ESTRUTURA DE DADOS - LISTA

Estrutura de Dados I – ED1

Listas Encadeada

- Estrutura que armazenam os dados em um formato de lista, ou seja, um conjunto de “nós”.
- Um nó é uma estrutura que armazena a informação a ser gerenciada por uma lista.
- Cada um dos nós de uma lista encadeada, além de conhecer o valor que está sendo armazenado em seu interior, também conhece o elemento posterior a ele: por isso ela é chamada de “lista encadeada”.

Listas Encadeada

- Lista estática
  - A lista é representada por um vetor, faz uso de endereços contíguos de memória e a ordem linear é definida pelos índices do vetor.
  - O tamanho do vetor deve contemplar o número máximo de elementos.
- Desvantagens
  - Tamanho máximo fixo. Dependendo da aplicação, é difícil prever o tamanho exato do vetor.
  - Maior custo computacional nas operações de inclusão e remoção.
    - Pode ser necessário o deslocamento de dados

Listas Encadeada

- Lista dinâmica
  - A estrutura da lista é representada por elementos, que além de conter o dado, possuem um ponteiro para o próximo elemento.
  - A lista pode aumentar ou diminuir de tamanho com base nos elementos inseridos e removidos da lista, respectivamente.

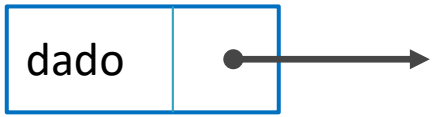
Listas Encadeada

- Lista dinâmica: vantagens
  - Cada elemento da lista é alocado dinamicamente quando necessário.
  - O tamanho máximo da lista está condicionado ao espaço disponível na memória!
  - Não há deslocamento de dados nas operação de inclusão e remoção, apenas endereçamento dos ponteiros da lista.

Listas Encadeada

Componentes/termos

- **Nó da lista (elemento):** contém o **dado** e um **ponteiro** para o próximo elemento.



- **Lista encadeada** : estrutura de dados que representa uma sequência encadeada de elementos/nós.

Listas Encadeada

Lista simplesmente encadeada

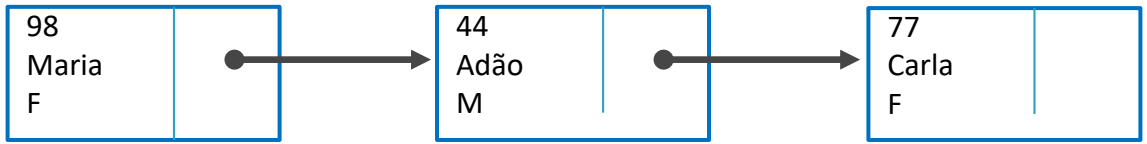
Lista circular

Lista duplamente encadeada

Listas Encadeada

- **Lista homogênea:** os elementos da lista contém apenas um dado primitivo

- **Lista heterogênea:** os elemento da lista contém um dado composto, como nome, idade e sexo.



Listas Encadeada

- **Regras**
  - Diferente de filas e pilhas, as quais possuem uma regra para inserção e retirada dos elementos da estrutura de dados, a lista encadeada não possui uma regra para a inserção e retirada dos elementos da lista.
  - Os elementos podem ser inseridos/retirados no início da lista, no final, no meio, etc.

Listas Encadeada

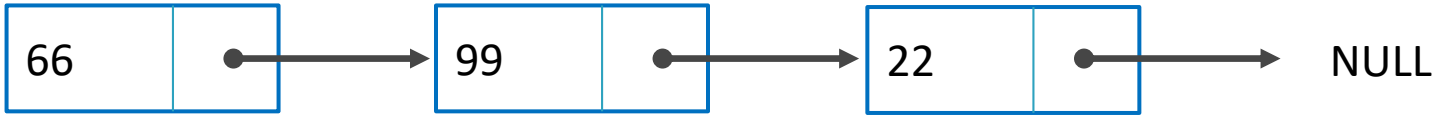
- Operações básicas:
  - inicializa() – cria uma lista vazia
  - insere() – insere um novo nó na lista
  - inserePosicao() - insere um novo nó em uma determinada posição da lista
  - remove() – remove um nó da lista
  - busca() – busca um elemento da lista
  - mostrar() – mostrar todos os elementos da lista
  - destrói() –destrói a lista, removendo todos os nós alocados

Implementação

Lista simplesmente encadeada

Listas Encadeada

- A lista encadeada é representada por um ponteiro para o primeiro nó.
- O ponteiro do último nó é **NULL**.



