada	A=5	Valor=2
6ª Chamada	A=2	Valor=1
7ª Chamada	A=1	Valor=0

6.1.1 Exercícios:

Explique qual será o resultado e o funcionamento dos seguintes programas:

a)-
```
PROGRAM Teste;
FUNCTION XXX(A :  WORD) :  WORD;
BEGIN
        IF a = 0 THEN
        BEGIN
                XXX :  = 1;
        ELSE
                XXX : =A * XXX(A - 1);
        END;
END;
BEGIN
        WRITE(XXX(5));
END.
```

b)
```
PROGRAM Teste;
        PROCEDURE Recursão(a :  BYTE);
BEGIN
        a : =  a - 1;
        IF a  > 0 THEN
        BEGIN
                Recursão(a);
        END;
        WRITE(a);
END;
BEGIN
        Recursão(5);
END.
```

c)

```
PROGRAM Teste;
PROCEDURE Recursão(VAR a: BYTE);
BEGIN
        a : = a - 1;
        IF a > 0 THEN
        BEGIN
                Recursão(a);
        END;
        WRITE(a);
END;
BEGIN
        Recursão(5);
END.
```

7. Unit

As sub-Rotinas foram criadas para facilitar a construção de programas pois eliminam a necessidade de duplicação de código, uma vez que blocos de comandos usados repetidas vezes podem ser transformados em sub-Rotinas. Este conceito se aplica muito bem para apenas um algoritmo / Programa, mas imagine que você necessite elaborar dois sistemas: Um para cadastro de clientes de uma loja qualquer e outro para o cadastro de alunos de um colégio. Os dois sistemas serão totalmente diferentes na Função que realizam, mas poderão ter sub-Rotinas idênticas (genéricas) , por Exemplo sub-Rotina que manipulem a tela, sub-Rotinas para programar a impressora, sub-Rotinas para armazenar/ recuperar informações no disco, sub-Rotinas para gerenciar memória do computador etc. Pelo conhecimento visto até agora, quando da construção destes sistemas, ou outros no futuro, seria necessário repetir a digitação destas mesmas sub-Rotinas tantas vezes quantos forem os sistemas a serem construídos.

Dentro de um programa através do uso de sub-Rotinas podemos compartilhar blocos de comandos, o que facilitou muito a construção de um sistema, mas quando se trata de elaborar vários sistemas o uso de sub-Rotinas não é o bastante, pois precisam também compartilhar sub-Rotinas genéricas entre sistemas diferentes. Pensando nisto, foi criado um novo conceito de programação, onde podemos construir um tipo especial de programa onde são definidos não apenas sub-Rotinas, mas também variáveis, constantes e tipos de dados que podem ser usados não apenas por um programa, mas sim por diversos programas diferentes. A este Exemplo de programação deu-se o Nome de programação modular e a este programa especial deu-se o Nome de módulo.

O Pascal dá a este modo o Nome de UNIT e a Sintaxe para a sua construção é a seguinte:

```
UNIT <Nome da Unit>;
INTERFACE
        USES <lista de UNITs importadas>
        < definição de variáveis, constantes e tipos exportados>
        <cabeçalho das sub-Rotinas exportadas>
IMPLEMENTATION
        USES <lista  de UNITs importadas privativas ao módulo>
        <definição de variáveis, constantes e tipos internos a UNIT>
        <sub-Rotinas internas a UNITs>
        <corpo das sub-Rotinas exportadas>
BEGIN
        <Comandos a serem executados na ativação da Unit>
END.
```

Obs :

- A seção conhecida por INTERFACE define todas as sub-Rotinas, variáveis, constantes e tipos de dados que são exportados, ou sejam, são visíveis em outros programas.
- Os tipos de dados, variáveis e constantes definidos n seção de IMPLEMENTATION serão visíveis somente dentro da UNIT , não sendo portanto exportados.
- As sub-Rotinas definidas na seção de IMPLEMENTATION e que não tenham o seu cabeçalho definido na seção de INTERFACE serão internas a UNIT, não sendo desta forma exportadas.
- Para usar as sub-Rotinas, variáveis, constantes e tipos de dados definidos em outras UNITs basta utilizar a palavra reservada USES seguido da relação de nomes de UNITs desejada.

Exemplo: Construir uma UNIT que contenha uma sub-Rotina para escrever uma STRING qualquer em uma determinada linha e coluna na tela do computador.

```
UNIT tela;
INTERFACE
        PROCEDURE Escreve_Str( linha, coluna : BYTE; Texto :  STRING);
IMPLEMENTATION
USES CRT;
        PROCEDURE Escreve_Str( linha, coluna :  BYTE;  texto :  STRING);
        BEGIN
                GOTOXY(coluna, linha);
                WRITE(texto);
        END;
END.
```

Como complementação do Exemplo vamos construir um pequeno programa que use a sub-*Rotina* definida acima:

```
PROGRAM Testa_Unit;
USES Tela;
BEGIN
        Escreve_Str(10, 10, 'Teste de Unit');
END;
```

7.1.1 Exercícios:

1 - Construi UNITs para:

- Definir constantes com os códigos das cores/tonalidades possíveis em uma tela tipo texto
- Definir constantes com os códigos das teclas especiais como PgUp, PgDn, Setas, Esc etc.
- Definir constantes com os códigos de programação dos caracteres de uma impressora
- Definir constantes com os caracteres da tabela ASCII necessários para criação de moldura

2 - Monte uma UNIT que contenha Rotinas para gerenciamento de tela tipo texto com montagens de molduras, efeitos de sombreamento de janelas, subRotinas que permitam salvar/restaurar a tela tipo texto, sub-Rotinas para pintar uma região da tela (X1, Y1, X2, Y2) com uma determinada cor/tonalidade etc.

3 - Monte uma UNIT que contenha sub-Rotinas para montagem de menus de barra horizontais, verticais e matriciais, sub-Rotinas que permitam realizar a edição de campos, podendo ser definido o tamanho do campo a ser editado, usando teclas tipo setas, Esc, Backspace, Insert/ OverWrite, sub-Rotinas que façam a centralização de STRING's dentro de determinadas coordenadas(X1,X2,Y1,Y2) do vídeo etc.

8. Arquivos

Um arquivo é de suma importância nos programas computacionais, desde o tempo em que o primeiro computador surgiu, pois, para que um programa faça algum tipo de operação , o mesmo precisa ser alimentado com informações: estas , ou são fornecidas pelo teclado, o que atualmente torna-se inviável, ou são fornecidos através de um arquivo.

O PASCAL, possui dois tipos de arquivos, os quais são:

1. Arquivos FILE
2. Arquivos TEXT

8.1 Arquivos FILE

Um arquivo do tipo FILE , também conhecido por arquivo randômico, ou de acesso aleatório, é o arquivo mais importante do Pascal, sendo desta forma também o mais utilizado. Um arquivo randômico é caracterizado pelo fato de ser possível buscar uma determinada informação em qualquer posição que a mesma se encontre, sem haver a necessidade de se percorrer todo o arquivo até se alcançar a informação desejada. O acesso a informação é direto.
Sintaxe :

<Nome da variável> : FILE OF<tipo>

Observação: Um arquivo FILE deve ser apenas um tipo de dado, ou seja : INTEGER, REAL, RECORD, STRING, BYTE, etc.
Exemplo: Crie um programa que defina uma variável como sendo um arquivo FILE de STRING's, crie também neste mesmo programa um tipo Arquivo de INTEGERs.

```
PROGRAM Exemplo;
TYPE
        Meu_tipo = FILE OF INTEGER;
VAR
        Minha_Variável = FILE OF STRING;
BEGIN
END.
```

Estrutura Interna do Arquivo:

Quando um arquivo FILE é criado, o mesmo possui a seguinte estrutura:

Posição Física	Informação
0	
1	
2	
…	
n	

A posição física corresponde a um número que é gerado automaticamente no instante que uma informação qualquer é incluída no arquivo. Este número, corresponde ao "Endereço" da informação no arquivo, sendo que é através deste Endereço que é possível recuperar qualquer informação, sem precisar percorrer todo o arquivo em busca da mesma, ao invés disto basta fornecer o número da posição física da informação no arquivo.

