Listas Lineares

Prof. Marcelo Dib

Listas Lineares

- Lista linear é uma estrutura de dados na qual elementos de um mesmo tipo de dado estão organizados de maneira sequencial.
- Não necessariamente, estes elementos estão fisicamente em sequência, mas a idéia é que exista uma ordem lógica entre eles.
- Um exemplo disto seria um consultório médico: as pessoas na sala de espera estão sentadas em qualquer lugar, porém sabe-se quem é o próximo a ser atendido, e o seguinte, e assim por diante.

.

Listas Lineares

- Assim, é importante ressaltar que uma lista linear permite representar um conjunto de dados afins (de um mesmo tipo) de forma a preservar a relação de ordem entre seus elementos.

- Cada elemento da lista é chamado de nó, ou nodo

Listas Lineares

- ***Definição:***

  Conjunto de N nós, onde $N \geq 0$, x_1, x_2, ..., x_n, organizados de forma a refletir a posição relativa dos mesmos.

- Se $N \geq 0$, então x_1 é o primeiro nó. Para $1 < k < n$, o nó x_k é precedido pelo nó x_{k-1} e seguido pelo nó x_{k+1} e x_n é o último nó.

- Quando N = 0, diz-se que a lista está vazia. Exemplos de listas lineares:

Listas Lineares

Exemplos

- Pessoas na fila de um banco;
- Letras em uma palavra;
- Relação de notas dos alunos de uma turma;
- Itens em estoque em uma empresa;
- Dias da semana;
- Vagões de um trem;
- Pilha de pratos;
- Cartas de baralho.

Listas Lineares

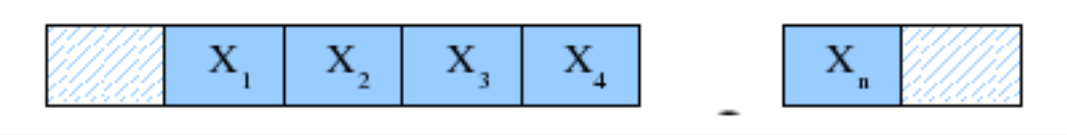
Alocação de uma lista

- Quanto a forma de alocar memória para armazenamento de seu elementos, uma lista pode ser:
  - Sequencial ou Contígua

    Numa lista linear contígua, os nós além de estarem em uma sequência lógica, estão também fisicamente em sequência. A maneira mais simples de acomodar uma lista linear em um computador é através da utilização de um vetor.

Listas Lineares

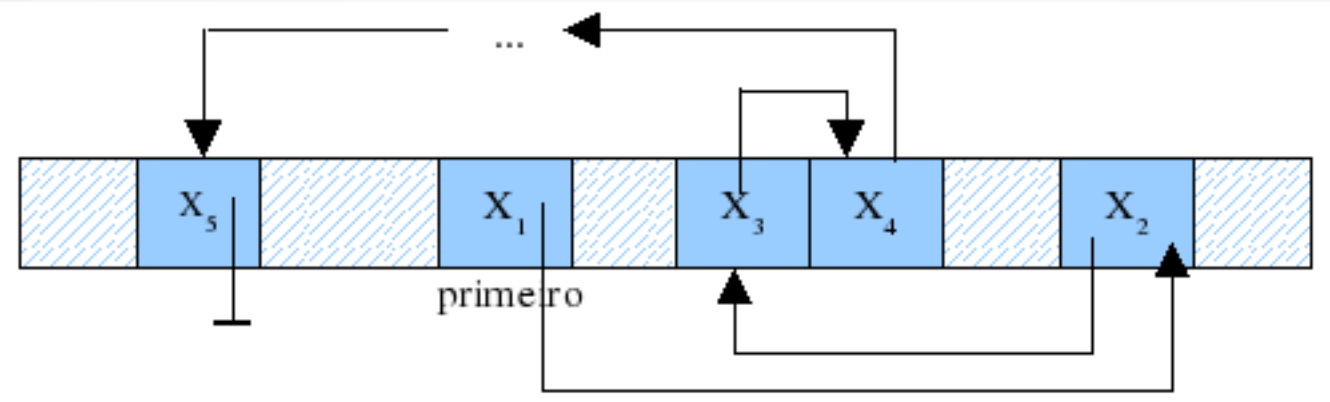
- A representação por vetor explora a sequencialidade da memória de tal forma que os nós de uma lista sejam armazenados em endereços contíguos.



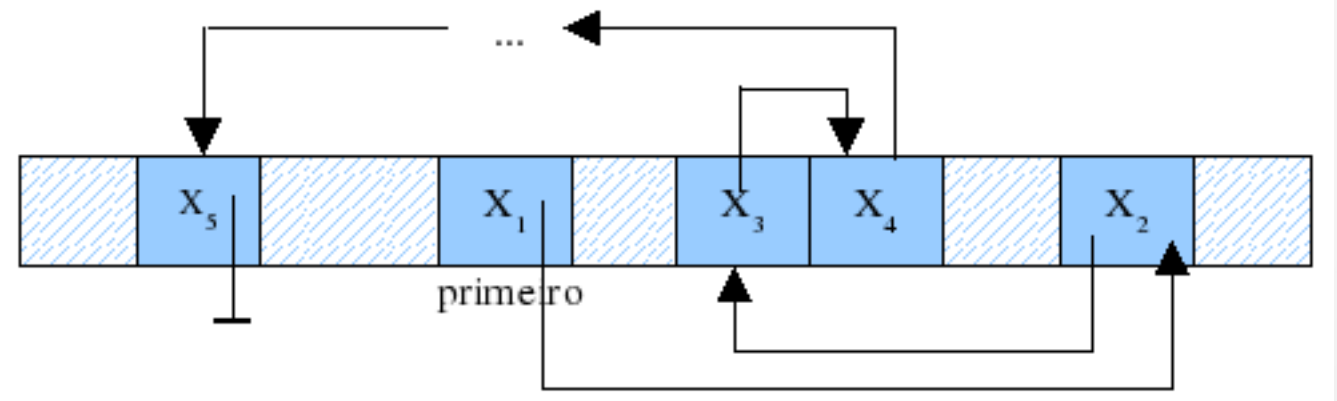
Listas Lineares

- Encadeada

  Os elementos não estão necessariamente armazenados sequencialmente na memória, porém a ordem lógica entre os elementos que compõem a lista deve ser mantida.



