

Estruturas de Dados

Pilha Encadeada e Fila

Departamento de Computação

Prof. Martín Vigil

Adaptado de prof. Jean Martina e Aldo Wangenheim

2020.1



UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Extensões do conceito de Lista Encadeada

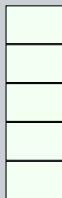
- ▶ A ideia da Lista Encadeada vista até agora é o modelo mais geral e simples;
- ▶ Pode ser especializada e estendida das mais variadas formas;
- ▶ Especializada:
 - ▶ Pilhas;
 - ▶ Filas;
- ▶ Estendida:
 - ▶ Listas Duplamente Encadeadas;
 - ▶ Listas Circulares Simples e Duplas.

Conceito de Pilhas

Pilha

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).

Pilha Vazia



Conceito de Pilhas

Pilha

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).

Empilhar 20

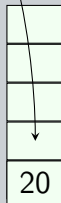


Conceito de Pilhas

Pilha

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).

Empilhar 55

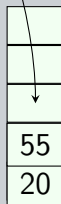


Conceito de Pilhas

Pilha

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).

Empilhar 4



Conceito de Pilhas

Pilha

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).

Empilhar 12



Conceito de Pilhas

Pilha

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).

Empilhar 7



Conceito de Pilhas

Pilha

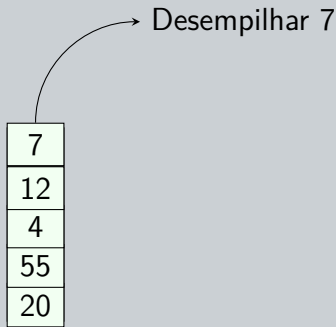
É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).

7
12
4
55
20

Conceito de Pilhas

Pilha

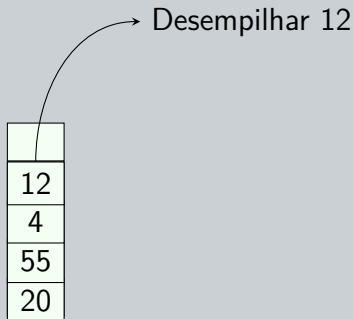
É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).



Conceito de Pilhas

Pilha

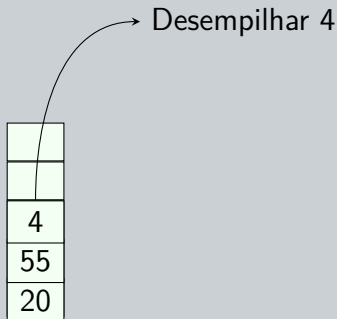
É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).



Conceito de Pilhas

Pilha

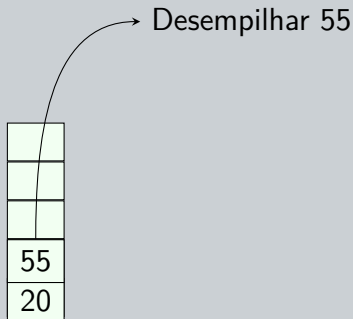
É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).



Conceito de Pilhas

Pilha

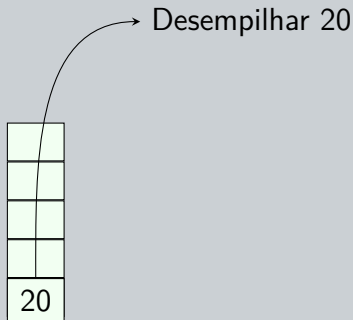
É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).



Conceito de Pilhas

Pilha

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).

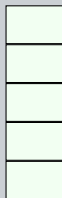


Conceito de Pilhas

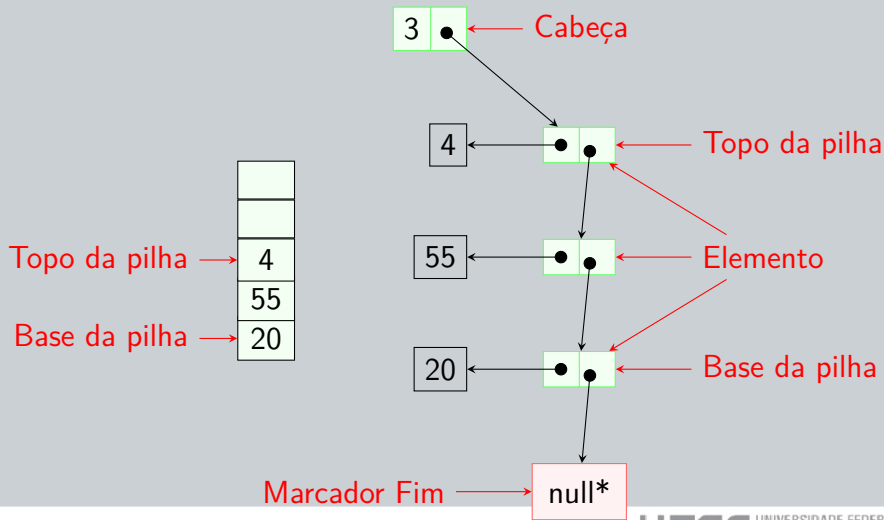
Pilha

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de empilhar, onde o **primeiro a entrar é o último a sair** (LIFO = *last in, first out*).

Pilha Vazia



Pilhas Encadeadas



Modelagem da Pilha

- ▶ Aspecto Estrutural:
 - ▶ Mesmas estruturas de uma lista encadeada:
 - ▶ Estrutura Lista
 - ▶ Estrutura Elemento
 - ▶ Ponteiro genérico de dados.