Observação: Passaremos daqui por diante a chamar as informações armazenadas em um arquivo de "Registros".

Sub-Rotinas para Tratamento de Arquivos FILES

Existem uma grande quantidade de sub-Rotinas construídas especialmente para manipular arquivos FILE. Iremos neste curso mostrar as principais.

Rotina : ASSIGN()
Função : Serve para associar um determinado Nome de arquivo, no disco ou disquete com o arquivo definido pelo programador.
Sintaxe : ASSIGN(Meu_Arquivo, STRING_Com_Nome_Arquivo_DOS).
Exemplo:

```
PROGRAM TESTE
TYPE
        Registro = RECORD
                        Nome :  STRING;
                        Idade :  BYTE;
                   END;
VAR
        Arquivo  :  FILE OF Registro;
BEGIN
        ASSIGN (Arquivo, 'dados.dat');
END.
```

Rotina : REWRITE()
Função : Cria e abre para E\S um arquivo. Caso o arquivo não exista, o mesmo será criado. Caso o arquivo já exista, todos os dados existentes nele serão apagados.
Sintaxe : REWRITE(Meu_Arquivo);
Exemplo:

```
PROGRAM Teste;
TYPE
        Registro = RECORD
                        Nome :  STRING;
                        Idade :  BYTE;
                   END;
VAR
        Arquivo :  FILE OF Registro;
BEGIN
        ASSIGN (Arquivo, 'Dados.Dat');
        REWRITE (Arquivo);
END.
```

Rotina : RESET()
Função : Abre para E/S um arquivo que já exista. Caso o arquivo não exista ocorrerá um erro de execução e o programa será abortado.
Sintaxe : RESET(Meu_Arquivo)
Exemplo:

```
PROGRAM Teste;
TYPE
        Registro = RECORD
                        Nome :  STRING;
                        Idade :  BYTE;
                    END;
VAR
        Arquivo :  FILE OF Registro;
BEGIN
        ASSIGN (Arquivo, 'Dados.Dat');
        RESET (Arquivo);
END.
```

Rotina : CLOSE()
Função : Fecha um arquivo que tenha sido aberto com RESET\REWRITE.
Sintaxe : CLOSE(Meu_Arquivo)
Exemplo:

```
PROGRAM Teste;
TYPE
        Registro = RECORD
                        Nome :  STRING;
                        Idade :  BYTE;
                    END;
VAR
        Arquivo :  FILE OF Registro;
BEGIN
        ASSIGN (Arquivo, 'Dados.Dat');
        REWRITE (Arquivo);
        CLOSE (Arquivo);
END.
```

Rotina : WRITE()
Função : A Rotina WRITE tem a mesma Função de saída de informações como até agora já tínhamos trabalhado, somente que ao invés da informação ser apresentada no vídeo, a mesma será armazenada em um arquivo.
Sintaxe : WRITE (Meu_Arquivo, Registro)
Exemplo:

```
PROGRAM Teste;
TYPE
        Registro = RECORD
                        Nome :  STRING;
                        Idade :  BYTE;
                   END;
VAR
        Arquivo :  FILE OF Registro;
        Reg     :  Registro;
BEGIN
        ASSIGN (Arquivo,  'Dados.Dat');
        REWRITE (Arquivo);
        WRITE ('Digite o Nome:  ');
        READ (Reg.Nome);
        WRITE ('Digite a Idade:  ');
        READ (Reg.Idade);
        WRITE (Arquivo, Reg);
        CLOSE (Arquivo);
END.
```

Rotina : READ()
Função : A Rotina READ tem a mesma Função de entrada de informações como até agora já tínhamos trabalhado, somente que ao invés da leitura ser feita pelo teclado, a mesma será feita de um arquivo.
Sintaxe : READ (Meu_Arquivo, Registro)
Exemplo:

```
PROGRAM Teste;
TYPE
        Registro = RECORD
                        Nome :  STRING;
                        Idade :  BYTE;
                   END;
VAR
        Arquivo :  FILE OF Registro;
        Reg     :  Registro;
BEGIN
        ASSIGN (Arquivo, 'Dados.Dat');
        RESET (Arquivo);
        READ (Arquivo);
        WRITE ('Nome = ', Reg.Nome);
        WRITE ('Idade = ', Reg.Idade);
        CLOSE (Arquivo);
END.
```

Observação: Após cada operação READ/WRITE no arquivo, o endereço do registro corrente no arquivo é incrementado em uma unidade. Assim por Exemplo, se o endereço do registro corrente é igual a 10, após uma operação de READ/WRITE, o registro corrente passará a ser o número 11.

Rotina : FILEPOS()
Função : Retorna um número inteiro indicando qual o registro corrente em um arquivo.
Sintaxe : Registro_Corrente : = FILEPOS (Meu_Arquivo)
Exemplo:

```
PROGRAM Teste;
TYPE
        Registro = RECORD
                        Nome :  STRING;
                        Idade :  BYTE;
                    END;
VAR
        Arquivo :  FILE OF Registro;
        Corrente:  INTEGER;
BEGIN
        ASSIGN (Arquivo, 'Dados.Dat');
        RESET (Arquivo);
        corrente : = FILEPOS(Arquivo);
        WRITE  (corrente);
        CLOSE (Arquivo);
END.
```

Rotina : FILESIZE()
Função : Retorna quantos registro existem armazenados no arquivo.
Sintaxe : Tamanho_Arquivo : = FILESIZE (Meu_Arquivo)
Exemplo:

```
PROGRAM Teste;
TYPE
        Registro = RECORD
                        Nome :  STRING;
                        Idade :  BYTE;
                    END;
VAR
        Arquivo :  FILE OF Registro;
        Total     :  INTEGER;
BEGIN
        ASSIGN (Arquivo, 'Dados.Dat');
        RESET (Arquivo);
        Total : = FILESIZE (Arquivo);
        WRITE  (Total);
        CLOSE (Arquivo);
END.
```

Rotina : SEEK ()
Função : Posiciona o ponteiro do arquivo em um registro determinado, para que o mesmo possa ser processado.
Sintaxe : SEEK(Meu_Arquivo, Endereço_Registro)
Exemplo:

```
PROGRAM Teste;
TYPE
        Registro = RECORD
                        Nome :  STRING;
                        Idade :  BYTE;
                   END;
VAR
        Arquivo :  FILE OF Registro;
        Reg     :  Registro;
BEGIN
        ASSIGN (Arquivo, 'Dados.Dat');
        RESET (Arquivo);
        SEEK (Arquivo, 10);
        READ (Arquivo, Reg);
        WRITE ('Nome = ', Reg.Nome);
        WRITE ('Idade = ', Reg.Idade);
        CLOSE (Arquivo);
END.
```

Rotina : EOF()
Função : Retorna TRUE caso se alcance o final do arquivo, FALSE caso contrário.
Sintaxe : Chegou_Final : = EOF (Meu_Arquivo)
Exemplo:

```
PROGRAM Teste;
TYPE
        Registro = RECORD
                        Nome :  STRING;
                        Idade :  BYTE;
                   END;
VAR
        Arquivo :  FILE OF Registro;
        Reg     :  Registro;
BEGIN
        ASSIGN (Arquivo, 'Dados.Dat');
        RESET (Arquivo);
        WHILE NOT EOF(Arquivo) DO
                BEGIN
                        READ (Arquivo, Reg);
                        WRITE ('Nome = ', Reg.Nome);
                        WRITE ('Idade = ', Reg.Idade);
                END;
        CLOSE (Arquivo);
END.
```

Rotina : ERASE ()
Função : Elimina o arquivo do disco. É importante notar que o arquivo a ser eliminado não pode estar aberto.
Sintaxe : ERASE (Meu_Arquivo)
Exemplo:

```
PROGRAM Teste;
TYPE
        Registro = RECORD
                        Nome    : STRING;
                        Idade   : BYTE;
                   END;
VAR
        Arquivo :  FILE OF Registro;
BEGIN
        ASSIGN (Arquivo, 'Dados.Dat');
        ERASE(Arquivo);
END.
```

