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
Professor responsável: ***Fernando Santos Osório***
Semestre: 2004/2
Horário: 63

E-mail: ***osorio@exatas.unisinos.br***
Web:
http://www.inf.unisinos.br/~osorio/prog2.html
Xerox : *Pasta 54* – LAB. II (Xerox do "Alemão")

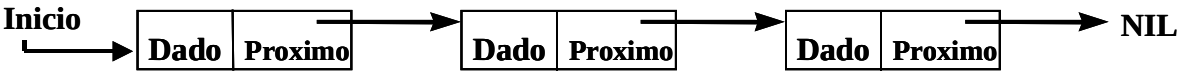
ALOCAÇÃO DINÂMICA - LISTAS SIMPLESMENTE ENCADEADAS

As estruturas com alocação dinâmica possuem um componente importante que permite "unir" cada "pedaço" de memória que for sendo alocado durante a execução do programa. Este componente é o **PONTEIRO**, que permite encadear diferentes blocos de memória, formando o que denominamos de *estruturas de dados encadeadas*, onde destacam-se as estruturas dos seguintes tipos: Listas Simplesmente Encadeadas, Listas Duplamente Encadeadas e Árvores.

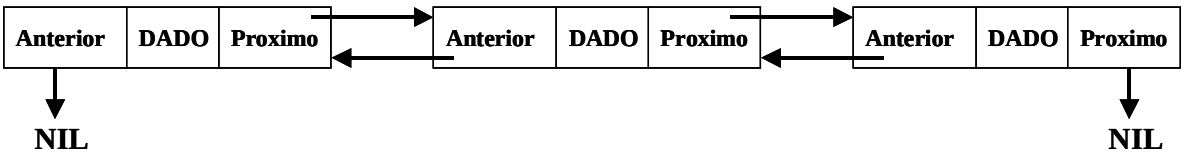
Estas estruturas irão nos permitir implementar inclusive as estruturas anteriormente estudadas, ou seja, PILHAS, FILAS e DEQUES podem ser implementados utilizando como base para o armazenamento dos dados uma estrutura encadeada do tipo LISTA ENCADEADA.

Abaixo apresentamos um esquema das principais estruturas encadeadas citadas acima:

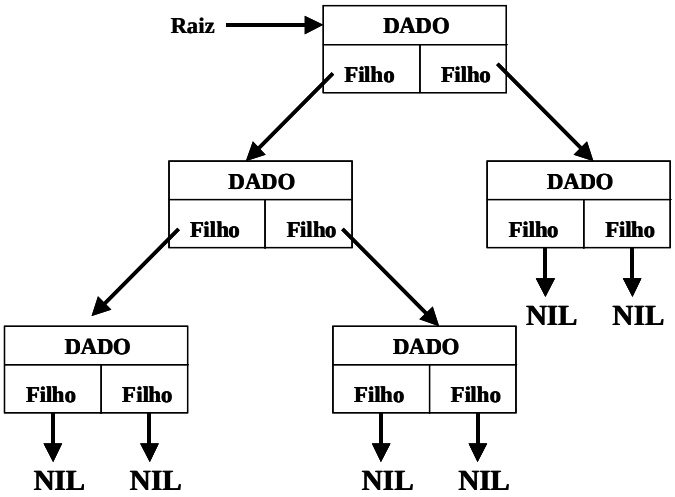
1) **Listas Simplesmente Encadeadas:** Os nodos possuem apenas 1 (um) elo de encadeamento. Só pode ser percorrida em um sentido. Esquema:



2) **Listas Duplamente Encadeadas:** Os nodos possuem 2 (dois) elos de encadeamento, possibilitando que a lista seja percorrida nos dois sentidos. Esquema:



3) **Árvores:** Os nodos possuem uma organização hierárquica, formando níveis, onde um nodo pode estar no nível superior, inferior ou no mesmo nível de um outro nodo. Dentro desta hierarquia, definimos o conceito de nodo pai e nodo filho, que são aqueles nodos que estão em níveis diferentes da hierarquia, mas que também possuem um elo de ligação. Esquema:



LISTAS SIMPLESMENTE ENCADEADAS

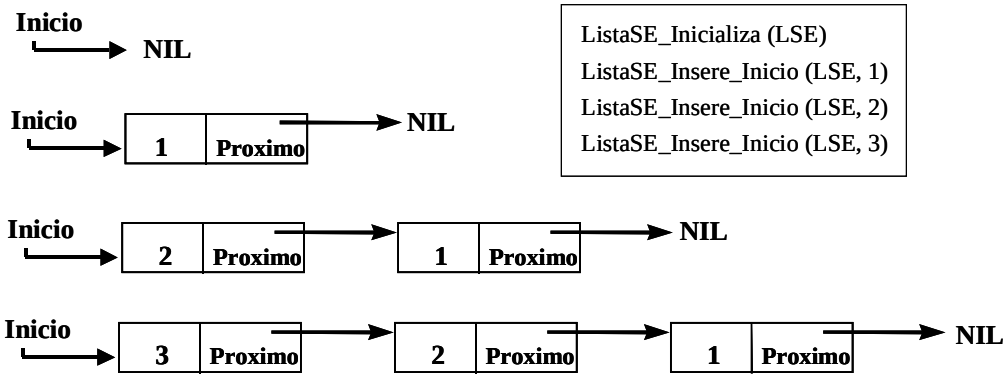
Uma lista simplesmente encadeada deverá ser constituída obrigatoriamente dos seguintes elementos:

- **NODO:** Registro de dados contendo um (ou mais) campo(s) para armazenar os dados e um campo que será do tipo "ponteiro para o próprio nodo" (elo/encadeamento);
- **INÍCIO:** Ponteiro que irá preservar (apontar) o nodo inicial da Lista Encadeada. Este ponteiro é indispensável para que seja possível localizar os dados da lista simplesmente encadeada na memória. *Se o início da lista for perdido, toda a lista será perdida!*

Além destes elementos obrigatórios, poderemos ainda ter um elemento opcional que é o ponteiro indicando o **FINAL da lista** (aponta para o último nodo da lista encadeada). O ponteiro para o final da lista pode ser particularmente útil se estivermos interessados em inserir novos nodos no final da lista, sem ter que obrigatoriamente percorrer toda a lista para alcançar o seu final.

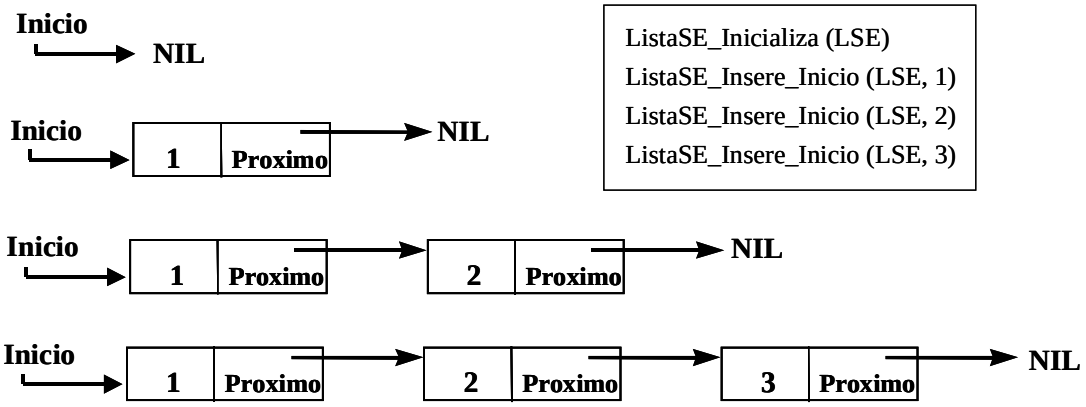
De acordo com o indicado acima, podemos facilmente identificar que existem diferentes maneiras de inserir e remover dados de uma lista encadeada, onde o fato da lista ser simplesmente ou duplamente encadeada vai simplificar e/ou dificultar algumas destas operações. Os *tipos básicos* de operações de inserção em listas encadeadas são:

1) **Inserção no início da lista:** Os nodos são sempre alocados (NEW) e depois inseridos antes do primeiro nodo da lista encadeada, tornando-se assim o novo início da lista. Exemplo:

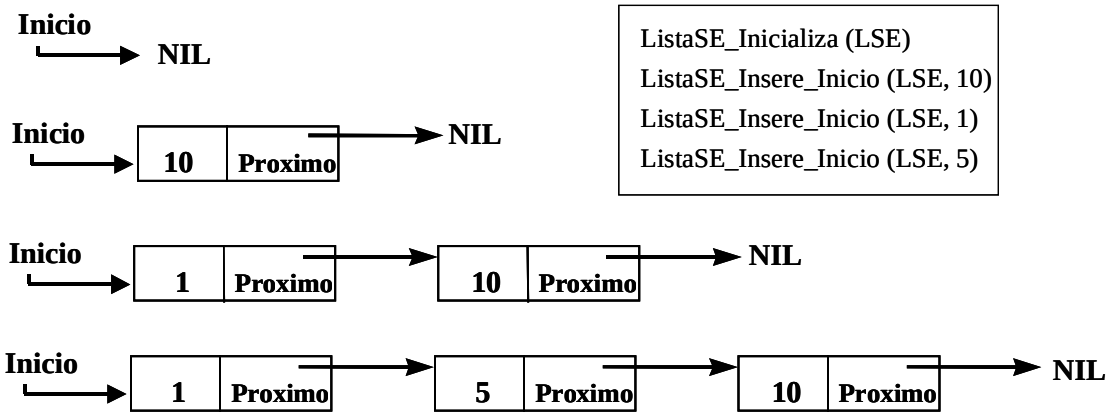



```
ListaSE_Inicializa (LSE)
ListaSE_Insere_Inicio (LSE, 1)
ListaSE_Insere_Inicio (LSE, 2)
ListaSE_Insere_Inicio (LSE, 3)
```

2) **Inserção no final da lista:** Os nodos são encadeados após o último nodo da lista, ou seja, o nodo inserido torna-se o último nodo da lista. Exemplo:



3) **Inserção ordenada:** Os nodos são inseridos respeitando uma regra de ordenação previamente estabelecida, por exemplo, em ordem crescente de um campo numérico. Exemplo:



4) **Inserção especial:** rotinas para implementar inserção em Pilha, Fila ou Deque.
Rotinas baseada nas rotinas acima de insere/remove no início/fim.

Além das rotinas de inserção, também iremos identificar que existem diferentes maneiras de remover dados de uma lista encadeada. Lembre-se que na remoção de dados de uma lista é muito importante que a memória seja liberada (DISPOSE) de modo a poder ser aproveitada por outras aplicações e/ou pedidos de alocação de memória. Os tipos básicos de operações de remoção em listas encadeadas são:

1) **Remoção no início da lista:** O primeiro nodo da lista encadeada nodos é sempre o que será removido, onde o nodo seguinte a ele torna-se assim o novo início da lista.

2) **Remoção do final da lista:** O último nodo da lista encadeada é removido, onde o penúltimo nodo torna-se o último após a remoção.

3) **Remoção de um nodo específico indicado por um ponteiro:** Neste caso, é necessário que o nodo tenha sido previamente localizado e que já tenhamos um ponteiro apontando para o nodo que se deseja remover. A remoção deste nodo implica em criar um elo de ligação entre seu anterior e o próximo nodo (caso existam).

4) **Remoção de um nodo contendo um determinado dado:** Este tipo de remoção implica em primeiramente localizar o dado (a sua primeira ocorrência, ou, repetir esta operação para todas as ocorrências deste mesmo dado presentes na lista) e em seguida realizar a remoção deste nodo, de forma análoga a remoção de um nodo específico indicado por um ponteiro (vide item 3).

5) **Remoção de todos os nodos da lista:** Todos os nodos da lista devem ser devidamente removidos e a memória ocupada por eles liberada, e por fim o ponteiro para o início da lista deve ser inicializado novamente (NIL). Lembre-se que não basta apenas atribuir NIL ao ponteiro de início de lista, devemos usar o comando DISPOSE para realmente liberar a memória alocada.

Vamos definir a seguir um conjunto de **rotinas genéricas** para manipulação de Listas Simplesmente Encadeadas, que simplificarão o seu uso e sua adaptação para uso em outras aplicações.

