Estruturas de dados

Listas Lineares

Estruturas Lineares

- Estruturas de Dados Básicos
  - **Listas Lineares**
  - Pilhas
  - Filas

Listas

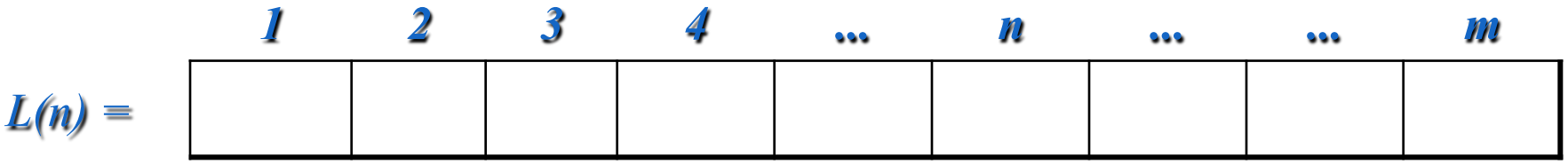
- Uma lista é um conjunto de dados e de número variável de elementos.
- Há 2 tipos de listas:
  - Lista seqüencial (por meio de Arranjos)
  - Lista encadeada (por meio de Estruturas Auto-Referenciadas)

Lista

- Uma lista é um conjunto de n nós ($n \geq 0$; X1, X2,..., Xn) com as seguintes propriedades:
  - Se $n > 0$, então X1 é o primeiro nó da lista e Xn é o último;
  - Para $1 < k < n$, Xk é precedido por Xk-1 e sucedido por Xk +1 ;
  - Se $n = 0$, então a lista é **vazia**.

Lista seqüencial

- Uma lista seqüencial aproveita a seqüencialidade 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.
- **A lista seqüencial é um vetor** .



m – número máximo de elementos que a lista L pode armazenar;

n – número de elementos existentes em um determinado momento.

Lista seqüencial

- 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

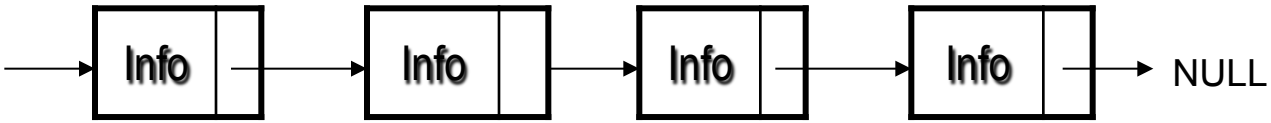
- As **listas encadeadas** permitem a utilização de estruturas flexíveis em relação à sua quantidade de elementos.
- Cada elemento é um nó composto por:
  - uma parte que **armazena dados** e
  - outra que **armazena campos de referencia** com outros nós.
- O campo de **referencia** dos nós **contém 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 do próximo nó.
  - 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**.

Listas encadeadas

- As listas encadeadas permitem fácil inserção e remoção de elementos sem um impacto global na estrutura.
- A lista encadeada é vantajosa quando há elementos que apresentam prioridade de acesso.
- **Desvantagem** da lista encadeada:
  - o acesso seqüencial.
    - Para acessar um nó no meio da lista, todos os nós anteriores (ou posteriores) devem ser visitados.
  - a necessidade de armazenar informações adicionais
    - os ponteiros para outros nós.

Listas simplesmente encadeadas

- Se um nó tem uma referencia **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**,
  - 1- a informação e
  - 2- o endereço de memória onde está armazenado o próximo elemento da lista.



Listas simplesmente encadeadas

O nó pode ser declarado por meio da seguinte estrutura:

```
struct node
{
   int info;
   struct node *proximo;
};
node usuario1, * usuario2;
```

- usuario1.info ➔ Membro info do objeto usuario1
- usuario2->info → Membro info do objeto apontado por usuario2

Listas simplesmente encadeadas

- 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->proximo = n
```

Listas simplesmente encadeadas

- Generalizando:

