Estrutura de Dados – 2º semestre de 2019

Professor Mestre Fabio Pereira da Silva

Alocação de memória

- Reservar na memória (principal), o espaço para guardar a informação através da declaração de uma variável.
- Estática: É a alocação do espaço de memória antes da execução de um programa em tempo de compilação:
  - int x; flot vet[10]; Produto vProd[500];
- Dinâmica: É a alocação do espaço de memória durante a execução do programa.
  - em tempo de execução.

Ponteiro

- Um ponteiro é uma variável que **aponta** para outra variável. Isto significa que um **ponteiro mantém o endereço de memória de outra variável**.
- Em outras palavras, **o ponteiro não contém um valor no sentido tradicional, mas sim o endereço de outra variável.** Um ponteiro "aponta para" esta outra variável mantendo uma cópia de seu endereço.
- Como um ponteiro contém um endereço, e não um valor, terá duas partes. O ponteiro contém um endereço e o endereço aponta para um valor.

Lista Encadeada

- É uma estrutura de dados linear e dinâmica.
- Linear, pois existe uma relação de ordem entre os elementos.
- Dinâmica porque é composta por elementos, chamados de nós ou nodos, cujo o espaço de memória é alocado em tempo de execução, conforme for necessário.
- Desta forma, ao invés dos elementos estarem em sequencia (em uma área contínua da memória – consecutiva), como na lista sequencial, os elementos podem ocupar quaisquer célula de memória.
- Para manter a relação de ordem entre os elementos, cada elemento indica qual é o seguinte (ou e o anterior também).

Lista Encadeada

- Uma lista linear ligada (ou simplesmente lista ligada) é uma lista linear na qual a ordem (lógica) dos elementos da lista (chamados “nós”) não necessariamente coincide com sua posição física (em memória).
- Pode ser implementada de forma estática (usando-se um vetor) ou, em linguagens de programação que oferecem suporte à alocação dinâmica, com uso de ponteiros.

Lista Encadeada

- Espaços de memória podem ser alocados no decorrer da execução do programa, quando forem efetivamente necessários.
- É possível alocar espaço para um elemento de cada vez.
- Espaços de memória também podem ser liberados no decorrer a execução do programa, quando não forem mais necessários.
- Também é possível liberar espaço de um elemento de cada vez.

Lista Encadeada

- Podem crescer e diminuir dinamicamente.
- Tamanho máximo não precisa ser definido previamente.
- Provêm flexibilidade, permitindo que os itens sejam reposicionados de maneira eficiente.
  - Perda no tempo de acesso a qualquer item arbitrário da lista, comparando com vetores.

Lista Encadeada

- Também pode ser considerada uma coleção linear de objetos auto-referenciados, chamados de **Nós** que são conectados por links de referência.
- Em geral um programa acessa uma lista, por meio de uma referência ao primeiro elemento da lista.
- O programa acessa cada link subsequente via a referência armazenada o **Nó** anterior.
- A referência do último Nó é marcada como **null** para indicar o final da lista.
- Um Nó pode conter dados de **qualquer tipo**, bem como referências a objetos de outras classes.

Vantagens

- Listas encadeadas **são dinâmicas**, seu tamanho pode aumentar ou diminuir **conforme necessário**.
- Listas encadeadas podem ser mantidas em ordem de classificação, para isso basta inserir o elemento no ponto adequado da lista.
- Uma alteração da lista não faz com que seja necessário mover todos os seus elementos como ocorre na lista estática. Apenas duas referências são modificadas.
- Em sistemas de softwares utilizados na indústria de software **é inviável** trabalhar com listas de alocação estática. Devido a necessidade continua de sempre inserir novos elementos.

Lista Estática x Lista Dinâmica

Alocação Estática	Alocação Dinâmica
Quantidade constante de elementos	Não há quantidade máxima de elementos (o limite é a memória do computador)
Aloca espaço de acordo com a quantidade de elementos	Utiliza somente o espaço de memória suficiente
Usa arrays	Utiliza ponteiros para indicar a posição de memória que o endereço inserido na lista será armazenado

Lista Estática x Lista Dinâmica

- 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 (nós).
- Em uma lista contígua é necessário mover todos os elementos da lista para uma nova lista para realizar essas operações, ou deslocar elementos, quando se deseja manter a ordem através de uma informação.
- Listas encadeadas são mais adequadas em situações onde a lista possui centenas ou milhares de nós, onde serão realizadas muitas operações de inserção ou remoção, que em uma lista contígua representaria uma perda notável no desempenho do processamento.

