Estruturas Lineares

Estruturas homogêneas

- vetores (tipos básicos repetidos)
- matrizes
- Listas
- Pilhas e filas

Listas

- Uma lista é um conjunto de dados ordenados e de número variável de elementos.
- Há 2 tipos de listas:
  - Lista seqüencial
  - Lista encadeada

Lista seqüencial

- Uma lista sequencial é um conjunto de n nós ($n \geq 0$; X_1, $\boldsymbol{X}_2$,..., X_n) com as seguintes propriedades:
  - Se $n > 0$, então X_1 é o primeiro nó da lista e X_n é o último;
  - Para $1 < k < n$, X_k é precedido por X_{k-1} e sucedido por X_{k+1} ;
  - Se $n = 0$, então a lista é vazia.
- Representação por contiguidade
- Uma lista sequencial aproveita a sequencialidade da memória.
- Uma lista L (com n nós e todos os nós de mesmo tamanho) ocupa um espaço consecutivo na memória equivalente a n*tamanho do nó.

	1	*2*	*3*	*4*	...	*n*	...	...	*m*
L(n)=									

- Operações
  - Acesso a um determinado elemento.
  - Inserção de um novo elemento.
  - Remoção de um elemento.
  - Concatenação de duas ou mais listas lineares.
  - Separação de uma lista em duas ou mais listas.
  - Ordenação.
  - Contagem de elementos.

Listas encadeadas

- As listas encadeadas permitem a utilização de estruturas flexíveis em relação à sua quantidade de elementos, tendo em vista sua característica dinâmica.
- Cada elemento é um nó composto por uma parte que armazena dados e outra que armazena campos de ligação com outros nós.
- O campo de ligação dos nós contém o endereço de memória onde o próximo nó está armazenado.
- A passagem de um nó para outro da estrutura encadeada é realizada através dos endereços de outros nós.
- Os nós podem estar em qualquer lugar na memória.
- As estruturas encadeadas podem ser implementadas de diferentes modos, no entanto, a implementação mais flexível é por meio de ponteiros.
- As listas encadeadas permitem fácil inserção e remoção de elementos sem um impacto global na estrutura.
- Uma desvantagem da lista encadeada é o acesso sequencial.
- Para acessar um nó no meio da lista, todos os nós anteriores (ou posteriores) devem ser visitados.
- Outra desvantagem é a necessidade de armazenar informações adicionais, no caso, os ponteiros para outros nós.
- A lista encadeada é vantajosa quando há elementos que apresentam prioridade de acesso.

Listas simplesmente encadeadas

- Se um nó tem um vínculo somente para o seu sucessor na seqüência, a lista é simplesmente encadeada.

- Os nós são formados por um registro que possui pelo menos 2 campos, a informação e o endereço de memória onde está armazenado o próximo elemento da lista.
- O nó pode ser declarado por meio da seguinte estrutura:

```
struct node
{
    int info;
    struct node *proximo;
}
```

- Para criar uma lista encadeada deve-se manter uma variável armazenando sempre o endereço do primeiro elemento da lista.

```
aloca(primeiro)
se (primeiro) <> nulo
    primeiro→dados = 10
    primeiro → proximo=nulo
```

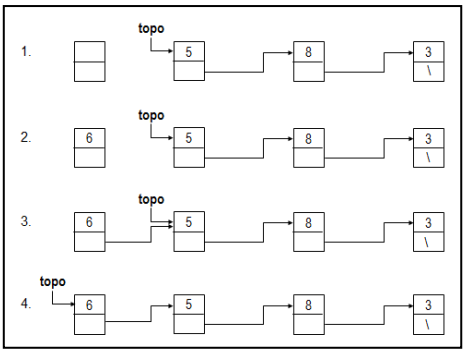
- Para o segundo elemento da lista.

```
aloca(n)
se (n) <> nulo
    n→dados = 8
    n→proximo=nulo
  primeiro = n
```

