

ESTRUTURA DE DADOS – LISTAS LINEARES

| | | |
|----------|---|----------|
| 4 | LISTAS LINEARES | 1 |
| 4.1 | DEFINIÇÃO | 1 |
| 4.2 | OPERAÇÕES MAIS COMUNS (A SEREM DESENVOLVIDAS NO MI-LISTA)..... | 1 |
| 4.3 | REPRESENTAÇÕES DE LISTAS | 1 |
| 4.4 | CONTIGÜIDADE DOS NÓS (TAMBÉM CHAMADO DE ALOCAÇÃO SEQUÊNCIAL)..... | 2 |
| 4.5 | ENCADEAMENTO DOS NÓS (OU ALOCAÇÃO ENCADEADA OU DINÂMICA) | 5 |
| 4.5.1 | <i>Lista Simplesmente Encadeada</i> | 5 |
| 4.5.2 | <i>Lista com Descritor</i> | 11 |
| 4.5.3 | <i>Lista Duplamente Encadeada</i> | 12 |
| 4.5.4 | <i>Listas Circulares</i> | 13 |

4 LISTAS LINEARES

4.1 Definição

A estrutura que permite representar um conjunto de dados de forma a preservar a relação de ordem linear entre eles é a **lista linear**. Uma lista linear é composta de nós, os quais podem conter, cada um deles, um tipo primitivo ou um tipo definido pelo programador.

Uma lista linear é uma coleção $L = [x_1, x_2, \dots, x_n]$, com $n \geq 0$, cuja propriedade estrutural baseia-se apenas na posição relativa dos elementos, que são dispostos linearmente. Se $n = 0$, diz-se que a lista L é **vazia**; caso contrário, são válidas as seguintes propriedades:

- x_1 é o primeiro elemento (nó) de L ;
- x_n é o último elemento (nó) de L ;
- x_k , $1 \leq k \leq n$, é precedido pelo nó x_{k-1} e seguido por x_{k+1} em L .

Em outras palavras, a característica fundamental de uma lista linear é o sentido de ordem unidimensional dos elementos que a compõem. Uma ordem que permite dizer com precisão onde a coleção inicia-se e onde termina, sem possibilidade de dúvida.

4.2 Operações mais comuns (a serem desenvolvidas no MI-LISTA)

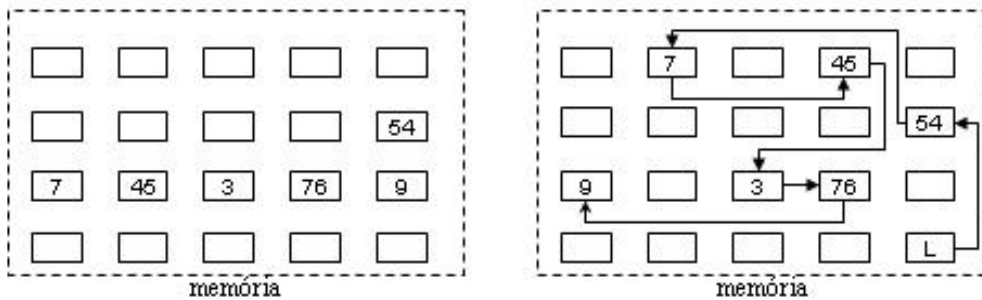
Uma vez decidido que um conjunto de dados será representado sob a forma de uma lista linear, deve-se também decidir que operações podem (ou devem) ser efetuadas sobre esta lista. Alguns exemplos destas operações são:

- a) acessar o nó x_k contido numa determinada posição (para obter o dado ou alterá-lo);
- b) inserir um novo nó (antes ou depois do nó x_k ou no extremo da lista);
- c) eliminar o nó x_k ;
- d) concatenar duas listas;
- e) determinar o número de nós (n) de uma lista (comprimento);
- f) localizar o nó que contém um determinado dado;
- g) partir uma lista em duas ou mais;
- h) fazer uma cópia de uma lista;
- i) ordenar os nós de uma lista;
- j) etc.

4.3 Representações de Listas

A seguir serão discutidas as duas formas mais freqüentemente usadas para representar listas lineares: o **encadeamento** dos nós e a **contigüidade** dos mesmos. A escolha da forma de representação dependerá da freqüência com que determinadas operações serão executadas sobre a lista, uma vez que algumas representações são favoráveis a algumas operações, enquanto que outras não o são, no sentido de exigir um maior esforço computacional para a sua execução.