8.1.1 Exercícios:

1. Crie um arquivo FILE formado somente por números inteiro de 1 à 10.000.
2. Faça a leitura do arquivo e escreva no vídeo somente os número ímpares.
3. Crie um arquivo com Nome e salário de n funcionários e armazene estas informações em um arquivo FILE.
4. Faça a leitura do arquivo anterior e escreva no vídeo o Nome e o salário dos funcionários que ganham mais de US$ 1.000,00.
5. Abra o arquivo anterior e escreva o seu conteúdo de trás para frente, ou seja, do último registro até o primeiro. Utilize os comandos SEEK, FILEPOS e FILESIZE.
6. Abra o arquivo anterior e altere o salário de10 funcionários para US$ 1.050,00.
7. Abra o arquivo anterior e aumente em 15% o salário de todos os funcionários que ganham menos de US$1.000,00.
8. Crie um arquivo FILE com a seguinte informação: Nome do produto. O código do produto será o próprio número físico do registro. Após a criação cadastre n produtos.
9. Crie um arquivo FILE com a seguinte informação: Nome do Fornecedor. O código do fornecedor será o próprio endereço físico do registro. Após a criação cadastre n fornecedores.
10. Crie um arquivo FILE com a seguintes informações: Código do Fornecedor, Código do Produto. Cadastre as informações.
11. Usando os arquivos criados nos itens 8, 9 e 10 imprima um relatório com o Nome dos Fornecedores e os nomes dos produtos que cada um fornece.
12. Imprima um relatório, usando os arquivos dos itens 8,9 e 10 com o seguinte layout.

FORNECEDOR	PRODUTOS
João da Silva	Geladeira
	Fogão
	Televisão
	MicroOndas
	Vídeo Cassete
Pedro de Alcântara	Sabão em Pó
	Detergente
	Sabão em barra
	Etc…

13. Construa uma Rotina que ordene alfabeticamente pelo Nome, o arquivo de fornecedores criado no item 9.
14. Apague todas as informações do arquivo criado no item 10. O arquivo deverá continuar existindo no disco.
15. Apague do disco o arquivo criado no item 10.
16 - Usando o arquivo de fornecedores, elimine(apague) os registros cujas posições físicas são de número par.
17 - Crie um arquivo de peças, com o seguinte Lay-Out: Nome de Peça, cor , quantidade, tamanho e deletado. O campo "Deletado" será um campo Boolean, setado inicialmente para FALSE , informando se o registro está ou não deletado do arquivo.
18 - Faça a uma Rotina para deletar um, ou mais, registros do arquivo de peças. A deleção consiste em setar o campo "deletado" do arquivo para TRUE.
19 - Percorrer o arquivo de peças imprimindo somente as peças que não foram deletadas
20 - Faça uma Rotina que elimine fisicamente os registros do arquivo de peças que foram marcadas para deleção, isto é , onde o campo "deletado" está setado para TRUE.

8.2 Arquivos TEXT

Um arquivo do tipo TEXT, também conhecido por arquivo seqüencial, é um tipo especial de arquivo que, ao contrário do arquivo FILE, pode ser editado normalmente através de um editor de textos qualquer. Ele é dito seqüencial porque a leitura tem que ser feita seqüencialmente do início ao fim do arquivo, não podendo desta forma, como é feito no arquivo FILE através do comando SEEK, posicionar de forma direta, o ponteira o ponteiro de arquivo em um registro em particular.

Sintaxe:

<Nome da variável> : TEXT

Exemplo: Crie um programa que defina uma variável como sendo um arquivo TEXT e um tipo de dado que represente um arquivo do tipo TEXT.

```
PROGRAM Exemplo;
TYPE
        Menu_Tipo : TEXT;
VAR
        Minha_Variável : TEXT;
BEGIN
End.
```

Nos arquivos do tipo TEXT, todas as informações lá armazenadas são texto (STRING's), mesmo assim, é possível escrever no arquivo informações de qualquer tipo de dado simples (INTEGER, REAL, STRING, BYTE, etc) as quais , ao serem fisicamente armazenadas no arquivo, serão automaticamente convertidas do seu tipo original para o tipo STRING. A leitura se processa de forma inversa, ou seja, quando é lida uma informação em um arquivo TEXT, a mesma será automaticamente convertida para o tipo da variável que irá armazenar a informação, isto é , do tipo STRING para o tipo da variável receptora da informação lida.

8.3 Sub-Rotinas para Tratamento de Arquivos TEXT.

Existem uma grande quantidades de Sub-Rotinas construídas especialmente para manipular arquivos TEXT, algumas das quais já foram vistas. Iremos neste curso mostrar as principais.

Rotina : ASSIGN()
Função : Serve para associar um determinado Nome de arquivo, no disco ou disquete com o arquivo definido pelo programador.
Sintaxe : ASSIGN(Meu_Arquivo, STRING_Com_Nome_do_Arquivo_DOS)
Exemplo:

```
PROGRAM Teste;
VAR
        Arquivo : TEXT;
BEGIN
        ASSIGN(Arquivo, 'Dados.Dat');
END.
```

Rotina : REWRITE()
Função : Crie e Abra um arquivo no formato Write-Only(somente para escrita). Caso o arquivo não exista, este será criado. Caso já exista, todos os dados existentes nele serão apagados.
Sintaxe : REWRITE(Meu_Arquivo)
Exemplo:

```
PROGRAM Teste;
VAR
        Arquivo : TEXT;
BEGIN
        ASSIGN(Arquivo , 'Dados.Dat');
        REWRITE(Arquivo);
END;
```

Rotina : RESET()
Função : Abre um arquivo que já exista, mas no formato Read-Only(somente para leitura). Caso o arquivo não exista ocorrerá um erro de execução e o programa será abortado.
Sintaxe : RESET(Meu_Arquivo)
Exemplo:

```
PROGRAM Teste;
VAR
        Arquivo :  TEXT;
INICIO
        ASSIGN(Arquivo, 'Dados.Dat');
        RESET( Arquivo);
END;
```

Rotina : APPEND()
Função : Abre um arquivo para inclusão de novas informações do tipo Write-Only (somente para escrita). Caso o arquivo não exista ocorrerá um erro de execução e o programa será abortado. É importante notar que as inclusões se processam sempre no final do arquivo.
Sintaxe : APPEND(Meu_Arquivo);
Exemplo:

```
PROGRAM Teste;
VAR
        Arquivo : Text;
BEGIN
        ASSIGN(Arquivo, 'Dados.Dat');
        APPEND(Arquivo);
END.
```

Rotina : CLOSE()
Função : Fecha um arquivo que tenha sido aberto com Reset\Rewrite\Append
Sintaxe : CLOSE(Meu_Arquivo)
Exemplo:

```
PROGRAM Teste;
VAR
        Arquivo : TEXT;
BEGIN
        ASSIGN(Arquivo, 'Dados.Dat');
        REWRITE(Arquivo);
        CLOSE(Arquivo);
END;
```

Rotina : WRITE() ou WRITELN()
Função : A Rotina WRITE ou WRITELN tem a mesma Função de saída de informações como até agora já tínhamos trabalhado, somente que ao invés da informação ser apresentada no vídeo, a mesma será armazenada no arquivo. Ao ser usado o comando WRITE, todas as informações serão escritas no arquivo na mesma linha, como acontece quando se usa este comando para escrever no vídeo. Por outro lado, ao ser usado o comando WRITELN,

todas as informações serão colocadas uma em cada linha, como acontece quando se usa este comando para escrever informações no vídeo.
Sintaxe : WRITE(Meu_Arquivo, informação) ou WRITELN(Meu_Arquivo, Informação)
Exemplo:

```
PROGRAM Teste;
TYPE
        Registro = RECORD
                Nome    : STRING;
                Idade   : BYTE;
        END;
VAR
        Arquivo : TEXT;
        Reg     : Registro;
BEGIN
        ASSIGN(Arquivo, 'Dados.Dat');
        REWRITE(Arquivo);
        WRITE('Digite o Nome');
        READ(Reg.Nome);
        WRITE('Digite a Idade:');
        READ( Reg. Idade);
        WRITELN(Arquivo, Reg.Nome);
        WRITELN(Arquivo, Reg. Idade);
        CLOSE(Arquivo);
END.
```