Listas Lineares



Listas Lineares

Operações com Listas

- Criação de uma lista
  - Inserção de um elemento da lista
  - Remoção de um elemento da lista
  - Acesso de um elemento da lista
  - Alteração de um elemento da lista
  - Combinação de duas ou mais listas
  - Classificação da lista
  - Cópia da lista

Listas Lineares

Estrutura de Dados

```
Typedef struct  {
  int capacidade;
  float *dados;
  int pos ;
  int nItens;
} Lista ;
```

Listas Lineares

Funções

```
void criarLista(  Lista *f, int c ) {
 f->capacidade = c;
 f->dados = (float*) malloc (f->capacidade * sizeof(float));
 f->pos = -1;
 f->nItens = 0;
                                    }
```

Listas Lineares

Funções

```
int inserir( Lista *f, int v) {
if(f->pos == f->capacidade-1)
                        return(0);
                    else {
                    f->pos++;
                    f->dados[f->pos] = v;
                    f->nItens++; return(1); }
}
```

Listas Lineares

Tipos de Listas Lineares

- Pilhas

  Uma pilha é uma lista linear do tipo LIFO - **L**ast **I**n **F**irst **O**ut, o último elemento que entrou, é o primeiro a sair. Ela possui apenas uma entrada, chamada de topo, a partir da qual os dados entram e saem dela. Exemplos de pilhas são: pilha de pratos, pilha de livros, pilha de alocação de variáveis da memória, etc.

Listas Lineares

- Filas

  Uma fila é uma lista linear do tipo FIFO - **F**irst **I**n **F**irst **O**ut, o primeiro elemento a entrar será o primeiro a sair. Na fila os elementos entram por um lado (“por trás”) e saem por outro (“pela frente”). Exemplos de filas são: a fila de caixa de banco, a fila do INSS, etc.

Listas Lineares

- Deques

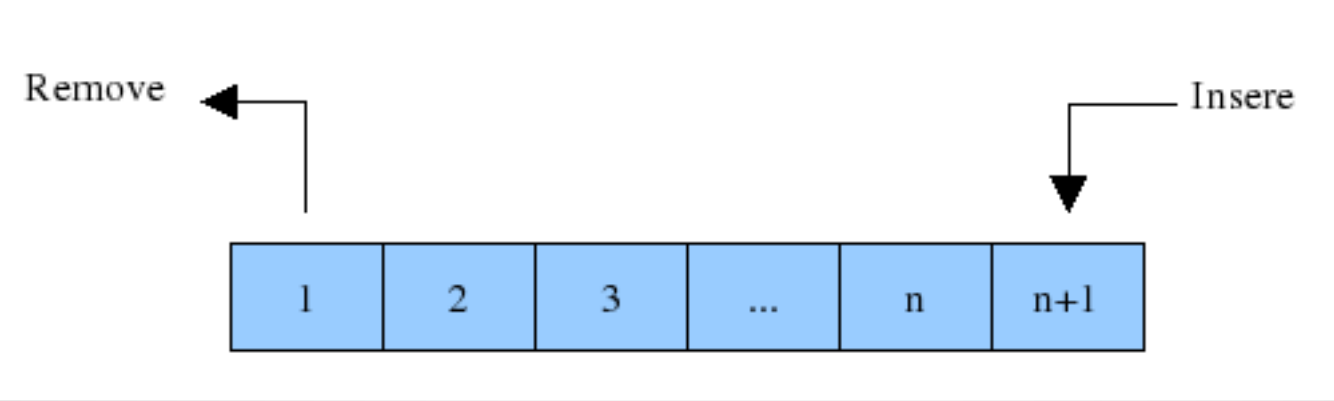
  Um deque - **D**ouble-**E**nded **QU**E**u**e) é uma lista linear na qual os elementos entram e saem tanto pela “pela frente” quanto“por trás”. Pode ser considerada uma generalização da fila.

Assim o que vai distinguir os diferentes tipos de listas são as operações que se podem realizar sobre as mesmas, podendo tanto serem implementadas com alocação sequencial quanto com alocação encadeada.

Listas Lineares

Filas - Queue

São estruturas de dados do tipo FIFO (first-in first-out), onde o primeiro elemento a ser inserido, será o primeiro a ser retirado, ou seja, adiciona-se itens no fim e remove-se do início.



Listas Lineares

São exemplos de uso de fila em um sistema:

- Controle de documentos para impressão;
- Troca de mensagem entre computadores numa rede;