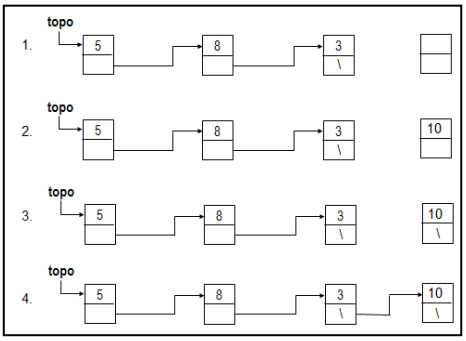
- Generalizando:

```
aloca(p)
se (p<>nulo)
    p→dados = valor
    p→proximo=nulo
    se (topo==nulo)      (se a lista estiver vazia)
        topo =p
    senão                (a lista contém nós)
        fim→proximo = NULL;
    fim=p
```

- Operações
  - Inserção de um nó no início da lista.
  - Inserção de um nó no fim da lista.
  - Inserção de um nó antes de um nó endereçado por k.
  - Remoção de um nó do início da lista.
  - Remoção de um nó do fim da lista.
  - Remoção de um nó antes de um nó endereçado por k.
  - Remoção de um nó com um valor determinado.
  - Busca.
- Inserção de um novo elemento no início da lista.
  - A adição de um nó no início de uma lista é realizada em 4 etapas.
    1. Um nó vazio é criado.
    2. O membro info do nó é inicializado com um valor particular.
    3. Como o nó é incluído no início da lista, o membro próximo se torna um ponteiro para o atual primeiro nó da lista.
    4. O novo nó precede todos os nós da lista de forma que o endereço do primeiro nó deve ser atualizado.



- Inserção de um novo elemento no fim da lista.
  - A adição de um nó no fim de uma lista é realizada em 4 etapas.
    1. Um nó vazio é criado.
    2. O membro info do nó é inicializado com um valor particular.
    3. Como o nó é incluído no final da lista, o membro proximo deste novo nó se torna nulo.
    4. O novo nó é incluído no fim lista de forma que o membro proximo do nó anterior deverá apontar para o novo nó.



- Inserção de um novo elemento.
  - Normalmente a inserção de elementos ocorre a partir do último elemento da lista.
  - Para evitar percorrer toda a lista, pode-se armazenar o endereço do último elemento da lista.

```
#include <iostream.h>
#include <stdio.h>
struct node
{
   int info;
   struct node *proximo;
};
```

```
void novodado( )
{
  node *novo = new node;
  cin >> novo->info;
}
```

- Inserir no início / inserir no fim → procedimentos diferentes
  - A função `novodado( )` adiciona um nó à lista.
  - O novo nó é inserido no final da lista.
  - Primeiramente, uma nova estrutura do tipo node é criada pela instrução: node `*novo = new node;`
  - A instrução reserva memória para armazenar a estrutura e atribui o endereço desta memória ao ponteiro novo.
  - Em seguida, a informação da nova estrutura é preenchida pelo usuário.
- Remoção
  - Uma operação de remoção consiste em remover um nó da lista e retornar o valor armazenado neste nó.
  - A operação de remoção permite liberar o espaço em memória referente ao nó especificado.
  - Há dois casos especiais a serem tratados:
    - Remoção de um nó de uma lista vazia
    - Remoção de um nó de uma lista com somente um nó
- Remoção de um nó do início da lista.

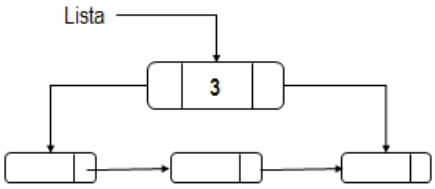
```
se (topo==nulo) status=falso      // a lista está vazia
senao{
      aux=topo
      valor = aux->dados
      topo=topo->proximo
      se topo = nulo   fim=nulo
      libera (aux)
      status = verdadeiro
  }
retorna valor
```

- Remoção de um nó do fim da lista.
  - Para remover um nó do fim da lista deve-se considerar 3 situações:
    - Lista Vazia: Para verificar se a lista está vazia basta verificar a variável topo, se o conteúdo for nulo, a lista está vazia.
    - Lista com apenas um elemento: Neste caso, verifica-se o conteúdo do campo próximo do primeiro nó da lista, se o conteúdo for nulo, a lista possui apenas um elemento.
    - Lista com dois ou mais elementos: Se nenhuma das situações anteriores for verdadeira, a lista possui dois elementos ou mais.

```
se (topo==nulo) status=falso      // a lista está vazia
senao{
      se (topo->proximo == nulo){      // a lista tem um único nó
            valor = topo->dados
            libera (topo)
            fim = nulo
            topo = nulo
        }
       p= topo
       enquanto (p->proximo <> nulo) {
             aux=p
             p=p->proximo
       }
      valor=p->dados
      aux->proximo=nulo
      fim=aux
      libera(p)
      status = verdadeiro
  }
   retorna valor
}
```