Rotina :READ() ou READLN()
Função :A Rotina READ ou READLN tem a mesma Função de entrada de informações como até agora já havíamos trabalhado, somente que ao invés da leitura ser feita pelo teclado, a mesma será feita de um arquivo. Ao ser usado o comando READ, a leitura será feita sempre na mesma linha , como acontece quando se usa este comando para ler informações pelo teclado. Por outro lado, ao ser usado o comando READLN, as leituras serão feitas linha a linha, como acontece quando se usa este comando para leitura pelo teclado.
Sintaxe : READ(Meu_Arquivo, Informação) ou READLN(Meu_Arquivo, Informação)
Exemplo:

```
PROGRAM  Testes;
TYPE
        Registro = RECORD
                Nome : STRING;
                Idade  : BYTE;
        END;
VAR
        Arquivo : TEXT;
        Reg :Registro;
BEGIN
        ASSIGN(Arquivo, 'Dados.Dat');
        RESET(Arquivo);
        READLN(Arquivo, Reg.Nome);
        READLN(Arquivo, Reg.Nome);
        READLN(Arquivo, Reg.Idade);
        WRITE('Nome = ', Reg. Nome);
        WRITE('Idade = ', Reg. Idade');
        CLOSE(Arquivo);
END.
```

Rotina : EOF()
Função : Retorna TRUE caso se alcance o final do arquivo, FALSE caso contrário.
Sintaxe : Chegou_Final := EOF(Meu_Arquivo);
Exemplo:

```
PROGRAM Teste;
TYPE
        Registro = RECORD
                Nome : STRING;
```

```
                Idade : BYTE;
        END;
VAR
        Arquivo : TEXT;
        Reg     : registro;
BEGIN
        ASSIGN(Arquivo, 'Dados.Dat');
        RESET(Arquivo);
        WHILE NOT EOF(Arquivo) DO
        BEGIN
                READLN(Arquivo, Reg.Nome);
                READLN(Arquivo, Reg.Idade);
                WRITE('Nome = ',Reg.Nome);
                WRITE('Nome = ', Reg.Idade);
        END;
        CLOSE(arquivo);
END.
```

Rotina : ERASE()
Função : Elimina o arquivo do disco. É importante notar que o arquivo a ser eliminado não pode estar aberto .
Sintaxe : ERASE(Meu_Arquivo);
Exemplo:

```
PROGRAM Teste;
VAR
        Arquivo : TEXT;

BEGIN
        ASSIGN(Arquivo, 'Dados.Dat');
        ERASE(Arquivo);
END.
```

8.3.1 Exercícios:

1. Crie um arquivo TEXT formado somente por números inteiros de 1 a 10.000.
2. Faça a leitura do arquivo anterior e escreva no vídeo somente os números ímpares.
3. Crie um arquivo com Nome e salário de n funcionários e armazene estas informações em um arquivo TEXT.
4. Faça a leitura do arquivo anterior e escreva no vídeo o Nome e salário dos funcionários que ganham mais que US$1.000,00.
5. Abra o arquivo anterior e altere o salário de 100 funcionários para US$3.050,00.
6. Abra o arquivo anterior e aumente em 15% o salário de todos os funcionários que ganham menos de US$1.000,00.
7. Abra o arquivo anterior e mais 10 funcionários.
8. Ordene crescentemente pelo Nome do funcionário o arquivo anterior.
9. Envie os dados do arquivo anterior para a impressora. Para isso , use USES PRINTER e o comando WRITE(LST, informação); ou WRITELN(LST, Informação).
10. Faça um programa que imprima os arquivos, caso existam, AUTOEXEC.BAT e CONFIG.SYS na impressora. Adicionalmente faça com que a impressora seja em negrito . Para isso consulte o manual da impressora para determinar os códigos de programação da mesma.
11. Faça um programa que permita ao usuário imprimir um arquivo texto qualquer dando a possibilidade de selecionar qual o tipo de formato de impressão o usuário deseja , isto é : negrito, sublinhado, itálico, comprimido, expandido, etc.
12. Faça uma sub-Rotina para posicionar a cabeça da impressora um uma determinada linha e coluna . Considere o topo da folha de papel como sendo a posição (1 , 1).

9. Alocação Dinâmica

9.1 Introdução

Até agora temos definido variáveis de forma estática, ou seja, reservamos o espaço na memória necessária para as variáveis que iremos utilizar no programa. Isto funciona bem quando sabemos o quanto de memória iremos utilizar, mas e quando não sabemos? Tome por Exemplo as definições dos ARRAY's. Será que sempre temos certeza do tamanho de um ARRAY poderá ter por toda a vida de um programa? Será que o meu sistema de cadastro de clientes, o qual usa um ARRAY com 10.000 posições é o suficiente? Será que nunca irá acontecer de se tentar cadastrar o cliente de número 10.001? E o que acontece quando os clientes cadastrados nunca passarem de 100? As posições de memória restantes (9.900) não poderão ser utilizadas por outras variáveis, pois já estão reservadas. O uso de ARRAY's é sem dúvida de grande ajuda para a construção de um programa, mas quando temos que super dimensionar uma variável ARRAY, por não sabermos qual o tamanho que esta mesma variável irá ter, então começa a ser questionável a sua utilização. Pensando neste tipo de problema, foi desenvolvido um novo conceito para alocação de memória, onde poderemos reservar espaço da memória disponível (HEAP) a medida que for necessário, da mesma forma que poderemos liberar posições de memória quando não mais precisarmos delas. A este conceito deu-se o Nome de alocação dinâmica, uma vez que a memória é alocada não no início do programa, mas sim no decorrer de sua utilização do sistema. De uma forma mais simples de falar, é como se pudéssemos definir um ARRAY com o seu tamanho sendo alterado a medida que fosse necessário.

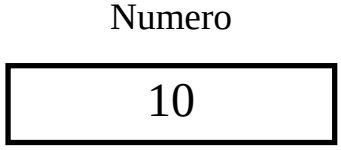
9.2 Definição de Pointers

Até agora ao definirmos uma variável, estávamos na verdade alocando um espaço na memória com um tamanho definido pelo tipo da variável (INTEGER, STRING, CHAR, etc...), sendo que ao invés de trabalharmos com o endereço físico de memória temos a facilidade de dar a este espaço alocado um Nome simbólico qualquer.

Exemplo: Alocar na memória para uma variável do tipo INTEGER e atribuir a esta posição de memória um valor qualquer

```
PROGRAM ESTATICO;
VAR
   Número   : INTEGER;
BEGIN
        Número := 10;
END.
```

No Exemplo acima ocorre as seguintes situações:
a) Reservamos espaço na memória suficiente para armazenar dois (2) BYTE's, ou seja, um INTEGER, e demos a esta posição de memória um Nome simbólico: "Número"
b) Atribuímos a variável "Numero" o valor dez (10), o que fará com que a memória ocorra a seguinte situação:



Bem, o que foi mostrado acima é o nosso modo habitual de trabalhar com variáveis, mas a partir de agora iremos trabalhar de uma maneira um pouco diferente, ou seja, ao invés de definirmos uma variável como sendo de um tipo qualquer e a esta variável atribuirmos uma informação propriamente dita, mas sim o endereço físico da memória onde a informação está armazenada. A este tipo de variável passaremos a chamar, a partir de agora, de variáveis pointer (apontadores ou ponteiro), pelo simples fato dela (a variável) apontar, indicar, a localização de uma informação na memória.

Sintaxe para definição:

<Nome da variável> : ^<tipo>

Exemplo 1: Definir variáveis pointer para os tipos STRING, INTEGER, REAL, CHAR, BOOLEAN.

```
PROGRAM DEFINE_POINTER;
VAR
        Ap_STRING     : ^STRING;
        Ap_INTEGER    : ^INTEGER;
        Ap_REAL       : ^REAL;
        Ap_BYTE       : ^BYTE;
        Ap_CHAR       : ^CHAR;
        Ap_BOOLEAN    : ^BOOLEAN;
BEGIN
        <comandos>;
END.
```

Caso seja necessário definir variáveis pointers para RECORD's e ARRAY's será preciso antes criar tipos de dados que representem estes mesmos RECORD's e ARRAY's.