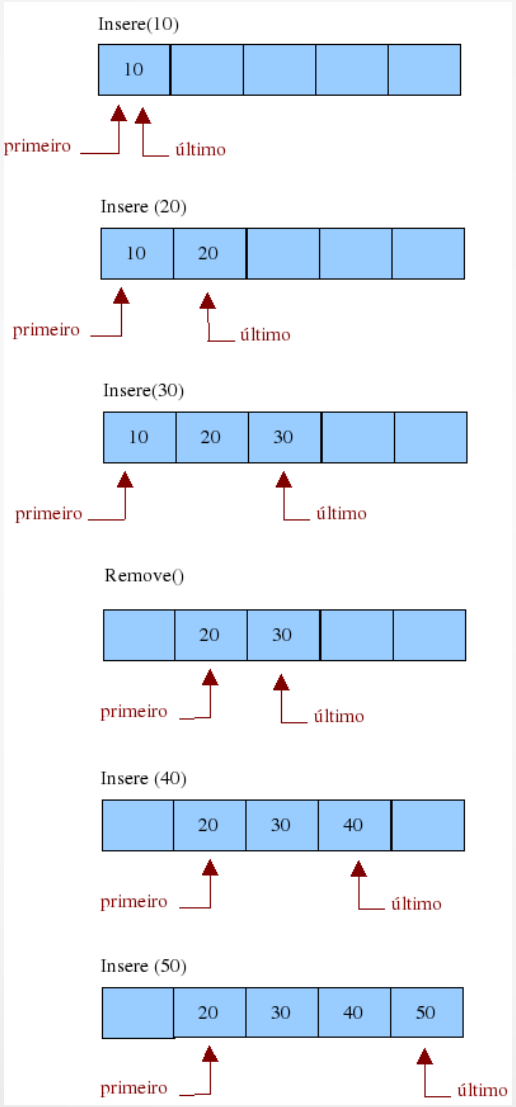
A implementação de filas pode ser realizada através de vetor (alocação do espaço de memória para os elementos é contígua) ou através de listas encadeadas.

Listas Lineares

Operações com Fila:

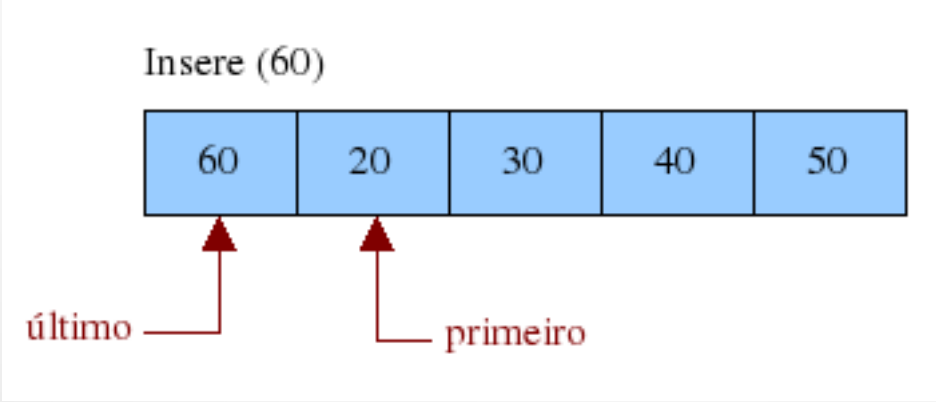
- criação da fila
  - enfileirar (enqueue) - o elemento é o parâmetro nesta operação;
  - desenfileirar (dequeue);
  - mostrar a fila (todos os elementos);
  - verificar se a fila está vazia (isEmpty);
  - verificar se a fila está cheia (isFull - implementação sequencial - vetor).

Listas Lineares



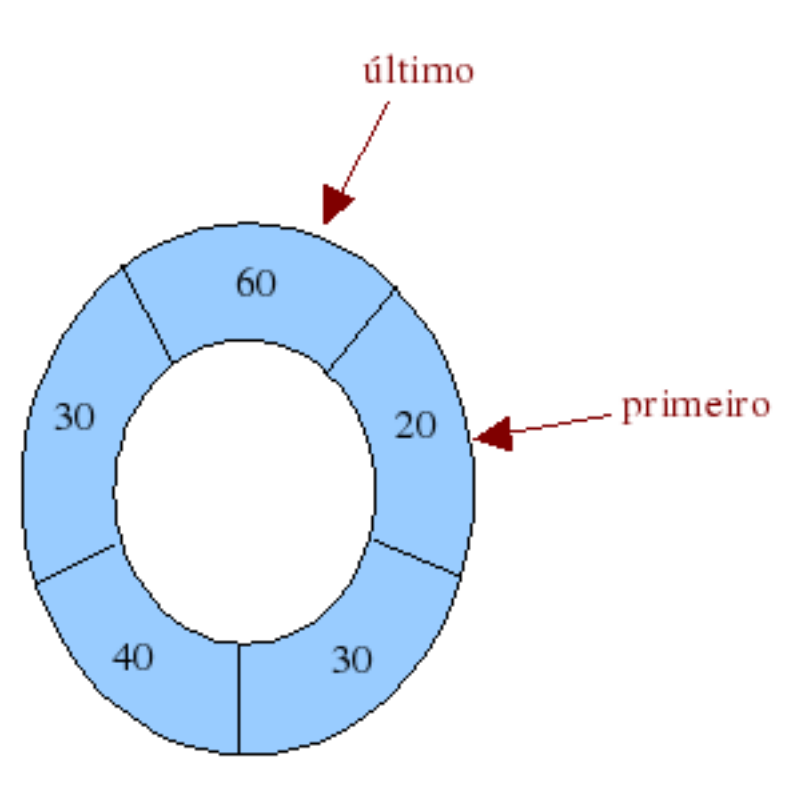
Listas Lineares

Para evitar problemas de não ser capaz de inserir mais elementos na fila, mesmo quando ela não está cheia, as referências primeiro e último circundam até o inicio do vetor, resultando numa fila circular.



Listas Lineares

Desta forma a fila simula uma representação circular:



Listas Lineares

```
Typedef struct  {
    int capacidade;
   float *dados;
   int primeiro;
    int ultimo;
   int nItens;
     } Fila ;
```

Listas Lineares

```
void criarFila(  Fila *f, int c ) {

f->capacidade = c;

f->dados = (float*) malloc (f->capacidade * sizeof(float));

f->primeiro = 0;

f->ultimo = -1;

f->nItens = 0;

}
```

Listas Lineares

```
void inserir( Fila *f, int v) {

if(f->ultimo == f->capacidade-1)

f->ultimo = -1;

f->ultimo++;

f->dados[f->ultimo] = v; // incrementa ultimo e insere

f->nItens++; // mais um item inserido

}
```