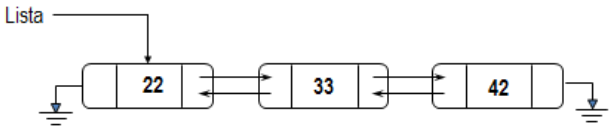
Listas encadeadas com header

- Para facilitar a gerência de informações de início e fim da lista pode-se reunir as referências em uma única estrutura chamada descritor, líder ou header da lista.
- O acesso aos elementos da lista é sempre realizado por meio do header.
- O header pode conter informações como: início e fim da lista, quantidade de nós da lista e outras informações que se deseje.



Listas duplamente encadeadas

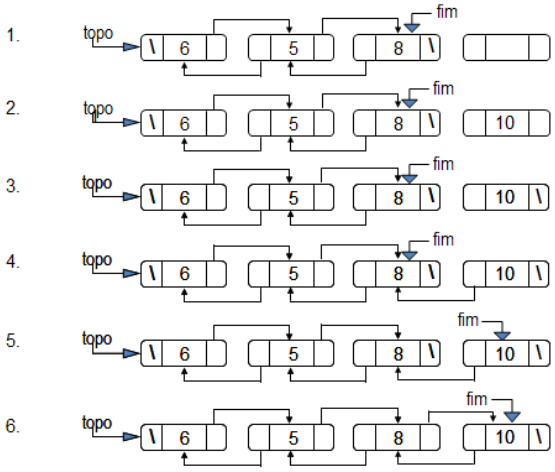
- Nas listas duplamente encadeadas, os nós contêm dois ponteiros um para o sucessor e outro para o predecessor.



- Definição da lista duplamente ligada com estruturas

```
struct Node{
          struct Node *prev;
          int dados;
          struct Node *proximo;
};
struct Node *topo=NULL, *fim=NULL;
```

- Inserção de um nó no final da lista
    1. O nó é criado.
    2. Insere-se o dado.
    3. O ponteiro para próximo torna-se nulo.
    4. O ponteiro para prev deve apontar para o último nó da lista (fim).
    5. O valor de fim é ajustado para o novo nó .
    6. O ponteiro proximo do nó anterior toma o endereço do novo nó.



- Inserção de um nó no final da lista

```
aloca (p)
    se (p<> nulo)
        p->dados = valor
        p->prev=fim
        p->prox = nulo
        fim->prox = p
        fim = p
        se (topo == nulo)
            topo = fim
```

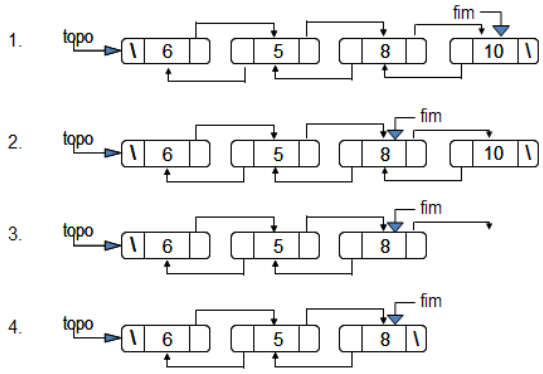
- Inserção de um nó no início da lista

```
aloca (p)
    se (p<> nulo)
        p->dados = valor
        p->prox=topo
        topo->prev=p
```

```
                p->prev = nulo
                topo = p
                se (fim == nulo)
                        fim=topo
```

- Remoção de um nó do fim da lista

```
        se (topo <> nulo)
                aux = fim
                fim = aux->prev
                se (fim == nulo)
                        topo = nulo
                senão fim->prox = nulo
                valor = aux->dados
                libera (aux)
                status=verdadeiro
                retorna valor
        senão
                 status=falso
```

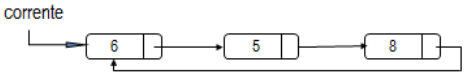


- Remoção de um nó do início da lista

```
        se (topo <> nulo)
                aux = topo
                topo = aux->prox
                se (topo == nulo)
                        fim = nulo
                senão topo->prev = nulo
                valor = aux->dados
                libera (aux)
                status=verdadeiro
                retorna valor
        senão
                 status=falso
```