A seguir é apresentado um exemplo de uma lista de números inteiros $L = [54, 7, 45, 3, 76, 9]$ mostrando ambas as representações: **contigüidade** e **encadeamento**. Na representação por contigüidade, os números (elementos da lista) estão contíguos na memória, isto é, um vem após o outro. Na representação por encadeamento, os elementos estão espalhados pela memória.



4.4 Contigüidade dos Nós (também chamado de alocação seqüencial)

A representação por contigüidade é considerada a possibilidade mais natural para colocar-se elementos no interior de uma lista. Assume-se simplesmente que a colocação dos elementos de uma lista ocorre em células com memórias consecutivas, uma após a outra. Neste caso a lista é representada por um vetor da forma:

```
.....
tipo LISTA = vet [1..n] tipo-t;
.....
LISTA: X;
```

sendo tipo-t o tipo do dado de um nó da lista.

Esquemáticamente:

| | | | | | | |
|-----|----------------|----------------|----------------|-----|------------------|----------------|
| I = | 1 | 2 | 3 | ... | n - 1 | n |
| X = | X ₁ | X ₂ | X ₃ | ... | X _{n-1} | X _n |

Outra forma é considerar ainda uma representação onde a lista de n nós é inicialmente parte de um vetor de m elementos, sendo $m \geq n$. Isto possibilita alterar a quantidade de nós (comprimento) da lista durante a execução. Desta forma tem-se:

```
.....
tipo LISTA = vet [1..m] tipo-t;
.....
LISTA: X;
```

Esquemáticamente:

| | | | | | | | | | |
|-----|----------------|----------------|----------------|-----|------------------|----------------|-----|------------------|----------------|
| I = | 1 | 2 | 3 | ... | n - 1 | n | ... | m - 1 | m |
| X = | X ₁ | X ₂ | X ₃ | ... | X _{n-1} | X _n | ... | X _{m-1} | X _m |

----- lista inicial (com n nós) -----

----- vetor X (com m elementos) -----

Neste caso deve-se conhecer o valor de n , isto é, o índice de X_n correspondente ao **último** nó da lista. A seguir são apresentados dois procedimentos do MI-LISTA, considerando X um vetor com m elementos.

Para procurar um determinado elemento VAL na lista, é necessário percorrê-la do início até o fim (1 até n). Quando o elemento VAL é encontrado, guarda-se o índice de VAL e a função termina retornando o índice de VAL.

Determina o índice do nó que tem o valor VAL na lista contígua X:

```

função IndiceCont(LISTA: X, int:N, VAL:tipo-t):int
{verifica se o nó VAL está na lista X e retorna a posição (índice) de sua primeira ocorrência}
  int:I;
  I ← 1;
  enquanto X[I] ≠ VAL e I ≤ N faça
    I ← I + 1;
  fim-enquanto;
  se I ≤ N entao
    IndiceCont ← I; {retorna o valor de I, índice de VAL na lista X}
  senao
    IndiceCont ← -1; {retorna -1 se não achou VAL na lista X}
  fim-se;
fim-funcao;

```

Na inserção de um novo elemento no meio da lista, na posição K, é necessário o deslocamento de todos os itens localizados após o ponto da inserção para que se abra o espaço a ser ocupado pelo novo elemento. O tamanho da lista deve ser incrementado de 1.

Inserir um novo nó na posição K da lista contígua X:

```

função InserirCont(LISTA: L; ref int:N; int: K; NOVO_NO:tipo-t):LISTA
{Inserir o nó NOVO_NO na posição K}
  int:I;
  I ← N;
  se K ≤ N e K ≥ 1 entao
    {desloca para a direita todos os nós de L após o índice K}
    enquanto I ≥ K faça
      L[I + 1] ← L[I];
      I ← I - 1;
    fim-enquanto;
    L[K] ← NOVO_NO;
    Inserircont ← L; {retorna a lista L com o novo no índice K}
    N ← N+1; {incrementa o tamanho da lista, N é passado por referência}
  senao
    imprima("Índice inválido");
  fim-se;
fimfuncao;

```

A função a seguir retira da lista o nó X_k . Os nós remanescentes, localizados após o nó retirado, são deslocados para preencher o espaço vazio deixado. O tamanho da lista deve ser decrementado de 1.