Modelagem da Pilha Encadeada

- ▶ Aspecto Funcional:
 - ▶ Inicializar a pilha.
 - ▶ Empilhar (***push***) dado na pilha;
 - ▶ Desempilhar (***pop***) dado da pilha;
 - ▶ Testar se a pilha está vazia;

Modelagem da Pilha Encadeada

- ▶ Inicializar ou limpar:
 - ▶ `inicializarPilha();`
 - ▶ `destruirPilha()`
- ▶ Testar se a pilha está vazia:
 - ▶ `bool pilhaVazia();`
- ▶ Colocar e retirar dados da pilha:
 - ▶ `push();`
 - ▶ `pop();`

Método *inicializarPilha*

- ▶ Equivalente ao `inicializarLista()`
- ▶ Complexidade temporal $\Theta(1)$

Método *destruirPilha*

- ▶ Similar a destruirLista;
- ▶ Função genérica em C não consegue desalocar qualquer tipo de dado na pilha
- ▶ Complexidade temporal $\Theta(n)$

Método *pilhaVazia*

- ▶ Equivalente a verificar se o tamanho da lista é nulo
- ▶ Complexidade temporal $\Theta(1)$

Método *push*

- ▶ Equivalente a adicionarNoInicio na lista encadeada.
- ▶ Complexidade temporal $\Theta(1)$

Método *pop*

- ▶ Equivalente a `retiraNoInicio` na lista encadeada.
- ▶ Complexidade temporal $\Theta(1)$

Aplicações de Pilhas

1. Algoritmos de Busca em Profundidade com *Backtracking*
 - ▶ Busca pela saída em labirintos
2. Chamada e retorno de funções na execução de um software
3. Mais exemplos <http://jcsites.juniata.edu/faculty/kruse/cs240/stackapps.htm>

Conceito de Filas

Fila

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de enfileiramento, onde o **primeiro a entrar é o primeiro a sair** (FIFO = *first in, first out*).

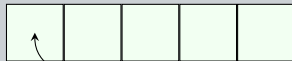
Fila Vazia



Conceito de Filas

Fila

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de enfileiramento, onde o **primeiro a entrar é o primeiro a sair** (FIFO = *first in, first out*).

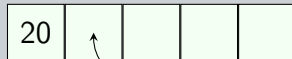


Enfileitar 20

Conceito de Filas

Fila

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de enfileiramento, onde o **primeiro a entrar é o primeiro a sair** (FIFO = *first in, first out*).



Enfileirar 55

Conceito de Filas

Fila

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de enfileiramento, onde o **primeiro a entrar é o primeiro a sair** (FIFO = *first in, first out*).

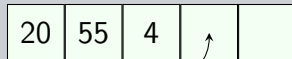


Enfileirar 4

Conceito de Filas

Fila

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de enfileiramento, onde o **primeiro a entrar é o primeiro a sair** (FIFO = *first in, first out*).

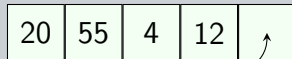


Enfileirar 12

Conceito de Filas

Fila

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de enfileiramento, onde o **primeiro a entrar é o primeiro a sair** (FIFO = *first in, first out*).

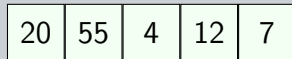


Enfilear 7

Conceito de Filas

Fila

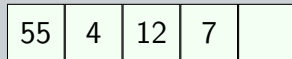
É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de enfileiramento, onde o **primeiro a entrar é o primeiro a sair** (FIFO = *first in, first out*).



Conceito de Filas

Fila

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de enfileiramento, onde o **primeiro a entrar é o primeiro a sair** (FIFO = *first in, first out*).

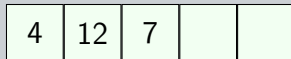


Desenfileirar 55

Conceito de Filas

Fila

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de enfileiramento, onde o **primeiro a entrar é o primeiro a sair** (FIFO = *first in, first out*).



Desenfiletrar 4

Conceito de Filas

Fila

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de enfileiramento, onde o **primeiro a entrar é o primeiro a sair** (FIFO = *first in, first out*).

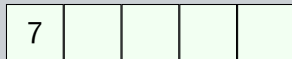


Desenfileirar 12

Conceito de Filas

Fila

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de enfileiramento, onde o **primeiro a entrar é o primeiro a sair** (FIFO = *first in, first out*).



Desenfileirar 7

Conceito de Filas

Fila

É uma estrutura de dados cujo funcionamento é inspirado no conceito “natural” de enfileiramento, onde o primeiro a entrar é o primeiro a sair (FIFO = *first in, first out*).

Fila Vazia



Fila

Modelagem funcional

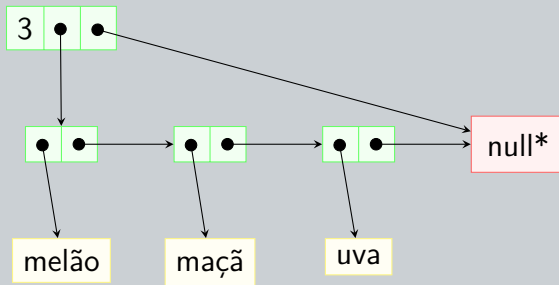
- ▶ Inserir dado ao fim da fila (***enqueue***);
- ▶ Remover dado do início da fila (***dequeue***);
- ▶ Verificar tamanho da fila;

Fila usando Lista Encadeada

Modelagem funcional

- ▶ Inserir dado ao fim da fila (**enqueue**) é $\Theta(1)$ 😊
- ▶ Remover dado do início da fila (**dequeue**) é $\Theta(n)$ 😞
- ▶ Verificar tamanho da fila é $\Theta(1)$ 😊

