Estrutura de Dados

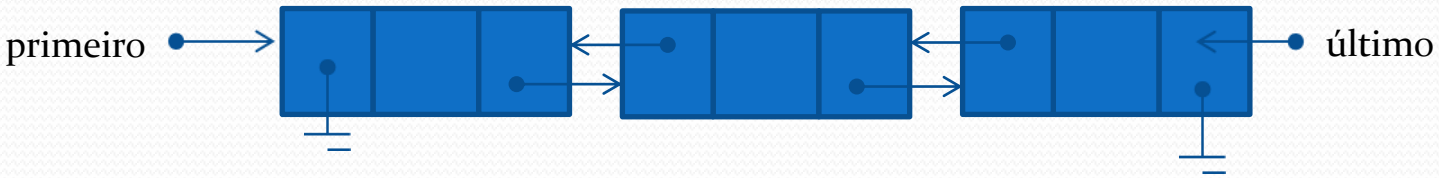
Aula 14

Listas Duplamente Encadeadas

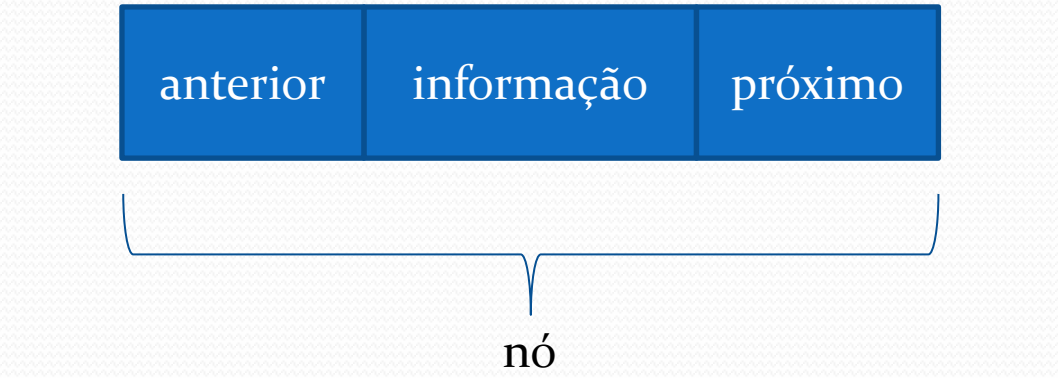
prof Leticia Winkler

Lista Duplamente Encadeada

- É um tipo de lista encadeada que pode ser vazia (NULL) ou que pode ter um ou mais nós, sendo que cada nó possui dois ponteiros: um que aponta para o nó *antecessor* e outro que aponta para o nó *sucessor*.
- 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.