```
aloca(p)
se (p<>nulo)
    p->dados = valor
    p->proximo=nulo
    se (primeiro==nulo)    (se a lista estiver vazia)
        primeiro=p
ultimo=p
```

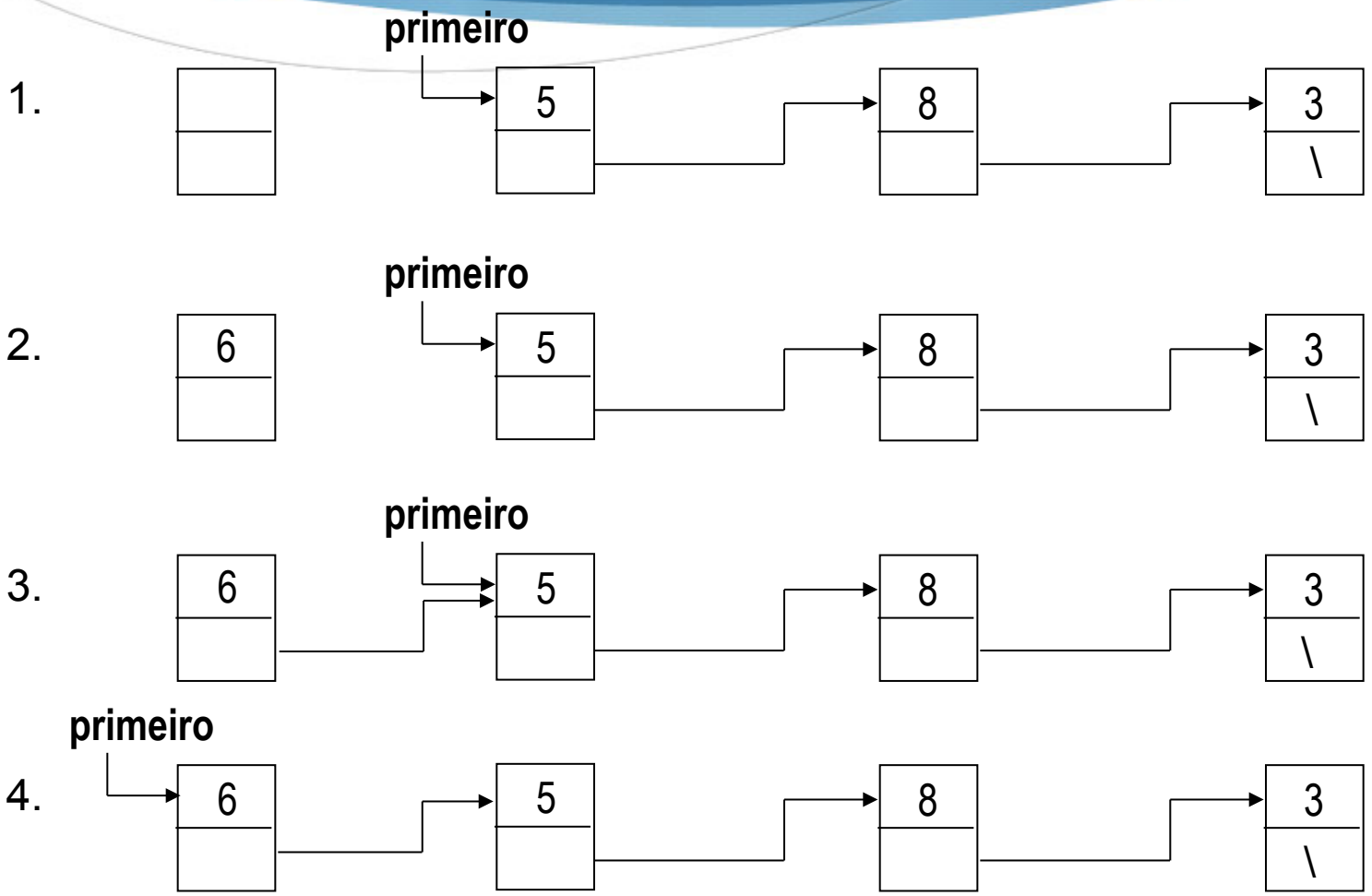
Listas simplesmente encadeadas

- 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**.

Listas simplesmente encadeadas

- Inserção de um novo elemento **no início da lista**.
  - A adição 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 proximo 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.

