



INSTITUTO FEDERAL
TRIÂNGULO MINEIRO
Campus Uberlândia Centro

Licenciatura em Computação
Algoritmos e Estrutura de Dados

LISTAS LINEARES

Prof. Walteno Martins Parreira Júnior

www.waltenomartins.com.br

waltenomartins@iftm.edu.br

2016

SUMÁRIO

1 - LISTAS LINEARES	2
1.1 - Introdução	2
1.2 - Definição de lista	2
1.3 - Lista Seqüencial	3
1.4 - Listas Lineares Ligadas ou Listas Encadeadas	3
1.5 - Definição Recursiva de Listas Ligadas.....	5
1.6 - Estrutura da Lista Ligada	5
1.7 - Implementação de Listas Ligadas utilizando vetores	6
1.8 - Gerenciamento da memória disponível para armazenar as listas	7
Exercícios.....	12
1.9 - Alocação Dinâmica de Memória	13
1.10 - Implementação de Listas Ligadas utilizando Ponteiros	14
2 - LISTAS DUPLAMENTE ENCADEADAS.....	18

1 - LISTAS LINEARES

1.1 - Introdução

Uma lista é uma forma de agrupar itens com a finalidade de melhorar sua manipulação. Em nosso dia-a-dia é comum utilizarmos as listas como forma de organizarmos algumas tarefas.

É a propriedade seqüencial de uma lista linear que é a base para a sua definição e para o seu uso. Observamos que em uma lista linear existe um início, onde está o primeiro elemento; e um final, onde encontramos o último elemento. A disciplina de acesso, ou seja, a forma pela qual se realizam as operações de inserção e de remoção de elementos nas listas é o que determinam algumas classificações das listas existentes.

1.2 - Definição de lista

Uma lista linear é uma coleção $L: [a_1, a_2, \dots, a_n]$ com $n \geq 0$, cuja propriedade estrutural baseia-se apenas na posição relativa dos elementos, que são dispostos linearmente.

Se $n \geq 0$, onde n representa o número de elementos, temos:

- a_1 é o primeiro elemento da lista;
- a_n é o último elemento da lista;
- para cada i , $1 < i < n$, o elemento a_i é precedido por a_{i-1} e seguido por a_{i+1} .

Quando $n = 0$, dizemos que a lista é vazia.

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

Entre as diversas operações que podemos realizar sobre listas, temos:

- acessar um elemento qualquer da lista;
- inserir um elemento em uma posição específica da lista;
- remover um elemento de uma posição específica da lista;
- combinar duas listas em uma única lista;
- dividir uma lista em duas novas listas;
- obter cópia de uma lista;
- calcular o total de elementos na lista;
- ordenar os elementos da lista;
- apagar uma lista

1.3 - Lista Seqüencial

Uma lista é seqüencial se para cada componente da lista, o sucessor deste componente estiver armazenado na posição física seguinte na lista.

Exemplo:

Seja L uma lista de números pares dispostos na ordem crescente de seus valores.

$L = [0, 4, 10, 18]$

0	4	10	18
---	---	----	----

Armazenamento físico da lista L

para inserir o elemento 14 na lista L, temos:

0	4	10	14	18
---	---	----	----	----

para remover o elemento 4 da lista L, temos:

0	10	14	18
---	----	----	----

No exemplo acima podemos observar que:

- A ordem lógica dos dados coincide com a ordem física dos mesmos;
- A inserção e remoção de elementos exigem considerável movimentação de dados;
- A implementação de listas seqüenciais é realizada utilizando-se vetores (Arrays) ou arquivos (Files);
- Nestas implementações, o tamanho da lista precisa ser conhecido e alocado antecipadamente.

1.4 - Listas Lineares Ligadas ou Listas Encadeadas

São listas formadas por seqüências de elementos, e que estão organizadas em uma determinada ordem, de tal forma, que o armazenamento lógico não coincide necessariamente com o armazenamento físico. Neste caso, em vez de manter os elementos em células consecutivas, podem-se utilizar qualquer célula para guardar os elementos da lista. Para preservar a relação de ordem linear da lista, cada elemento armazena sua informação e também o endereço da próxima célula de memória válida.