Complexidade assintótica

- A inserção e remoção de um elemento que esteja disposto no início da lista encadeada possuí um número de operações fixo e independente do tamanho da lista, sendo da ordem de O(1).
- As operações de busca, remoção e a inserção após um determinado elemento da lista possuí complexidade assintótica de O(N), tal como ocorre nas listas sequenciais. Entretanto, nas listas encadeadas não é necessário mover os itens para a entrada de um novo elemento da lista.
- A implementação de operações do tipo mostrar a lista é mais simples numa lista sequencial. Assim, se não serão realizadas muitas operações de inserção e remoção, a lista sequencial é uma boa opção.

Operações

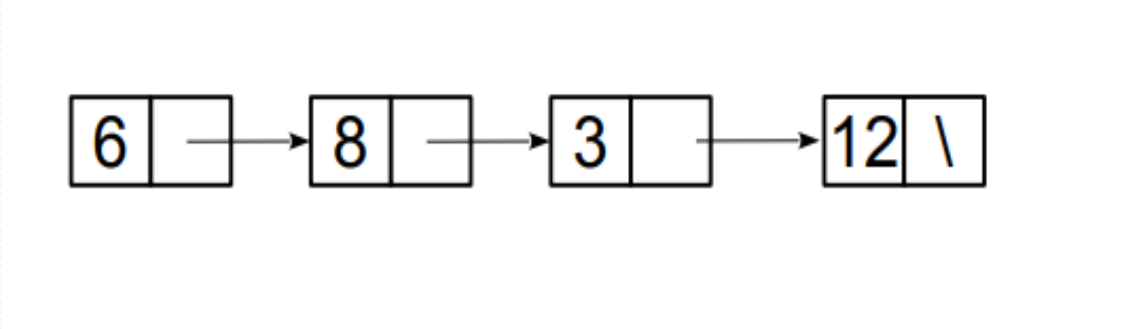
- Inicializar a lista
- Inserir um elemento no final da lista
- Inserir um elemento no início da lista
- Inserir um elemento em qualquer posição da lista
- Pesquisar um elemento
- Remover um elemento no final da lista
- Remover um elemento no início da lista
- Remover um elemento em qualquer posição da lista
- Ordenar a lista utilizando algoritmos de ordenação como Merge Sort, Quick Sort ou Heap Sort
- Concatenar a lista em outra lista ou até mesmo Pilhas ou Filas a partir de uma dada condição

- Exemplo
  - Dinâmica

  - Linear

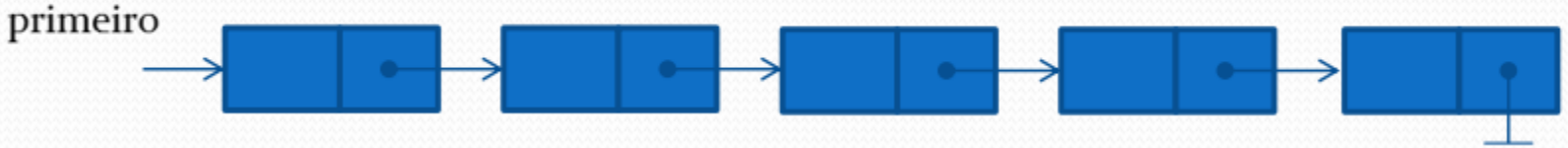
Lista Encadeada

- A lista encadeada ou lista ligada é uma sequencia de elementos (nós), onde cada nó possui:
  - Um ou mais campos de informações
  - Um ponteiro para o próximo nó da lista



Lista Simplesmente Encadeada

- Uma lista é denominada como simplesmente encadeada se em cada Nó só existe um ponteiro que aponta para o próximo Nó.
- É preciso que um ponteiro aponte para o primeiro nó, determinando assim, o início da sequência de dados armazenados na memória.
- Este tipo de lista pode ser vazia ou não, circular ou não, assim como seus dados podem estar ou não em ordem (crescente ou decrescente).