Listas Lineares

```
int remover(  Fila *f ) { // pega o item do comeÃ§o da fila
float temp = f->dados[f->primeiro++]; // pega o valor e incrementa o primeiro
if(f->primeiro == f->capacidade)
f->primeiro = 0;

f->nItens--;  // um item retirado
return temp;
}
```

Listas Lineares

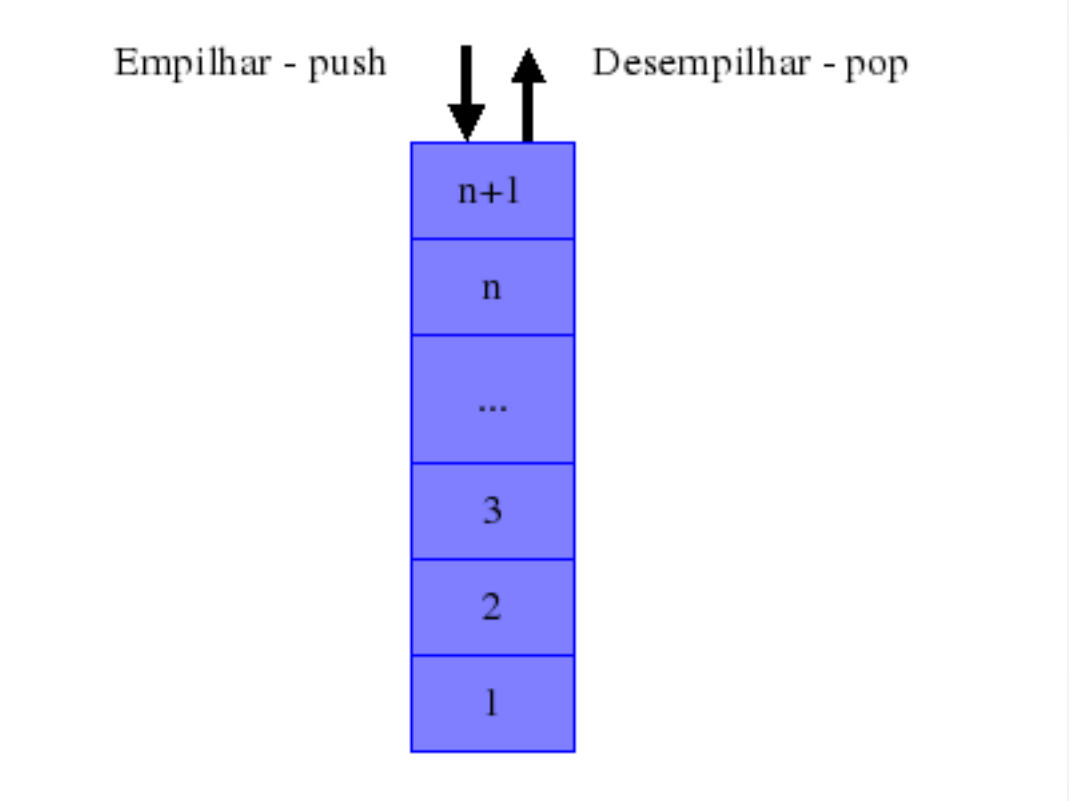
```
void mostrarFila( Fila *f){

        int cont, i;

        for ( cont=0, i= f->primeiro; cont < f->nItens; cont++){

                printf("%.2f\t",f->dados[i++]);

                if (i == f->capacidade)
                        i=0;

        }
        printf("\n\n");

}
```

Listas Lineares

Pilhas - Stack

São estruturas de dados do tipo LIFO (last-in first-out), onde o último elemento a ser inserido, será o primeiro a ser retirado. Assim, uma pilha permite acesso a apenas um item de dados - o último inserido. Para processar o penúltimo item inserido, deve-se remover o último.