1.4.1 – Lista com Inserção na Cauda e Remoção na Cabeça

Exemplo:

Frutas = [Banana, Laranja, Caqui, Uva]

End	Informação	Próximo
1		
2	Caqui	7
3	Laranja	2
4		
5	Banana	3
6		
7	Uva	0

Frutas: 5
Disp = 1

Controle da
Lista

Área de memória para
armazenamento da
lista

Para inserir Melão no final da lista:

Frutas = [Banana, Laranja, Caqui, Uva, Melão]

End	Informação	Próximo
1	Melão	0
2	Caqui	7
3	Laranja	2
4		
5	Banana	3
6		
7	Uva	1

Frutas: 5
Disp = 4

Para remover o primeiro elemento da lista:

Frutas = [Laranja, Caqui, Uva, Melão]

End	Informação	Próximo
1	Melão	0
2	Caqui	7
3	Laranja	2
4		
5		
6		
7	Uva	1

Frutas: 3
Disp = 4

No exemplo pode-se observar:

- A ordem lógica dos dados não coincide necessariamente com a ordem física dos mesmos;
- A ordem lógica dos dados é feita através de ponteiros;
- As operações de inserção e remoção não requerem movimentação de elementos;
- O endereço do primeiro elemento é essencial para a localização do ponto inicial da lista;
- Existe um endereço fictício que indica o final da lista;
- É trabalhoso encontrar um determinado elemento.

1.5 - Definição Recursiva de Listas Ligadas

Uma lista é vazia ou possui 2 componentes:

- a) Cabeça – é o primeiro elemento da lista;
- b) Cauda – é a lista sem a cabeça.

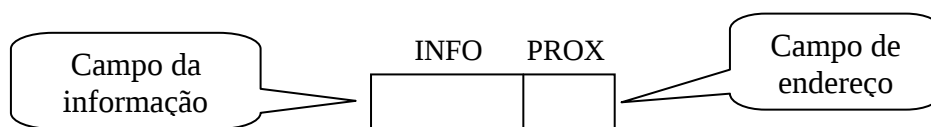
Exemplos:

- a) $L = [A, B, C]$ onde temos: Cabeça = A e Cauda = [B, C]
- b) $L = [B, C]$ onde temos: Cabeça = B e Cauda = [C]
- c) $L = [C]$ onde temos: Cabeça = C e Cauda = []

1.6 - Estrutura da Lista Ligada

Listas ligadas utilizam a denominada alocação encadeada, que é composta de elementos chamados NÓS (ou nodos ou nódulos). Cada nó contém duas partes distintas, chamadas de campos:

- a) O primeiro campo contém a informação;
- b) O segundo campo contém o endereço do próximo nó.



O campo **INFO** armazena a informação armazenada e o campo **PROX** armazena o endereço do próximo nó da lista.

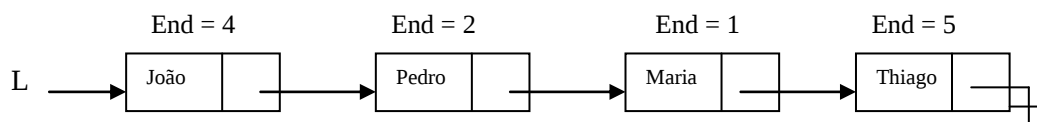
Exemplo: $L = [\text{João, Pedro, Maria, Thiago}]$

1	Maria	5
2	Pedro	1
3		6
4	João	2
5	Thiago	0
6		0

$L = 4$

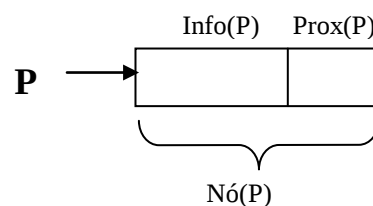
$\text{Disp} = 3$

Em formato gráfico, pode-se representar da seguinte forma:



No exemplo pode-se observar:

- A lista é acessada através de seu ponteiro externo que aponta para a cabeça da lista, que no exemplo é L;
- O acesso aos elementos da lista é seqüencial, elemento após elemento;
- O campo Prox do último nó contém um valor especial para indicar o fim da lista;
- A lista vazia é representada por L = [] ou L = NULL;
- Dado um nó apontado por P, tem-se:



1.7 - Implementação de Listas Ligadas utilizando vetores

Pode-se implementar uma lista ligada utilizando-se vetores (Arrays) com gerenciamento do armazenamento disponível.

a) Definição da estrutura de dados em algoritmo

Constantes

N=30;

Capacidade máxima da memória

$$L_V=0;$$

Lista vazia

Tipo

```
Tipo_Info=String[15];
```

Tipo de informação da lista

Pont=0..N;

No = Registro

```
Info : Tipo_info;
```

Prox : Pont;

Fim;

Vet = vetor[1..n] de No;

Variaveis

Mem : Vet;

Área para armazenar as listas

L, L1 : Pont;

Ponteiros externos das Listas

Disp : Pont;

Variável global

Exemplo: L2 = [João, Pedro, Maria, Thiago] e L1 = [André, Sara]

Representação gráfica:

End	Info	Prox
1	Thiago	0
2	Pedro	3
3	Maria	1
4		8
5	João	2
6	Sara	0
7	Andre	6
8		9
...		
N		0

Memória

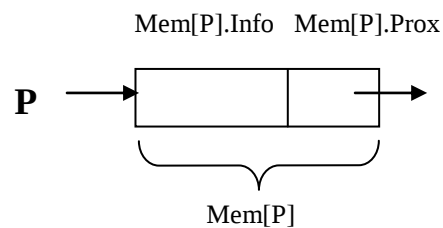
L2 = 5
L1 = 7
Disp = 4

Fazer a representação em
forma de caixas.

b) Acesso ao Nó apontado por P

Exemplos: Mem[6].Info = Sara

Mem[6].Prox = 0



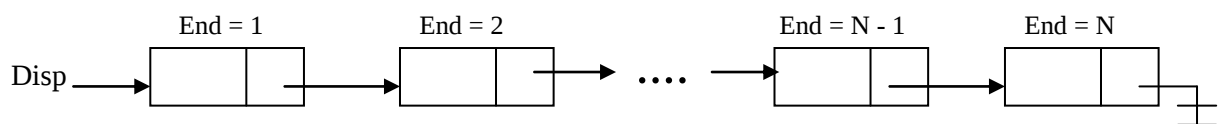
1.8 - Gerenciamento da memória disponível para armazenar as listas

Inicialmente temos somente a alocação da área, portanto temos uma área disponível.

End	Info	Prox
1		
2		
3		
...		
N - 1		
N		

Memória

Disp = 1



a) Alocação da lista de Nós disponíveis

Este procedimento aloca a lista DISP de nós disponíveis

```
Procedimento Aloca_memoria;  
Variaveis  
  I: Inteiro;  
Inicio  
  Para I de 1 até (N-1) faça  
    Mem[I].prox <- I+1;  
  Fim_para;  
  Mem[N].prox <- LV;  
  Disp <- 1;  
Fim;
```

b) Inserção e remoção de elementos

Para manipular um nó, precisamos de um nó livre. A função Pega_no retorna o endereço de um nó livre e requer que DISP seja diferente de lista vazia.

```
Função Pega_No: Pont;  
Inicio  
  Pega_no <- Disp;  
  Disp <- Mem[Disp].prox;  
Fim;
```

Este procedimento torna disponível o nó apontado por P, ou seja insere o nó P na lista DISP, liberando-o para futura utilização. Requer o endereço de P.

```
Procedimento Libera_No(P: Pont);  
Inicio  
  Mem[P].prox <- Disp;  
  Disp <- P;  
Fim;
```

Desenvolver as passagens:

- Situação inicial
- Armazenar na memória a lista $L = [a, b, c]$
- Inserir **z** na cabeça da lista L
- Remover b da lista L

c) Inserção na Cabeça da lista

Este procedimento insere uma informação lida X na cabeça de uma lista L.