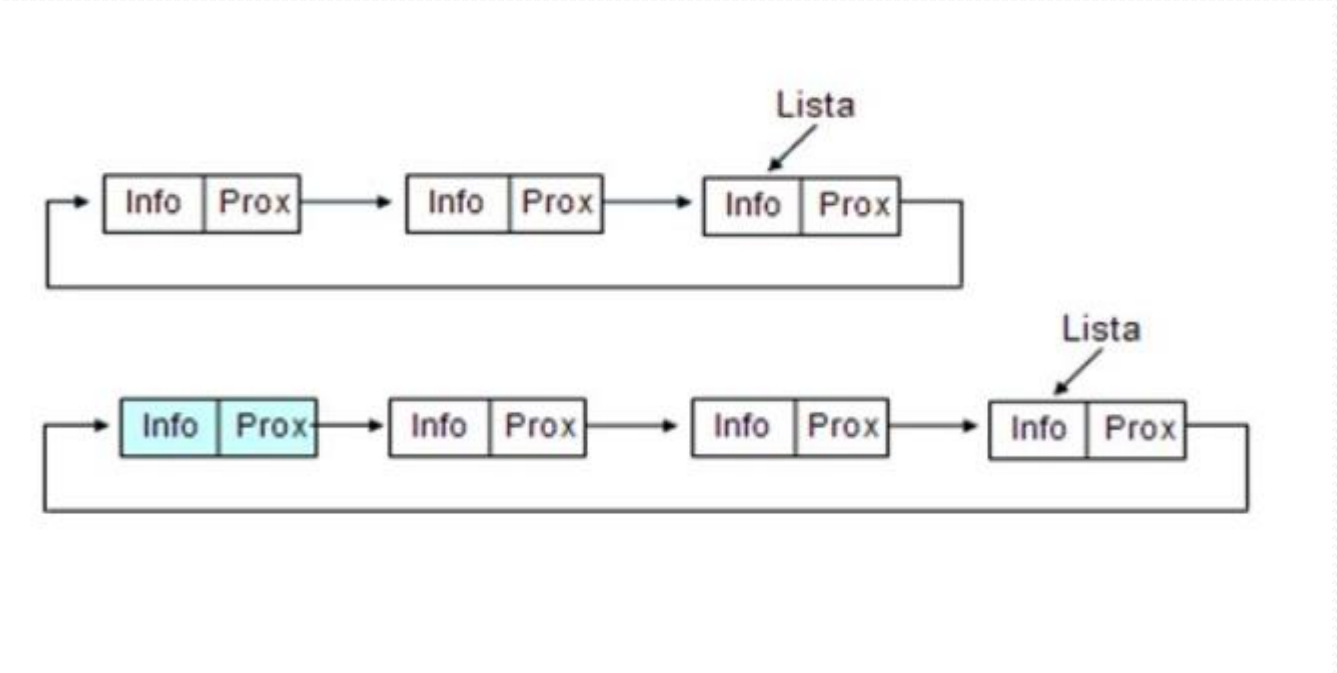


Lista Duplamente Encadeada

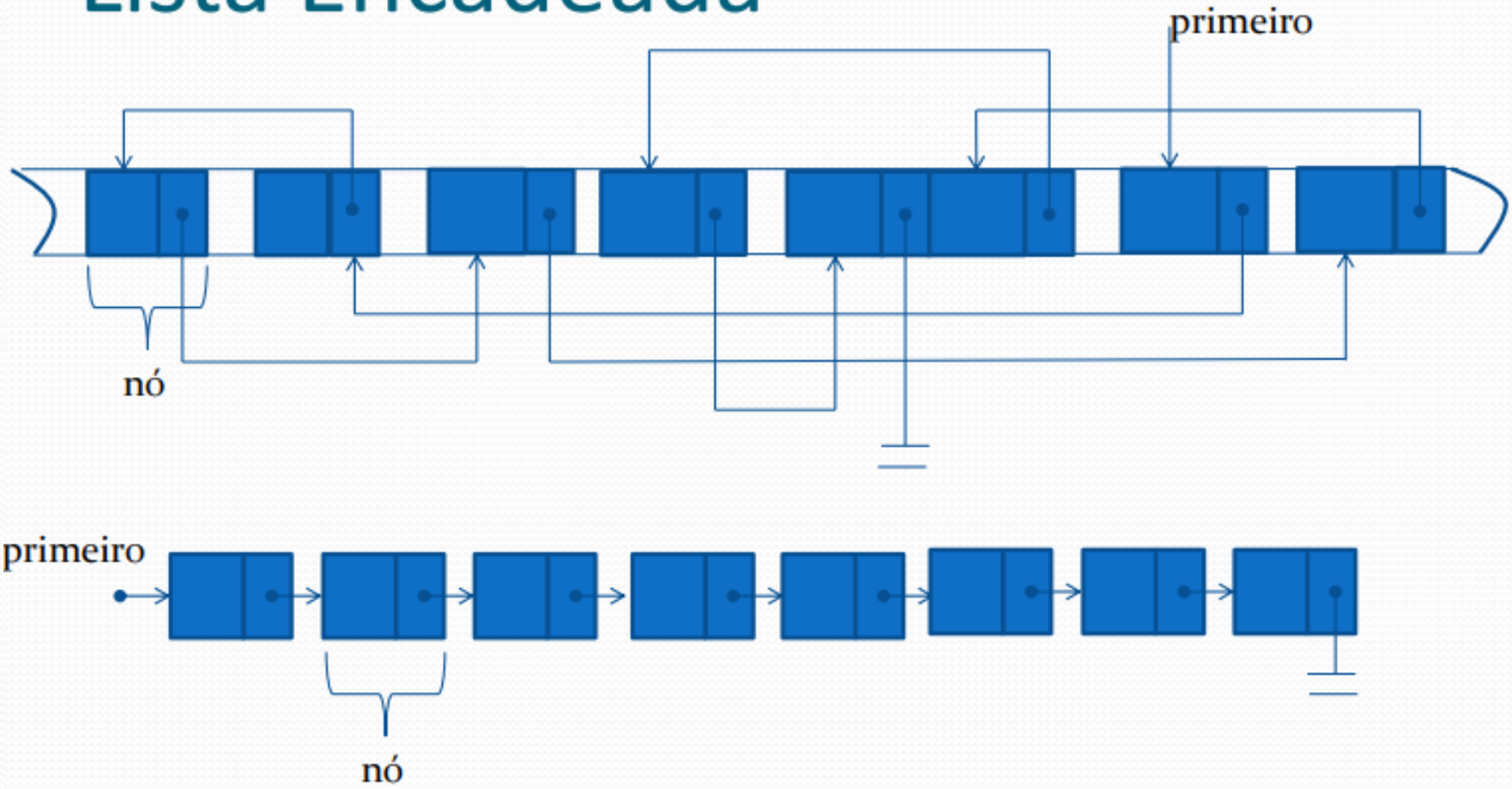
- Quando cada nó referência tanto o próximo nó da lista quanto o nó anterior.
- O importante é que, neste tipo de lista, o ponteiro externo pode apontar para qualquer nó da lista, pois é possível caminhar para a direita ou para a esquerda com igual facilidade.
- Uma lista duplamente encadeada pode ser circular ou não e ainda, pode estar em ordem (crescente/decrescente) ou não.

Lista Encadeada Circular

- Em uma lista circular o último elemento aponta para o primeiro.
- Não se guarda o endereço do primeiro e do último Nó da lista, guarda-se o endereço de apenas um deles.
- Neste tipo de lista é possível acessar qualquer Nó a partir de qualquer ponto, porque nela podemos considerar qualquer Nó como sendo o primeiro Nó da lista.