Listas circulares

- Em uma lista circular os nós formam um anel.
- Um exemplo da utilização das listas circulares é quando diversos processos estão usando os mesmos recursos simultaneamente.
- Deve-se assegurar que os processos sigam uma ordem de utilização do recurso.
- Colocam-se os recursos em uma lista circular acessíveis através do ponteiro “corrente”.
- Depois que o processo é ativado o ponteiro se move para o próximo nó para ativar o processo seguinte.
- Na implementação de uma lista simplesmente encadeada circular usa-se apenas um ponteiro permanente.
- Lista circular simplesmente encadeada



- Lista circular duplamente encadeada
- Inserção de um nó antes do nó corrente

```
        aloca (p)
            se (p<> nulo){
                p->dados = valor
                se (corrente == nulo){
                        corrente = p
                        p->prox = p
                 }
                senao{
                        aux = corrente
                        enquanto (aux->prox <> corrente) aux = aux->prox
                        aux->prox=p
                         p->prox=corrente
                 }
            }
```

- Inserção de um nó antes do nó corrente (o novo nó se torna o nó corrente)

```
aloca (p)
    se (p<> nulo){
        p->dados = valor
        se (corrente == nulo){
            corrente = p
            p->prox = p
        }
        senao{
            aux = corrente
            enquanto (aux->prox <> corrente) aux = aux->prox
            aux->prox=p
            p->prox=corrente
            corrente=p
        }
    }
```

- Remoção do nó do corrente

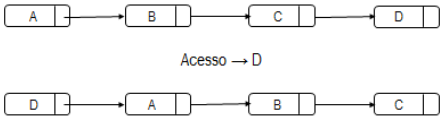
```
se (corrente <> nulo){
        aux = corrente
        se (aux->prox)<> corrente{
            enquanto (aux->prox <> corrente) aux = aux->prox
            aux ->prox = corrente->prox
           valor = corrente->dados
           libera(corrente)
           corrente = aux->prox
           status=verdadeiro
           retorna valor
       }
       senão{
            valor = corrente->dados
           libera(corrente)
           corrente = nul
           status=verdadeiro
           retorna valor

      }
senão
        status=falso
}
```

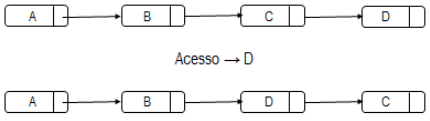
Listas auto-organizadas

- As listas encadeadas exigem a busca seqüencial para localizar um elemento ou descobrir que ele não está na lista.
- Pode-se melhorar a eficiência da busca organizando a lista dinamicamente.
- Existem diferentes métodos de organização de listas.

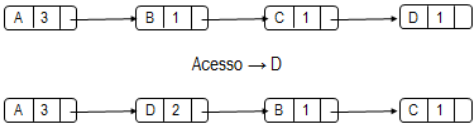
1. Método de mover para frente: O elemento localizado deve ser colocado no início da lista.



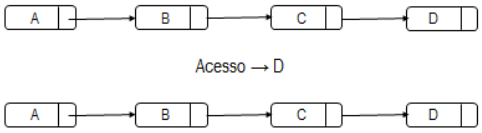
2. Método da transposição: O elemento localizado deve ser trocado com seu predecessor, exceto se ele estiver no topo da lista.



3. Método da contagem: A lista deve ser ordenada pelo número de vezes que os elementos são acessados.



4. Método da ordenação: A lista é ordenada de acordo com sua informação.



- Os três primeiros métodos permitem colocar os elementos mais prováveis de serem acessados no início da lista.
- O método da ordenação tem a vantagem na busca de informação, sobretudo se a informação não está na lista pois a busca pode terminar sem pesquisar toda a lista.

Exercícios

- Criar um programa para manipular uma lista duplamente encadeada
    - O programa deverá apresentar um menu com as seguintes opções:
        1. Inclusão no início da lista
        2. Inclusão no fim da lista
        3. Remoção do início da lista
        4. Remoção do fim da lista
        5. Impressão da lista com valor do topo, e todos os nós, contendo o endereço do nó, valor, endereço do ponteiro direito e do ponteiro esquerdo.