Estrutura de Dados

Aula 10
Alocação Dinâmica de Memória
Listas Encadeadas

prof Leticia Winkler

Alocação Dinâmica de Memória

Alocação de Memória

- Reservar na memória (principal), o espaço para guardar a informação através da declaração de uma variável
- Tipos:
  - Estática
    - É a alocação do espaço de memória antes da execução de um programa – em tempo de compilação:

      int x;

      flot vet[10];

      struct Produto vProd[500];
  - Dinâmica
    - É a alocação do espaço de memória durante a execução do programa – em tempo de execução.

Alocação Dinâmica de Memória

- Em C++, a alocação dinâmica é feita através do operador **new**, usando um ponteiro:

  nomePonteiro = new tipoDado;

- Após a operação acima, o ponteiro aponta (referencia) a área de memória alocada.
- Exemplo:

```
int *pInt, *pVet;
pInt = new int;  // aloca um espaço para armazenar 1 inteiro
pVet = new int[3]; /* aloca um espaço para armazenar um vetor
                      de inteiros de tamanho 3 */
```

Testando a Alocação

- É uma boa prática testar se a alocação ocorreu com sucesso, pois podem ocorrer erros a alocação:
  - falta de espaço, permissão, entre outros.
- Em caso de erro, o ponteiro irá conter o o valor null – não aponta para posição alguma.

  if (nomePonteiro == NULL) {...}

  ou

  if (! nomePonteiro) {...}
- Exemplos:

  if (pInt == NULL){ cout << "Erro!";} **ou** **if (!pInt){ cout << "Erro!";}**

  if (pVet == NULL) {...} **ou** **if (!pVet) {...}**
- A partir daí utiliza-se os ponteiros para manipular as variáveis para as quais os mesmos apontam.

Liberando o Espaço Alocado

- Após o término da utilização do espaço de memória alocado, é importante lembrar de liberá-lo. Para isto utiliza-se o operador *delete:*
  - delete nomePonteiro; e
  - delete [] nomePonteiro; // no caso de um vetor
- Exemplo:
  - delete pInt;
  - delete [] pVet;

Exemplo #1

```
#include <iostream>
using namespace std;

int main() {
    int *pInt;
    pInt = new int; /* cria a área necessária para 1 inteiro e coloca em 'pInt' o endereço desta área. */
    // testando se a alocação foi realizada com sucesso
    if (pInt == NULL) {
      cout << "Memória insuficiente!\n";
      exit(1);
    }
    cout << "Endereço de pInt: " <<  pInt << endl;
    *pInt = 90;
    cout << "Conteúdo de pInt: " << *pInt << endl;  // imprime 90
    delete pInt;  // Libera a área alocada
    return 0;
}
```

Exemplo #2

```
#include <iostream>
using namespace std;

int main() {
int *pVet, *pIni;
int qElem;

cout << "Quant. de elem. do vetor?
cin >> qElem;
/* cria a área necessária para um vetor com
qElem elementos */
pVet = new int [qElem];

if (!pVet) {
   cout << "Memória insuficiente!\n";
   exit(1);
}
pVet = pIni;
```

```
cout << "Digite os elementos do vetor:\n";
for (int i=0; i<qElem; i++) {
     cout << "Elemento " << i << "? ";
     cin >> *pVet;
     pVet++;
}
pVet = pIni;

cout << "Conteudo do vetor:\n";
for (int i=0; i<qElem; i++) {
     cout << *pVet << endl;
     pVet++;
}
pVet = pIni;
delete [] pVet;
cout << endl;
return 0;
}
```

Exemplo #3

```
#include <iostream>

using namespace std;

struct Candidato{
        int mat;
        char nome[30];
        int pontos;
};

void lerCandidato(struct Candidato *pc){
        cout << "Matricula? ";
        cin >> pc->mat;
        cout << "Nome.....? ";
        cin.get(); cin.get(pc->nome,30);
        cout << "Pontos...? ";
        cin >> pc->pontos;
}
```

```
void mostrarCandidato(struct Candidato *p){
    cout << "Matricula: " << p->mat << endl;
    cout << "Nome.....? " << p->nome << endl;
    cout<< "Pontos...? "<< p->pontos<<"\n\n";
}

int main(){
    struct Candidato *pCand, *pIni, *pMelhor;
    int qCand;

    cout << "Quantidade de candidatos? ";
    cin >> qCand;
    // Alocando o espaco p/ os candidatos
    pCand = new Candidato[qCand];

    pIni = pMelhor = pCand;
```