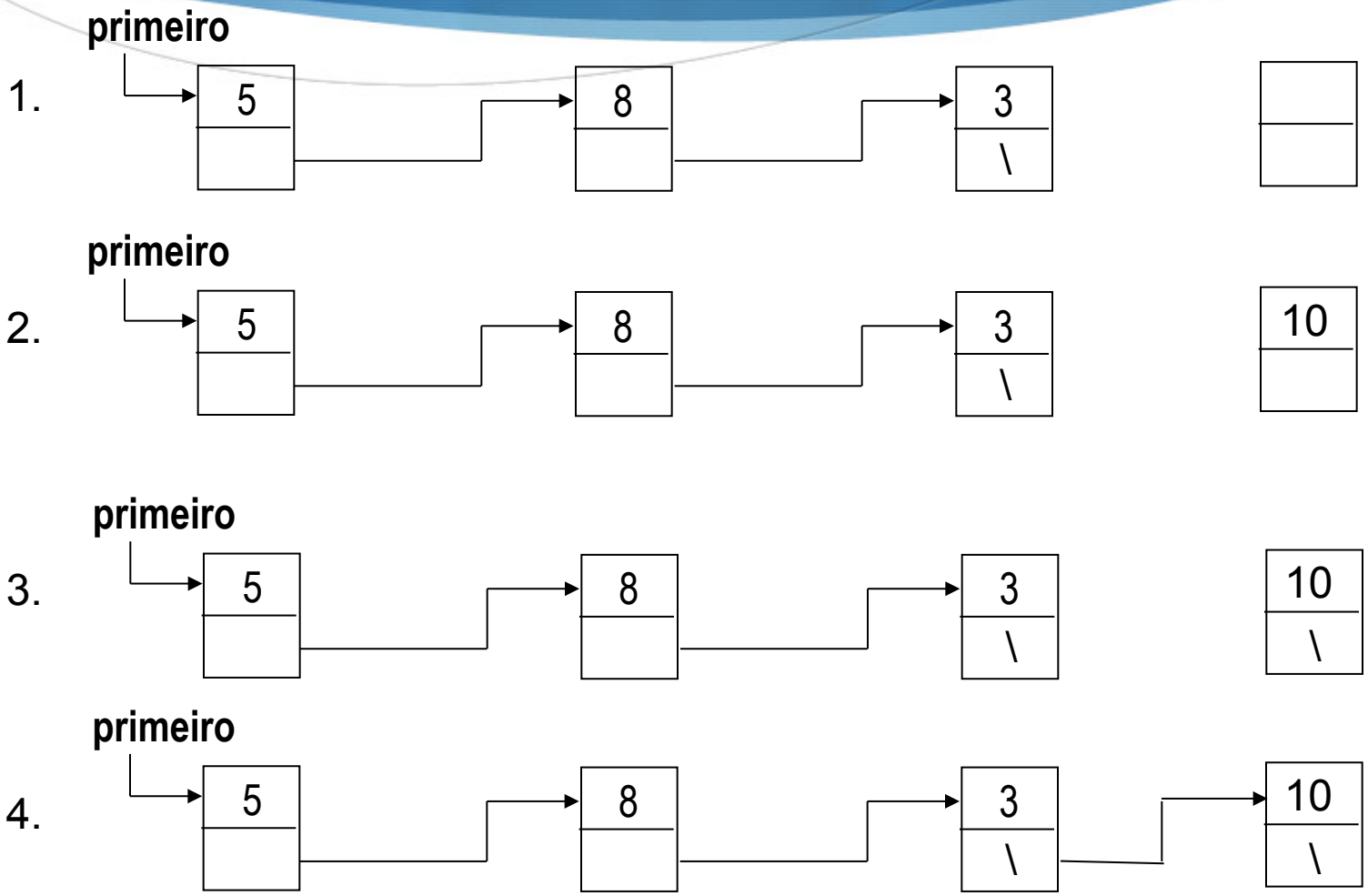
Listas simplesmente encadeadas



Listas simplesmente encadeadas

- Inserção de um novo elemento **no fim da lista**.
  - A adição 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**.

Listas simplesmente encadeadas



Listas simplesmente encadeadas

- 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.

Listas simplesmente encadeadas