Fila usando Lista Encadeada Modificada



Fila usando Lista Encadeada

Modelagem funcional

- ▶ Inserir dado ao fim da fila (**enqueue**) é $\Theta(1)$ 😊
- ▶ Remover dado do início da fila (**dequeue**) é $\Theta(1)$ 😊
- ▶ Verificar tamanho da fila é $\Theta(1)$ 😊

Modelagem da Cabeça de Fila Encadeada

- ▶ Aspecto Estrutural:
 - ▶ Necessitamos um ponteiro para o primeiro elemento da fila;
 - ▶ Necessitamos um ponteiro para o último elemento da fila;
 - ▶ Necessitamos um inteiro para indicar quantos elementos a fila possui.

```
estrutura Fila {  
    Elemento *_primeiro;  
    Elemento *_ultimo;  
    int _quantidade;  
};
```

Método *inicializaFila()*

- ▶ Inicializamos `_primeiro` como nulo;
- ▶ Inicializamos `_ultimo` como nulo;
- ▶ Inicializamos o `_quantidade` como “0”;

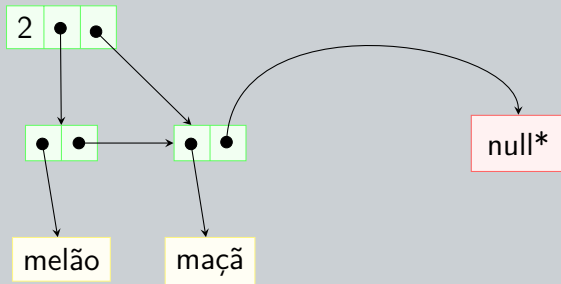
```
inicializaFila()  
inicio  
  Fila* novo <- ALOQUE(Fila)  
  novo._primeiro <- null;  
  novo._ultimo <- null;  
  novo._quantidade <- 0;  
  RETORNE Fila;  
fim;
```

Método adiciona(T^* dado)

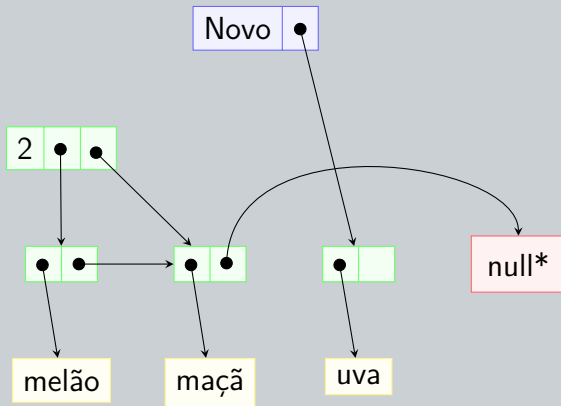
Ilustração simplificada em

<https://www.cs.usfca.edu/~galles/visualization/QueueLL.html>

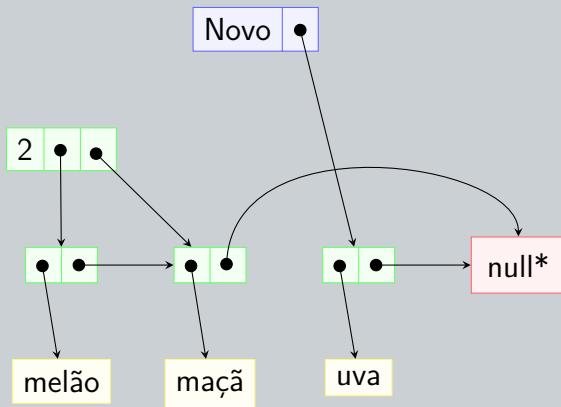
Método *adiciona*



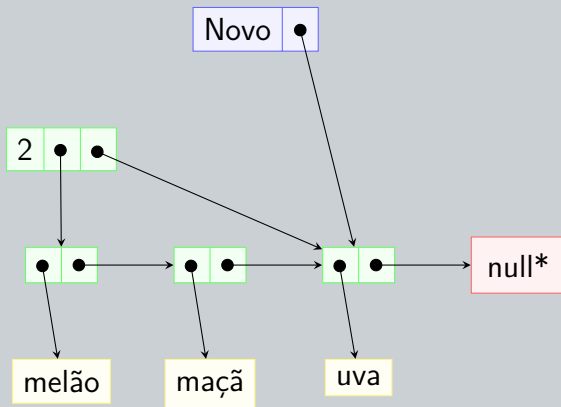
Método *adiciona*



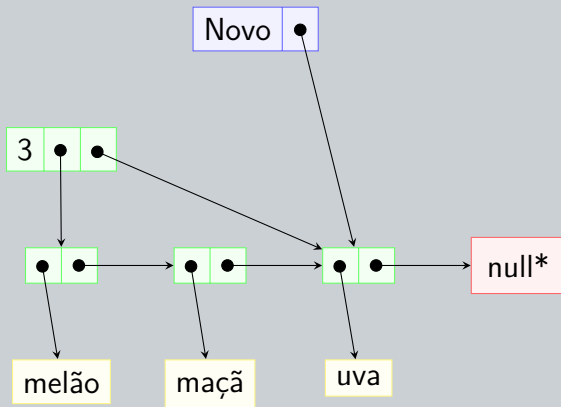
Método *adiciona*



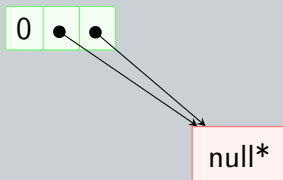
Método *adiciona*



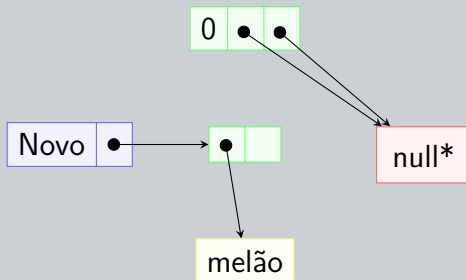
Método *adiciona*



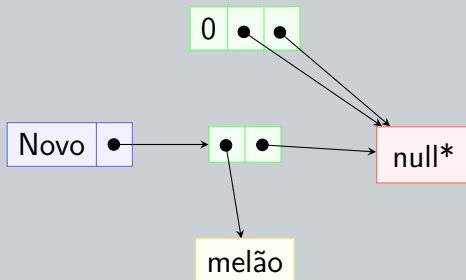
Método *adiciona* - Caso Especial Fila Vazia