(continua)

Exemplo #3 (cont)

```
    for (int i=0; i<qCand; i++){
        lerCandidato(pCand);
        if (pCand->pontos > pMelhor->pontos){
          pMelhor = pCand;
        }
        pCand++;
    }
    pCand = pIni;
    cout << "Listagem dos Candidatos";
    for (int i=0; i<qCand; i++){
        mostrarCandidato(pCand);
        pCand++;
    }
    pCand = pIni;
    cout << "Candidato aprovado:\n";
    mostrarCand(pMelhor);
    delete [] pCand;
    return 0;
}
```

Exercícios

- O que significa o operador asterisco em cada um dos seguintes casos:
  - int *p;
  - cout << *p;
  - *p = x*5;
  - cout << *(p+1);
- Explique a diferença entre:
  - p++;
  - (*p)++;
  - *(p++);

Exercícios

- Qual é a saída deste programa?

```
#include <iostream.h>
using namespace std;
int  main() {
    int i=5, *p;
    p = &i;
    cout << p << '\t' << (*p+2) << ' \t' << **&p<< '\t' << (3**p) << ' \t' <<
    (**&p+4);
}
```

Exercícios

- Qual o erro do trecho de programa a seguir:

```
{ ...
float x = 333.33;
int *p = &x;
cout << *p;
...
}
```

Exercícios

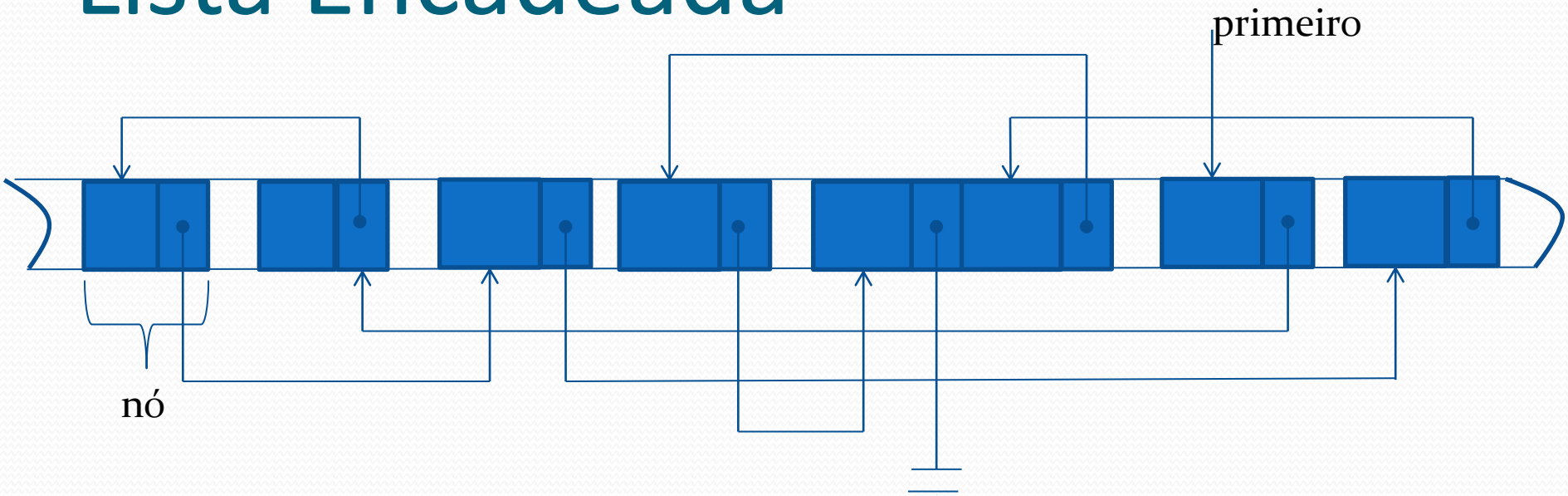
- Faça um programa que leia os dados dos veículos de uma revendedora de carros usados (placa, marca e modelo, ano e preço – defina uma struct que descreve o veículo). A quantidade de veículos deve ser informada em tempo de execução (aloque dinamicamente o espaço para um vetor de veículos) e, após lidos todos os dados, apresente um menu com as seguintes opções:
  - Listar os dados de todos os veículos;
  - Realizar uma consulta pela placa – dada a placa, apresente os dados do veículo solicitado;
  - Ordenar por preço e mostrar os dados ordenados (crescente)