```
#include <iostream>
#include <new>
using namespace std;
struct node{
    int info;
    struct node *proximo;
};
void novoDado( ){
     node *novo = new node;
     cout << "Insira a informacao: ";
     cin >> novo->info;
}
```

Inserir no início / inserir no fim -> procedimentos diferentes

Listas simplesmente encadeadas

- A função novoDado( ) adiciona um nó à 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.

Listas simplesmente encadeadas

- 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

Listas simplesmente encadeadas

- Remoção de um nó do início da lista.

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

Listas simplesmente encadeadas

- 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 **primeiro**, 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.

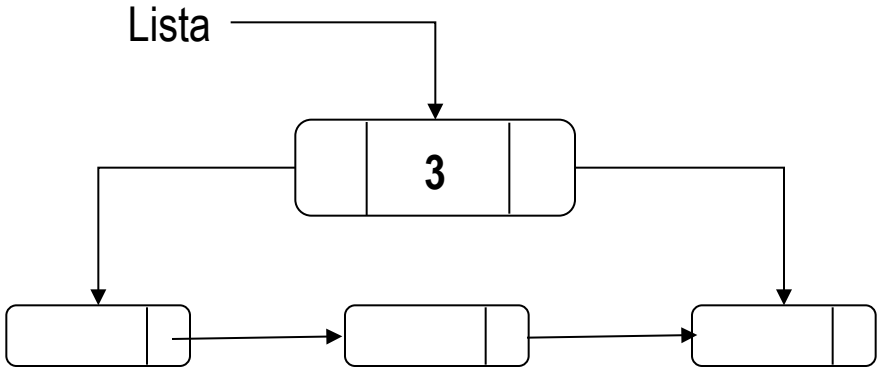
Listas simplesmente encadeadas

- Remoção de um nó do fim da lista.

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

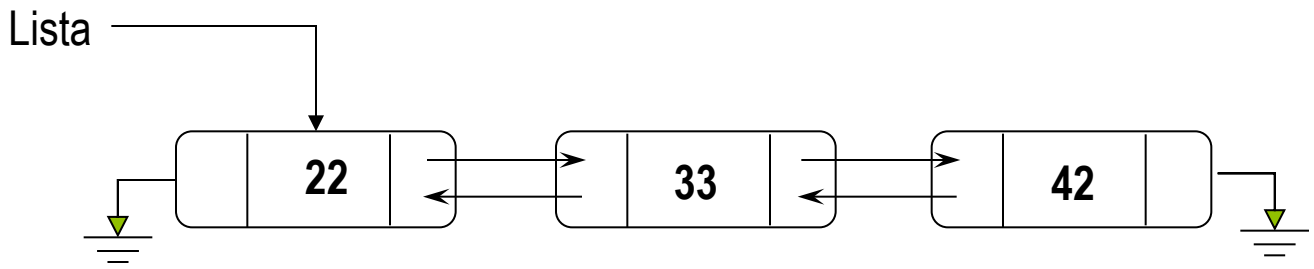
Listas encadeadas com header

- 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**.



