UNISINOS - **UNIVERSIDADE DO VALE DO RIO DOS SINOS**
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS (C6/6) – Curso: Informática

Programação II

Disciplina: Linguagem de Programação PASCAL **E-mail:** ***osorio@exatas.unisinos.br***
Professor responsável: ***Fernando Santos Osório*** **Web:**
Semestre: 2004/2 ***http://www.inf.unisinos.br/~osorio/prog2.html***
Horário: 63 **Xerox : *Pasta 54* – LAB. II (Xerox do "Alemão")**

ALOCAÇÃO DINÂMICA DE MEMÓRIA – PONTEIROS

Alocação Estática: Os arrays (vetores uni ou multi-dimensionais) são alocados seqüencialmente na memória do computador, e sendo assim possuem sempre um endereço fixo na memória onde começa o seu primeiro elemento. Para sabermos onde se encontra o elemento "N" de um array uni-dimensional (array [1..Maximo] of Tipo_Dado), basta somarmos ao endereço inicial do array o valor de N-1 multiplicado pelo Tamanho_do_dado.

Curiosidades:
- Para obter o endereço inicial de uma estrutura de dados qualquer, podemos usar o comando "addr": Ptr_Inicio_Vetor := addr(Vetor[1]);
- Para sabermos quantos bytes ocupa uma certa estrutura de dados, podemos usar o comando "sizeof": Tamanho_do_Dado := sizeof(Tipo_Dado);
- Endereços de memória nos PC's (TurboPascal DOS) são definidos por uma base (16 bits) e um deslocamento (16 bits), que juntos permitem que possamos endereçar e acessar qualquer posição de memória da máquina. O comando "Ptr" converte uma base e deslocamento em um ponteiro. Para compreender melhor os detalhes de endereçamento de um PC é necessário estudar a arquitetura dos processadores 80x86 (8086, 80286, 80386, 80486 e Pentium). Se você quiser ver em detalhes o 80x86 em ação - no menu do TPascal: *Options, Debugger*, marcar [X] *Standalone Debugging. Compile. Tools, Turbo Debugger*.

Como as variáveis de alocação estática possuem um endereço fixo e previsível na memória do computador, é necessário que estas sejam declaradas COM SEU TAMANHO EXATO, para que possamos compilar e executar um programa. Um vetor sempre terá as suas dimensões definidas com exatidão, antes de começarmos a executar o programa.

Surge então a pergunta: se eu estiver fazendo um programa para criar um cadastro de alunos, mas onde não sei ainda quantos alunos terei ao total, como farei para prever a quantidade de elementos do vetor de registros que armazena os dados do cadastro ? Existem basicamente duas soluções: usar um método de alocação estática de memória e dimensionar o vetor com um tamanho superior ao máximo necessário... ou então usar um *método de alocação dinâmica de memória*, onde vou alocando mais memória a medida que isto vai se fazendo necessário (não especifico o tamanho total da minha estrutura de dados).

Alocação Dinâmica: Os elementos básicos para a criação de estruturas de dados com alocação dinâmica de memória, são os comandos de alocação **NEW** e de liberação **DISPOSE** de memória, assim como o uso dos ponteiros ^, que permitem "achar" onde se encontram os dados na memória do computador, uma vez que estes estarão espalhados por toda parte (isto se deve ao fato que não teremos mais uma estrutura com alocação seqüencial de memória).

Características da *alocação dinâmica* de memória:

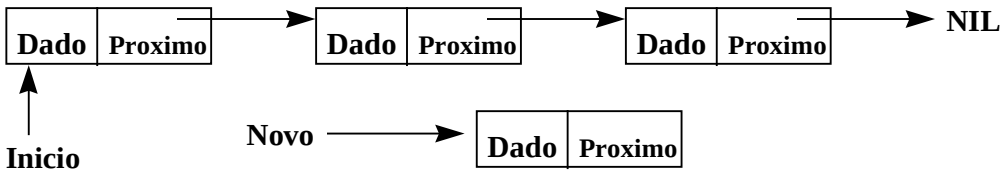
- Os dados se encontram espalhados na memória do computador;
- Geralmente as estruturas de dados são compostas por registros que incluem um campo do tipo ponteiro, o que nos permite encadear os dados e indicar onde está o dado seguinte na memória (visto que os dados estão espalhados na memória do computador);
- Podemos ir solicitando mais e mais memória a medida em que precisamos de mais espaço para armazenar as informações. Isso nos permite criar programas que usam apenas a memória necessária... e por conseqüência, podemos rodar outros programas sem que um único programa monopolize toda a memória disponível na máquina.
- Podemos liberar espaços de memória quando estes não forem mais necessários ao programa.