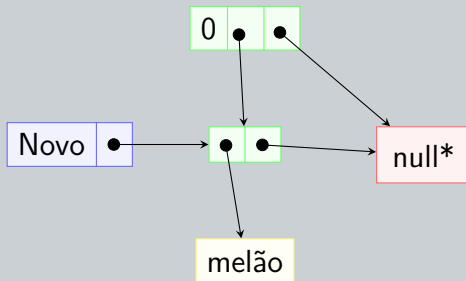
Método *adiciona* - Caso Especial Fila Vazia



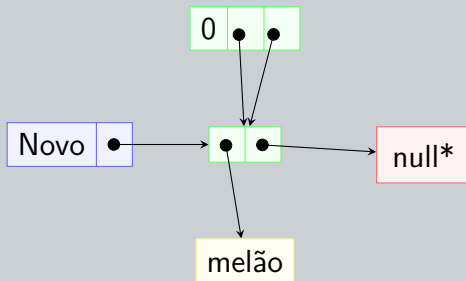
Método *adiciona* - Caso Especial Fila Vazia



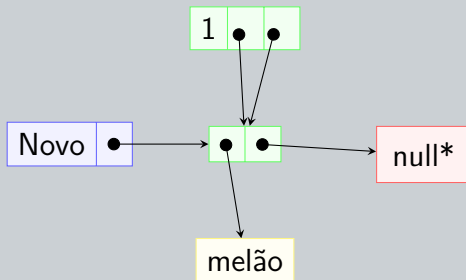
Método *adiciona* - Caso Especial Fila Vazia



Método *adiciona* - Caso Especial Fila Vazia



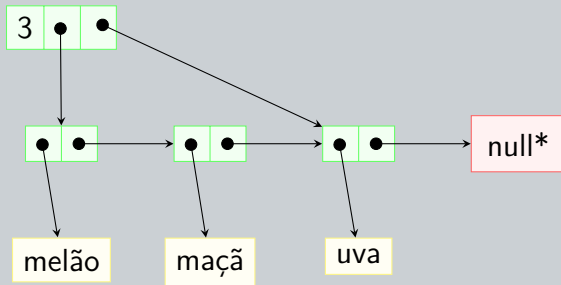
Método *adiciona* - Caso Especial Fila Vazia



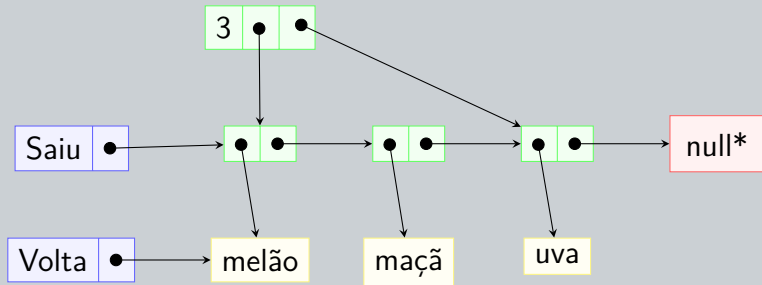
Método *adiciona*

```
adiciona(Fila* fila, T* dado)
Elemento *novo; // auxiliar.
inicio
  novo <- ALOQUE(Elemento);
  SE ( novo == null) THROW(ERROFILACHEIA);
  SE filaVazia(fila) ENTAO
    fila._primeiro <- novo
  SENA0
    fila._ultimo._proximo <- novo;
  FIM SE
  novo._proximo <- NULO;
  novo._dado <- dado;
  lista._ultimo <- novo;
  lista._quantidade <- lista._quantidade + 1;
  FIM SE
fim;
```