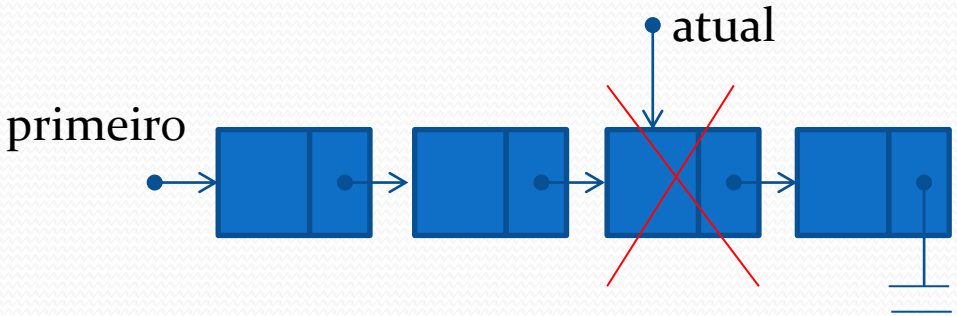
Nó da Lista

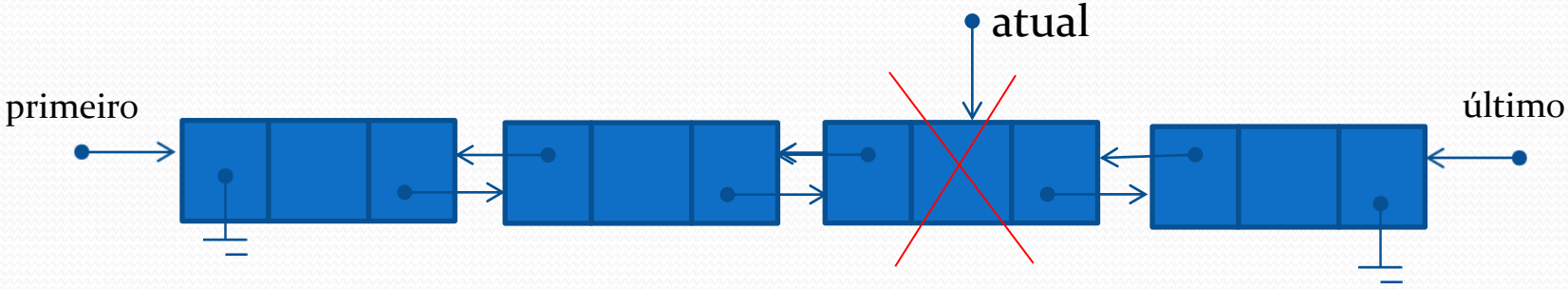


Lista Duplamente Encadeada vs. Lista Simplesmente Encadeada

- 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.

Lista Duplamente Encadeada vs. Lista Simplesmente Encadeada

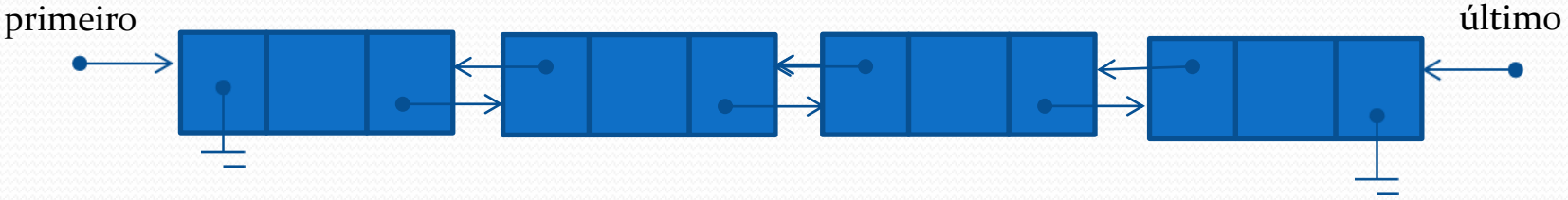




Representação do Nó

- Um nó da lista é representado por uma estrutura (struct), que deverá conter, no mínimo, três campos : um campo com a informação ou dado a ser armazenado e dois ponteiros: um para o próximo nó da lista e outro para o nó anterior.

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



Representação do Nó

```
struct noDupla {
   tipo info1;
   tipo info2;
   ...
   (struct) no *nomePonteiro1, *nomePonteiro2;
};
```

- Exemplo:

```
struct noDupla {
   int dado;
   struct noDupla *prox;  // ponteiro p/ à  direita
   struct noDupla *ant;   // ponteiro p/ à  esquerda
};
```

Operações com Listas Duplamente Encadeadas

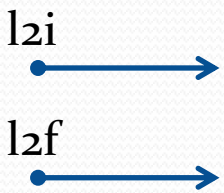
- Criação
- Inicialização
- Inserção (no início, no fim, após um determinado valor, etc...)
- Percurso (mostrar a lista – de trás para frente; de frente para trás)
- Substituição de um valor por outro
- Remoção (do primeiro nó, do último nó, de um nó em particular, etc..)
- Busca ou pesquisa de forma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

Criação e Inicialização da Lista

- Declara-se o ponteiro para o primeiro nó da lista e o ponteiro para o último nó da lista. Aqui chamado de **l2i** e de **l2f** respectivamente.

```
// criação
noDupla *l2i; // ponteiro para o primeiro elemento da lista
noDupla *l2f; // ponteiro para o último elemento da lista

l2i = l2f = NULL;      // inicialização
```