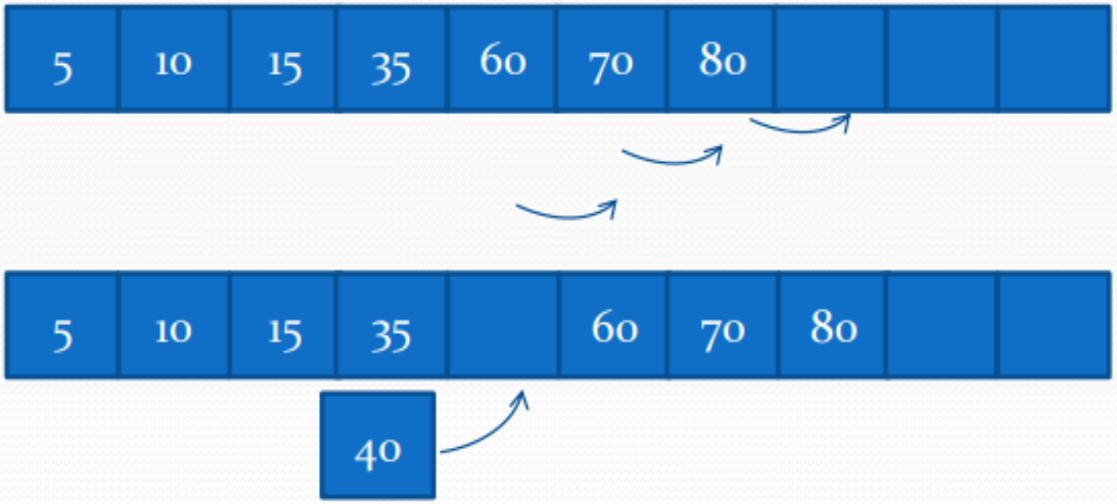


Lista Encadeada



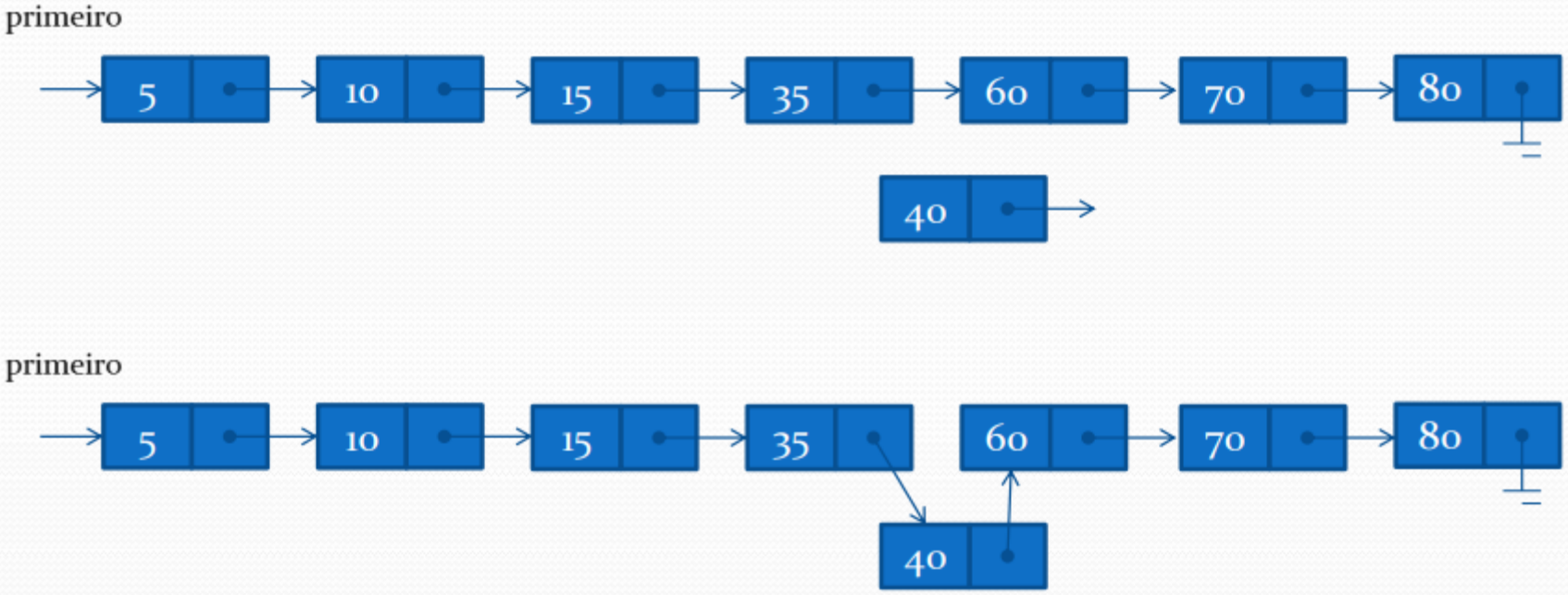
Inserção – Lista Sequencial

- Inserir o 40 – lista sequencial

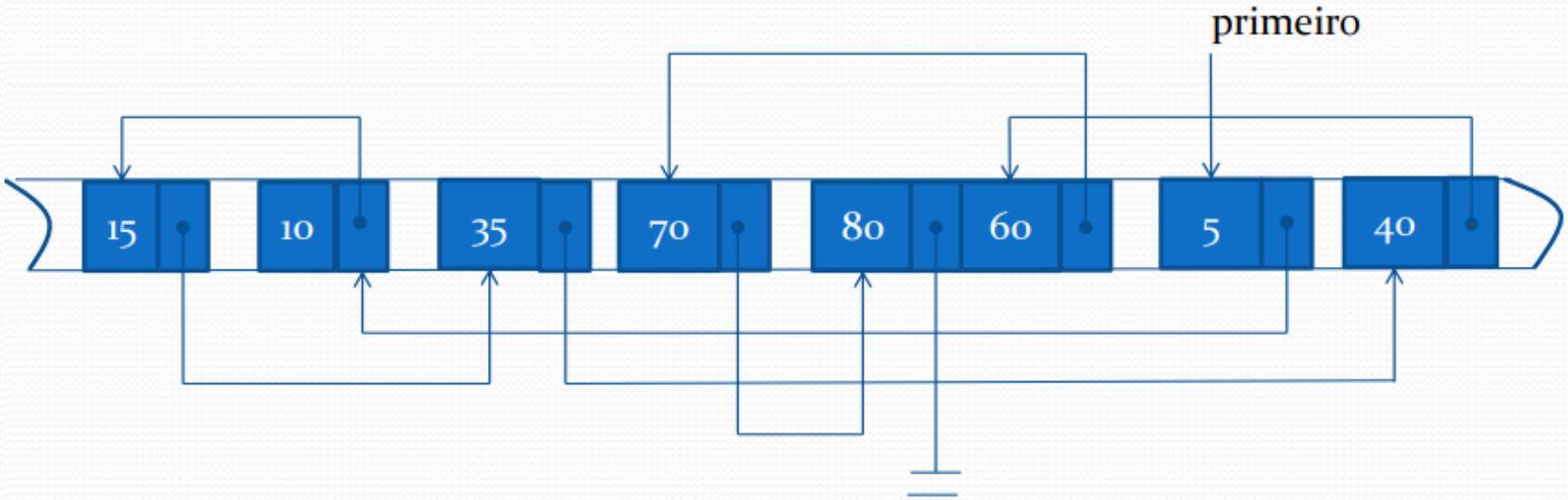


Inserção – Lista Encadeada

- Inserir o 40



Mostrar a Lista

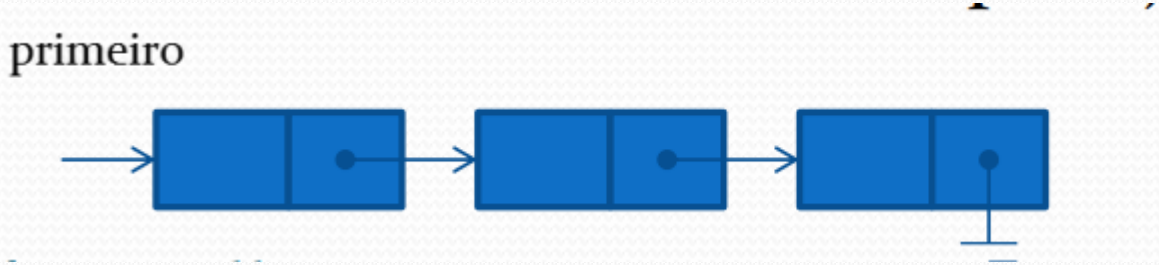


Lista Encadeada x Lista Duplamente Encadeada

- 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.
- Se não existe a necessidade de se percorrer a lista de trás para frente, a lista simplesmente encadeada é a mais interessante, pois é mais simples.

Representação do Nó