Retirar o nó X_k da lista:

```

função RetiraCont(LISTA: X, ref int:N, Xk:tipo-t):LISTA
{Retira o nó  $X_k$  da lista X. O valor de N é decrementado de 1}
  int:K,I;
  K ← IndiceCont(X, N, Xk); {determina o índice de  $X_k$ }
  se K ≠ -1 entao {achou o índice de  $X_k$ }
    para I de K até N - 1 faca
      X[I] ← X[I + 1]; {desloca para a esquerda os nós após o nó  $X_k$ }
    fimpara;
    N ← N - 1; {decrementa o tamanho da lista}
  senao
    imprima("Nó não encontrada na lista");
  fimse;
  RetiraCont ← X;
fimfuncao;

```

Complexidade ou ordem de grandeza

Como vimos no capítulo 1, a complexidade em tempo de um programa ou uma rotina nos dá o número de operações realizadas por um determinado programa ou rotina em função do tamanho da entrada; no caso N , o número de elementos na lista. Para as funções desenvolvidas anteriormente, têm-se as seguintes complexidades:

| Rotina | Complexidade |
|-------------|--------------|
| RetiraCont | $O(N)$ |
| InserirCont | $O(N)$ |
| IndiceCont | $O(N)$ |

Isso quer dizer que se, por exemplo, $N = 10000$, no pior caso, estas funções realizarão 10000 operações.

Aplicações:

Desenvolver o TAD LISTA CONTÍGUA com n elementos, considerando X um vetor com m elementos (capacidade máxima da lista), com as seguintes operações:

- Criar o TAD;
- Alterar o valor do nó X_k para VAL ;
- Localizar todos os índices do nó que contém o valor VAL , considerando que a lista pode possuir nós iguais.
- concatenar duas listas;
- partir uma lista em duas ou mais;
- fazer uma cópia de uma lista;
- ordenar os nós de uma lista,

Outra representação contígua:

| | | | | | | | | | |
|-------|---|-------|-----|-----------|-----|-----------|-----|-----------|-------|
| $I =$ | 1 | 2 | ... | pri | ... | ult | ... | m-1 | m |
| $X =$ | X_1 | X_2 | ... | X_{pri} | ... | X_{ult} | ... | X_{m-1} | X_m |
| | --- lista inicial (com n nós) --- | | | | | | | | |
| | ----- vetor X (com m elementos) ----- | | | | | | | | |

Uma vantagem da representação contígua que se pode ressaltar é a facilidade para calcular o endereço de memória de um certo elemento de índice I . Já uma desvantagem que aparece na representação contígua de nós de uma lista surge no esforço do momento de inserir ou remover nós no meio da lista. Isso ocorre porque é necessário movimentar os nós para liberar (inserção) ou diminuir (remoção) o espaço entre eles.

Exercícios

- Dadas duas listas $L1$ e $L2$, que possam estar desordenadas e contendo elementos repetidos :
- Verifique se $L1$ está ordenada ou não (a ordem pode ser crescente ou decrescente)
- Faça uma cópia da lista $L1$ em outra lista $L2$;
- Faça uma cópia da Lista $L1$ em $L2$, eliminando elementos repetidos;
- Inverta $L1$ colocando o resultado em $L2$;
- Inverta $L1$ colocando o resultado na própria $L1$;
- Intercale $L1$ com a lista $L2$, gerando a lista $L3$. Considere que $L1$, $L2$ são ordenadas.
- Gere uma lista $L2$ onde cada registro contém dois campos de informação: `elem` contém um elemento de $L1$, e `count` contém quantas vezes este elemento apareceu em $L1$

9. Eliminam de L1 todas as ocorrências de um elemento dado, L1 ordenada;
10. Assumindo que os elementos da lista L1 são inteiros positivos, forneça os elementos que aparecem o maior e o menor número de vezes (forneça os elementos e o número de vezes correspondente).
11. Usando lista simplesmente encadeada, implementar um Cadastro de Clientes de uma empresa com as seguintes especificações: Campos: Código, Nome e Cidade. Operações: Inclusão (no final da Lista), Exclusão de um Nome, Pesquisa de um Nome, Alteração de qualquer dado (dado um Nome) e, por fim, Listagem dos dados na tela;

4.5 Encadeamento dos nós (ou alocação encadeada ou dinâmica)

A representação por encadeamento do nó é segunda forma para agrupar os elementos numa lista linear. Nesse caso, em vez de manter os elementos em células contíguas, pode-se utilizar quaisquer células (não necessariamente consecutivas) para guardar os elementos da lista. Esse esquema implica que, para preservar a relação de ordem linear da lista, cada elemento armazena sua informação e também o endereço da próxima célula de memória válida.