Listas Lineares

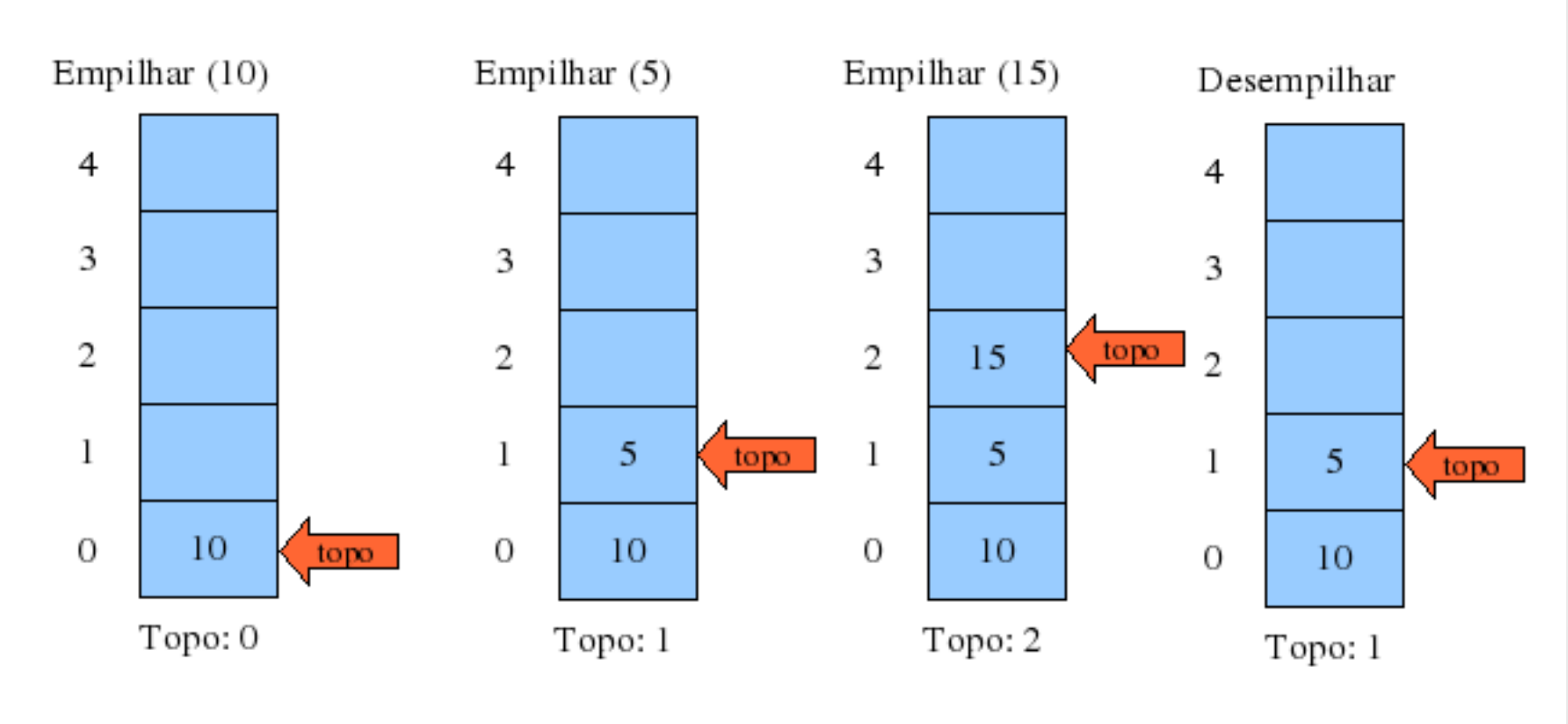


Listas Lineares

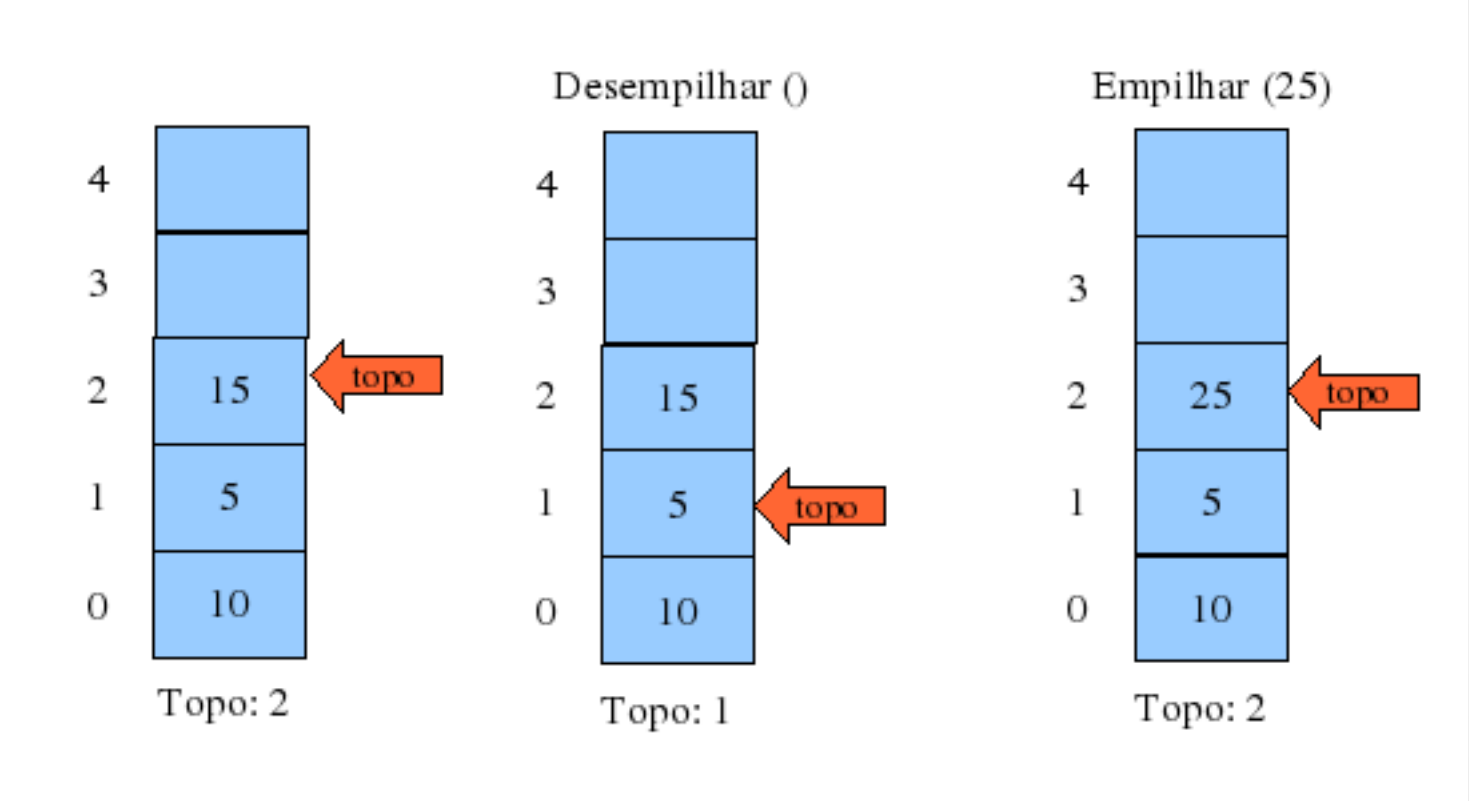
Exemplos :

- Funções recursivas em compiladores;
- Mecanismo de desfazer/refazer dos editores de texto;
- Navegação entre páginas Web;
- etc.