Usualmente teremos a seguinte estrutura de dados:

```
Type
    Tipo_Dado  =  Integer;
    Ptr_Registro =  ^Registro;
    Registro       =  Record
                          Dado: Tipo_Dado;
                          ...
                          Proximo : Ptr_Registro;
                      End;
Var
    Inicio: Ptr_Registro;  { Uma vez criada a lista encadeado não devemos
                             perder o ponteiro que aponta para o início da lista ! }
    Novo: Ptr_Registro;  { Usado para criar novos registros com o comando NEW }

{ Criando uma lista de 2 nodos apenas[1] }
Begin
    New (Novo);                 { Alocação dinâmica: cria o primeiro registro    }
    Inicio:= Novo;              { Inicio aponta para o primeiro registro da lista }
    Novo^.Dado := 1;
    New(Novo);                  { Alocação dinâmica: cria o segundo registro        }
    Inicio^.Proximo:=Novo;      { O proximo depois do inicio é o segundo registro }
    Novo^.Dado := 2;
    Novo^.Proximo := NIL;       { O proximo depois do segundo é o FIM da Lista }
End.
```

A figura abaixo mostra um exemplo de lista encadeada, baseada na estrutura de dados descrita logo acima:



[1] O Turbo Debugger possui uma ferramenta muito útil para visualizar listas encadeadas: Data - Inspect

EXERCÍCIOS – ***Listas simplesmente encadeadas***

1. Faça um programa para a manipulação de listas simplesmente encadeadas com alocação dinâmica na memória do computador. Crie as seguintes rotinas genéricas de manipulação de dados:

   Definições elementares:

```
Type
    TListaSE_Dado     = Integer;
    Ptr_ListaSE_Nodo  = ^ListaSE_Nodo;
    ListaSE_Nodo      = Record
                          Dado: TListaSE_Dado;
                          Prox : Ptr_ListaSE_Nodo;
                        End;
```

   Rotinas:

   Procedure **ListaSE_Inicializa** (Var LSE: Ptr_ListaSE_Nodo);
   {LSE = Ponteiro para o início da Lista Simplesmente Encadeada, inicializa com NIL }

   Procedure **ListaSE_Insere_Inicio** (Var LSE: Ptr_ListaSE_Nodo; Dado: TListaSE_Dado);
   { Insere um dado no início da lista, ou seja, como primeiro da lista }

   Procedure **ListaSE_Insere_Final** (Var LSE: Ptr_ListaSE_Nodo; Dado: TListaSE_Dado);
   { Insere um dado no final da lista, ou seja, como último da lista }

   Procedure **ListaSE_Insere_Ordenado** (Var LSE: Ptr_ListaSE_Nodo; Dado: TListaSE_Dado);
   { Insere os dados de modo ordenado, localizando a sua correta posição na lista ordenada }

   Function **ListaSE_Remove_Inicio** (Var LSE: Ptr_ListaSE_Nodo;
   Var Dado: TListaSE_Dado): Boolean;
   { Remove o primeiro dado da lista }

   Function **ListaSE_Remove_Final** (Var LSE: Ptr_ListaSE_Nodo;
   Var Dado: TListaSE_Dado): Boolean;
   { Remove o último dado da lista }

   Function **ListaSE_Remove_Elemento** (Var LSE: Ptr_ListaSE_Nodo;
   Dado: TListaSE_Dado): Boolean;
   { Procura o dado informado e, se encontrar, remove ele da lista }

   Function **ListaSE_Quantidade** (LSE: Ptr_ListaSE_Nodo): Integer;
   { Retorna a quantidade total de nodos da lista }

   Procedure **ListaSE_Exibe_Lista** (LSE: Ptr_ListaSE_Nodo);
   { Exibe na tela o conteúdo da lista encadeada }

Function **ListaSE_Pesquisa** (Var LSE: Ptr_ListaSE_Nodo; Dado: TListaSE_Dado): Boolean;
{ Procura na lista o dado informado, retornando se encontrou e posicionando o ponteiro nele }

Function **ListaSE_Percorre** (Var LSE: Ptr_ListaSE_Nodo;
Var Dado: TListaSE_Dado): Boolean;
{ Dado um nodo apontado por LSE, avança o ponteiro p/o nodo seguinte, e retorna seu valor }

Procedure **ListaSE_Apaga** (Var LSE: Ptr_ListaSE_Nodo);
{ Remove todos os nodo da lista }

2. Faça um programa que receba como entrada duas listas encadeadas ordenadas (o usuário deve poder digitar os valores que vão fazer parte de cada uma das listas) e gere como resultado uma terceira lista encadeada, que concatene as duas listas encadeadas, mantendo ordenados seus elementos. Utilize as rotinas implementadas no exercício anterior.