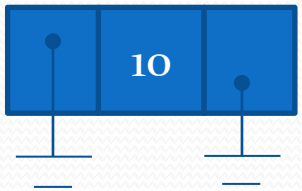


Exemplo

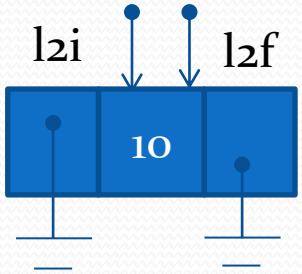
- Construir uma lista com um nó apenas com o valor 10:

novo = new noDupla;

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



l2i = novo;
l2f = novo;

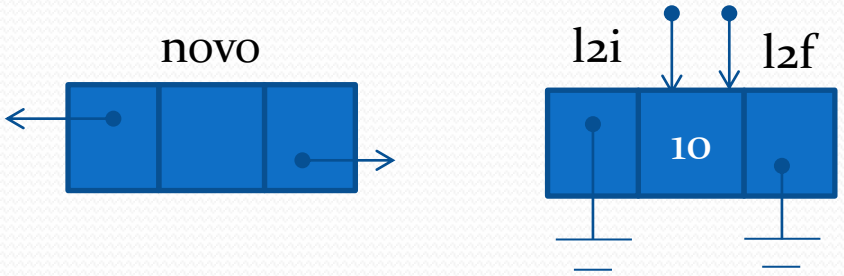


← Lista com um nó apenas

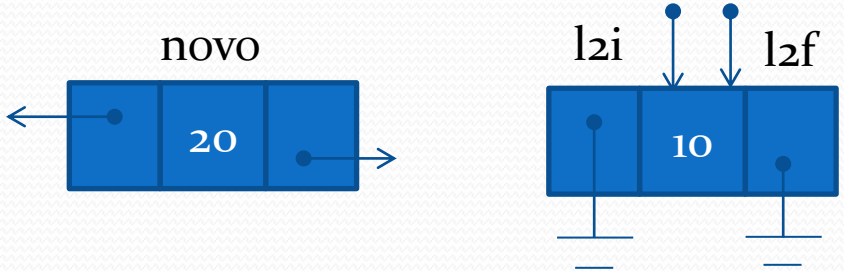
Exemplo

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

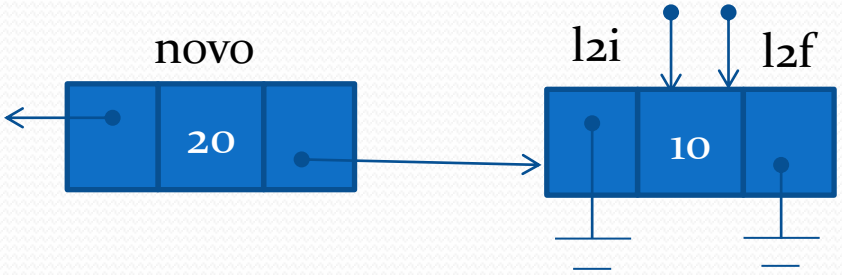
novo = new noDupla;



novo-> dado = 20;



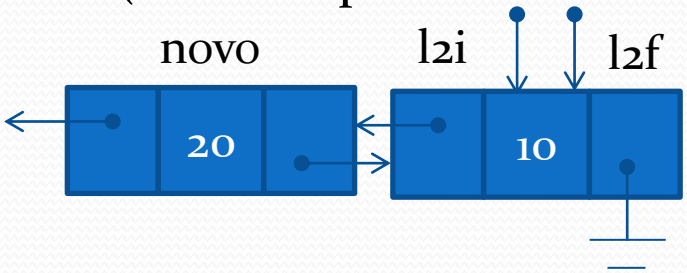
novo-> prox = l2i;



