UNISINOS - **UNIVERSIDADE DO VALE DO RIO DOS SINOS**
CIÊNCIAS EXATAS E TECNOLÓGICAS – Curso: Informática / Ciência da Computação

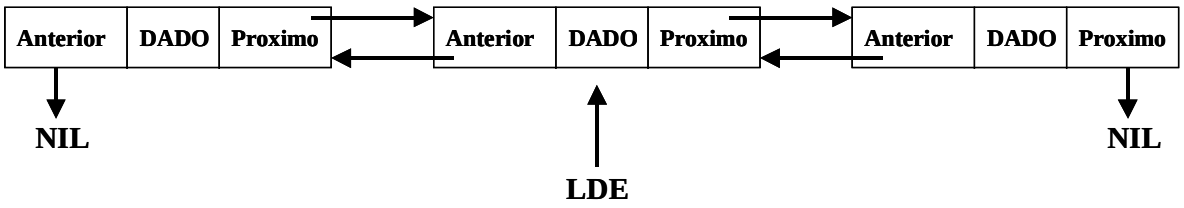
Programação II

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 - LISTAS DUPLAMENTE ENCADEADAS

LISTAS DUPLAMENTE ENCADEADAS

Cada nodo de uma lista duplamente encadeada possui 2 (dois) elos de encadeamento – *anterior* e *próximo*, possibilitando que a lista seja percorrida nos dois sentidos. Sendo assim, podemos a partir de um ponteiro para um nodo qualquer da lista, alcançar as suas duas extremidades, onde podemos nos deslocar tanto do início para o final, quanto do final para o início da lista. Esquema:



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

- **NODO:** Registro de dados contendo um (ou mais) campo(s) para armazenar os dados e dois campos que serão do tipo "ponteiro para o próprio nodo" (elo/encadeamento). Os ponteiros devem apontar para o nodo anterior (ou NIL se este for o primeiro nodo da lista) e para o nodo seguinte (ou NIL se este for o último nodo da lista). Obs.: em casos muito raros e específicos podemos imaginar uma lista duplamente encadeada circular, onde o primeiro nodo aponta para o último e o último aponta para o primeiro, mas isto não é usual.

- **PONTEIRO:** Ponteiro que irá apontar para um dos nodos da Lista Encadeada. Este ponteiro é indispensável para que seja possível localizar os dados da lista duplamente encadeada na memória. Este ponteiro pode estar posicionado no início da lista, no final da lista, ou mesmo em qualquer outro elemento da lista, podendo estar fixo em um nodo/extremidade, ou mesmo, se deslocar livremente pelos nodos que compõem a lista.

Note que sempre teremos um compromisso entre performance, flexibilidade e requisitos de alocação de memória, relacionados com a opção que fazemos por uma ou outra ou outra estrutura de dados. No caso da lista simplesmente encadeada, os nodos são menores (1 ponteiro por dado armazenado), mas onde a flexibilidade de deslocamento entre os nodos é menor (só em um sentido) e mesmo a performance pode ser menor em operações como a inserção no final da lista (considerando as rotinas estudadas anteriormente). As listas duplamente encadeadas irão ocupar mais memória (2 ponteiros para cada dado), mas são mais flexíveis e podem ser mais eficientes na em certas operações (por exemplo, na inserção/remoção do final da lista).

De acordo com o indicado acima, podemos facilmente identificar que existem diferentes maneiras de inserir e remover dados de uma lista duplamente encadeada, onde o fato da lista ser. Em nossas rotinas vamos criar formas de deslocar o ponteiro para as extremidades (posiciona inicio/fim), o que irá facilitar as operações de inserção/remoção de nodos. Os *tipos básicos* de operações de inserção em listas encadeadas continuarão sendo:

- **Inserção no início da lista**
- **Inserção no final da lista**
- **Inserção ordenada**

Mas poderemos considerar que estas rotinas serão derivadas do uso de outras rotinas mais genéricas que são:

- **Posiciona ponteiro no início da lista**
- **Posiciona ponteiro no final da lista**
- **Posiciona ponteiro em um nodo específico (pesquisa dado)**
- **Inserção antes de um determinado nodo**
- **Inserção após um determinado nodo**

Além das rotinas de inserção, também iremos identificar que existem diferentes maneiras de remover dados de uma lista encadeada. No caso da lista duplamente encadeada, vamos considerar apenas 2 modos de remoção: **Remoção de um nodo específico indicado por um ponteiro** e **Remoção de um nodo contendo um determinado dado** (procura um dado e se achar e remove). As rotinas de remove nodo do início/fim da lista podem ser compostas com o auxílio da rotina posiciona no início/fim e a remove o nodo correntemente apontado.