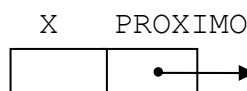
4.5.1 Lista Simplesmente Encadeada

Neste caso, um nó deve conter além de seu valor, uma indicação (apontador, ponteiro ou referência) para o nó seguinte, representando uma contigüidade lógica.

Esquematicamente:



Sendo um nó qualquer representado por:



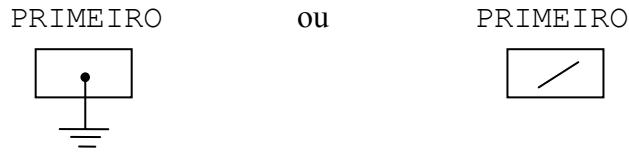
Assim, pode-se construir a seguinte estrutura (recursiva):

```
tipo R = ref NO;
tipo NO = reg(tipo-t: X, R: PROXIMO);
```

É necessário ainda declarar uma variável do tipo R para indicar o **primeiro** nó da lista, que deve ser inicializada com nil para criar uma lista encadeada vazia:

```
PRIMEIRO ← nil;
```

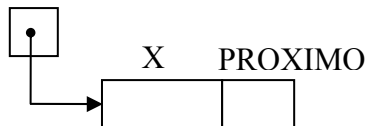
O que pode ser representado por:



Sendo assim, pode-se desenvolver a função para se criar uma lista vazia:

```
funcao CriaListaVazia:R
  CriaListaVazia ← nil;
fim-funcao;
```

Para se criar um nó, é necessário usar uma variável auxiliar P do tipo R (ref NO) e executar o comando aloque (P) , obtendo:



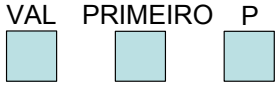

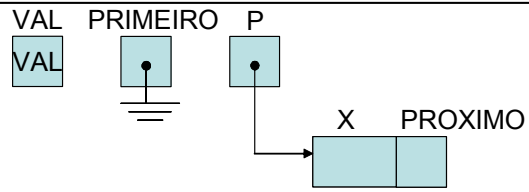
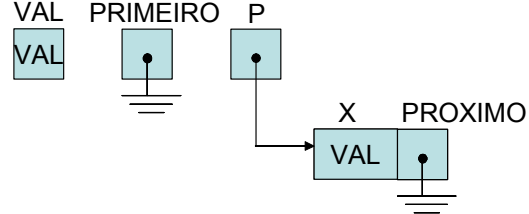
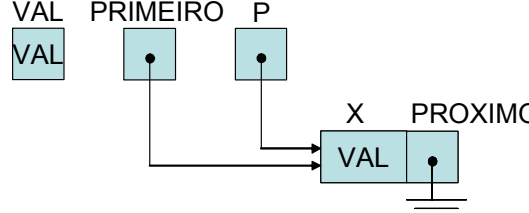
Sendo que para acessar as componentes do (registro) nó, usa-se a seguinte notação:

$P \uparrow . X$ para referenciar a componente do registro NO correspondente ao dado X
 $P \uparrow . PROXIMO$ referencia o apontador PROXIMO.

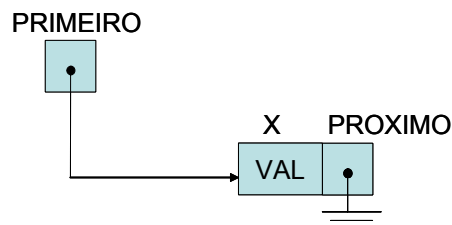
De acordo com esta notação, pode-se identificar:

| Notação | Lê-se | Identifica |
|------------------------|---------------------------------|---|
| P | P (parte de posição) | O endereço do (<u>reg</u>) NO |
| $P \uparrow$ | O que P aponta (parte de valor) | O (<u>reg</u>) NO |
| $P \uparrow . X$ | O que P aponta, ponto X | A componente X do NO apontado por P |
| $P \uparrow . PROXIMO$ | O que P aponta, ponto PROXIMO | A componente PROXIMO do NO apontado por P |