Definição dos Dados:

```
Type
    TListaSE_Dado     = Integer;

    Ptr_ListaSE_Nodo  = ^ListaSE_Nodo;

    ListaSE_Nodo      = Record
                            Dado: TListaSE_Dado;
                            Prox : Ptr_ListaSE_Nodo;
                        End;
```

Definição das Rotinas:

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

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  }
Var
   novo: Ptr_ListaSE_Nodo;
Begin
  New (novo);
  novo^.Dado := Dado;
  novo^.Prox := LSE;
   LSE := novo;
End;
```

```
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   }
Var
   novo, aux: Ptr_ListaSE_Nodo;
Begin
   New (novo);
   novo^.Dado := Dado;
   novo^.Prox := NIL;

   If  LSE = NIL  Then  LSE := novo              { Lista vazia ? Insere 1º. nodo }
   Else   Begin
               aux := LSE;
               While  aux^.Prox  <>  NIL         { Avança até o último nodo }
               Do      aux :=  aux^.Prox;
               aux^.Prox := novo;
          End;
End;
```

```
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   }
Var
   novo, aux, ant: Ptr_ListaSE_Nodo;
Begin
   New (novo);
   novo^.Dado := Dado;

   aux := LSE;
   While  (aux <> NIL)  AND (aux^.Dado < Dado)
   Do      Begin
               ant := aux;
               aux := aux^.Prox;
           End;

   If  aux = LSE  Then  LSE := novo        { Lista vazia ? }
   Else   ant^.Prox := novo;

   novo^.prox := aux;
End;
```

```
Function ListaSE_Remove_Inicio (Var LSE: Ptr_ListaSE_Nodo;  Var Dado: TListaSE_Dado): Boolean;
{ Remove o primeiro dado da lista }
Var
  aux: Ptr_ListaSE_Nodo;
Begin
   aux := LSE;
   If   aux = NIL  Then ListaSE_Remove_Inicio := False
   Else  Begin
              Dado := LSE^.Dado;
              LSE := LSE^.Prox;
              Dispose (aux);
              ListaSE_Remove_Inicio := True;
         End;
End;
```

```
Function ListaSE_Remove_Final (Var LSE: Ptr_ListaSE_Nodo; Var Dado: TListaSE_Dado): Boolean;
{ Remove o último dado da lista }
Var
   aux: Ptr_ListaSE_Nodo;
Begin
   aux := LSE;
   If  aux = NIL
   Then  ListaSE_Remove_Final := False
   Else   Begin
             If  aux = NIL      { Lista vazia ? }
             Then  ListaSE_Remove_Final := False
             Else   Begin
                       If aux^.Prox = NIL
                       Then  Begin
                                 LSE := NIL;
                                 Dado := aux^.Dado;
                                 Dispose (aux);
                             End
                       Else    Begin
                                  While  aux^.Prox^.Prox  <> NIL  { Localiza o penúltimo nodo }
                                  Do  aux := aux^.Prox;
                                  Dado := aux^.Prox^.Dado;
                                  Dispose (aux^.Prox);
                                  aux^.Prox := NIL;
                             End;
                       ListaSE_Remove_Final := True;
                    End;
          End;
End;

Procedure ListaSE_Exibe (LSE: Ptr_ListaSE_Nodo);
{ Exibe na tela o conteúdo da lista encadeada }
Var
   aux: Ptr_ListaSE_Nodo;
Begin
   aux := LSE;
   While  aux <> NIL
   Do  Begin
          Writeln (aux^.Dado);
          aux := aux^.prox;
       End;
   Write ('Tecle <enter> para continuar...');
   Readln;      { Faz uma pausa... }
End;
```

Rotinas complementares...