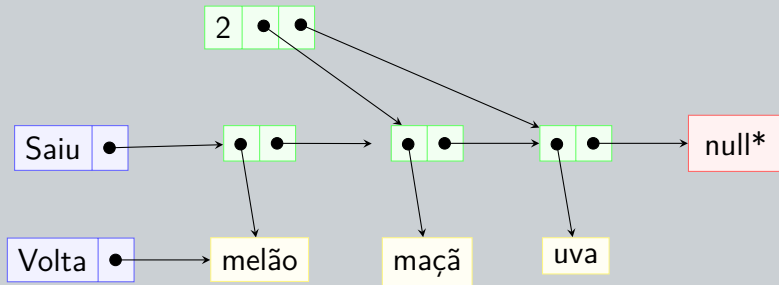
Método *T* *retira()*



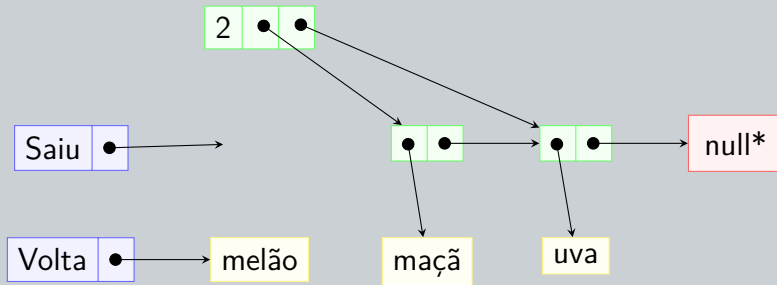
Método *T* *retira()*



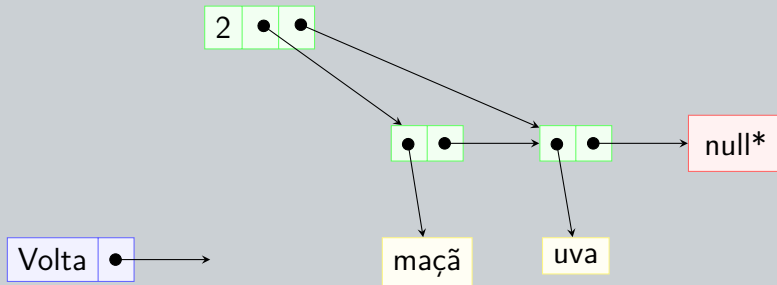
Método *T* *retira()*



Método *T* *retira()*

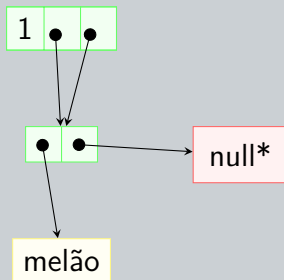


Método *T* *retira()*

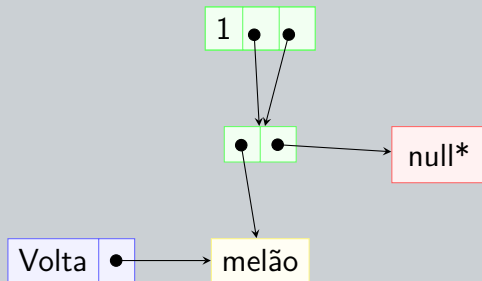


► Semelhanças??

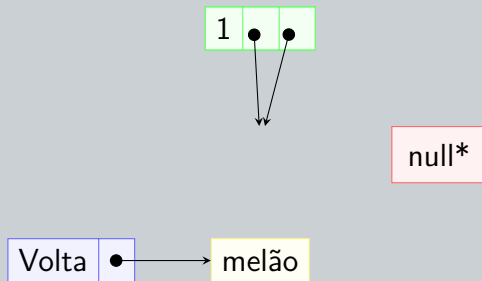
Método $T\text{retira}()$ - Caso Especial Fila Unitária



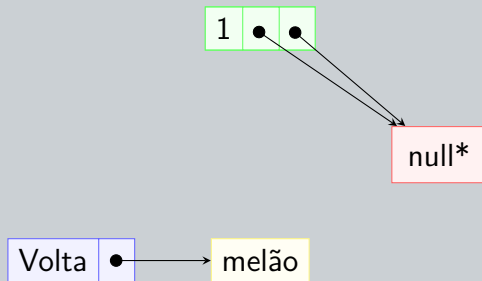
Método $T\ retira()$ - Caso Especial Fila Unitária



Método $T\text{retira}()$ - Caso Especial Fila Unitária



Método $T\text{retira}()$ - Caso Especial Fila Unitária



Método *T* *retira()*

```
T retira()
  Elemento *saiu; //Variável auxiliar elemento.
  T *volta; //Variável auxiliar tipo T.
  início
    SE (listaVazia()) ENTAO
      THROW(ERROLISTAVAZIA);
    SENA0
      saiu <- _dados;
      volta <- saiu->_info;
      _dados <- saiu->_próximo;
      //Se SAIU for o único, próximo é NULO e está certo.
      SE (_quantidade = 1) ENTAO
        //Fila unitária: devo anular o _fim também.
        _fim <- NULO;
      FIM SE
      _quantidade <- _quantidade - 1;
      LIBERE(saiu);
      RETORNE(volta);
    FIM SE
  fim;
```

Aplicações de Filas

- ▶ É importante para gerência de dados/processos por ordem cronológica:
 - ▶ Fila de impressão em uma impressora de rede;
 - ▶ Fila de pedidos de uma expedição ou tele-entrega;
- ▶ É importante para simulação de processos sequenciais:
 - ▶ chão de fábrica: fila de camisetas a serem estampadas;
 - ▶ comércio: simulação de fluxo de um caixa de supermercado;
 - ▶ tráfego: simulação de um cruzamento com um semáforo;
- ▶ Útil para algoritmos de busca em largura.

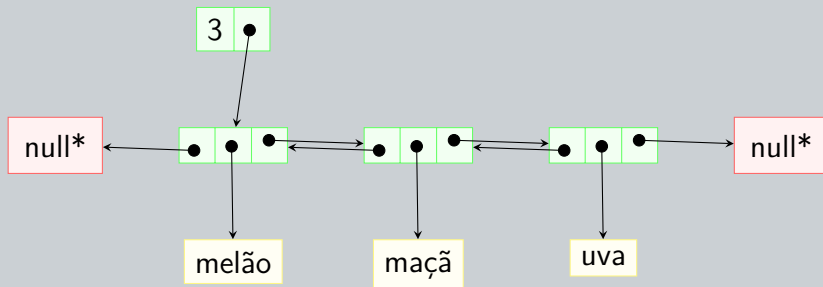
Trabalho Fila Encadeada

- ▶ Implemente uma classe Fila todas as operações vistas;
- ▶ Implemente a fila usando Templates;
- ▶ Use as melhores práticas de orientação a objetos;
- ▶ Documente todas as classes, métodos e atributos;
- ▶ Aplique os testes unitários disponíveis no moodle da disciplina para validar sua estrutura de dados;
- ▶ Entregue até a data definida no moodle.