Um trecho de programa para criar uma lista encadeada contendo um único nó com o valor VAL e sua respectiva representação esquemática de alocação de memória, poderia ser:

| Comandos | Representação esquemática |
|--|--|
| <u>tipo</u> R = <u>ref</u> NO; <u>tipo</u> NO = <u>reg</u> (tipo-t: X, R: PROXIMO); | |
| R: PRIMEIRO, P; <u>tipo-t</u> : VAL; |  |
| {cria lista vazia} PRIMEIRO ← <u>nil</u> ; <u>leia</u> (VAL); |  |
| <u>aloque</u> (P); |  |
| {monta o novo nó} P↑.X ← VAL; P↑.PROXIMO ← PRIMEIRO; |  |
| {liga o novo nó à lista} PRIMEIRO ← P; |  |

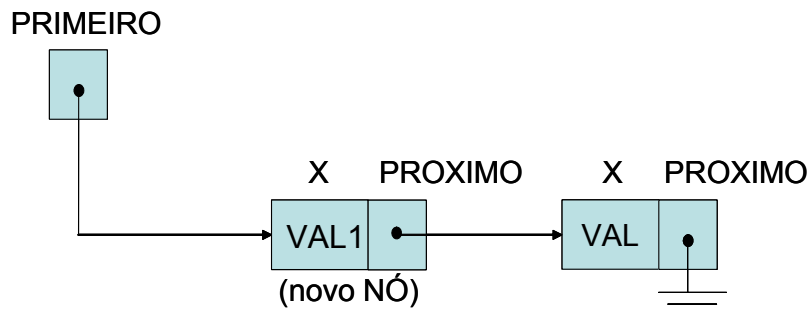
E assim, cria-se a lista:



Para incluir mais um nó no início da lista com o valor VAL1, basta executar:

| Comandos | Representação esquemática |
|--|---------------------------|
| <pre>..... <code>aloque (P);</code></pre> | |
| <pre>{monta o novo nó} <code>P↑.X ← VAL1;</code> <code>P↑.PROXIMO ← PRIMEIRO;</code></pre> | |
| <pre>{liga o novo nó à lista} <code>PRIMEIRO ← P;</code></pre> | |

A lista passa a ter, desta forma, o seguinte aspecto:



A seguir é apresentado o procedimento para adicionar um nó no início de uma lista cujo primeiro nó é apontado pela variável PRIMEIRO:

```

procedimento AddNoLEnc(ref R:PRIMEIRO, tipo-t:VAL)
{adiciona um nó com valor VAL na lista encadeada PRIMEIRO}
  R:P;
  aloque(P);
  P↑.X ← VAL;
  P↑.PROXIMO ← PRIMEIRO;
  PRIMEIRO ← P;
fim-procedimento;

```

O procedimento anterior só funciona se a lista já foi criada. Através deste procedimento, pode-se criar outro procedimento simples para adicionar o primeiro nó na lista.

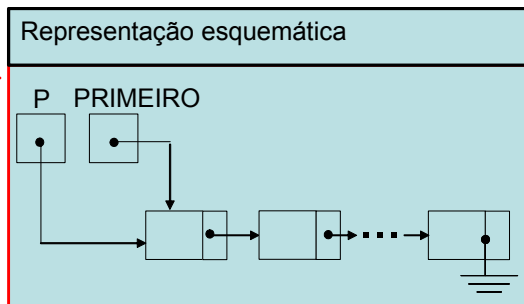
```

procedimento AddPrimNoLEnc(ref R:PRIMEIRO, tipo-t:VAL)
{adiciona o primeiro nó com valor VAL na lista ainda não criada}
    PRIMEIRO ← CriaListaVazia;
    AddNoLEnc(PRIMEIRO, VAL);
fim-procedimento;

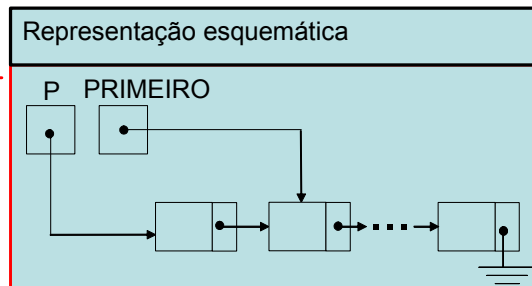
```

Para eliminar o primeiro NÓ da lista apontada por PRIMEIRO, pode-se executar:

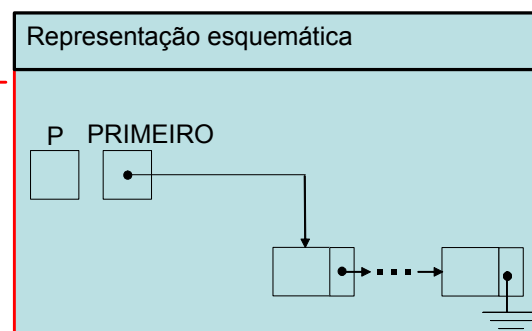
P ← PRIMEIRO;