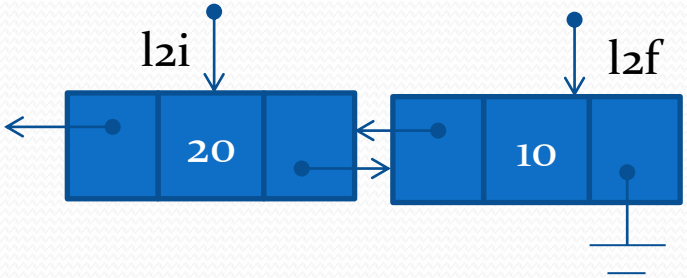
Exemplo

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

l2i->ant = novo;



l2i = novo;



← Lista com mais um nó

Código da Inserção no Início

```
void inserirInic(int valor) {
    // cria um novo no
    noDupla *novo = new noDupla;

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

    // inserindo no inicio da lista
    if (l2i != NULL) {
      novo->prox = l2i;
      l2i->ant = novo;
    }
    else
      l2f = novo;
    l2i = novo;
}
```

Remoção

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

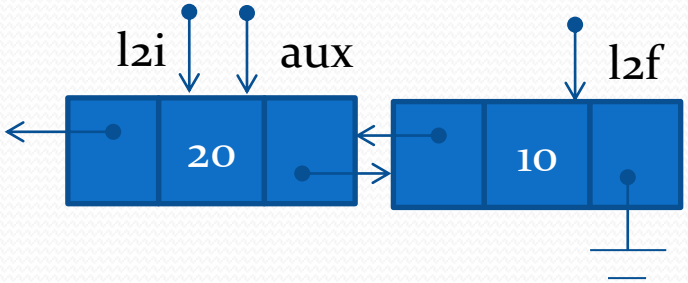
```
if (l2i == NULL) {
    cout << "\nLista vazia!!!\n\n";
}
```

- Ou, usando uma função:

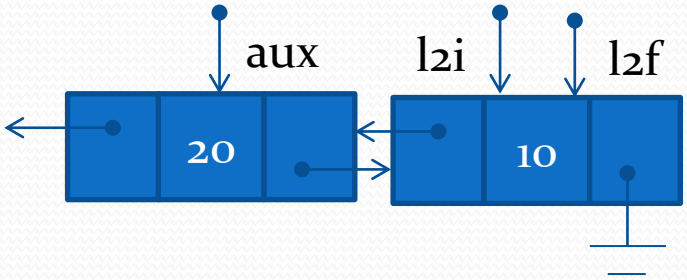
```
bool estaVazia() {
    return (l2i == NULL);
}
```