Listas duplamente encadeadas

- Definição da lista duplamente ligada com estruturas

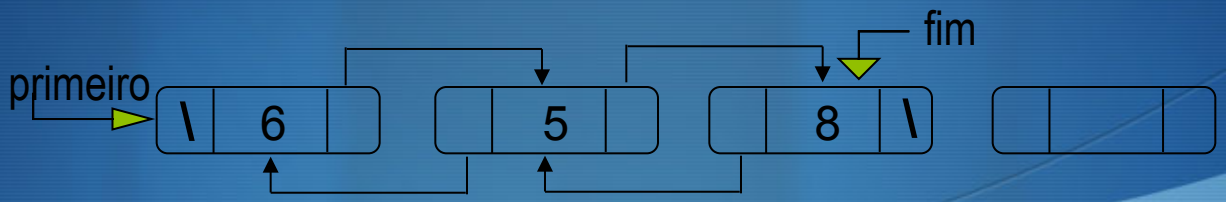
```
struct Node{
        struct Node *anterior;
        int dados;
        struct Node *proximo;
};
struct Node *primeiro=NULL, *fim=NULL;
```

Listas duplamente encadeadas

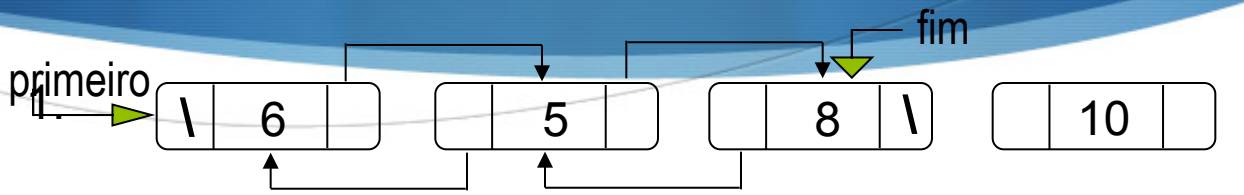
- Inserção de um nó no final da lista
  1. O nó é criado.
  2. Insere-se o dado.
  3. O ponteiro para próximo torna-se nulo.
  4. O ponteiro para prev deve apontar para o último nó da lista (fim).
  5. O valor de fim é ajustado para o novo nó .
  6. O ponteiro proximo do nó anterior toma o endereço do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

Listas duplamente encadeadas

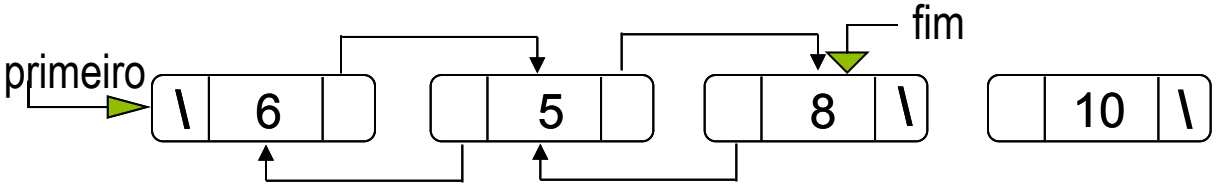
1.



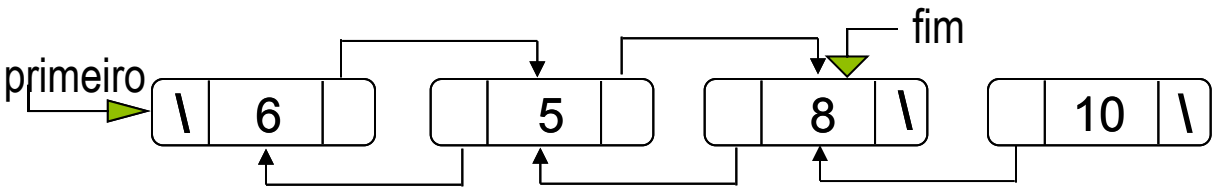
2.



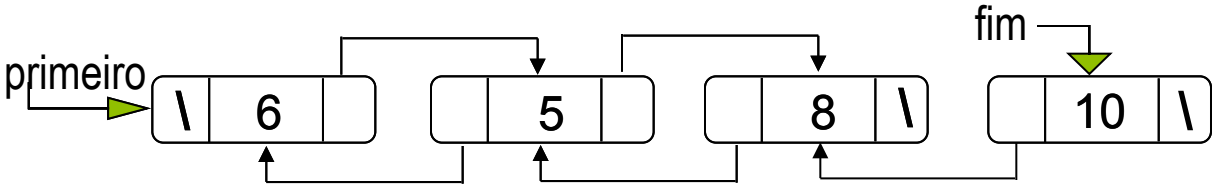
3.



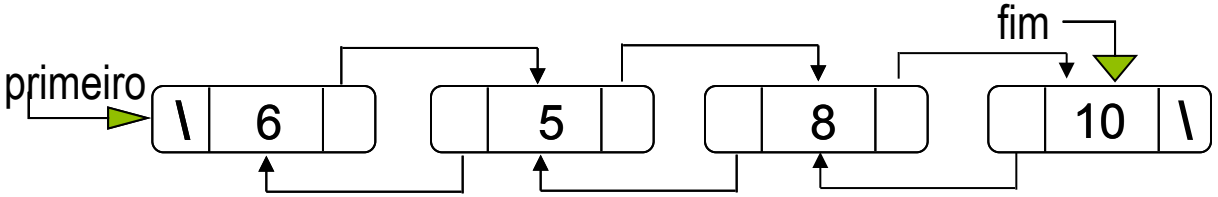
4.



5.



6.



Listas duplamente encadeadas

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 (primeiro== nulo)
            primeiro= fim
```