PRIMEIRO ← P↑.PROXIMO;
 {ou PRIMEIRO ← PRIMEIRO↑.PROXIMO;}



desaloque (P);



Obs: Caso a lista esteja vazia, o procedimento de eliminação de um nó pode prever a execução de ações específicas, como:

- terminar a execução do programa;
- simular a eliminação do nó sem executá-la efetivamente:

```

se PRIMEIRO ≠ nil então
    P ← PRIMEIRO;
    PRIMEIRO ← P↑.PROXIMO;
    desaloque (P);
fim-se;

```

- trabalhar com uma **rotina de tratamento de exceções** cujo parâmetro pode ser um valor inteiro que identifique uma exceção específica para as devidas providências. A seguir é apresentado o procedimento para a exclusão do primeiro nó da lista:

```

procedimento ExcluiPrimNoLEnc(ref R:PRIMEIRO)
  se PRIMEIRO = nil então
    ROTINA-ERRO (1);
  senão
    P ← PRIMEIRO;
    PRIMEIRO ← P↑.PROXIMO;
    desalogue (P);
  fim-se;
fim-procedimento;

```

Para percorrer uma lista, isto é, visitar todos os seus nós do primeiro ao último (por exemplo, para imprimir o valor de cada nó) pode-se executar o procedimento a seguir.

Procedimento para percorrer e imprimir o conteúdo de cada nó de uma lista simplesmente encadeada:

```

procedimento ImprimeValLEnc(R: Primeiro)
  R:P;
  P ← PRIMEIRO;
  enquanto P ≠ nil faca
    imprima(P↑.X);
    P ← P↑.PROXIMO;
  fim enquanto;
fim-procedimento;

```

Em alguns casos é necessário verificar se uma lista está vazia, isto é, se PRIMEIRO é igual a nil. A função a seguir retorna verdadeiro ou falso se a lista está vazia ou não, respectivamente.

```

funcao VaziaLEnc(R: PRIMEIRO):logico;
  se PRIMEIRO = nil então
    VaziaLEnc ← verdadeiro;
  senao
    VaziaLEnc ← falso;
fim-funcao;

```

Uma lista pode ter uma sub-rotina para localizar um nó cujo valor X é igual a um valor VAL dado. A função a seguir retorna uma referência (ponteiro) para o nó P tal que $P↑.X = VAL$; se VAL não estiver na lista, retorna nil.

```

funcao BuscaLEnc(R: PRIMEIRO): R
  R:P;
  P ← PRIMEIRO;
  enquanto P ≠ nil e P↑.X ≠ VAL faca
    P ← P↑.PROXIMO;
  fim enquanto;
  BuscaLEnc ← P;
  {observe que, neste ponto, PRIMEIRO é igual a nil (passado por ref)}
fim-procedimento;

```

Após o uso da lista em um programa, ou numa sub-rotina, a mesma deve ser destruída, ou seja, os espaços da memória ocupados por seus nós devem ser liberados para outros usos. A seguir é implementado um procedimento para destruir uma lista (desalocar todos os nós da lista).

```

procedimento DestroiLEnc(ref R: PRIMEIRO)
  R:P;
  P ← PRIMEIRO;
  enquanto P ≠ nil faca
    PRIMEIRO ← PRIMEIRO↑.PROXIMO;
    desaloque(P);
    P ← PRIMEIRO;
  fim enquanto;
  {neste ponto, PRIMEIRO é igual a nil e é passado por referencia}
fim-procedimento;

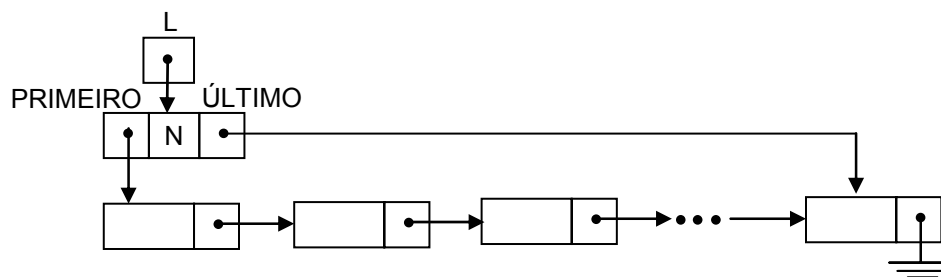
```