Listas Duplamente Encadeadas

- ▶ A Lista Encadeada e a Fila Encadeada possuem a desvantagem de somente podermos caminhar em uma direção:
 - ▶ Vimos que para olhar um elemento pelo qual “acabamos de passar” precisamos de uma variável auxiliar “anterior”;
 - ▶ Para olhar outros elementos ainda anteriores não temos nenhum meio, a não ser começar de novo.
- ▶ A Lista Duplamente Encadeada é uma estrutura de lista que permite deslocamento em ambos os sentidos:
 - ▶ Útil para representar conjuntos de eventos ou objetos a serem percorridos em dois sentidos;
 - ▶ Útil também quando realizamos uma busca aproximada e nos movemos para a frente e para trás.

Lista Duplamente Encadeada



Modelagem da Cabeça de Lista Dupla

- ▶ Aspecto Estrutural:
 - ▶ Necessitamos um ponteiro para o primeiro elemento da lista;
 - ▶ Necessitamos um inteiro para indicar quantos elementos a lista possui.

```
classe ListaDupla {  
    ElementoDuplo *_dados;  
    inteiro _tamanho;  
};
```


Modelagem da Elemento de Lista Dupla

- ▶ Aspecto Estrutural:
 - ▶ Necessitamos um ponteiro para o próximo elemento;
 - ▶ Necessitamos um ponteiro para o elemento anterior;
 - ▶ Necessitamos um ponteiro do tipo da informação que vamos armazenar.
 - ▶ T necessita de um destrutor próprio, assim como a lista (neste caso a cabeça) vai precisar de um também;

```
classe ElementoDuplo {  
    Elemento *_proximo;  
    Elemento *_anterior;  
    T *_info;  
};
```

Modelagem da Lista Duplamente Encadeada

- ▶ Aspecto Funcional:
 - ▶ Temos que colocar e retirar dados da lista;
 - ▶ Temos que testar se a lista está vazia (dentre outros testes);
 - ▶ Temos que inicializar a lista e garantir a ordem de seus elementos.

Modelagem da Lista Duplamente Encadeada

- ▶ Inicializar ou limpar:
 - ▶ ListaDupla();
 - ▶ limpaListaDupla();
 - ▶ ~ListaDupla();
- ▶ Testar se a lista está vazia ou cheia e outros testes:
 - ▶ bool listaVaziaDupla();
 - ▶ int posicaoDupla(dado);
 - ▶ bool contemDupla(dado);

Modelagem da Lista Duplamente Encadeada

- ▶ Colocar e retirar dados da lista:
 - ▶ adicionaDupla(T dado);
 - ▶ adicionaNoInicioDupla(T dado);
 - ▶ adicionaNaPosicaoDupla(T dado, int posicao);
 - ▶ adicionaEmOrdemDupla(T dado);
 - ▶ T retiraDupla();
 - ▶ T retiraDoInicioDupla();
 - ▶ T retiraDaPosicaoDupla(int posicao);
 - ▶ T retiraEspecificoDupla(dado);

Método *ListaDupla()*

- ▶ Inicializamos o ponteiro para nulo;
- ▶ Inicializamos o tamanho para “0”;

```
ListaDupla()  
inicio  
    _dados = null;  
    _tamanho <- 0;  
fim;
```

Método *~ListaDupla()*

► Chamamos DestroiLista();

```
~ListaDupla()
```

```
inicio
```

```
    DestroiListaDupla();
```

```
fim;
```

Método *listaVaziaDupla()*

```
bool listaVaziaDupla()  
inicio  
SE (_tamanho = 0) ENTÃO  
    RETORNE(Verdadeiro)  
SENÃO  
    RETORNE(Falso);  
fim;
```

- Um algoritmo ListaCheia não existe na Lista Duplamente Encadeada;

Método *listaVaziaDupla()*

```
bool listaVaziaDupla()  
inicio  
SE (_tamanho = 0) ENTÃO  
    RETORNE(Verdadeiro)  
SENÃO  
    RETORNE(Falso);  
fim;
```

- ▶ Um algoritmo ListaCheia não existe na Lista Duplamente Encadeada;
- ▶ Verificar se houve espaço na memória para um novo elemento será responsabilidade de cada operação de adição.

Método *adicionaNoInicioDupla*(*T dado*)

- ▶ Testamos se é possível alocar um elemento;

Método *adicionaNoInicioDupla(T dado)*

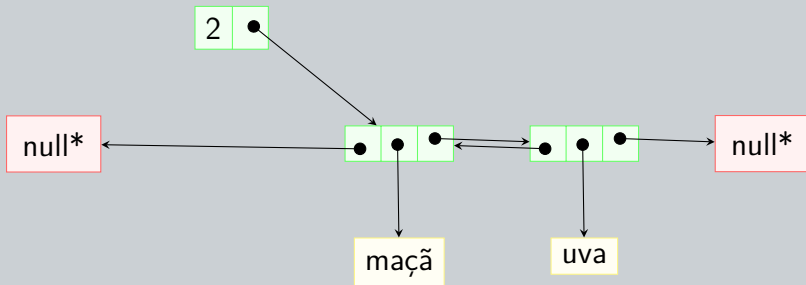
- ▶ Testamos se é possível alocar um elemento;
- ▶ Fazemos o próximo deste novo elemento ser o primeiro;

Método *adicionaNoInicioDupla*(*T dado*)

- ▶ Testamos se é possível alocar um elemento;
- ▶ Fazemos o próximo deste novo elemento ser o primeiro;
- ▶ Fazemos a cabeça de lista apontar para o novo elemento.