Função Insere_na_cabeca(L:pont; X:Tipo_Info):pont;

Variaveis

Aux:Pont;

Inicio

Aux <- Pega_no;

Mem[aux].info <- X;

Mem[aux].prox <- L;

Insere_na_cabeca <- aux;

Fim;

d) Impressão da Lista

O procedimento imprime as informações contidas nos nós de L. Requer a lista L.

Procedimento Imprime(L:pont);

Inicio

Escreva('[');

Enquanto (L <> LV) faça

Escreva(Mem[L].info, ' ');

L <- Mem[L].prox;

Fim-Enquanto;

escreva(']');

Fim;

e) Inserir uma informação no final da lista

O módulo insere uma informação lida X no final da lista, observando a situação da lista.

Função insere_final(L:pont; x:tipoinfo):pont;

variaveis p, aux:pont;

Inicio

P <- L;;

Aux <- pega_no;

men[aux].info <- x;

men[aux].prox <- lv;

Se (p = lv)

então L <- aux;

senão

Enquanto (men[p].prox<>lv) faça

P <- men[p].prox;

Fim-Enquanto;

men[p].prox <- aux;

Fim-Se;

insere_final <- L;

Fim;

f) Remover uma informação qualquer da lista

O módulo remove (apaga) uma informação lida X na lista, observando a situação da lista.

Função remove_da_lista(L:pont; x:tipoinfo):pont;

variáveis p, aux:pont;

Início

p <- L;

aux <- lv;

Enquanto (p <> lv) e (x <> men[p].info) faça

Aux <- p;

p <- men[p].prox;

Fim-enquanto;

Se (p = lv)

Então escreva(x, ' não pertence a lista');

Senão

Se p = L

Então

L <- men[p].prox;

Libera_no(p);

Senão

Mem[aux].prox <- men[p].prox;

Libera_no(p);

Fim-se;

Fim-se;

Remove_da_lista <- L;

Fim;

Atenção:

Se o campo **info** for uma string, na programação em linguagem C, tem-se que usar a função STRCMP(S1,S2) que retorna zero quando forem iguais.

g) Programa Principal

Chama os procedimentos.

Variáveis

L1, i : pont;

Nome : Tipo_Info;

Início

Aloca_Memoria;

L1 <- lv;

Para i de 1 até 5 faça

Escreva('Entre com o Nome: ');

leia(Nome);

L1 <- Insere_na_cabeca(L1,Nome);

Fim-Para

Imprime(L1);

```
Escreva('Entre com o Nome: ');  
leia(Nome);  
L1 <- Insere_final(L1,Nome)  
Imprime(L1);  
Escreva('Entre com o Nome: ');  
leia(Nome);  
L1 <- Remove_da_Lista(L1,Nome);  
Imprime(L1);  
Fim.
```

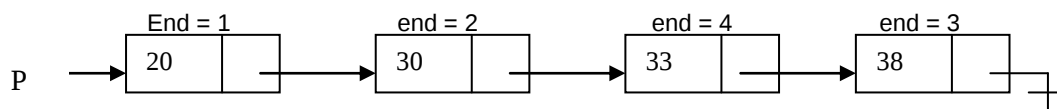
h) Remover uma informação no final da lista

O módulo remove (apaga) a informação que está no último nó da lista, retornando a informação para o programa principal.

```
procedimento remove_final(L:pont);  
variaveis ant:pont;  
Inicio  
  Ant <- lv;  
  Enquanto (men[L].prox <> lv faça  
    Ant <- L;  
    L <- men[L].prox;  
  Fim-enquanto;  
  Libera_no(L);  
  men[ant].prox <- lv;  
Fim;
```

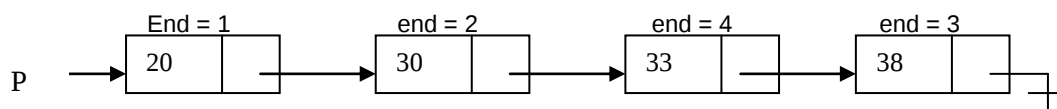
Exercícios

1) Supondo a lista ligada apresentada na figura abaixo:



- Desenvolver graficamente a memória disponibilizada com 10 posições. Os nós vazios devem ser encadeados de modo a que o nó de posição menor aponte para o de maior posição, considerando que DISP = 5 e P = 1.
- Considerando a inserção no final da lista, inserir o numeral **25** na lista. Mostre como fica a memória, faça a representação gráfica da lista. Quais os valores de DISP e de P.
- Considerando a inserção no final da lista, inserir o numeral **35** na lista. Mostre como fica a memória, faça a representação gráfica da lista. Quais os valores de DISP e de P.
- Remover o primeiro item da lista. Mostre como fica a memória, faça a representação gráfica da lista. Quais os valores de DISP e de P.
- Remover o primeiro item da lista. Mostre como fica a memória, faça a representação gráfica da lista. Quais os valores de DISP e de P.
- Considerando a inserção no final da lista, inserir o numeral **45** na lista. Mostre como fica a memória, faça a representação gráfica da lista. Quais os valores de DISP e de P.
- Considerando a inserção no final da lista, inserir o numeral **50** na lista. Mostre como fica a memória, faça a representação gráfica da lista. Quais os valores de DISP e de P.

2) Supondo a lista ligada apresentada na figura abaixo:



- Desenvolver graficamente a memória disponibilizada com 10 posições. Os nós vazios devem ser encadeados de modo a que o nó de posição menor aponte para o de maior posição, considerando que DISP = 5 e P = 1.
- Considerando a inserção na cabeça da lista, inserir o numeral **25** na lista. Mostre como fica a memória, faça a representação gráfica da lista. Quais os valores de DISP e de P.
- Considerando a inserção na cabeça da lista, inserir o numeral **35** na lista. Mostre como fica a memória, faça a representação gráfica da lista. Quais os valores de DISP e de P.
- Remover o primeiro item da lista. Mostre como fica a memória, faça a representação gráfica da lista. Quais os valores de DISP e de P.
- Remover o primeiro item da lista. Mostre como fica a memória, faça a representação gráfica da lista. Quais os valores de DISP e de P.
- Considerando a inserção na cabeça da lista, inserir o numeral **45** na lista. Mostre como fica a memória, faça a representação gráfica da lista. Quais os valores de DISP e de P.
- Considerando a inserção na cabeça da lista, inserir o numeral **50** na lista. Mostre como fica a memória, faça a representação gráfica da lista. Quais os valores de DISP e de P.

1.9 - Alocação Dinâmica de Memória

As listas estudadas até o momento são organizadas de maneira fixa. Isto é, criamos as variáveis e estas contam com um tamanho fixo disponível em memória. Arquivos permitem uma estrutura com um número indeterminado de elementos, porém sempre arrumados na forma de uma seqüência.

Quando tanto o número de elementos quanto sua forma de organização variam dinamicamente? Para resolver este problema tem-se necessidade de um mecanismo que permita:

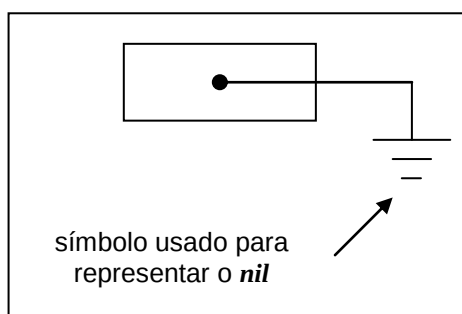
- criar espaço para novas variáveis em tempo de execução;
- definir “ligações” entre estas variáveis, de uma forma dinâmica.

Variáveis dinâmicas não possuem nome próprio, portanto não são referenciadas por seus nomes. A referência a uma variável dinâmica é feita por ponteiros. Um ponteiro é uma variável cujo conteúdo é o endereço de uma posição de memória. Um ponteiro é declarado fornecendo-se o tipo de variável por ele apontada. Ou seja, “P” é um ponteiro para um tipo “T” se houver a declaração:

$P : ^T;$

Ao iniciar o programa, o valor de “P” estará indefinido. Existe uma constante predefinida, do tipo ponteiro, chamada *nil* que não aponta para objeto algum. Note que o significado de *nil*: não apontar para objeto algum é diferente de indefinido que significa variável não inicializada.