Listas duplamente encadeadas

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
        primeiro= p
        se (fim == nulo)
            fim=topo
```

Listas duplamente encadeadas

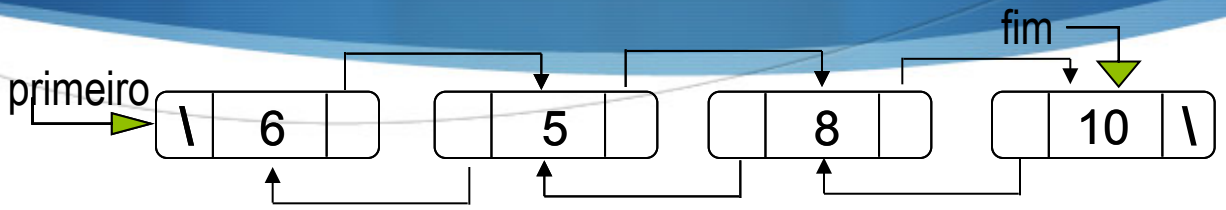
Remoção de um nó do fim da lista

```
se (primeiro<> 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
```

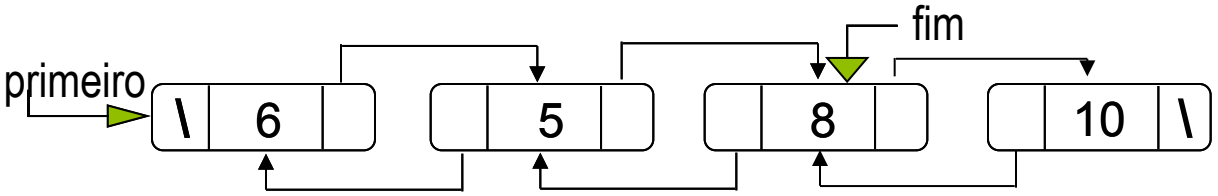
Listas duplamente encadeadas

Remoção de um nó do fim da lista

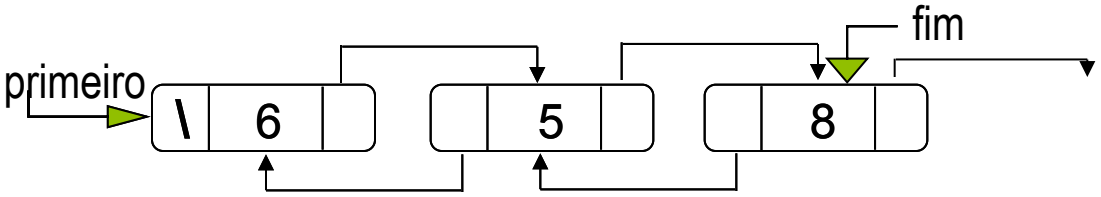
1.



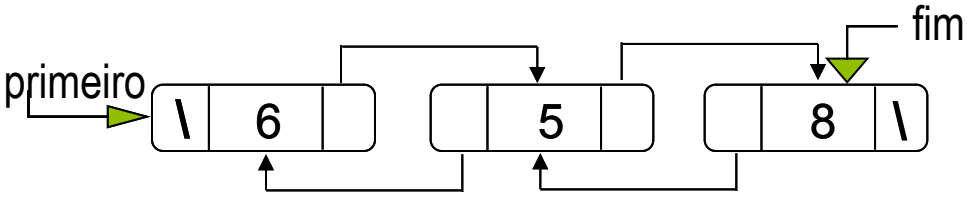
2.



3.



4.



Listas duplamente encadeadas

Remoção de um nó do início da lista

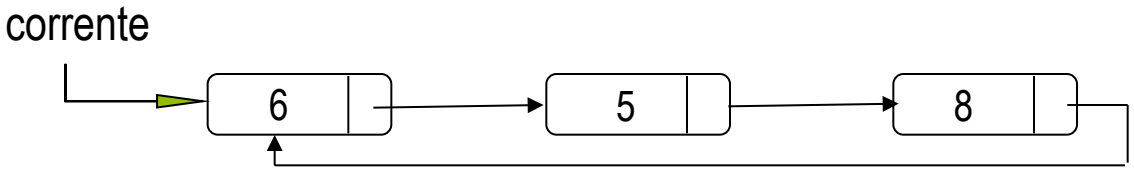
```
se (primeiro<> nulo)
        aux = topo
        primeiro= aux->prox
        se (primeiro== nulo)
            fim = nulo
        senão primeiro->prev = nulo
        valor = aux->dados
        libera (aux)
        status=verdadeiro
        retorna valor
senão
         status=falso
```

Listas circulares

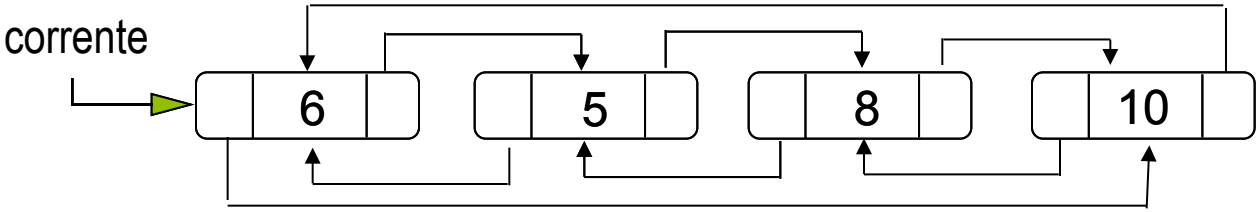
- Em uma lista circular os nós formam um anel.
  - Um exemplo da utilização das listas circulares é quando diversos processos estão usando os mesmos recursos simultaneamente.
  - Deve-se assegurar que os processos sigam uma ordem de utilização do recurso.
  - Coloca-se os recursos em uma lista circular 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**.

Listas circulares

- Lista circular simplesmente encadeada



- Lista circular duplamente encadeada



Listas circulares

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
         }
    }
```