- Um nó da lista deverá conter, no mínimo, dois campos: um campo com a informação ou dado a ser armazenado e um segundo campo, com o ponteiro para o próximo nó da lista, permitindo o encadeamento dos nós.
- É preciso que o primeiro nó seja apontado por um ponteiro, para que assim, a lista possa ser manipulada através de suas diversas operações.



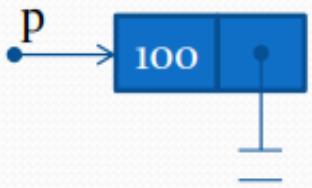
Exemplo

- Construir uma lista com um nó apenas com o valor 100:

p = new no;

p -> dado = 100;

p -> prox = NULL;

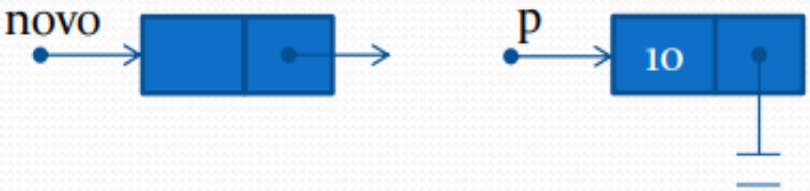


← Lista com um nó apenas

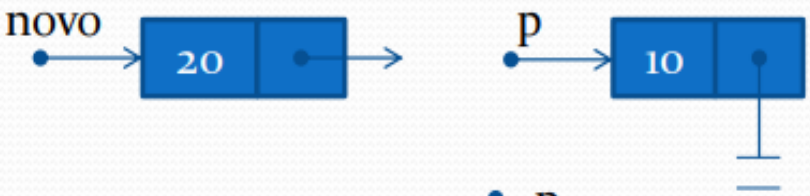
Exemplo

- Inserindo mais um nó na lista (antes do primeiro nó – inserir na frente)