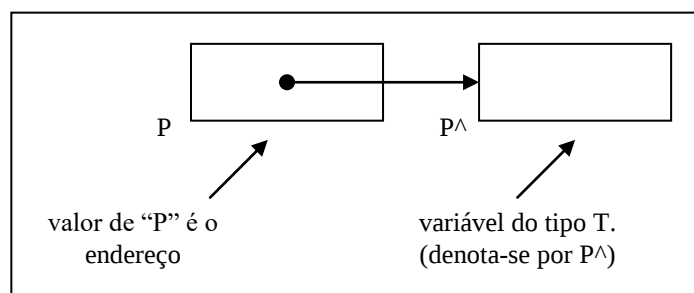
$P \leftarrow nil;$



A criação de uma variável dinâmica do tipo T é feita pelo operador aloque. Assim, o procedimento padrão:

$aloque(P)$

cria uma variável do tipo T, sem nome, e coloca em “P” o endereço desta variável. Graficamente podemos indicar a geração de uma variável dinâmica da seguinte forma:



Note que a variável dinâmica anterior é referenciada como **P[^]**, que significa variável apontada por “**P**”.

A remoção de uma variável criada dinamicamente, apontada por “**P**”, pode ser realizada através do seguinte comando:

```
desaloque(P);
```

1.10 - Implementação de Listas Ligadas utilizando Ponteiros

Podemos implementar uma lista ligada utilizando-se de ponteiros (apontadores) com armazenamento na memória disponível. As variáveis são criadas e destruídas em tempo de execução.

a) Definição da estrutura da lista em algoritmo

```
Const Lv = Nil;  
Tipo Tipo_Info = String[15];  
Tipo Pont = ^No;  
tipo No = registro  
    Info : Tipo_info;  
    Prox : Pont;  
Fim-registro;  
Variável  
    L : Pont;
```

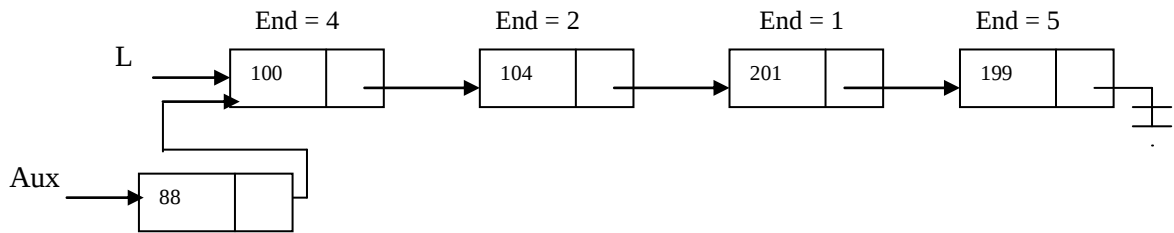
b) criar um novo Nó para a lista

```
Função Pega_no : Pont  
Variável P : Pont;  
Início  
    aloque(P);  
    Pega_no <- P;  
Fim;
```

c) Liberar uma área

```
Procedimento Libera_No ( P : Pont)  
Início  
    Desaloque (P);  
Fim;
```

d) Este procedimento insere uma informação lida X na cabeça de uma lista L.



Função Insere_Cabeca (var L: Pont; x: tipo_info) : Pont
variavel

Aux : Pont;

Inicio

Aux <- Pega_no;

Aux^.Info <- x;

Aux^.Prox <- L;

Insere_cabeca <- Aux;

fim;

e) Impressão da Lista

O procedimento imprime as informações contidas nos nós de L. Requer a lista L.

Procedimento Imprime(L:pont);

Inicio

Escreva('[');

Enquanto (L <> LV) faça

Escreva(L^.Info, ' ');

L <- L^.Prox;

Fim-Enquanto;

escreva(']');

Fim;

f) Este procedimento insere uma informação lida X no final de uma lista L.