Listas circulares

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
         }
    }
```

Listas circulares

Remoção do nó do corrente

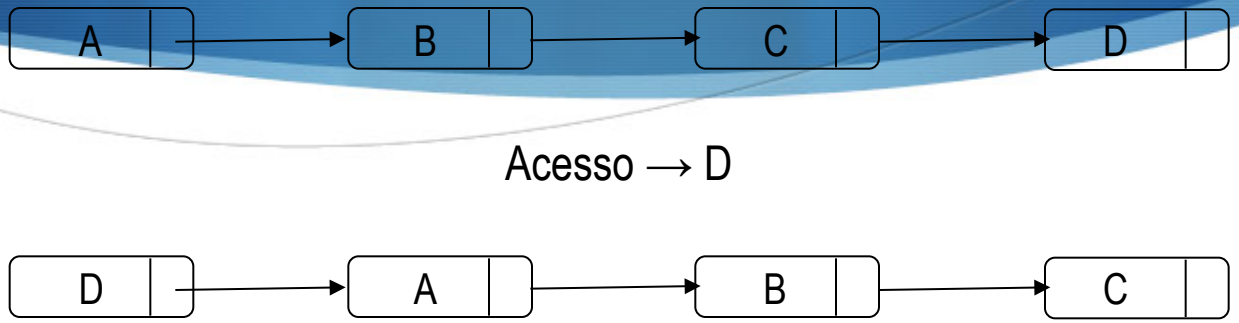
```
se (corrente <> nulo){
        aux = corrente
        se (aux->prox)<> corrente{
            enquanto (aux->prox <> corrente) aux = aux->prox
            aux ->prox = corrente->prox
           valor = corrente->dados
           libera(corrente)
           corrente = aux->prox
           status=verdadeiro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