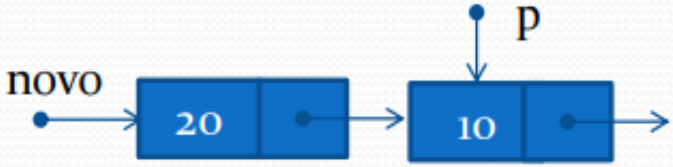
novo = new no;



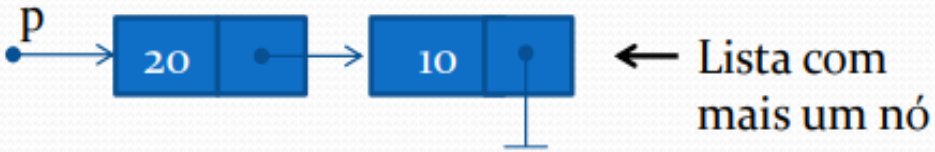
novo-> dado = 20;



novo-> prox = p;

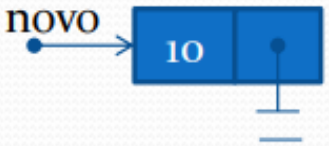


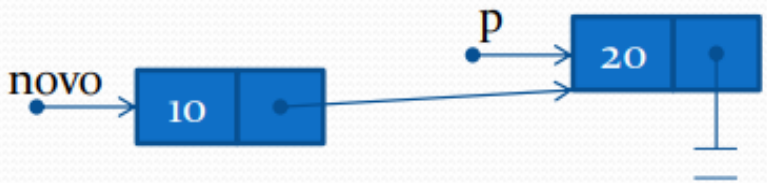
p = novo;



Código da Inserção

```
// cria um novo no
no *novo = new no;

cout << "Valor? ";
cin >> valor;

novo->dado = valor;
novo->prox = NULL;

// inserindo no inicio da lista
if (p != NULL) {
    novo->prox = p;
}
p = novo;
```

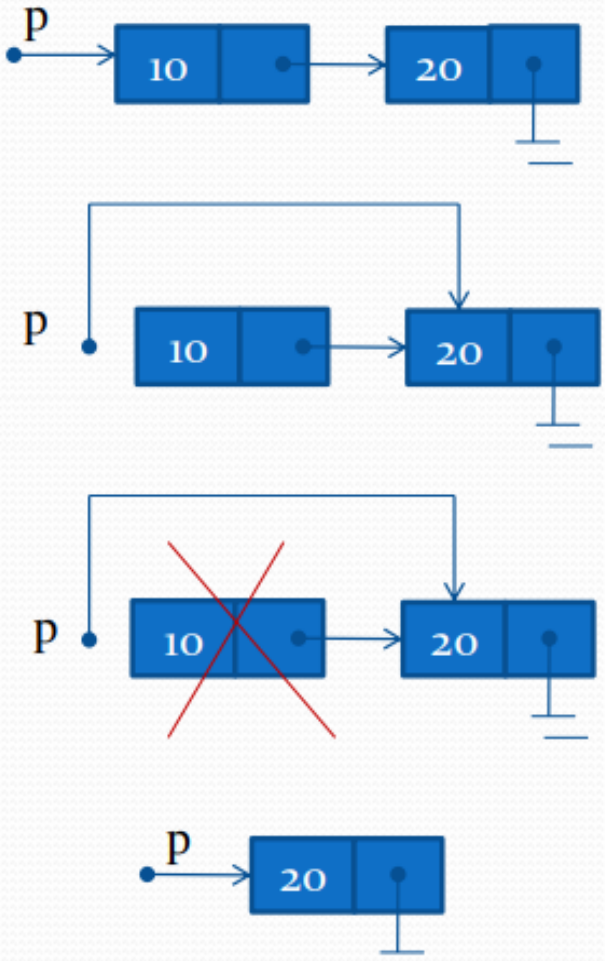




Remoção

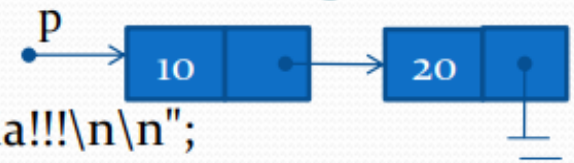
- Retirada de um Nó da lista
- Antes deve se verificar se a lista está vazia
- Para isso, verifique se o ponteiro para o primeiro elemento da lista está apontando para algum Nó