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

Definição dos Dados:

```
Type
    TListaDE_Dado     =  Integer;

    Ptr_ListaDE_Nodo  =  ^ListaDE_Nodo;

    ListaDE_Nodo      =  Record
                           Dado: TListaDE_Dado;
                           Ant, Prox : Ptr_ListaDE_Nodo;
                         End;
```

Definição das Rotinas:

```
Procedure ListaDE_Inicializa (Var LDE: Ptr_ListaDE_Nodo);
{   LDE  = Ponteiro para o início da Lista Duplamente Encadeada, inicializa com NIL  }
Begin
   LDE := NIL;
End;
```

```
Procedure ListaDE_Posiciona_Inicio (Var LDE: Ptr_ListaDE_Nodo);
{ Move o ponteiro para o inicio da lista }
Begin
    If  LDE <> NIL
    Then  While  LDE^.ant  <> NIL
            Do       LDE := LDE^.ant;
End;

Procedure ListaDE_Posiciona_Final (Var LDE: Ptr_ListaDE_Nodo);
{ Move o ponteiro para o final da lista }
Begin
    If  LDE <> NIL
    Then  While LDE^.prox  <> NIL
            Do      LDE := LDE^.prox;
End;


Procedure ListaDE_Insere_Antes (Var LDE: Ptr_ListaDE_Nodo; Dado: TListaDE_Dado);
{   Insere um dado "antes" do nodo correntemente apontado }
Var
   novo,aux: Ptr_ListaDE_Nodo;
Begin
   New (novo);
   novo^.Dado := Dado;

   If LDE = NIL
   Then Begin
              LDE := novo;
              novo^.ant   := NIL;
              novo^.prox := NIL;
         End
   Else  Begin
              aux := LDE^.ant;
              LDE^.ant     := novo;
              novo^.prox  := LDE;
              novo^.ant    := aux;
              If aux <> NIL
              Then aux^.prox := novo;
         End;
    LDE := novo; { Opcional: LDE passa a apontar para o novo nodo }
End;

Procedure ListaDE_Insere_Depois (Var LDE: Ptr_ListaDE_Nodo; Dado: TListaDE_Dado);
{   Insere um dado "depois" do nodo correntemente apontado }
Var
    novo,aux: Ptr_ListaDE_Nodo;
Begin
    New (novo);
    novo^.Dado := Dado;
```

```
    If  LDE = NIL
   Then Begin
              LDE := novo;
              novo^.ant   := NIL;
              novo^.prox := NIL;
         End
   Else Begin
              aux := LDE^.prox;
              LDE^.prox  := novo;
              novo^.ant    := LDE;
              novo^.prox  := aux;
              If  aux <> NIL
              Then aux^.ant := novo;
         End;
   LDE := novo; { Opcional: LDE passa a apontar para o novo nodo }
End;
```

Procedure ListaDE_Exibe (LDE: Ptr_ListaDE_Nodo);