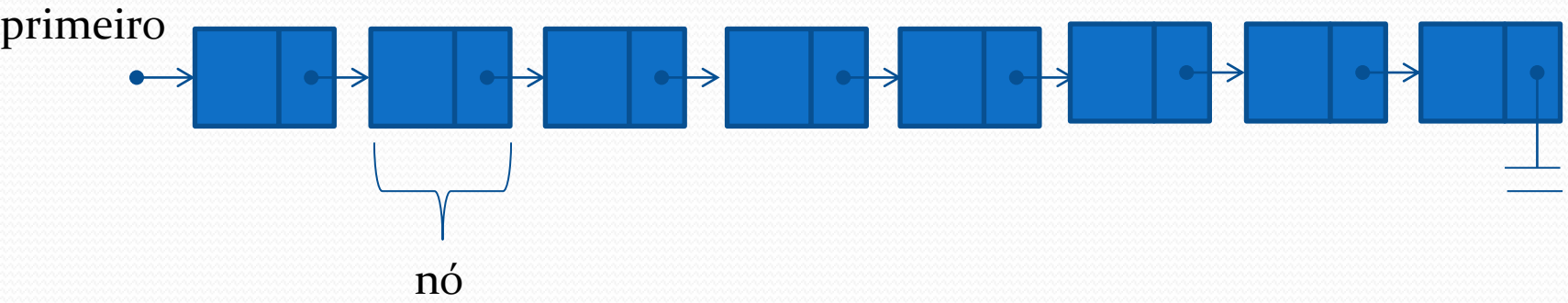
Listas Encadeadas

Definição

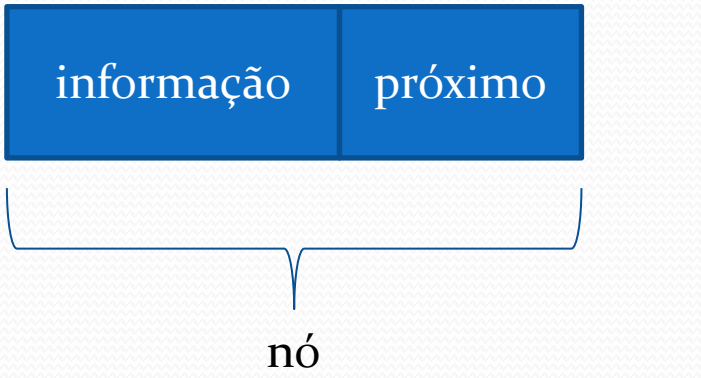
- É uma estrutura de dados linear e dinâmica.
- Linear, pois existe uma relação de ordem entre os elementos;
- Dinâmica porque é composta por elementos, chamados de nós ou nodos, cujo o espaço de memória é alocado em tempo de execução, conforme for necessário.
- Desta forma, ao invés dos elementos estarem em sequencia (numa 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



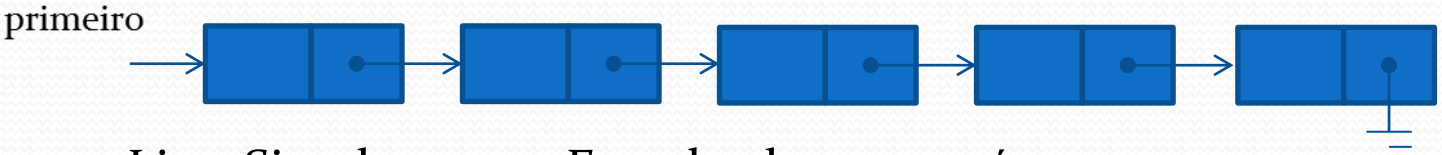


Nó da Lista



Tipos de Listas Encadeadas

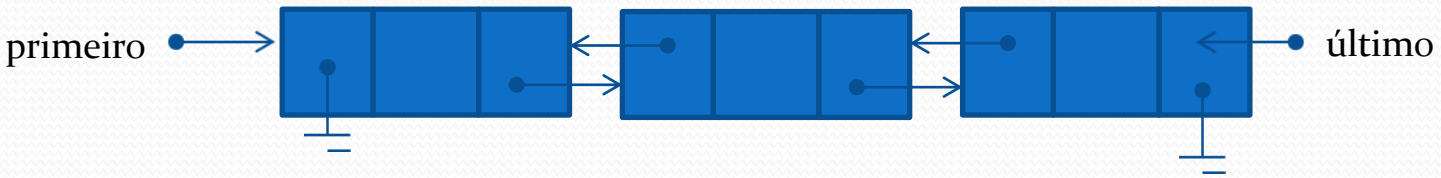
- Lista Linear Simplesmente Encadeada
  - Quando cada nó referência para o próximo nó da lista



  - Lista Simplesmente Encadeada com 5 nós
  - 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).