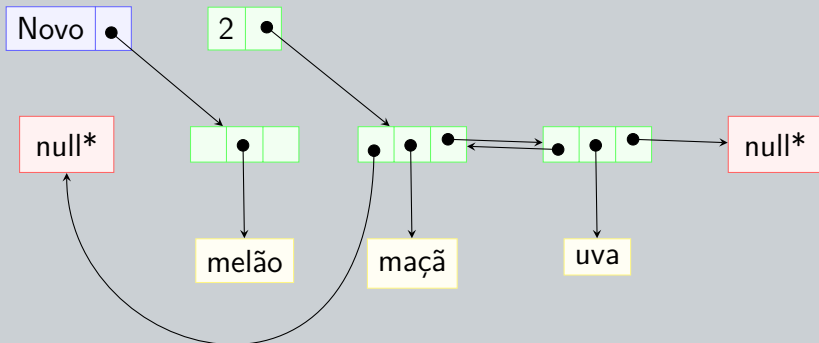
Método *adicionaNoInicioDupla*(*T* dado)

- ▶ Testamos se é possível alocar um elemento;
- ▶ Fazemos o próximo deste novo elemento ser o primeiro;
- ▶ Fazemos a cabeça de lista apontar para o novo elemento.



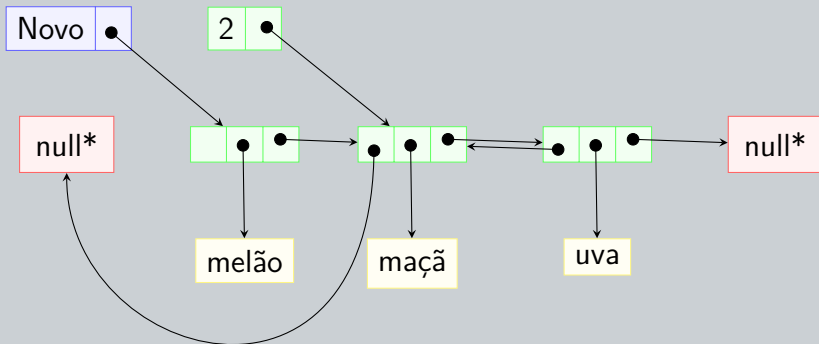
Método *adicionaNoInicioDupla*(*T* dado)

- ▶ Testamos se é possível alocar um elemento;
- ▶ Fazemos o próximo deste novo elemento ser o primeiro;
- ▶ Fazemos a cabeça de lista apontar para o novo elemento.



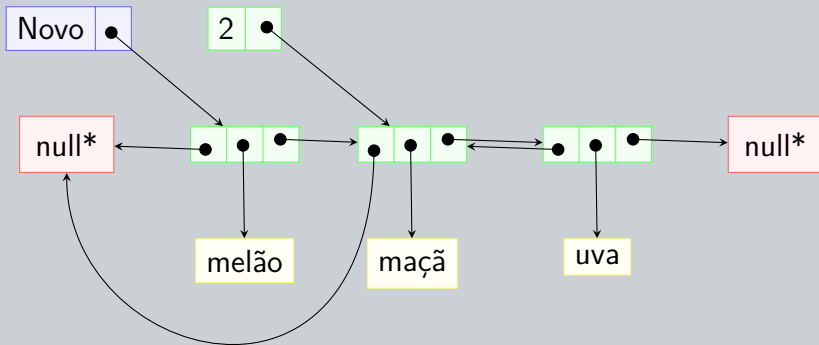
Método *adicionaNoInicioDupla*(T dado)

- ▶ Testamos se é possível alocar um elemento;
- ▶ Fazemos o próximo deste novo elemento ser o primeiro;
- ▶ Fazemos a cabeça de lista apontar para o novo elemento.



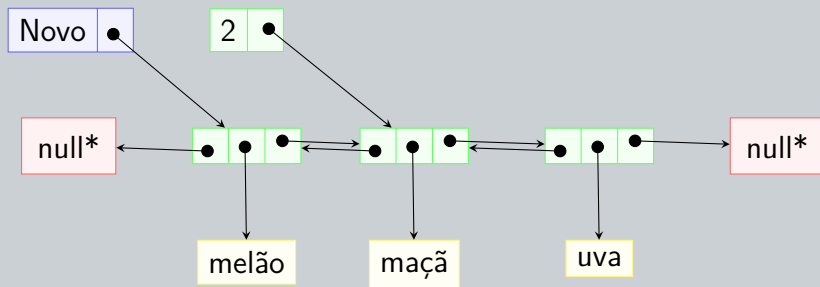
Método *adicionaNoInicioDupla*(*T* dado)

- ▶ Testamos se é possível alocar um elemento;
- ▶ Fazemos o próximo deste novo elemento ser o primeiro;
- ▶ Fazemos a cabeça de lista apontar para o novo elemento.



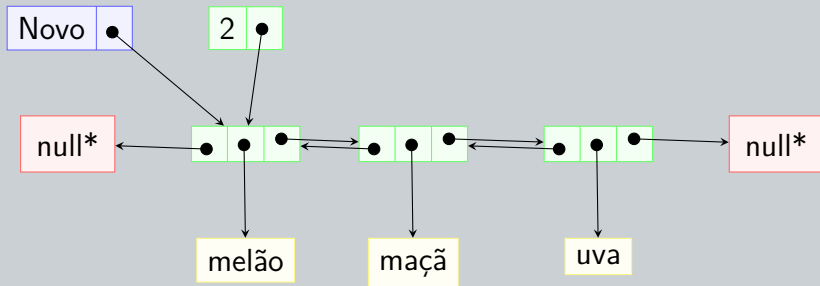
Método *adicionaNoInicioDupla*(*T* dado)

- ▶ Testamos se é possível alocar um elemento;
- ▶ Fazemos o próximo deste novo elemento ser o primeiro;
- ▶ Fazemos a cabeça de lista apontar para o novo elemento.