Listas Lineares



Listas Lineares



Listas Lineares

```
struct Pilha {

        int topo; /* elemento topo */
        int capa;
        float *pElem;

};
```

Listas Lineares

```
void criarpilha( struct Pilha *p, int c ){

   p->topo = -1;
   p->capa = c;
   p->pElem = (float*) malloc (c *
sizeof(float));

}
```

Listas Lineares

```
Void empilhar ( struct Pilha *p, float v){

        p->topo++;
        p->pElem [p->topo] = v;

}
```

Listas Lineares

```
float desempilhar ( struct Pilha *p ){

   float aux = p->pElem [p->topo];
   p->topo--;
   return aux;

}
```

Listas Lineares

Listas Lineares Encadeadas

Em listas lineares encadeadas, diferentemente das listas lineares sequenciais (ou contíguas), os elementos não estão necessariamente armazenados sequencialmente na memória. Assim, para manter a ordem lógica entre os elementos, as listas lineares encadeadas podem ser implementadas de duas formas:

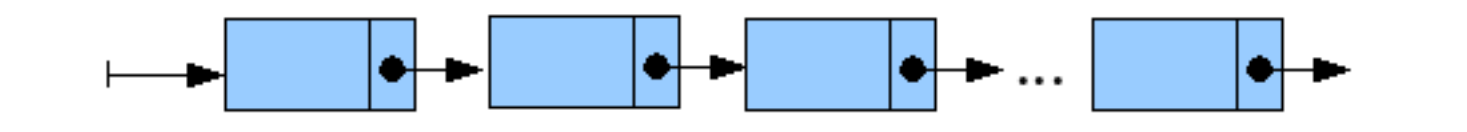
Listas Lineares

- **Simplesmente encadeada**

  Numa lista linear simplesmente encadeada cada elemento possui, além do espaço para armazenamento da informação, um espaço para armazenar uma referência da localização na memória onde o próximo elemento da lista (ou o anterior) se encontra.

Listas Lineares

A representação simbólica para a lista linear simplesmente encadeada é apresentada a seguir:

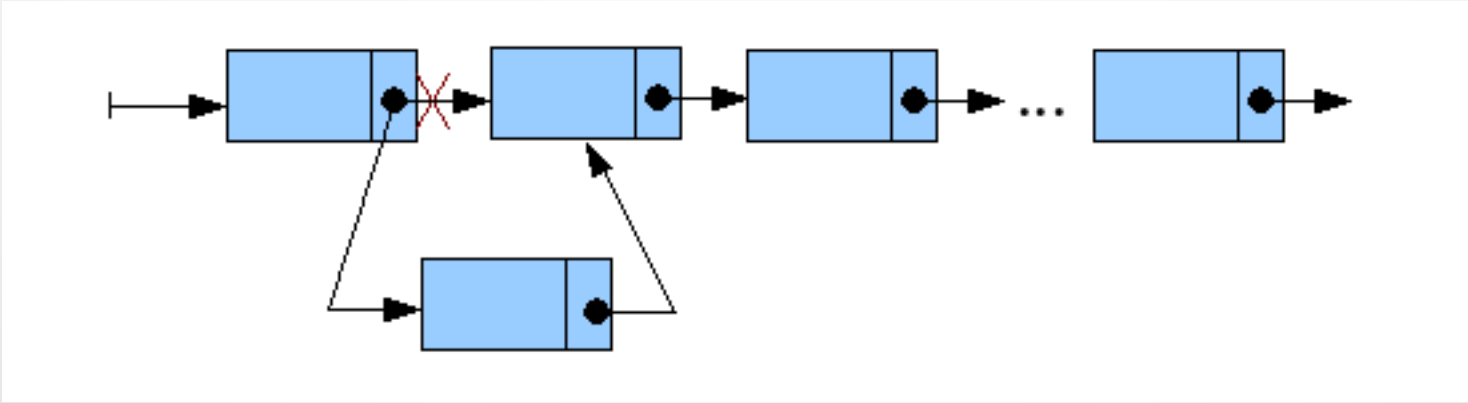


Listas Lineares

Rotinas de Manipulação Lista Simplesmente Encadeada

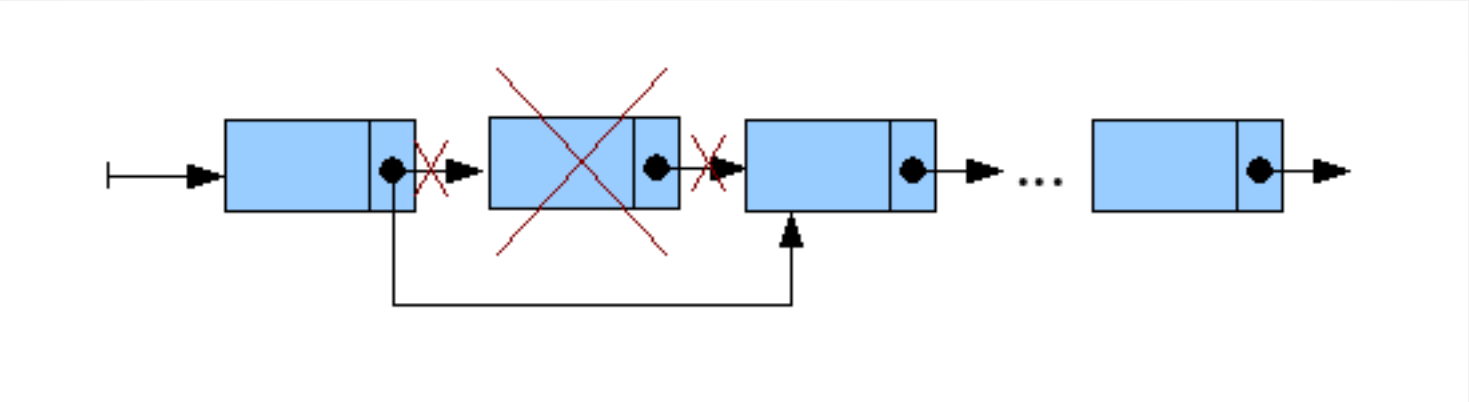
As rotinas inserção e remoção de elementos numa lista são realizadas manipulando-se a referência ao próximo nó da lista.