Tipos de Listas Encadeadas

- Lista Linear Duplamente Encadeada
  - Quando cada nó referência tanto o próximo nó da lista quanto o nó anterior



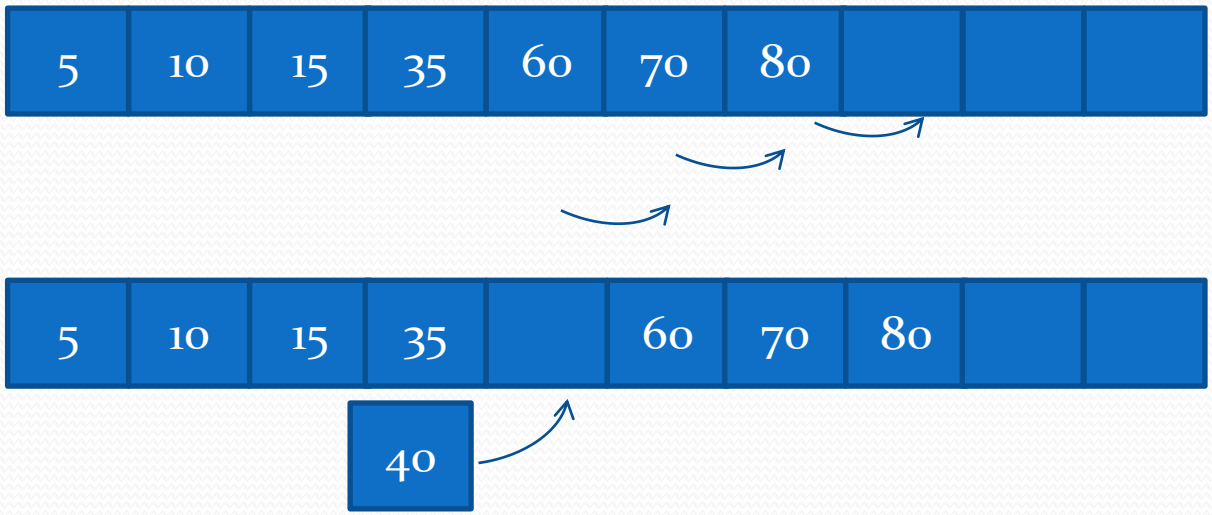
    - Lista Duplamente Encadeada com 3 nós
  - O importante é que, neste tipo de lista, o ponteiro externo pode apontar para qualquer nó da lista, pois é possível caminhar para a direita ou para a esquerda com igual facilidade.
  - Uma lista duplamente encadeada pode ser circular ou não e ainda, pode estar em ordem (crescente/decrescente) ou não.

Lista Encadeada vs Lista Sequencial

- 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
  - 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 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.
- 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.