Aplicações:

1. Desenvolver um procedimento para eliminar o último nó de uma lista encadeada (o nó mais à direita).
2. Desenvolver um procedimento para eliminar o nó apontado pela variável AP válida (lista não está vazia) do tipo ref NO, de uma lista simplesmente encadeada.
3. Desenvolver uma função para comparar se duas listas do mesmo tipo são iguais. Esta função deve retornar verdadeiro (se as listas são iguais) falso (se as listas **não** são iguais).
4. Desenvolver uma função que conta e retorna o número de nós de uma lista simplesmente encadeada.

4.5.2 Lista com Descritor

Em algumas aplicações torna-se interessante dispor de outras informações sobre a lista, destacando-se o número de nós (N) e o endereço do último nó. Para estes casos pode-se criar um registro **descritor**, ficando a estrutura na forma:



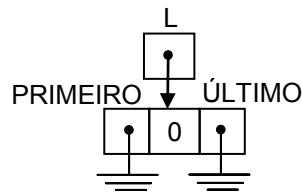
A seguir é apresentado o código para a definição do tipo lista com descritor:

```

tipo DESCRITOR = reg (RN: PRIMEIRO; int: N; RN: ULTIMO);
tipo RD = ref DESCRITOR;
tipo NO = reg (tipo-t: X; RN: PROXIMO);
tipo RN = ref NO;
.....
RD: L; {L: referencia para o descritor}

```

Uma lista vazia não é mais representada por uma referência nil, mas por uma referência ao descritor que contém as seguintes informações: PRIMEIRO = nil, N = 0 e ULTIMO = nil. Esquematicamente, uma lista encadeada vazia com descritor fica:



Um procedimento para criar uma lista vazia com descritor seria:

```

procedimento CRIALD(ref RD: L)
  aloque (L);
  L↑.PRIMEIRO ← nil;
  L↑.N ← 0;
  L↑.ULTIMO ← nil;
fim-procedimento; {CRIALD}

```

Aplicações: Considerando uma lista com descritor, desenvolver procedimentos para:

- incluir um nó à esquerda com o valor VAL;
- incluir um nó à direita com o valor VAL;
- excluir o nó da esquerda;
- excluir o nó da direita;
- considerando a lista com descritor, desenvolver os mesmos procedimentos e funções criados para a lista simplesmente encadeada.

4.5.3 Lista Duplamente Encadeada

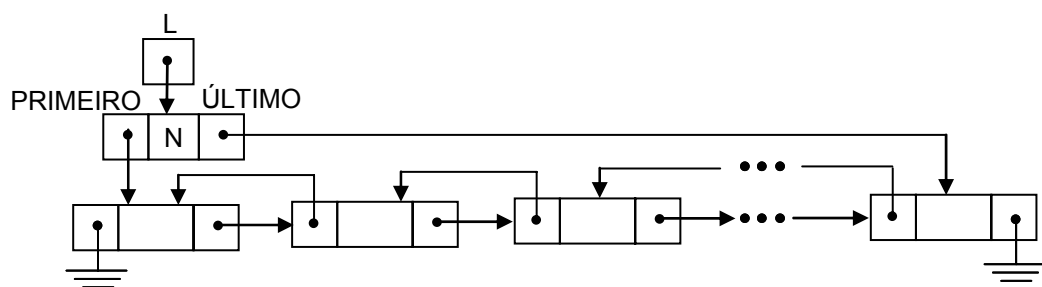
É uma lista (geralmente com descritor) onde cada nó possui dois apontadores, sendo um para o nó anterior e outro para o nó seguinte, isto é:

```

tipo NO = reg (RN: ESQ; tipo-t: X; RN: DIR);

```

Esquematicamente:



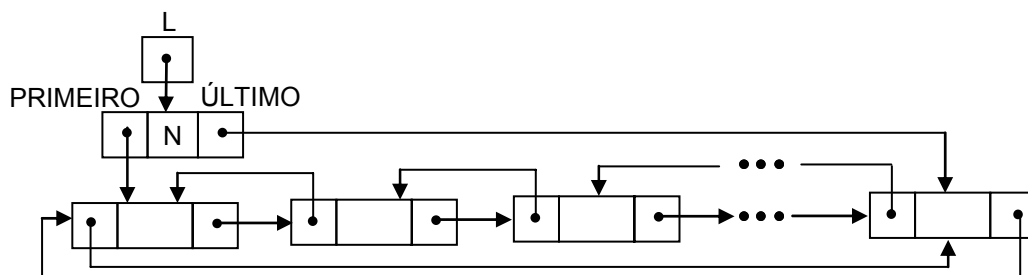
Obs: A lista duplamente encadeada permite o caminhamento nos dois sentidos.