Exemplo 2: Definir uma variável pointer para um ARRAY[1..2] OF STRING.

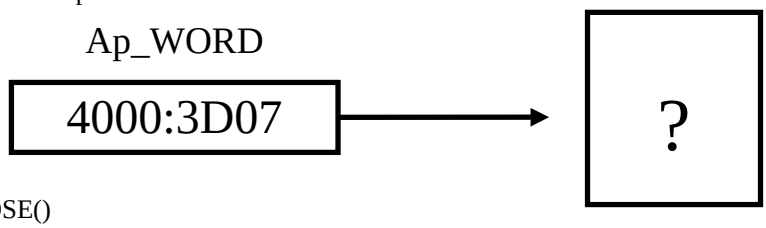
```
PROGRAM DEFINE_ARRAY_POINTER;
TYPE
        Vetor = ARRAY [1..2] OF STRING;
VAR
        Ap_vetor : ^Vetor;
BEGIN
        <comandos>;
END.
```

9.3 Rotinas para Alocação de Memória:

Rotina : NEW()
Função : Aloca espaço na memória para uma informação, com o tamanho definido pelo tipo da variável pointer.
Sintaxe : NEW(Variável Pointer)
Exemplo:

```
PROGRAM ALOCA;
VAR
        Ap_WORD : ^WORD;
BEGIN
        NEW(Ap_WORD);
END.
```

Obs.: No **Exemplo** acima, após o comando NEW, será alocado na memória HEAP, dois BYTE's (uma WORD), sendo que poderemos representar a memória como é mostrado abaixo:



Rotina : DISPOSE()
Função : Libera espaço na memória, o número de BYTE's liberados dependerá do tipo da variável pointer utilizada. Uma vez liberada memória, o valor lá armazenado estará perdido.
Sintaxe : DIPOSE(Variável Pointer)
Exemplo:

```
PROGRAM LIBERA;
VAR
        Ap_WORD : ^WORD;
BEGIN
        NEW(Ap_WORD);
        DISPOSE (Ap_WORD);
END.
```

Obs.: No Exemplo anterior, após o comando DISPOSE, serão liberado dois (2) BYTE's devido ao fato de uma WORD ocupar este espaço de memória.

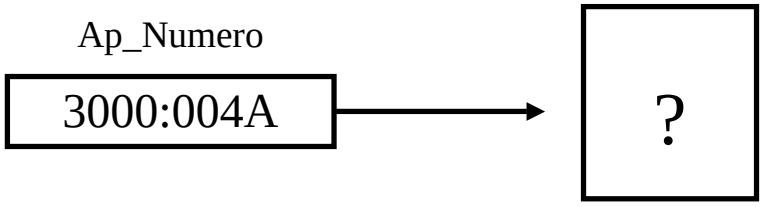
9.4 Atribuição de Valores

A Sintaxe para atribuição de valores é a mesma utilizada em variáveis simples, a única diferença é que devemos colocar após o Nome da variável apontadora o símbolo " ^ "
Exemplo:

```
PROGRAM ATRIBUI;
VAR
        Ap_Número : ^INTEGER;
BEGIN
        NEW(Ap_Número);
        Ap_Número ^:= 10;
        DISPOSE (Ap_Número);
END.
```

No **Exemplo** acima ocorre o seguinte:

a) Criamos uma variável que irá apontar ara dois (2) BYTE's (um INTEGER) na memória.
b) Alocamos espaço suficiente para armazenar um valor do tipo INTEGER e fazemos com que a variável "Ap_Número" aponte para a posição de memória alocada.

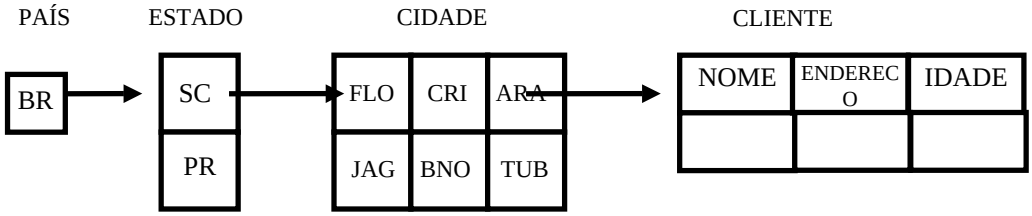


c) Colocamos na posição de memória apontada por "Ap_Número" o valor dez (10).
d) Liberamos os dois (2) BYTE's apontados por "Ap_Número". A informação não mais poderá ser acessada.

9.4.1 Exercícios:

1. Crie um vetor com n elementos, sendo que cada posição do vetor corresponderá a um pointer para um valor do tipo REAL. Faça a leitura de n valores e armazene-os na memória.

2. Crie uma variável pointer do tipo ARRAY[1..20] OF CHAR, faça a leitura de 20 caracteres e os armazene na memória.
3. Percorra o ARRAY definido acima e escreva quantos caracteres “A”, “E”, “I”, “O” e “U” existem no mesmo.
4. Defina um tipo (TYPE) de dado que represente um pointer para um RECORD com os seguintes campos: Nome e Idade.
5. Usando a definição de tipo anterior, crie uma sub-Rotina para ler as informações de uma única pessoa.
6. Defina uma variável como sendo um ARRAY com 10 posições, sendo que cada posição corresponderá aos dados de uma pessoa, conforme definido no item 5.
7. Use a sub-Rotina definida no item 5, para ler o vetor definido no exercício acima.
8. Defina uma variável pointer do tipo matriz N x N, sendo que cada posição desta matriz também será um pointer mas para um valor do tipo WORD.
9. Faça a leitura da matriz definida no item acima.
10. Percorra a matriz acima e escreva os valores existente na diagonal principal.
11. Faça as definições necessárias para obter a seguinte representação de pointer.



12. Preencha a estrutura acima com as informações de 12 pessoas.

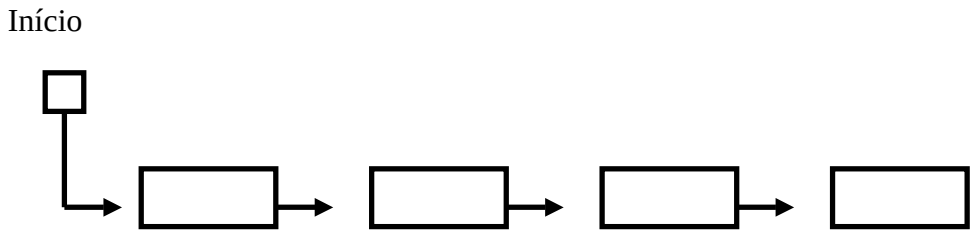
13. Liste no vídeo o Nome e idade das pessoas que tem idade ímpar.

10. Lista Simplesmente Encadeada

Até agora vimos que é possível alocar espaço para uma informação na memória e liberar este mesmo espaço quando não for mais necessário. O problema é que no mundo computacional necessitamos trabalhar não apenas com uma informação mas com várias. Da mesma forma como vínhamos trabalhando até agora, quando era necessário guardar várias informações na memória nós utilizávamos o ARRAY. Já foi explicado todos os problemas inerentes ao uso de um ARRAY, por isso é necessário definir um a outra estrutura que permita armazenar informações na memória independente da quantidade. Esta estrutura será a partir de agora chamada de **lista encadeada**.

10.1 Definição

Uma lista encadeada é uma seqüência de informação armazenadas em algum lugar da memória, sendo que as mesmas estão ligadas entre si por um endereço (pointer).
Exemplo: abaixo colocarei um desenho representando uma lista encadeada na memória.



10.2 Criando Listas na Memória

Para criarmos uma lista, para colocarmos na memória uma seqüência de valores sendo que os mesmos estejam ligados entre si por um endereço, ou pointer, vamos utilizar a estrutura RECORD. Esta RECORD será usada basicamente para definir dois tipos de campos: O primeiro tipo corresponde aos campos de informações, aquelas quereremos armazenar na memória, e o segundo tipo corresponde ao campo apontador (pointer), cuja *Função* será armazenar o endereço da próxima informação existente na memória.