      }
senão
        status=falso
}
```

Listas auto-organizadas

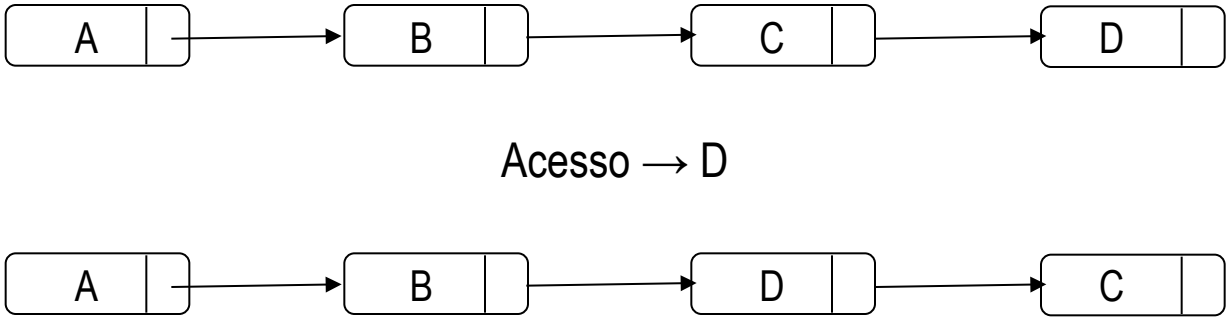
- As listas encadeadas exigem a busca seqüencial para localizar um elemento ou descobrir que ele não está na lista.
- Pode-se melhorar a eficiência da busca organizando a lista dinamicamente.
- Existem diferentes métodos de organização de listas.
  - **Método de mover para frente**: O elemento localizado deve ser colocado no início da lista.
  - **Método da transposição**: O elemento localizado deve ser trocado com seu predecessor, exceto se ele estiver no topo da lista.
  - **Método da contagem**: A lista deve ser ordenada pelo número de vezes que os elementos são acessados.
  - **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.

Listas auto-organizadas

1. Método de mover para frente

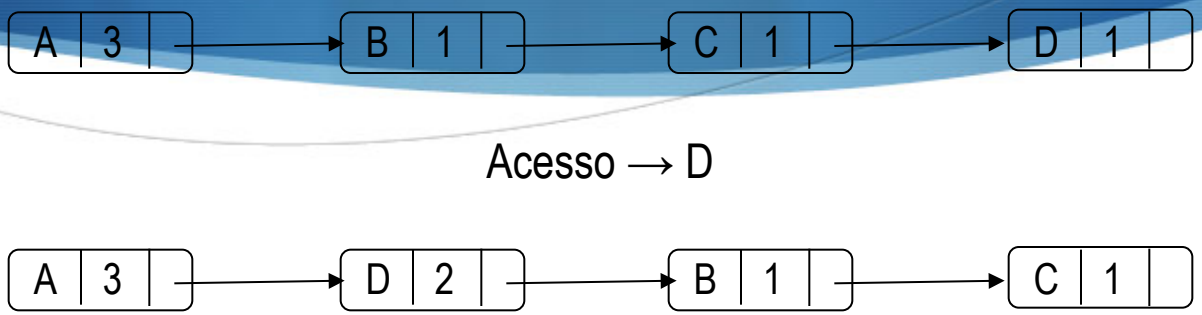


2. Método da transposição

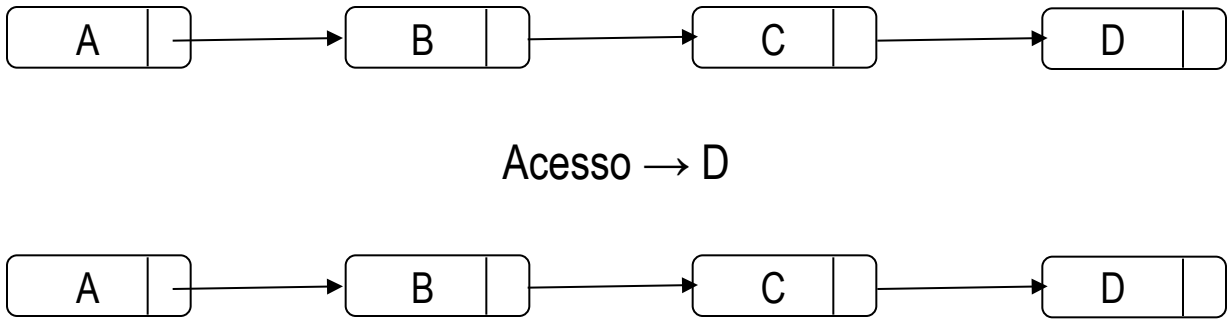


Listas auto-organizadas

3. Método da contagem



4. Método da ordenação



Exercício

- **Faça um programa que crie uma** Lista encadeada com as seguintes funções.
  - **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.