Aplicações: Considerando uma lista duplamente encadeada, desenvolver procedimentos para:

- criar uma lista vazia (idêntica ao CRIALD, com descritor);
- incluir o primeiro nó com o valor VAL;
- incluir um nó à direita com o valor VAL;
- excluir um nó à direita.
- considerando a lista duplamente encadeada, desenvolver os mesmos procedimentos e funções criados para a lista simplesmente encadeada.

4.5.4 Listas Circulares

É uma lista duplamente encadeada com os dois nós extremos unidos, isto é:



Obs: Neste caso, um nó qualquer apontado por P satisfaz a condição:

$$(P \uparrow .DIR) \uparrow .ESQ = P = (P \uparrow .ESQ) \uparrow .DIR$$

Aplicações:

- 1) Considerando uma lista circular, desenvolver procedimentos para:
 - a) criar uma lista vazia;
 - b) incluir o primeiro nó;
 - c) incluir um nó à direita;
 - d) excluir um nó à esquerda.
 - e) considerando a lista duplamente encadeada, desenvolver os mesmos procedimentos e funções criados para a lista simplesmente encadeada.

Exercícios:

- 1) Quais as vantagens e desvantagens de representar um grupo de itens como um vetor versus uma lista encadeada linear?
- 2) Desenvolver um procedimento para ler diversos valores inteiros e positivos e montar uma lista representada por contigüidade dos nós utilizando um vetor de 100 elementos. O último valor a ser lido é um FLAG igual a -1;
- 3) Idem ao anterior montando uma lista simplesmente encadeada (sem descritor);
- 4) Considerando as listas das questões 2 e 3, montar, respectivamente dois procedimentos para, dada a lista e um valor inteiro VAL, excluir da lista todos os nós cujo valor seja igual a VAL;
- 5) Desenvolver procedimentos para classificar os nós das listas das questões 2 e 3, em ordem crescente;
- 6) Dadas duas listas simplesmente encadeadas onde cada nó possui um valor inteiro e os nós estão em ordem crescente, desenvolver um procedimento para fundir estas duas listas, isto é, gerar somente uma lista mantendo a ordenação;
- 7) Dada uma lista duplamente encadeada (com descritor) onde cada nó possui uma informação X do tipo vet [1..5] int, desenvolver um procedimento para excluir o nó central caso o número de nós da lista seja ímpar;
- 8) Dado o valor VAL do tipo vet [1..5] int e uma lista idêntica a da questão 7, desenvolver um procedimento para, caso o número de nós seja par, incluir no centro da lista um novo nó contendo VAL;
- 9) Dada uma lista circular (duplamente encadeada com descritor) e uma variável P do tipo ref NO, desenvolver um procedimento para excluir o nó apontado por P;
- 10) Escreva um procedimento (ou função) para efetuar cada uma das seguintes operações sobre listas simplesmente encadeada:

- a. concatenar duas listas;
 - b. inverter uma lista de modo que o último elemento se torne o primeiro, e assim por diante;
 - c. eliminar o último elemento de uma lista;
 - d. eliminar o i -ésimo elemento de uma lista;
 - e. formar uma lista contendo a união dos elementos de duas listas (não pode haver elementos repetidos, assim esta operação é diferente da operação de fundir/concatenar);
 - f. inserir um elemento depois do i -ésimo elemento de uma lista;
 - g. eliminar os elementos nas posições pares, isto é, 2º, 4º, 6º, ...;
 - h. considerando um lista de inteiros, calcular e retornar a soma dos elementos numa lista;
 - i. desenvolver um procedimento para deslocar o nó apontado pela variável P válida (lista não está vazia) n posições a frente;
 - j. criar uma cópia de uma lista;
 - k. escreva uma rotina para intercambiar os elementos m e n da uma lista.
- 11) Escreva algoritmos para executar cada uma das operações do exercício 10, pressupondo que cada lista seja uma lista:
- a. com descritor;
 - b. duplamente encadeada;
 - c. circular.