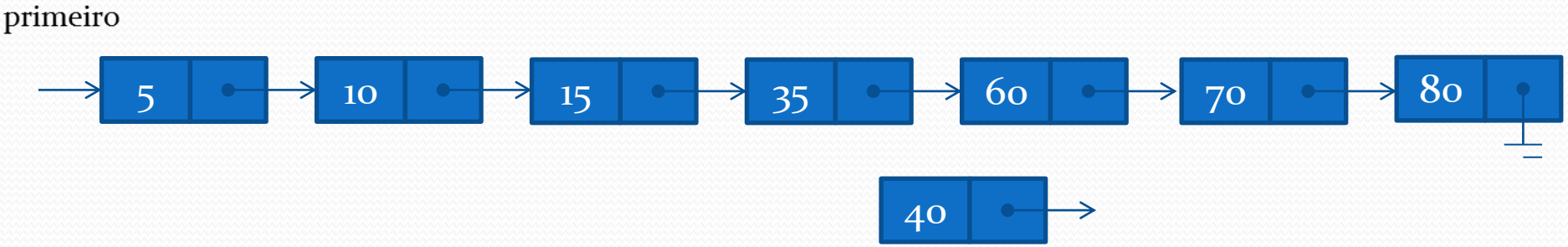
Inserção – Lista Sequencial

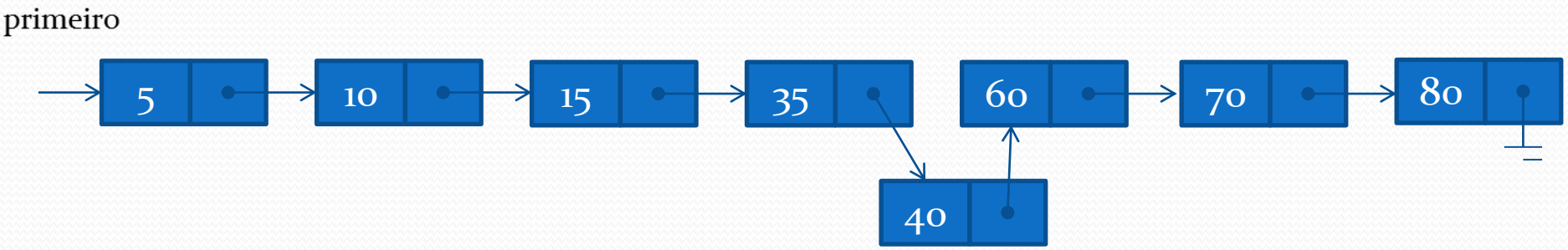
- Inserir o 40 – lista sequencial



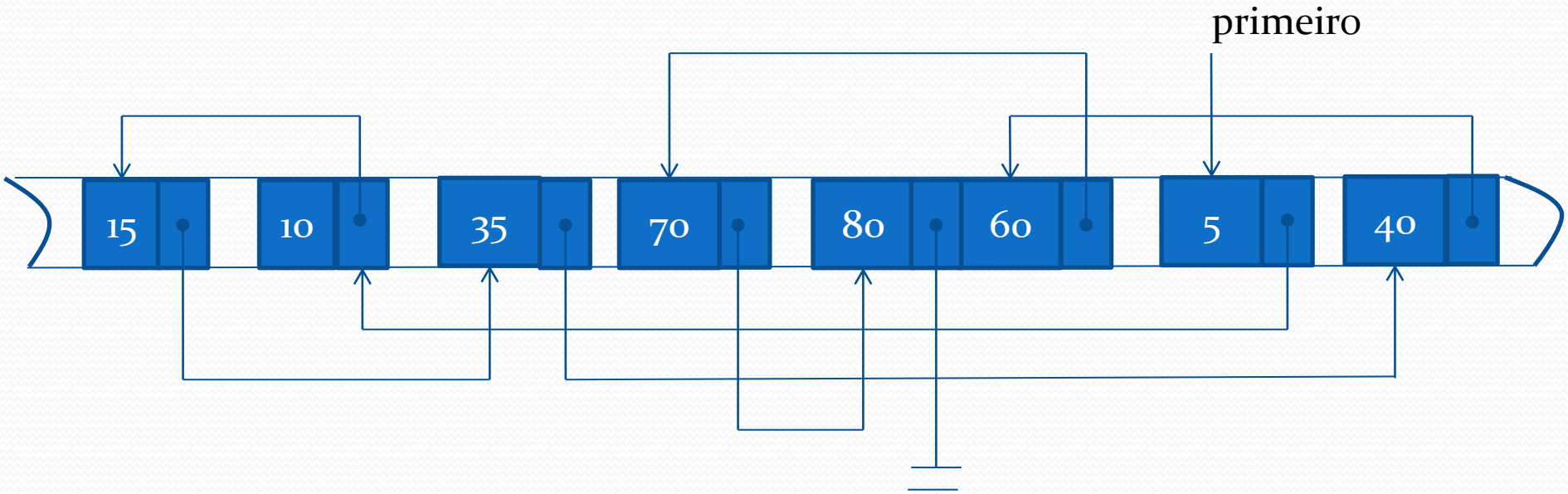
Inserção – Lista Encadeada

- Inserir o 40





Mostrar a Lista



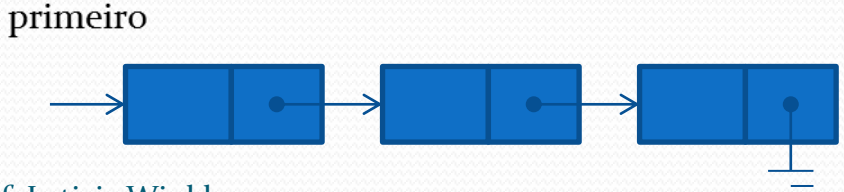
Lista Duplamente Encadeada vs. Lista Simplesmente Encadeada