Listas Encadeada

- Estrutura da lista
  - O nó contém o dado a ser armazenado e o ponteiro para o próximo elemento.

```
typedef int DadoNoLista;

struct No
{
    DadoNoLista dado;
    struct No *prox;
};
```

Declaração da lista

```
No* lista;
```

Listas Encadeada

- Operação inicializa()
  - Inicializa a lista, atribuindo NULL para o nó.

```
void inicializaL(No **lista)
{
    *lista = NULL;
}
```

Listas Encadeada

- Operação insere()
  - Aloca-se memória para o novo nó, grava-se o dado.
  - O ponteiro do próximo nó aponta para o início da lista.
  - O ponteiro do início da lista aponta para o novo nó.

```
//insere no início da lista
bool insereInicioL(No **lista, DadoNoLista valor)
{

    No *novo = new No(); // aloca memória para o nó
    if (novo == NULL)
        return false;

    novo->dado = valor;
    novo->prox = *lista;
    *lista = novo;

    return true;
}
```

Listas Encadeada

- Operação mostrar()

Percorre a lista, mostrando os dados

O laço começa no primeiro nó da lista e segue para os demais nós através do ponteiro (n->prox)

```
void mostraL(No **lista)
{
    No *n = *lista;

    cout << "L:{";
    while(n != NULL ) /// enquanto n não for NULL fica no laço
    {
        cout << n->dado;
        n = n->prox;

        if(n != NULL)
            cout << ", ";
    }
    cout << "}\n";
}
```

Listas Encadeada

- Operação remover()

Remove um nó com base no valor passado por parâmetro

```
///remover um nó da lista com um valor específico
bool removeL(No **lista, DadoNoLista valor)
{
    No *anterior = NULL;
    No *atual = *lista;
    ///fica no laço enquanto tiver elementos na lista
    /// e não encontrar o valor procurado
    while(atual!=NULL && atual->dado != valor)
    {
        anterior = atual;
        atual = atual->prox;
    }
    /// pode sair do laço sem encontrar o valor (atual==NULL)
    /// se encontrar -> atual tem o endereço do elemento para excluir
    if(atual == NULL) /// se atual é NULL -> não encontrou
        return false;

    if (anterior == NULL)
    {
        /// o elemento a ser excluído está no início da lista
        *lista = atual->prox;
    }
    else   /// elemento está no meio ou no fim
    {
        anterior->prox = atual->prox;
    }

    /// libera a memória do elemento
    delete(atual);
    return true;
}
```

Listas Encadeada

- Operação leInicio()
  - Retorna o valor do primeiro nó;
  - O nó não é removido da lista;
  - Verifique se a lista não está vazia antes de ler o valor do primeiro nó.

```
//retorna o valor do primeiro elemento da lista SEM removê-lo
DadoNoLista leInicioL(No **lista)
{
    DadoNoLista dado;
    dado = 0;//se DadoNoLista for uma struct, inicializar os atributos!

    if(vaziaL(lista) == false)
        dado = (*lista)->dado; //atribui o dado do início da lista para a variável dado

    return dado;
}
```

Listas Encadeada

- Operação removeInicio()
  - Retorna o valor do primeiro nó;
  - O primeiro nó é removido da lista;
  - Verifique se a lista não está vazia antes de remover o primeiro nó.

```
//retorna o valor do primeiro elemento da lista e REMOVE o elemento
DadoNoLista removeInicioL(No **lista)
{
    No* aux;
    DadoNoLista dado;
    dado = 0;//se DadoNoLista for uma struct, inicializar os atributos!

    if(vaziaL(lista) == false)
    {
        dado = (*lista)->dado; //atribui o dado do início da lista para a variável dado
        aux = *lista; //pega a referenccia do primeiro nó, o qual será removido

        *lista = (*lista)->prox; //atualiza o início da lista para o próximo elemento

        delete aux;
    }

    return dado;
}
```

Listas Encadeada

- Operação vazia()

Verifica se a lista está vazia, ou seja, nula, sem elementos.

```
bool vaziaL(No **lista)
{
    if(! (*lista) )
        return true;
    else
        return false;
}
```

Listas Encadeada

- Operação busca()

Busca por um determinado valor na lista.