Inserção:



Listas Lineares

Remoção:



Listas Lineares

```
#include <stdio.h>
#include <stdlib.h>

struct Node{
int num;
struct Node *prox;
};

typedef struct Node node;
```

Listas Lineares

```
void inicia(node *LISTA);
int menu(void);
void opcao(node *LISTA, int op);
node *criaNo();
void insereFim(node *LISTA, int x);
void insereInicio(node *LISTA, int x);
void exibe(node *LISTA);
void libera(node *LISTA);
```

Listas Lineares

```
int main(void)
{
node *LISTA = (node *) malloc(sizeof(node));
if(!LISTA){printf("Sem memoria disponivel!\n");
            exit(1);}
inicia(LISTA);
int opt;
do{
opt=menu();
opcao(LISTA,opt);
}while(opt!=5);

free(LISTA);
return 0;
}
```

Listas Lineares

```
void inicia(node *LISTA)
{LISTA->prox = NULL;}

int menu(void)
{int opt;
printf("Escolha a opcao\n");
printf("1. Exibir lista\n");
printf("2. Adicionar node no inicio\n");
printf("3. Adicionar node no final\n");
printf("4. Zerar lista\n");
printf("5. Sair\n");
printf("Opcao: "); scanf("%d", &opt);
return opt;
}
```

Listas Lineares

```
void opcao(node *LISTA, int op)
{   int x ;
switch(op){
Case 1: exibe(LISTA);break;
case 2: printf("Novo elemento: ");   scanf("%d", &x);
        insereInicio(LISTA,x);break;
case 3: printf("Novo elemento: ");   scanf("%d", &x);
        insereFim(LISTA,x);break;
Case 4: libera(LISTA);break;
Case 5: libera(LISTA);break;
default:printf("Comando invalido\n\n");
}
}
```

Listas Lineares

```
int vazia(node *LISTA)
{
if(LISTA->prox == NULL)
return 1;
else
return 0;
}
```

Listas Lineares

```
void insereFim(node *LISTA,int x)
{
node *novo=(node *) malloc(sizeof(node));
if(!novo){ printf("Sem memoria disponivel!\n");exit(1);}
novo->prox = NULL;
novo->num = x ;
if(vazia(LISTA))LISTA->prox=novo;
else{node *tmp = LISTA→prox;

while(tmp->prox != NULL)
tmp = tmp->prox;

tmp->prox = novo;
}}
```

Listas Lineares

```
void insereInicio(node *LISTA, int x)
{ node *novo=(node *) malloc(sizeof(node));
  if(!novo){ printf("Sem memoria disponivel!\n");
             Exit(1);}

node *oldHead = LISTA->prox;

LISTA->prox = novo;
novo->num = x ;
novo->prox = oldHead;
}
```

Listas Lineares

```
void exibe(node *LISTA){
if(vazia(LISTA)){printf("Lista vazia!\n\n");return;}

node *tmp;
tmp = LISTA->prox;

while( tmp != NULL){
printf("%5d", tmp->num);
tmp = tmp->prox;
}
printf("\n\n");
}
```

Listas Lineares

```
void libera(node *LISTA)
{ if(!vazia(LISTA)){
    node *proxNode,*atual;
  atual = LISTA→prox;
  while(atual != NULL){
    proxNode = atual->prox
    free(atual)
    atual = proxNode; }
}}
```

Listas Lineares

```
node * busca (node *LISTA, int x, node ** ant)
{ node *ptr ;
  *ant = LISTA ;
  ptr = NULL ;
  if(vazia(LISTA)) return NULL ;
                else
                { ptr = LISTA->prox;
 while (ptr != NULL)
   { if (ptr->num != x) { *ant = ptr ;
                        ptr = ptr->prox ;}
                 else
                 {if (ptr->num == x)  return ptr ;
        else  return (NULL);
 break;
 }
   }
```

Listas Lineares

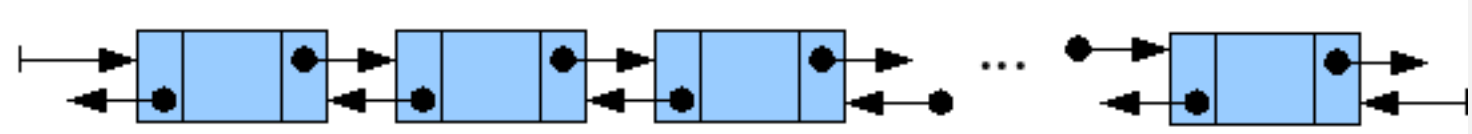
```
int remove(node *LISTA, int x)
{ node * ant ;
  node *aux = buscaOrdenada(LISTA, x, &ant);
  if (aux)  { ant->prox = aux->prox ;
              free(aux);
              return(1);
            }
              else return 0 ;
 }
```

Listas Lineares

- **Duplamente encadeada**

  Numa lista linear duplamente encadeada cada elemento possui, além do espaço para armazenamento da informação, um espaço para armazenar a referencia da localização de memória onde se encontra o próximo elemento da lista e outro espaço para armazenar a referencia da localização de memória onde se encontra o elemento anterior.

Listas Lineares



Listas Lineares

Uma primeira vantagem da utilização de lista duplamente encadeada sobre a lista simplesmente encadeada é a maior facilidade para navegação, que na lista duplamente encadeada pode ser feita nos dois sentidos, ou seja, do início para o fim e do fim para o início. Isso facilita a realização de operações tais como inclusão e remoção de nós, pois diminui a quantidade de variáveis auxiliares necessárias.

Listas Lineares

- A principal vantagem da utilização de listas encadeadas sobre listas sequenciais é o ganho em desempenho em termos de velocidade nas inclusões e remoções de elementos.
- Em uma lista contígua é necessário mover todos os elementos da lista para uma nova lista para realizar essas operações.
- Com estruturas encadeadas, como não existe a obrigatoriedade dos elementos estarem em posições contíguas da memória, basta realizar alterações nas referências dos nós, sendo um novo nó rapidamente inserido ou removido.