- 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

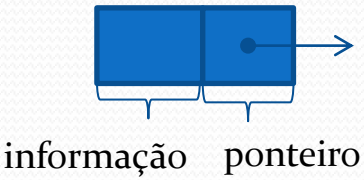
- Um nó da lista é representado por uma estrutura (struct), que deverá conter, no mínimo, dois campos : um campo com a informação ou dado a ser armazenado e um segundo campo, com o ponteiro para o próximo nó da lista, permitindo o encadeamento dos nós.

- É preciso que o primeiro nó seja apontado por um ponteiro, para que assim, a lista possa ser manipulada através de suas diversas operações.



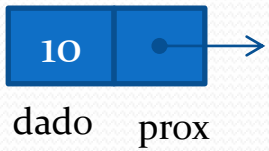
Representação do Nó

```
struct no {
    tipo info1;
    tipo info2;
    ...
    (struct) no *nomePonteiro;
};
```



- Exemplo:

```
struct no {
     int dado;
     struct no *prox;
 };
```



Operações com Listas Simplesmente Encadeadas

- Criação
- Inicialização
- Inserção (no início, no fim, após um determinado valor, etc...)
- Percurso (mostrar a lista)
- Substituição de um valor por outro
- Remoção (do primeiro nó, do último nó, de um nó em particular, etc..)
- Busca ou pesquisa de forma seqüencial
- Exemplos de aplicações com listas simplesmente encadeadas: todos os já mencionados para listas lineares.

Criação e Inicialização da Lista

- Declara-se o ponteiro para o primeiro nó da lista.Aqui chamado de **p**

```
struct no  *p;   // criação

p = NULL;        // inicialização
```

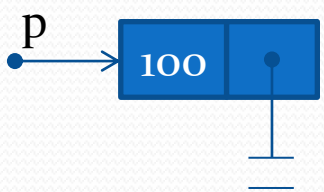
Exemplo

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

p = new no;

p -> dado = 100;

p -> prox = NULL;

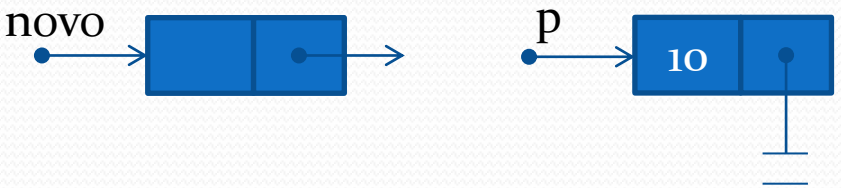


← Lista com um nó apenas

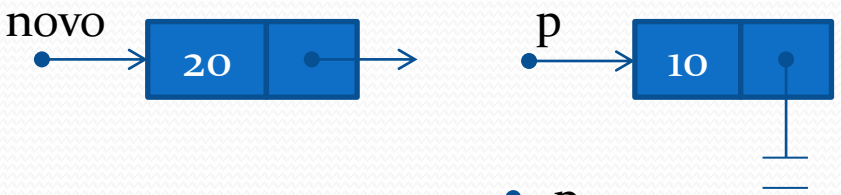
Exemplo

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

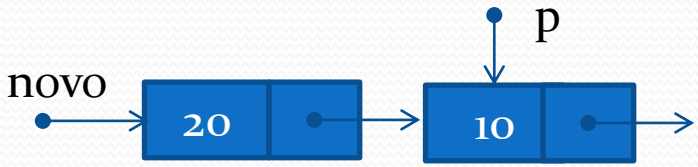
novo = new no;