Exemplo: Definir um tipo de dado que permita armazenar na memória as informações de um cliente: Nome, Idade e Sexo.

```
PROGRAM TÁ_FALTANDO_ALGO;
TYPE
REGISTRO = RECORD
                        Nome    : STRING;
                        Idade   : BYTE;
                        Sexo    : CHAR;
                        Ender   : <tipo apontador>;
              END;
BEGIN
        <comandos>;
END.
```

O programa acima não está completo quanto a definição do RECORD, pois um campo chamado “Ender” que não tem o seu tipo definido. Mas então, qual será este tipo? O campo “Ender” deverá ser usado para armazenar o endereço de um a informação na memória que seja do tipo “Registro”, pois é ela que contém a definição dos dados

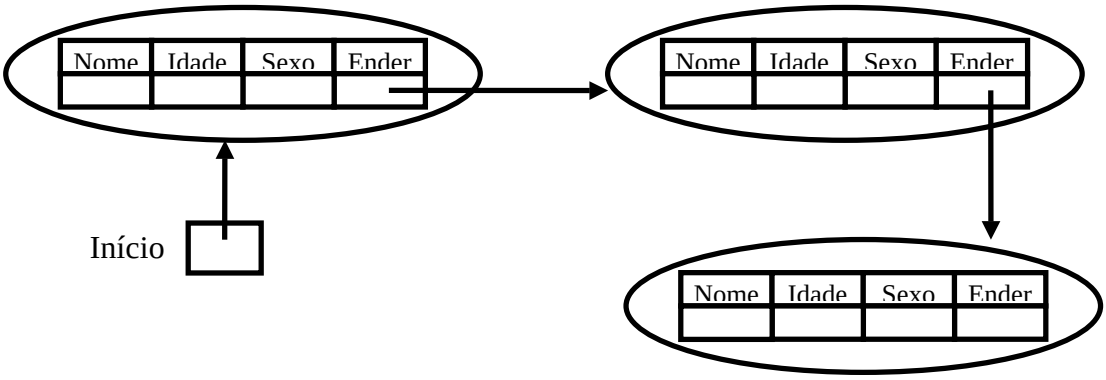
dos clientes. Desta forma, o campo “Ender” deverá ser um tipo apontador de registros. Só que temos um pequeno problema de Sintaxe. Caso façamos a seguinte definição do tipo “Registro”:

```
PROGRAM AINDA_TÁ_ERRADO;
TYPE
        REGISTRO = RECORD
                                Nome    : STRING;
                                Idade   : BYTE;
                                Sexo    : CHAR;
                                Ender   : ^Registro;
                        END;
BEGIN
        <comandos>;
END.
```

O PASCAL irá acusar um erro de compilação, pois ele precisa que um apontador de estruturas complexas como RECORD’s, tenham que ter definidos um tipo (TYPE) específico para ele, por isso eu necessito definir antes, um tipo que seja um apontador de “Registro. Abaixo é mostrado como isto será feito:

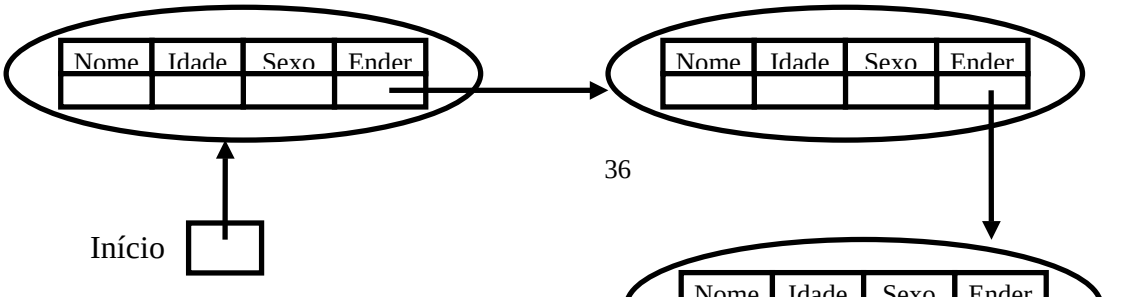
```
PROGRAM OK;
TYPE
        Ap_Registro = ^REGISTRO;
REGISTRO = RECORD
                        Nome    : STRING;
                        Idade   : BYTE;
                        Sexo    : CHAR;
                        Ender   : Ap_Registro;
                END;
BEGIN
        <comandos>;
END.
```

Pode parecer estranho, definir o tipo “Ap_Registro” como sendo um apontador de registro sendo que o tipo “Registro” ainda não tinha sido definido. Não se assuste, é assim mesmo. Com o tempo você se acostuma.
No futuro, quando já tivermos a lista definida criada, a mesma poderá ter a seguinte representação:



Obs.: Um apontador não pode ficar sem um valor, por isso o apontador do último elemento da lista deverá receber um valor especial do PASCAL, que indica que aquele apontador não aponta para ninguém. Este valor é uma variável pré-definida do PASCAL chamada NIL.

Desta forma, a estrutura acima com o uso da variável NIL, terá a seguinte representação:



Obs.: O termo lista simplesmente encadeada significa que a lista possui somente um apontador para apontar as informações na memória.

Exemplo1: Faça um PROGRAMA que armazene o Nome, idade, e sexo de uma pessoa em uma estrutura simplesmente encadeada na memória.

```
PROGRAM LISTA_ENCADEADA;
USES CRT;
TYPE
        Ap_Nodo = ^Nodo
Nodo = RECORD
                Nome    : STRING;
                Idade   : BYTE;
                Sexo    : CHAR;
                Prox    : Ap_Nodo;
        END;
VAR
        Raiz : Ap_Nodo;
BEGIN
        NEW(RAIZ)
        WRITE (‘Digite o Nome : ‘);
        READLN (RAIZ^.Nome);
        WRITE (‘Digite o Idade : ‘);
        READLN (RAIZ^.Idade);
        WRITE (‘Digite o Sexo : ‘);
        READLN (RAIZ^.Sexo);
        RAIZ^.Prox := NIL;
END.
```

Exemplo 2: Aproveitando a lista criada no Exemplo anterior, crie uma sub-Rotina que recebendo como parâmetro as informações de uma pessoa, acrescente esta pessoa no início da lista.

```
PROCEDURE Inclui_Inicio_da_Lista (Var Raiz : Ap_Nodo; Reg: Nodo);
VAR
        Novo_Nodo : Ap_Nodo;
BEGIN
        NEW ( Novo_Nodo);
        Novo_Nodo^:= Reg;
        Novo_Nodo^.Prox := Raiz;
        Raiz := Novo_Nodo;
END.
```

Exemplo 3: Aproveitando a lista criada no Exemplo anterior, crie uma sub-Rotina que recebendo como parâmetro as informações de uma pessoa, acrescente esta pessoa na lista, de forma que a mesma seja a segunda da lista.

```
PROCEDURE Segunda_Posição_da_Lista (Var Raiz : Ap_Nodo; Reg: Nodo);
VAR
        Novo_Nodo : Ap_Nodo;
BEGIN
        NEW ( Novo_Nodo);
        Novo_Nodo^:= Reg;
        Novo_Nodo^.Prox := Raiz^.Prox;
        Raiz^.Prox  := Novo_Nodo;
END.
```

Exemplo 4: Faça uma sub-Rotina genérica que recebendo como parâmetro as informações de uma pessoa, acrescente a mesma no final da lista.

```
PROCEDURE Inclui_Final_da_Lista (Var Raiz : Ap_Nodo; Reg: Nodo);
VAR
        Novo_Nodo, Atual : Ap_Nodo;
BEGIN
        NEW ( Novo_Nodo);
        Novo_Nodo^:= Reg;
        Novo_Nodo^.Prox := NIL;
        Atual := Raiz;
        WHILE Atual^.Prox <> NIL DO
        Atual := Atual^.Prox;
        Atual^.Prox := Novo_Nodo;
END.
```