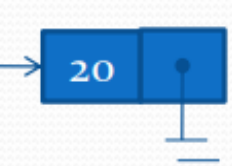
Removendo o primeiro da Lista

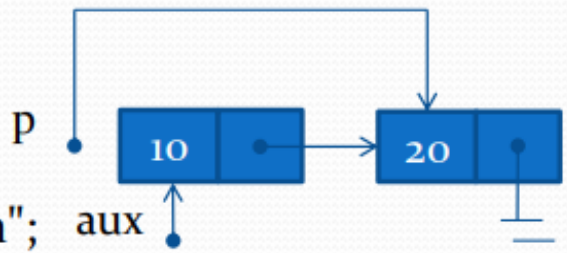


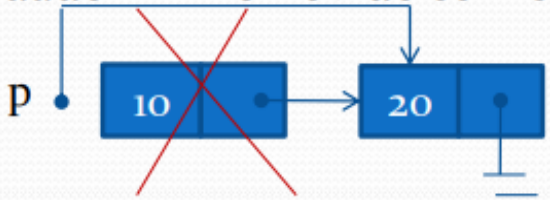
Código da Remoção de um nó

```
if (p == NULL) {
    cout << "\nLista vazia!!!\n\n";
}
else {
    no *aux;

    aux = p;

    p = p->prox;
    cout << aux->dado << " removido com sucesso\n";
    delete aux;
}
```



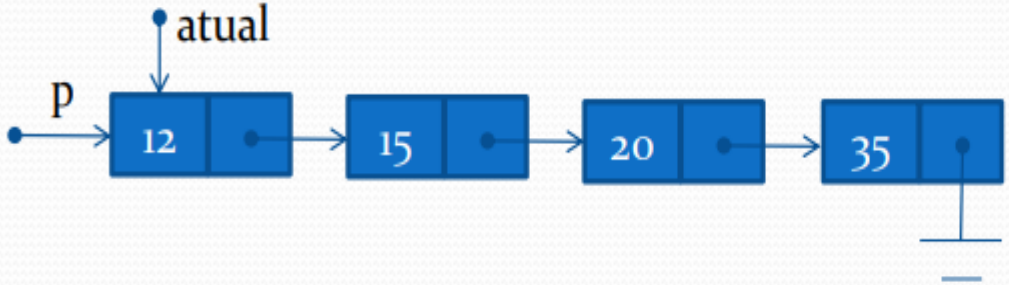






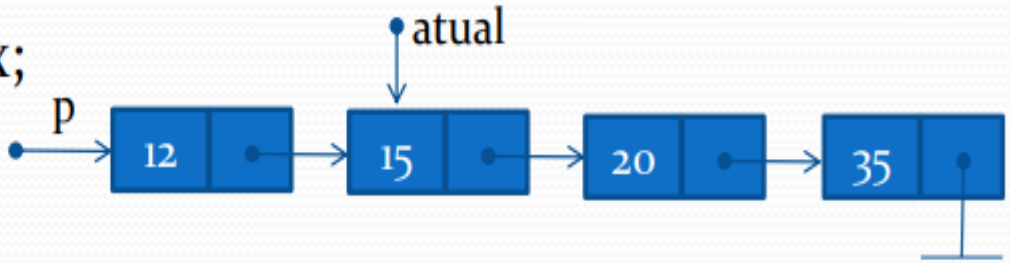
Percorrer a lista

- Verificar se a lista não está vazia
- Usando um ponteiro auxiliar, percorrer a lista a partir do primeiro elemento



- Move-se o ponteiro auxiliar pela lista

atual= atual-> prox;



Código para Percorrer

```
if (p == NULL) {
    cout << "\nLista vazia!!!\n\n";
}
else {
    no *atual;
    atual = p;
    cout << "\nLista => ";
    while (atual != NULL) {
            cout << atual->dado << "\t";
            atual = atual->prox;
    }
    cout << endl;
}
```

Contatos

- Email: fabio.silva321@fatec.sp.gov.br
- Linkedin: https://br.linkedin.com/in/b41a5269