Se o valor for encontrado, o nó contendo o valor é retornado.

```
No* buscaL(No **lista, DadoNoLista valor)
{
    No *n = *lista;
    while (n != NULL)
    {
        if (n->dado == valor)
            return n;

        n = n->prox;
    }

    return NULL;
}
```

Listas Encadeada

- Operação destrói()
  - Faz um laço para percorrer a lista e liberar cada nó alocado
  - O laço começa no primeiro nó da lista e segue para os demais nós através do ponteiro (n->prox)
  - Atribuir NULL para a lista

```
/// excluir todos os elementos da lista
void destroiL(No **lista)
{
    No *n = *lista;
    while (n != NULL)
    {
        No *aux  = n;
        n = n->prox;
        delete aux;
    }
    *lista = NULL;
}
```

Implementação

Lista com descritor

Lista com descritor

- Descritor é uma estrutura composta por informações sobre a própria lista.
  - Ponteiro para o primeiro nó
  - Ponteiro para o último nó
  - Número de nós (tamanho da lista)
  - Soma dos valores de todos os nós
  - Valor mínimo ou índice do nó com menor valor
  - Valor máximo ou índice do nó com maior valor

Lista com descritor

- Considere os seguintes problemas
  - Inserir um nó no final da lista
  - Descobrir o número de nós de uma lista
- Resolver os problemas acima com uma lista simplesmente encadeada **sem descritor**:
  - É necessário percorrer a lista do primeiro nó até o último.
- Resolver os problemas com uma lista simplesmente encadeada **com descritor**:
  - Para inserir um nó no final da lista, basta acessar o ponteiro do último nó e realizar a inserção no nó desejado.
  - Para descobrir o número de nós de uma lista, basta ler o valor armazenado no descritor.
  - Em ambos os casos, não é necessário percorrer todos os elementos da lista, ou seja, mais eficiente.

Lista com descritor

- Inicializar a lista com o uso de construtor;
- Destruir a lista (apagar todos os nós) com o uso de um destrutor
- Quando uma nó é adicionado ou removido da lista, os valores dos descritores devem ser atualizados, mantendo a consistência do descritor.

```
typedef int DadoNoLista;

struct No
{
    DadoNoLista dado;
    struct No *prox;
    No() // construtor
    {
        dado = 0;
        prox = nullptr;
    }
};
```

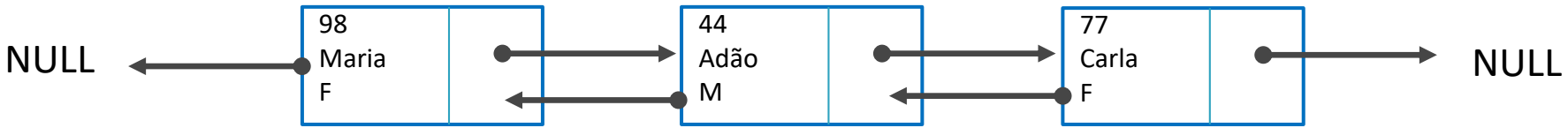
```
struct Lista
{
    No *inicio;
    No *fim;
    int tamanho;
    Lista() /// construtor
    {
        inicio = nullptr;
        fim = nullptr;
        tamanho =0;
    }
    ~Lista() /// desconstrutor
    {
        No *n = inicio;
        while(n)
        {
            No *aux = n;
            n = n->prox;
            delete aux;
        }
    }
};
```

Implementação

Lista duplamente encadeada

Lista duplamente encadeada

- Cada **nó** é composto pelo dado, um ponteiro para o próximo elemento e um ponteiro para o elemento anterior
- O ponteiro para o elemento anterior do primeiro nó é NULL
- O ponteiro para o próximo elemento do último nó é NULL
- Vantagens: A partir do ponteiro para o último nó da lista é possível percorrê-la em ordem inversa.



Lista duplamente encadeada

- Estrutura do nó

```
struct No
{
    DadoNoLista dado;
    No *prox;
    No *ant;
    No() // construtor
    {
        prox = nullptr;
        ant = nullptr;
    }
};
```

Lista duplamente encadeada

- Estrutura da lista com o descritor

```
struct Lista
{
    No *inicio;
    No *fim;
    int tamanho;
    Lista() /// construtor
    {
        //cout << "\n executando o construtor...\n";
        inicio = nullptr;
        fim = nullptr;
        tamanho =0;
    }
    ~Lista() /// desconstrutor /// quando chame delete(ponteiro)
    {
        //cout << "\n executando o desconstrutor....\n";
        No *n = inicio;
        while(n)
        {
            No *aux = n;
            n = n->prox;
            delete aux;
        }
    }
};
```

Lista duplamente encadeada

- A implementação das operações de inclusão e remoção precisam levar em conta a atualização do ponteiro para o próximo elemento e a atualização do ponteiro para o nó anterior.