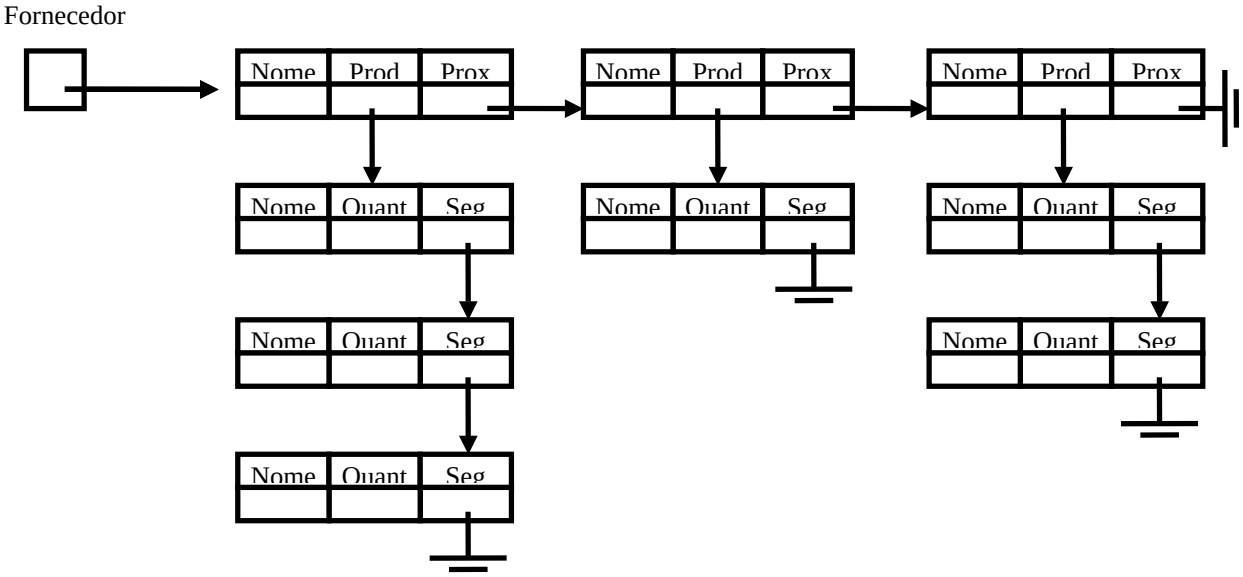
Exemplo 5 : Faça uma sub-Rotina genérica que recebendo como parâmetro as informações de uma pessoa, acrescente a mesma em uma lista. No Inicio, considere a lista com o valor NIL. As pessoas serão incluídas sempre no final da lista.

```
PROCEDURE Inclui_Final_da_Lista (Var Raiz : Ap_Nodo; Reg: Nodo);
VAR
        Novo_Nodo, Atual : Ap_Nodo;
BEGIN
        NEW ( Novo_Nodo);
        Novo_Nodo^:= Reg
        Novo_Nodo^.Prox := NIL;
        IF  RAIZ = NIL THEN
                Raiz := Novo_Nodo
        ELSE
            BEGIN
                Atual := Raiz;
                WHILE Atual^.Prox <> NIL DO
                Atual := Atual^.Prox;
                Atual^.Prox := Novo_Nodo;
            END;
END.
```

10.2.1 Exercícios:

1. Faça uma sub-Rotina que conte quantos elementos existem em uma lista simplesmente encadeada.
2. Faça uma sub-Rotina que verifique se uma determina pessoa existe na lista. A consulta pode ser feita pelo Nome da pessoa. Use a lista definida nos exemplos anteriores.
3. Faça uma sub-Rotina que elimine o primeiro elemento da lista se o mesmo existir.
4. Faça uma sub-Rotina que elimine o segundo elemento da lista se o mesmo existir.
5. Faça uma sub-Rotina que elimine o ultimo elemento da lista se o mesmo existir.
6. Faça uma sub-Rotina genérica que elimine um determinado elemento de uma lista. Deverá ser fornecido o número do elemento a ser eliminado. Pode ser que a lista não contenha este elemento. Por Exemplo, a lista tem 10 nodos e deseja-se eliminar o nodo de número 11.
7. Faça uma sub-Rotina que permita a inclusão de um elemento na lista em uma determinada posição. A sub-Rotina deverá receber a lista, a informação e a posição em que a nova informação será incluída. Caso a posição seja maior que o número de elementos existentes na lista, a mesma deverá ser incluída no final. Considere a possibilidade da lista, no início estar vazia.
8. Faça uma sub-Rotina para ordenar uma lista de números inteiros em ordem crescente. Antes de construir a Rotina, faça a definição do tipo da lista: RECORD e tipo apontador.

9. Faça a definição de um ou mais tipos que possam no futuro criar uma lista como a que é mostrada abaixo:



10. Faça uma sub-Rotina genérica para incluir um novo produto na lista de produtos de um fornecedor qualquer. Os parâmetros de entrada serão as informações do produto e a lista dos produtos de cada fornecedor (Fornecedor^.Prod). A inclusão de novo produto se dará sempre no final da lista de produtos.

11. Faça uma sub-Rotina genérica para incluir um novo fornecedor na lista de fornecedores. A inclusão se dará sempre no final da lista.

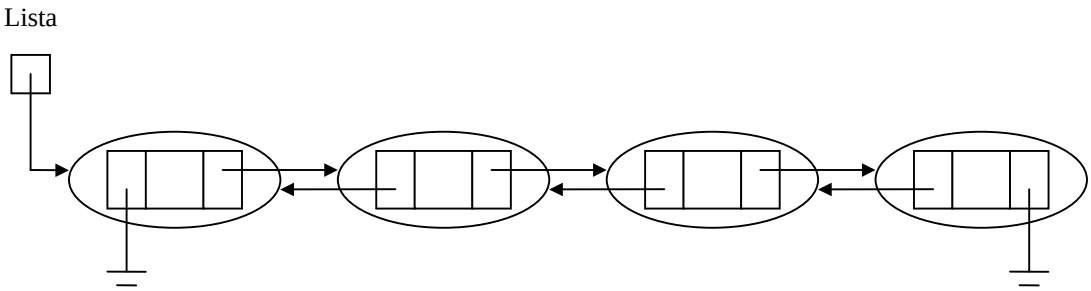
12. Faça uma sub-Rotina genérica que inclua um produto para um determinado fornecedor. Os parâmetros de entrada serão: A lista de fornecedores, as informações do produto e as informações do fornecedor, que terá incluído no seu campo “Prod” este novo produto. A Rotina deverá antes de mais nada percorrer a lista de fornecedores até achar o fornecedor cujo Nome combine com o Nome passado como parâmetro. Só após é que será feito o processo de inclusão do novo produto. Caso o fornecedor não exista na lista, o mesmo deverá antes de tudo ser incluído na lista de fornecedores, para só então ter o produto incluído no seu corpo “Prod”. Use também as Rotinas criadas nos itens 10 e 11.

11. Lista Duplamente Encadeada

Continuado o estudo de listas encadeadas, apresentamos agora uma nova estrutura conhecida como lista duplamente encadeada. Este termo é dado pelo fato de existir dois endereços (pointers) que apontam para nodos da mesma lista, isto é, um endereço aponta para o nodo imediatamente posterior e outro endereço aponta para o nodo imediatamente anterior, isto com exceção do primeiro e do último nodo, cujos endereços posterior e antecessor são respectivamente NIL.
Abaixo está uma figura representando este tipo de lista encadeada na memória

Abaixo colocaremos exemplos de *Rotina*s para criação de listas duplamente encadeada na memória.