Função Insere_final(L:pont; x:Tipo_Info) : pont;

Variáveis

Aux, P : pont;

inicio

Aux <- pega_no;

Aux^.info <- x;

Aux^.prox <- Lv;

Se L = Lv

então

```

    L <- Aux
  Senão
    P <- L;
    Enquanto ( P^.Prox <> LV ) faça
      P <- P^.Prox;
    Fim_enquanto;
    P^.prox:=Aux;
  Fim_se;
  Insere_final <- L;
Fim;

```

g) Remover uma informação qualquer da lista

O módulo remove (apaga) uma informação lida X na lista, observando a situação da lista.

```

Função Remove (L:pont ; x:Tipo_info) : Pont;
Variavel
  Ant , Q : Pont;
Início
  Ant <- Lv;
  Q <- L;
  Enquanto ( Q <> Lv ) faça
    Se ( Q^.info <> x)
      Então
        Ant <- Q;
        Q <- Q^.prox;
      Senão
        Se ( Q = L )
          Então
            L <- L^.prox;
            Libera_no(Q);
          Senão
            Ant^.prox <- Q^.prox;
            libera_no(Q);
        Fim_se;
      Fim_se;
  Remove <- L;
Fim;

```

h) Programa Principal

Chama os procedimentos.

Variaveis

L1, i : pont;

Nome : Tipo_Info;

Inicio

Para i de 1 até 5 faça

Escreva('Entre com o Nome: ');

leia(Nome);

L1 <- Insere_cabeca(L1, Nome);

Fim-Para

Imprime(L1);

Escreva('Entre com o Nome: ');

leia(Nome);

L1 <- Insere_final(L1, Nome);

Imprime(L1);

Escreva('Entre com o Nome: ');

leia(Nome);

L1 <- Remove(L1, Nome);

Imprime(L1);

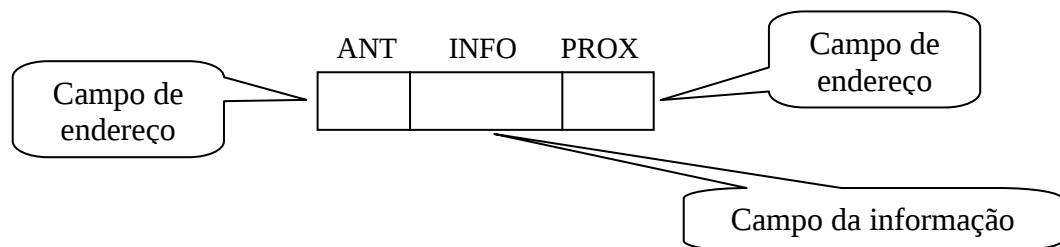
Fim.

2 - LISTAS DUPLAMENTE ENCADEADAS

Uma lista linear duplamente encadeada é aquela em que cada nó possui dois ponteiros, ao invés de um só. O primeiro é usado para indicar o nó predecessor, enquanto que o segundo aponta para o nó sucessor.

Logo, lista duplamente encadeada é composta de elementos chamados NÓS (ou nodos ou nódulos). Cada nó contém três partes distintas, chamadas de campos:

- O primeiro campo contém o endereço do nó anterior;
- O segundo campo contém a informação;
- O terceiro campo contém o endereço do próximo nó.



a) Definição da estrutura da lista em algoritmo

```

Const Lv = Nil;
Tipo Tipo_Info = String[15];
Tipo Pont = ^No;
tipo No = registro
    Ant : Pont;
    Info : Tipo_info;
    Prox : Pont;
Fim-registro;
Variável
    L : Pont;
  
```

b) Criar um novo Nó para a lista

```

Função Pega_no : Pont
Variável P : Pont;
Início
    aloque(P);
    Pega_no <- P;
Fim;
  
```

c) Liberar uma área

```

Procedimento Libera_No ( P : Pont)
  
```

```

Inicio
  Desaloque (P);
Fim;