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

Function ListaDE_Pesquisa_Dado (Var LDE: Ptr_ListaDE_Nodo;
Dado:TListaDE_Dado): Boolean;

```
{ Procura na lista pela primeira ocorrência do dado e retorna se achou ou não e o ponteiro }
Begin
    If  LDE <> NIL
    Then Begin
              ListaDE_Posiciona_Inicio(LDE);
              While  (LDE^.dado <> Dado) AND (LDE^.prox <> NIL)
              Do  LDE := LDE^.prox;
              If  LDE^.dado = Dado
              Then ListaDE_Pesquisa_Dado := True
              Else ListaDE_Pesquisa_Dado  := False;
         End
    Else  ListaDE_Pesquisa_Dado := False;
End;
```

```
Function ListaDE_Pesquisa_Prox (Var LDE: Ptr_ListaDE_Nodo;
                                Dado: TListaDE_Dado): Boolean;
{ Procura na lista pela próxima ocorrência do dado a partir da posição atual
  e retorna se achou ou não e o ponteiro }
Begin
    If (LDE <> NIL)
    Then Begin
              While (LDE^.dado <> Dado) AND (LDE^.prox <> NIL)
              Do LDE := LDE^.prox;
              If   LDE^.dado = Dado
              Then ListaDE_Pesquisa_Prox := True
              Else ListaDE_Pesquisa_Prox := False;
            End
    Else ListaDE_Pesquisa_Prox := False;
End;

Function ListaDE_Remove_Nodo (Var LDE: Ptr_ListaDE_Nodo;
                              Var Dado: TListaDE_Dado): Boolean;
{ Remove o nodo correntemente apontado, retornando seu conteudo e se teve sucesso }
Var
   aux, ant,prox: Ptr_ListaDE_Nodo;
Begin
    If  LDE <> NIL
    Then Begin
              aux   := LDE;
              dado := LDE^.dado;
              ant    := LDE^.ant;
              prox  := LDE^.prox;
              If  ant <> NIL
              Then  ant^.prox := prox;
              If  prox <> NIL
              Then  prox^.ant := ant;

              If  ant <> NIL
              Then LDE := ant
             Else  If  prox <> NIL
                    Then LDE := prox
                    Else  LDE := NIL;

             dispose(aux);
             ListaDE_Remove_Nodo := True;
           End
    Else ListaDE_Remove_Nodo := False;
End;

Function ListaDE_Remove_Dado (Var LDE: Ptr_ListaDE_Nodo;
                              Dado: TListaDE_Dado): Boolean;
{ Procura pela primeira ocorrência do nodo a partir da posição atual, e se
  encontrar remove, retornando seu conteúdo e se teve sucesso }
Var
   dadoaux: TListaDE_Dado;
   achou:boolean;
```

```
Begin
    achou:=ListaDE_Pesquisa_Dado (LDE, Dado);
    If achou
    Then ListaDE_Remove_Dado := ListaDE_Remove_Nodo(LDE,dadoaux)
    Else  ListaDE_Remove_Dado := False;
End;
```

Rotinas complementares...

Function ListaDE_Quantidade (LDE: Ptr_ListaDE_Nodo): Integer;
{ Retorna a quantidade total de nodos da lista }

Function ListaDE_Proximo (Var LDE: Ptr_ListaDE_Nodo;
Var Dado: TListaDE_Dado): Boolean;
{ Dado um nodo apontado por LDE, avança o ponteiro p/o nodo seguinte, e retorna seu valor }

Function ListaDE_Anterior (Var LDE: Ptr_ListaDE_Nodo;
Var Dado: TListaDE_Dado): Boolean;
{ Dado um nodo apontado por LDE, avança o ponteiro p/o nodo anterior, e retorna seu valor }

Procedure ListaDE_Apaga (Var LDE: Ptr_ListaDE_Nodo);
{ Remove todos os nodo da lista }

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

Function ListaDE_Le (Var ArqTxt: Text; Var LDE: Ptr_ListaDE_Nodo);
{ Lê os dados do disco e insere na LDE, 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_Duplamente_Encadeada;

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

Var
   Lista: Ptr_ListaDE_Nodo;
   Dado:TlistaDE_Dado;
Begin
   writeln('>>> Teste das rotinas LDE <<<');
   writeln;
   writeln('# Insere dados no inicio: 10, 20, 30');
   ListaDE_inicializa (Lista);
   ListaDE_Insere_Antes (Lista,10);
   ListaDE_Insere_Antes (Lista,20);
   ListaDE_Insere_Antes (Lista,30);

   writeln('# Lista Duplamente Encadeada com insercao no inicio:');
   ListaDE_Exibe(Lista);
```

```
    writeln('# Insere dados. Adiciona dados no final: 40, 50, 60');
    ListaDE_Posiciona_Final(Lista);
    ListaDE_Insere_Depois (Lista,40);
    ListaDE_Insere_Depois (Lista,50);
    ListaDE_Insere_Depois (Lista,60);

    writeln('# Lista Duplamente Encadeada com insercao inicio e no final:');
    ListaDE_Exibe(Lista);
End.
```

EXERCÍCIOS:

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

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

3) Baseado nas rotinas já desenvolvidas, implemente um novo conjunto de rotinas de manipulação de listas duplamente encadeadas, do tipo DEQUE, adotando 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). Note que neste caso, as rotinas posiciona início e posiciona final não serão mais necessárias. 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 dos ponteiros que já estão posicionados no início e no final da lista.