Listas Lineares

- Esta estrutura é mais adequada para listas com centenas ou milhares de nós, onde uma inserção ou remoção em uma lista contígua representará uma perda notável no desempenho do processamento.

Listas Lineares

```
#include <stdio.h>
#include <stdlib.h>

struct Node{
int num;
struct Node *ant;
struct Node *prox;
};
typedef struct Node node;
```

Listas Lineares

```
void inicia(node *LISTA);
int menu(void);
void opcao(node *LISTA, int op);
void exibe(node *LISTA);
void exibeInversa(node *LISTA);
Node * buscaOrdenada(node *LISTA, int x, node
** ant);
void insere(node *LISTA, int x);
void retira(node *LISTA,int x);
void libera(node *LISTA);
```

Listas Lineares

```
int main(void)
{  node *LISTA = (node *) malloc(sizeof(node));
   if(!LISTA){
   printf("Sem memoria disponivel!\n");exit(1);}
   inicia(LISTA);
   int opt;

   do{
    opt=menu();
    opcao(LISTA,opt);
    }while(opt!=7);

    free(LISTA);
    return 0;}
```

Listas Lineares

```
Void inicia(node *LISTA)
{
LISTA->ant = NULL;
LISTA->prox = NULL;
LISTA->num = -1;
}
```

Listas Lineares

```
int menu(void)
{ int opt;
printf("Escolha a opcao\n");
printf("1. Exibir lista\n");
printf("2. Exibir lista Inversa\n");
printf("3. BUSCA\n");
printf("4. INSERIR\n");
printf("5. RETIRAR \n");
printf("6. Zerar lista\n");
printf("7. Sair\n");
printf("Opcao: "); scanf("%d", &opt);
return opt;
}
```

Listas Lineares

```
void opcao(node *LISTA, int op)
{   int x ; node *ptr=NULL , *ant = NULL ;
   switch(op){
        Case 1: exibe(LISTA);break;
        Case 2: exibeInversa(LISTA);break;
        Case 3: printf("\n BUSCA elemento: ");     scanf("%d", &x); ptr =
buscaOrdenada(LISTA,x,&ant); if (ptr==NULL) printf(" \n Elemento nao
encontrado %d \n\n",ant→num); else printf("\n elemento: %d \n \n ",ptr-
>num); break;
        case 4: printf("\n Novo elemento: "); scanf("%d", &x);
insere(LISTA,x); printf("\n");break;
        case 5: printf("\n BUSCA elemento: "); scanf("%d",
&x);retira(LISTA,x);printf("\n");break;
        case 6:libera(LISTA); break;
        case 7:break;
        Default: printf("Comando invalido\n\n");
            }
}
```

Listas Lineares

```
int vazia(node *LISTA)
{
if(LISTA->prox == NULL)
return 1;
else
return 0;
}
```

Listas Lineares

```
void exibe(node *LISTA)
{    if(vazia(LISTA)){ printf("\nLista vazia!\n\n");
                       exit(1) ;}

node *tmp;
tmp = LISTA->prox;

while( tmp != LISTA){
printf("%5d", tmp->num);
tmp = tmp->prox;
}
printf("\n\n");
}
```

Listas Lineares

```
void exibeInversa(node *LISTA)
{ if(vazia(LISTA)){ printf("\n Lista vazia!\n\n");
                    return ;}

node *tmp;
tmp = LISTA->ant;

while( tmp != LISTA){
printf("%5d", tmp->num);
tmp = tmp->ant;
}
printf("\n\n");
}
```

Listas Lineares

```
void insere(node *LISTA,int x)
{   node *tmp1 = NULL , *ant ;
    node *novo=(node *) malloc(sizeof(node));
    if(!novo){printf("Sem memoria disponivel!\n"); exit(1);}
    tmp1 = buscaOrdenada(LISTA,x,&ant) ;
    if (tmp1 == NULL ) { novo->num = x ;
                    if(vazia(LISTA)){ LISTA->prox=novo; novo-
>ant = LISTA ; LISTA->ant = novo ;novo->prox = LISTA;}

                else{  node *tmp = ant->prox ; ant->prox = novo ;
                          novo->ant = ant ; novo->prox = tmp ;
                        tmp->ant =novo; }
                      }
  else   printf("\nElemento ja esta na lista!\nn");
   }
```

Listas Lineares

```
void retira(node *LISTA, int x)
{ node *tmp = NULL , *ant ;
  tmp = buscaOrdenada(LISTA,x,&ant) ;
  if(tmp==NULL )  printf("\nElemento n'ao Existe!\n\n");
             else
              if (LISTA->prox->prox == LISTA)
                  { node *tmp = LISTA->prox;
                    LISTA->prox = NULL ;
                    LISTA->ant = NULL ;
                    free(tmp);}
                  else {
                    node *aux1 = tmp->prox;
                    node *aux2 = tmp->ant;
                    aux2->prox= aux1;
                    aux1->ant = aux2; ;
                    free(tmp);
                        }

}
```

Listas Lineares

```
node * buscaOrdenada(node *LISTA, int x, node ** ant)
{ node *ptr ;   *ant = LISTA ;   ptr = NULL ;
  if(vazia(LISTA)) return NULL ;
                else
                if (x>LISTA->ant->num)
                    {*ant = LISTA->ant ; return NULL ;}
                  else
                    { ptr = LISTA->prox;
                      while (ptr != LISTA)
                       { if (ptr->num < x) { *ant = ptr ;
                                             ptr = ptr->prox ;}
                                  else
                                  {if (ptr->num == x)  return ptr ;
                                      else  return (NULL);
                                   Break; }
                       }
                     }
 }
```

Listas Lineares

```
void libera(node *LISTA)
{  if(!vazia(LISTA)){ node *proxNode,*atual;
                      atual = LISTA->prox;
                      while(atual != NULL){
                      proxNode = atual->prox;
                      free(atual);
                      atual = proxNode;}
                    }
}
```

Listas Lineares

Listas Lineares