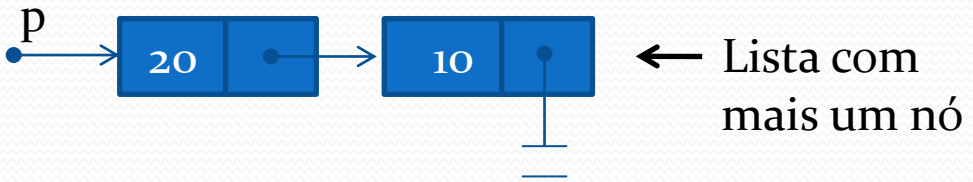
novo-> dado = 20;



novo-> prox = p;

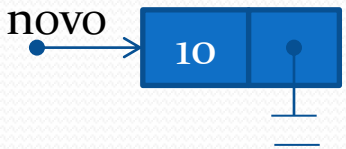


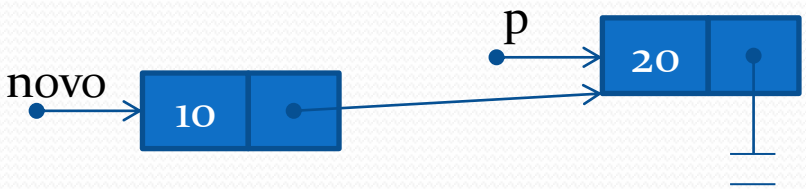
p = novo;

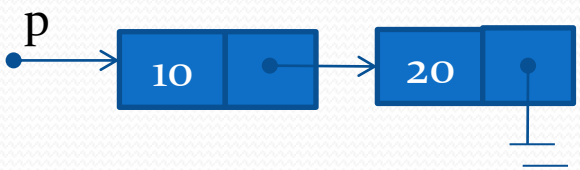


Código da Inserção

```
// 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

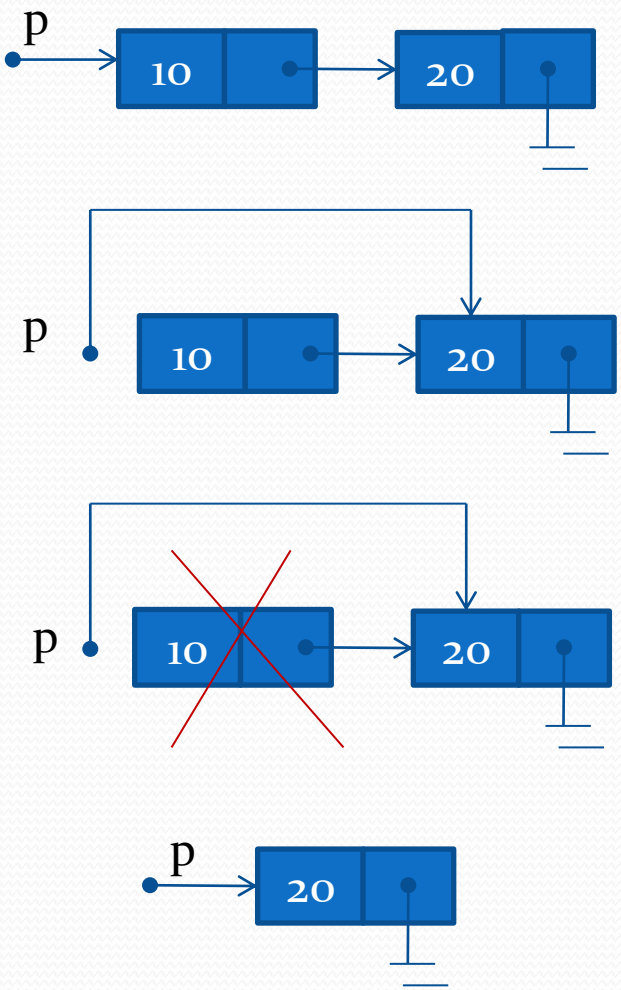
- Retirada de um nó da lista
- Antes deve-se verificar se a lista não está vazia.
- Como se sabe se uma lista encadeada está vazia?

Verificando se a Lista Encadeada está Vazia

- Verifica-se se o ponteiro para o primeiro elemento da lista está apontando para algum nó

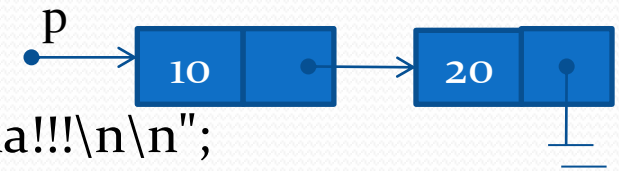
```
if (p == NULL) {
    cout << "\nLista vazia!!!\n\n";
}
```

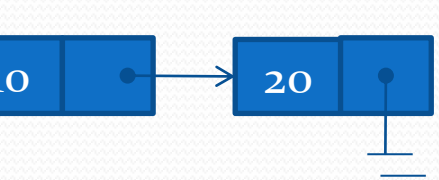
Removendo o primeiro da Lista

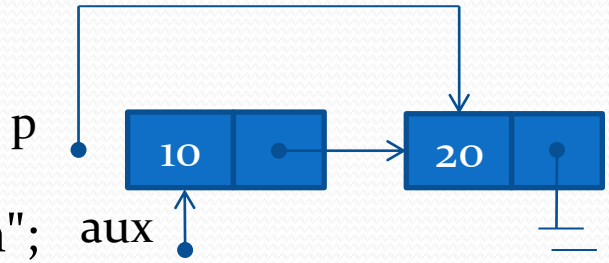


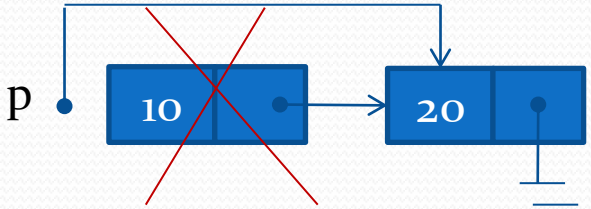
Código da Remoção de um nó

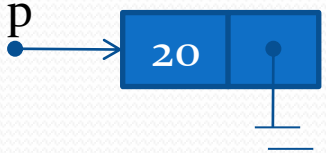
```
if (p == NULL) {
    cout << "\nLista vazia!!!\n\n";
}
else {
    no *aux;