```

- d) Este procedimento insere uma informação lida X na cabeça de uma lista L.

```

Função Insere_Cabeca (var L: Pont; x: tipo_info) : Pont
variavel
  Aux : Pont;
Inicio
  Aux <- Pega_no;
  Aux^.Ant <- Lv;
  Aux^.Info <- x;
  Aux^.Prox <- L;
  Se ( L <> Lv)
    Então L^.Ant <- Aux;
    Senão Aux^.Prox <- Lv;
  Fim_se;
  Insere_cabeca <- Aux;
fim;

```

- e) Impressão da Lista

O procedimento imprime as informações contidas nos nós de L. Requer a lista L.

```

Procedimento Imprime(L:pont);
Inicio
  Escreva('[');
  Enquanto (L <> LV) faça
    Escreva(L^.Info, ' ');
    L <- L^.Prox;
  Fim-Enquanto;
  escreva(']');
Fim;

```

- f) Este procedimento procura uma informação lida X em uma lista L e devolve a sua posição.

```

Função Busca(L:pont; x:Tipo_Info) : pont;
Variáveis
  P : inteiro;
inicio
  Pos <- 0;
  Enquanto (L <> Lv) e (x <> L^.Info) faça

```

```

    L <- L^.Prox;
    Pos <- Pos + 1;
Fim_enquanto;
Se (Pos > 0)
    Então Busca <- L;
    Senão Busca <- Lv;
Fim_se;
Fim;

```

g) Remover uma informação qualquer da lista

O módulo remove (apaga) uma informação lida X na lista, observando a situação da lista.

```

Função Remove (L:pont ; x:Tipo_info) : Pont;
Variavel
    P : Pont;
Inicio
    P <- Busca(L,x);
    Se ( P = Lv)
        Então
            Escreva(x,' não pertence a lista');
        Senão
            Se ( P = L )
                Então
                    L <- L^.prox;
                    Libera_no(P);
                    Se (L <> Lv)
                        Então L^.Ant <- Lv;
                    Fim_se;
                Senão
                    P^.Ant^.prox <- P^.prox;
                    Se (P^.Prox <> Lv)
                        Então P^.Prox.Ant <- P^.Ant;
                    Fim_se;
                    libera_no(P);
                Fim_se;
            Fim_se;
        Remove <- L;
    Fim;

```

h) Programa Principal

Ele chama os procedimentos que executam a manipulação da lista.

Variaveis

L1, i : pont;

Nome : Tipo_Info;

Inicio

Para i de 1 até 5 faça

Escreva('Entre com o Nome: ');

leia(Nome);

L1 <- Insere_cabeca(L1, Nome);

Fim-Para

Imprime(L1);

Escreva('Entre com o Nome: ');

leia(Nome);

L1 <- Insere_final(L1, Nome);

Imprime(L1);

Escreva('Entre com o Nome: ');

leia(Nome);

L1 <- Remove(L1, Nome);

Imprime(L1);

Fim.

REFERENCIAS:

CORMEN, T. H. *Algoritmos: Teoria e Prática*. Rio de Janeiro: Campus. 2001.

KNUTH, D. E. *The Art of Computer Programming*. Massachusetts: Addison-Wesley Longman, 1997. v. 1 e 2.

SALVETTI, D. D.; BARBOSA L M. *Algoritmos*. São Paulo: Makron Books, 1998.

TANENBAUM, A. M. *Estruturas de dados Usando C*. São Paulo: Makron Books, 1995.

TERADA, R. *Desenvolvimento de Algoritmo e Estruturas de Dados*. São Paulo: Makron Books, 1991.

ZIVIANI, N. *Projeto de Algoritmos - Com Implementações em PASCAL e C*. São Paulo: Editora Pioneira, 1999.

Nota do Professor:

Este trabalho é um resumo do conteúdo da disciplina, para facilitar o desenvolvimento das aulas, devendo sempre ser complementado com estudos nos livros recomendados e o desenvolvimento dos exercícios indicados em sala de aula e a resolução das listas de exercícios propostas.