Removendo o primeiro da Lista

- Guarda nó a ser removido através de um ponteiro auxiliar

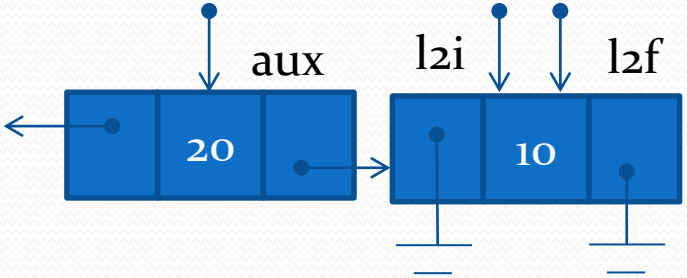


- Ponteiro para o inicio da lista (primeiro) aponta para o próximo da lista

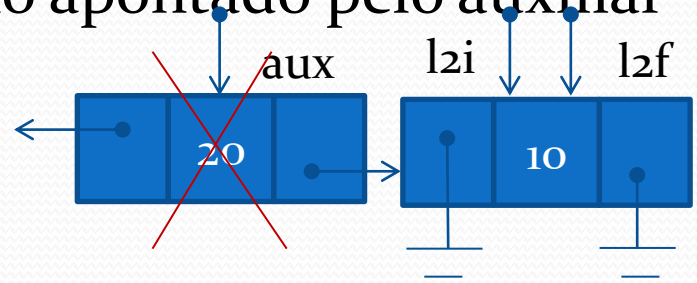


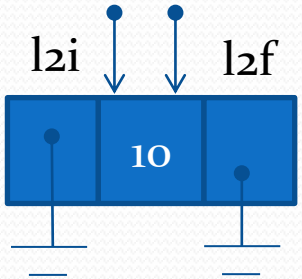
Removendo o primeiro da Lista

- Ponteiro *anterior* do *início da lista* é aterrado



- Remove nó apontado pelo auxiliar



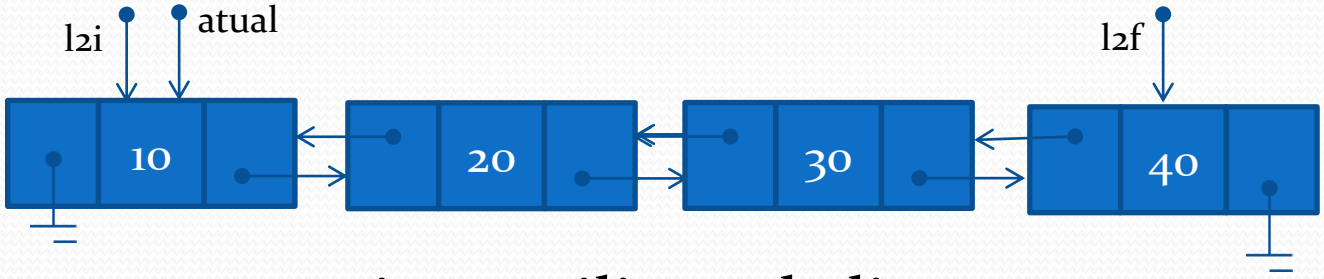


Código da Remoção de um nó do Início

```
noDupla* removerInic () {
   noDupla *aux = l2i;
   l2i = l2i->prox;
   if (l2i == NULL) // se foi removido o ultimo elemento que existia na lista
      l2f = NULL; // ultimo tb ficara apontando para nada
   else
      l2i->ant = NULL;
   return aux;
}
```

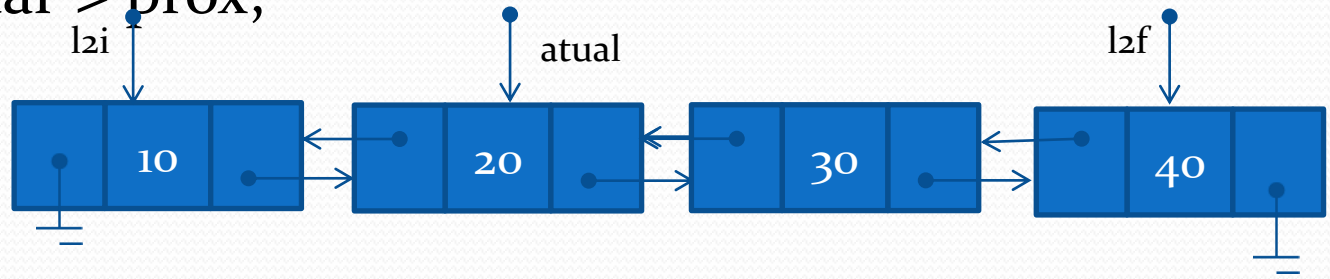
Percorrer a Lista (mostrar)

- Verifica-se se a lista não está vazia
- Usando um ponteiro auxiliar (aqui chamado de atual) percorre-se a lista a partir do primeiro (l2i)



- Move-se o ponteiro auxiliar pela lista:

atual= atual-> prox;



Código para Percorrer (frente para traz)

```
void percorrerFrenteTraz () {
    noDupla *atual; // ponteiro para percorrer a lista

    atual = l2i;
    cout << "\nLista => ";
    while (atual != NULL) {
        cout << atual->dado << "\t";
        atual = atual->prox;
    }
    cout << endl;
}
```

Exercício #1

- Altere o programa listadupla.cc (está no SIA), concluindo o trecho do código para:
  - Inserir um elemento no final da lista;
  - Remover o último da lista;
  - Remover determinado elemento da lista, cujo valor deve ser informado pelo usuário;
  - Percorrer de traz para frente.