    aux = p;

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





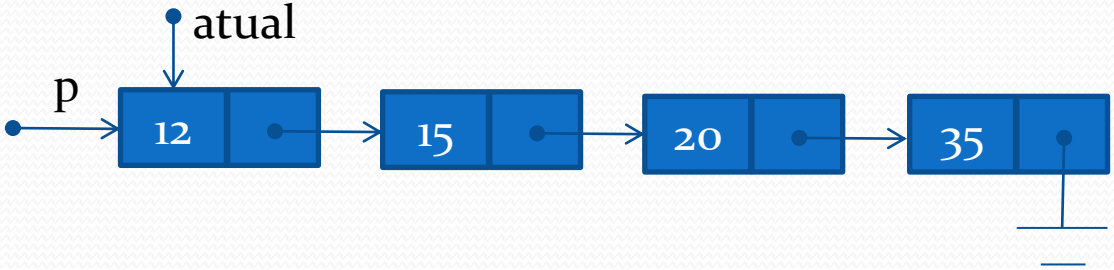






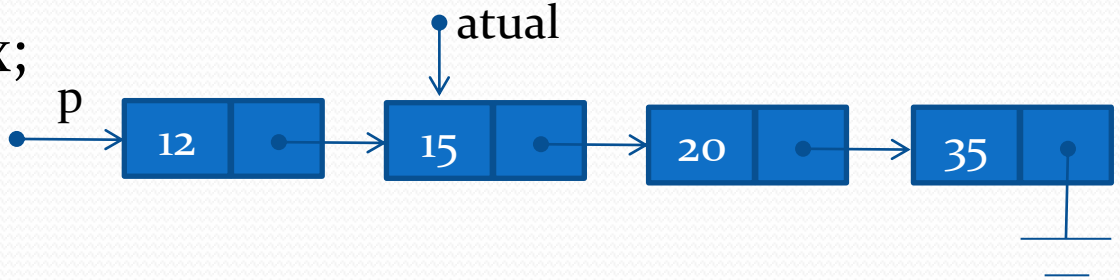
Percorrer a Lista (mostrar)

- Verifica-se se a lista não está vazia
- Usando um ponteiro auxiliar (aqui chamado de atual) percorre-se a lista a partir do primeiro



- 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;
}
```

Animação em uma Lista Simplesmente Encadeada

http://www.cosc.canterbury.ac.nz/mukundan/dsal/LinkListAppl.html

Exercício #1

- Altere o programa listasimples.cc (está no SIA), concluindo o trecho do código para inserir um elemento no final da lista e também para remover o último da lista.

Exercício #2

- A partir do exercício #1, altere o programa para que o mesmo utilize uma função para cada operação solicitada no menu.