Exemplo1: Faça um algoritmo que armazene o Nome, idade e sexo de uma pessoa em uma estrutura duplamente encadeada na memória.




```
PROGRAM LISTA_ENCADEADA;
USES CRT;
TYPE
        Ap_Nodo = ^Nodo
Nodo = RECORD
                Nome    : STRING;
                Idade   : BYTE;
                Sexo    : CHAR;
                Ant, Prox    : Ap_Nodo;
        END;
VAR
        Raiz : Ap_Nodo;
BEGIN
        NEW(RAIZ)
        WRITE (‘Digite o Nome : ‘);
        READLN (RAIZ^.Nome);
        WRITE (‘Digite o Idade : ‘);
        READLN (RAIZ^.Idade);
        WRITE (‘Digite o Sexo : ‘);
        READLN (RAIZ^.Sexo);
        RAIZ^.Ant := NIL;
        RAIZ^.Prox := NIL;
END.
```

Exemplo 2: Aproveitando a lista criada no Exemplo anterior, crie uma sub-*Rotina* que recebendo como parâmetro as informações de uma pessoa, acrescente esta pessoa no início da lista.

```
PROCEDURE Inclui_Inicio_da_Lista (Var Raiz : Ap_Nodo; Reg: Nodo);
VAR
```

```
        Novo_Nodo : Ap_Nodo;
BEGIN
        NEW ( Novo_Nodo);
        Novo_Nodo^:= Reg;
        Novo_Nodo^.Ant := NIL;
        Novo_Nodo^.Prox := Raiz;
        Raiz^.Ant:= Novo_Nodo;
        Raiz := Novo_Nodo;
END.
```

Exemplo 3: Aproveitando a lista criada no Exemplo anterior, crie uma sub-Rotina que recebendo como parâmetro as informações de uma pessoa, acrescente esta pessoa na lista, de forma que a mesma seja a segunda da lista.

```
PROCEDURE Segunda_Posição_da_Lista (Var Raiz : Ap_Nodo; Reg: Nodo);
VAR
        Novo_Nodo, Aux : Ap_Nodo;
BEGIN
        NEW ( Novo_Nodo);
        Novo_Nodo^:= Reg;
        Aux := Raiz^.Prox;
        Novo_Nodo^.Prox := Aux;
        Novo_Nodo^.Ant := Raiz;
        Raiz^.Prox  := Novo_Nodo;
        IF Aux <> NIL THEN
                Aux^.Ant := Novo_Nodo;
END.
```

Exemplo 4: Faça uma sub-Rotina genérica que recebendo como parâmetro as informações de uma pessoa, acrescente a mesma no final da lista.

```
PROCEDURE Inclui_Final_da_Lista (Var Raiz : Ap_Nodo; Reg: Nodo);
VAR
        Novo_Nodo, Atual : Ap_Nodo;
BEGIN
        NEW ( Novo_Nodo);
        Novo_Nodo^:= Reg;
        Novo_Nodo^.Prox := NIL;
        Atual := Raiz;
        WHILE Atual^.Prox <> NIL DO
        Atual := Atual^.Prox;
        Atual^.Prox := Novo_Nodo;
        Novo_Nodo^.Ant := Atual;
END.
```

Exemplo 5 : Faça uma sub-Rotina genérica que recebendo como parâmetro as informações de uma pessoa, acrescente a mesma em uma lista. No Inicio, considere a lista com o valor NIL. As pessoas serão incluídas sempre no final da lista.

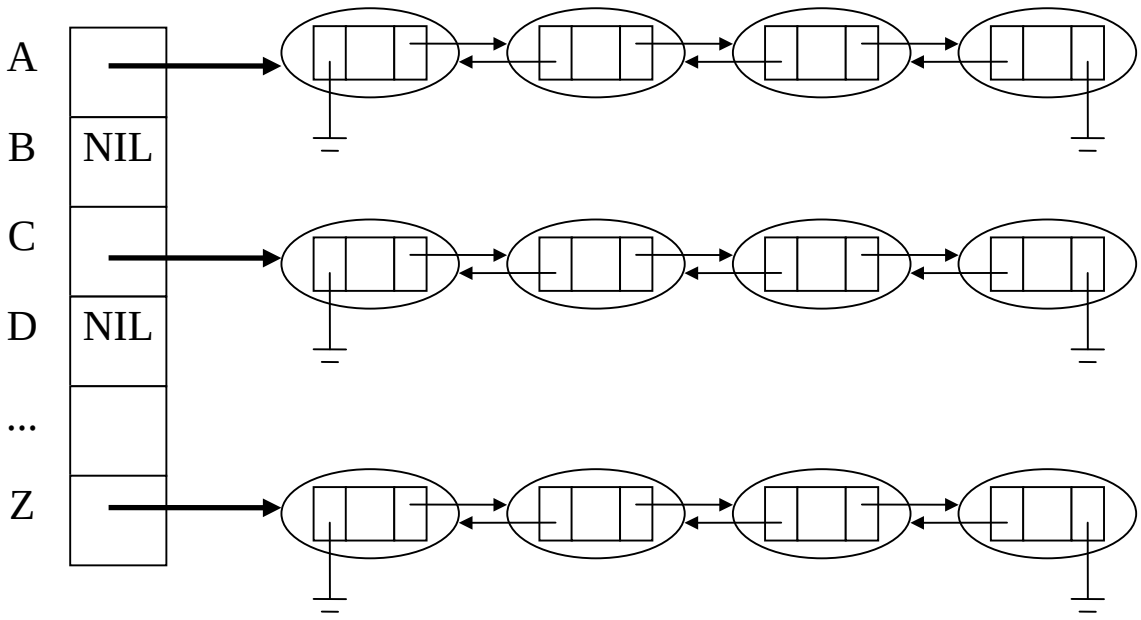
```
PROCEDURE Inclui_Final_da_Lista (Var Raiz : Ap_Nodo; Reg: Nodo);
VAR
        Novo_Nodo, Atual : Ap_Nodo;
BEGIN
        NEW ( Novo_Nodo);
        Novo_Nodo^:= Reg
        Novo_Nodo^.Ant := NIL;
        Novo_Nodo^.Prox := NIL;
        IF  RAIZ = NIL THEN
                Raiz := Novo_Nodo
        ELSE
             BEGIN
                Atual := Raiz;
                WHILE Atual^.Prox <> NIL DO
                        Atual := Atual^.Prox;
                Atual^.Prox := Novo_Nodo;
                Novo_Nodo^.Ant := Atual;
            END;
END.
```

11.1.1 Exercícios:

1. Faça uma sub-Rotinas que elimine o primeiro elemento da lista, se o mesmo existir.
2. Faça uma sub-Rotinas que elimine o segundo elemento da lista, se o mesmo existir.
3. Faça uma sub-Rotina que elimine o último elemento da lista, se o mesmo existir.
4. Faça uma sub-Rotina genérica que elimine um determinado elemento de uma lista. Deverá ser fornecido o número do elemento a ser eliminado. Pode ser que a lista não contenha este elemento. Por Exemplo, a lista tem 10 nodos e deseja-se eliminar o nodo de número 11.

5. Faça uma sub-Rotina que permita a inclusão de um elemento na lista em uma determinada posição. A sub-Rotina deverá receber a lista, a informação e a posição em que a nova informação será incluída. Caso a posição seja maior que o número de elementos atualmente existentes na lista, a mesma deverá ser incluída no final. Considere a possibilidade da lista no início estar vazia.
6. Faça uma sub-Rotina para ordenar uma lista de números inteiros em ordem crescente. Antes de construir a Rotina, faça a definição do tipo da lista: RECORD e o tipo apontador.
7. Aproveitando a lista ordenada no exercício anterior, faça uma sub-Rotina genérica para listar o conteúdo da lista em ordem crescente.
8. Aproveitando a lista ordenada no exercício anterior, faça uma sub-Rotina genérica para listar o conteúdo da lista em ordem crescente.

9. Faça as definições de tipo necessárias para criar, no futuro, uma estrutura na memória como a que é representada na figura abaixo:



Obs.: a) Dicionário é um vetor (de “A” até “Z”) de pointers para uma estrutura duplamente encadeada onde além dos campos “Prox” e “Ant” , existem mais dois campos STRING os quais representam a palavra e o seu significado.

b) A lista de palavras que estão ligadas ao índice “A” do vetor, correspondem aquelas palavras que iniciam com a letra “A”, e assim por diante.

10. Usando a definição feita no exercício anterior, faça uma sub-Rotina para inclusão de uma palavra e o seu significado no dicionário de palavras. Lembre-se que a palavra e o seu significado não devem ser incluídas em qualquer lista de forma aleatória, a inclusão dever ser feita somente na lista cujo índice do vetor corresponde à primeira letra da palavra. Desta forma, caso a palavra, fosse “Cavalo”, esta palavra, junto com o seu significado, deveriam ser incluídos na lista existente no índice “C” do vetor.