```
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_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 }

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 }

Procedure ListaSE_Grava (Var ArqTxt: Text; LSE: Ptr_ListaSE_Nodo);
{ Salva em disco os dados da lista de modo que esta possa ser posteriormente recuperada }

Function ListaSE_Le (Var ArqTxt: Text; Var LSE: Ptr_ListaSE_Nodo);
{ Lê os dados do disco e insere na LSE, na mesma ordem em que se encontram no arquivo }

ATENÇÃO: ***Ponteiros não devem jamais serem salvos em disco!***

Exemplo de uso das rotinas...

```
Program Testa_Lista_Simplesmente_Encadeada;

>>> Incluir rotinas de LSE, {$I Lse.pas }, ou  utilizar uma Unit Lse.tpu: “Uses  Lse;”  <<<

Var
   Lista1, Lista2: Ptr_ListaSE_Nodo;
   Dado:TlistaSE_Dado;
Begin
   ListaSE_inicializa (Lista1);
   ListaSE_Insere_Inicio (Lista1,10);
   ListaSE_Insere_Inicio (Lista1,20);
   ListaSE_Insere_Inicio (Lista1,30);
   writeln(‘Lista Simplesmente Encadeada com inserção no início:‘);
   ListaSE_Exibe(Lista1);

   ListaSE_inicializa (Lista2);
   ListaSE_Insere_Final (Lista2,10);
   ListaSE_Insere_Final (Lista2,20);
   ListaSE_Insere_Final (Lista2,30);
   writeln(‘Lista Simplesmente Encadeada com inserção no final:‘);
   ListaSE_Exibe(Lista2);

End.
```

EXERCÍCIOS:

1) Implemente e teste rotinas de manipulação de pilha (insere_pilha = Push e remove_pilha = Pop) baseadas nas rotinas de manipulação de listas simplesmente encadeadas.

2) Implemente e teste rotinas de manipulação de filas (insere_fila e remove_fila) baseadas nas rotinas de manipulação de listas simplesmente encadeadas.

3) Altere as rotinas já desenvolvidas de forma a implementar a manipulação de lista simplesmente encadeada de modo a adotar um ponteiro para início e um ponteiro para o final da lista, ou seja, todas as rotinas sempre receberão como parâmetros 2 ponteiros, um que aponta para o início e outro que aponta para o final da lista (ao contrário das rotinas atuais que recebem sempre apenas 1 ponteiro). As rotinas devem ser adaptadas de modo a (i) manter corretamente atualizados os ponteiros de início e final de lista e (ii) otimizar as rotinas que podem tirar proveito do uso de um ponteiro que já esteja posicionado no final da lista.

4) Faça um programa, usando as rotinas de manipulação de listas simplesmente encadeadas, para gerenciar o estoque de produtos de uma loja de venda por atacado. Este programa deve possuir as seguintes opções: 1) Compra de produtos - inserir no estoque; 2) Venda de produtos - dar baixa no estoque; 3) Listar quantidade de produtos em estoque; 4) Consulta produto; 5) Gravar em disco os dados; 6) Ler do disco os dados. Cada produto deve ser descrito pelo seu código, nome, quantidade e preço. Mantenha a lista de produtos em estoque ordenada pelo código do produto, sendo que um mesmo produto (produto com o mesmo código) não devem ser cadastrado mais de uma vez, onde devemos sempre manter um registro do total de produtos de um determinado código. Segue abaixo um exemplo de interação entre o usuário e o programa que você deve implementar:

```
>> Loja - Atacado ACME <<
1) Compra de produtos
2) Venda de produtos
3) Listar produtos
4) Consultar o estoque de produtos
5) Gravar em disco o cadastro
6) Ler do disco o cadastro
0) Sair do programa

Entre com a sua opção: 1
> Código (4 dígitos): 123
> Descrição: Dinamite ACME
> Quantidade: 10
> Preço unitário: 1.00
Cadastrar mais produtos (s/n) ? s
> Código (4 dígitos): 456
> Descrição: Tinta invisível ACME
> Quantidade: 10
> Preço Unitário: 10.00
Cadastrar mais produtos (s/n) ? n

Entre com a sua opção: 2
> Código (4 dígitos): 123
= Produto: Dinamite ACME
> Quantidade: 2
= Total a ser pago: 2.00
> Confirma (s/n) ? s

Entre com a sua opção: 3
123 - Dinamite ACME - 8 und. - R$1.00
456 - Tinta invisível ACME - 10 und. - R$10.00

Entre com a sua opção: 4
> Código: 456
= Tinta invisível ACME - 10 und. - R$10.00

Entre com a sua opção: 5
= Arquivo "estoque.txt" salvo no disco

Entre com a sua opção: 6
Dados do disco vão sobrepor dados atuais (s/n)? s
= Arquivo "estoque.txt" lido do disco
```