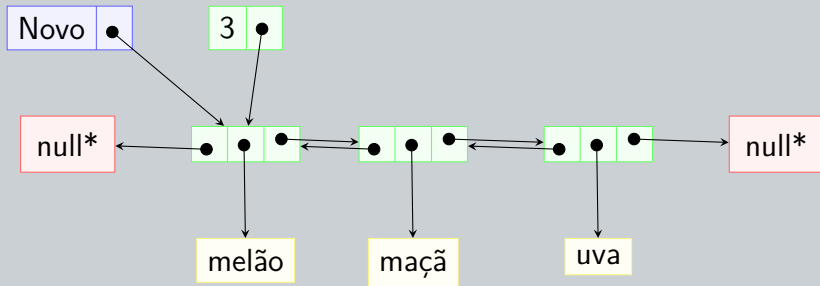
Método *adicionaNoInicioDupla*(*T dado*)

- ▶ Testamos se é possível alocar um elemento;
- ▶ Fazemos o próximo deste novo elemento ser o primeiro;
- ▶ Fazemos a cabeça de lista apontar para o novo elemento.



Método *adicionaNoInicioDupla*(*T* dado)

- ▶ Testamos se é possível alocar um elemento;
- ▶ Fazemos o próximo deste novo elemento ser o primeiro;
- ▶ Fazemos a cabeça de lista apontar para o novo elemento.



Método *adicionaNoInicioDupla*(*T* dado)

```
adicionaNoInicioDupla(T dado)
  ElementoDuplo *novo; //Variável auxiliar.
  início
  novo <- aloque(ElementoDuplo);
  SE (novo = NULO) ENTAO
    THROW(ERROLISTACHEIA);
  SENAO
    novo->_proximo <- _dados;
    novo->_anterior <- NULO;
    novo->_info <- dado;
    _dados <- novo;
    SE (novo->_proximo ~= NULO) ENTAO
      novo->_proximo->_anterior <- novo;
    FIM SE;
    _tamanho <- _tamanho + 1;
  FIM SE
  fim;
```

Método *T* *retiraDoInicioDupla()*

- ▶ Testamos se há elementos;

Método *T* *retiraDoInicioDupla()*

- ▶ Testamos se há elementos;
- ▶ Decrementamos o tamanho;

Método *T* *retiraDoInicioDupla()*

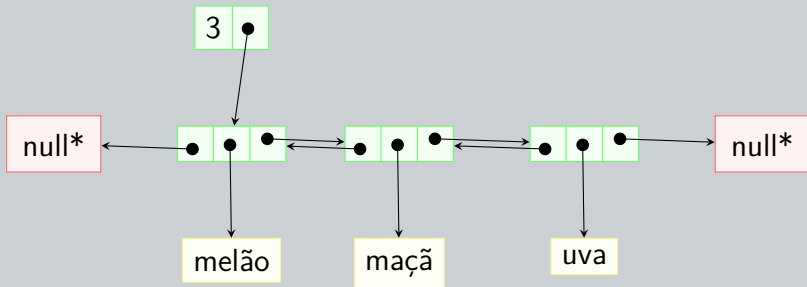
- ▶ Testamos se há elementos;
- ▶ Decrementamos o tamanho;
- ▶ Liberamos a memória do elemento;

Método *T retiraDoInicioDupla()*

- ▶ Testamos se há elementos;
- ▶ Decrementamos o tamanho;
- ▶ Liberamos a memória do elemento;
- ▶ Devolvemos a informação.

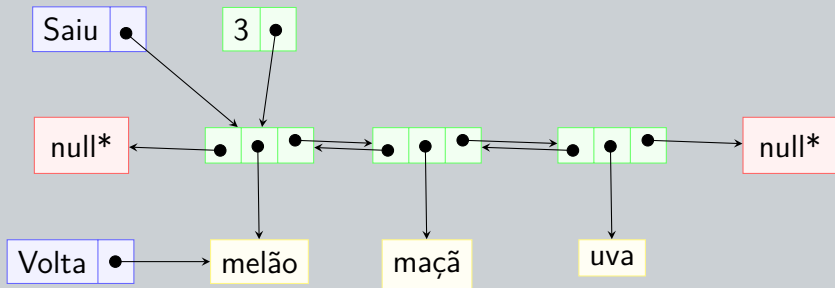
Método *T retiraDoInicioDupla()*

- ▶ Testamos se há elementos;
- ▶ Decrementamos o tamanho;
- ▶ Liberamos a memória do elemento;
- ▶ Devolvemos a informação.



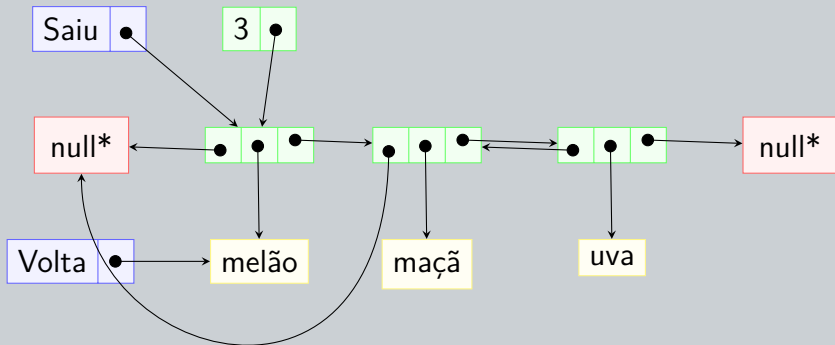
Método *T retiraDoInicioDupla()*

- ▶ Testamos se há elementos;
- ▶ Decrementamos o tamanho;
- ▶ Liberamos a memória do elemento;
- ▶ Devolvemos a informação.



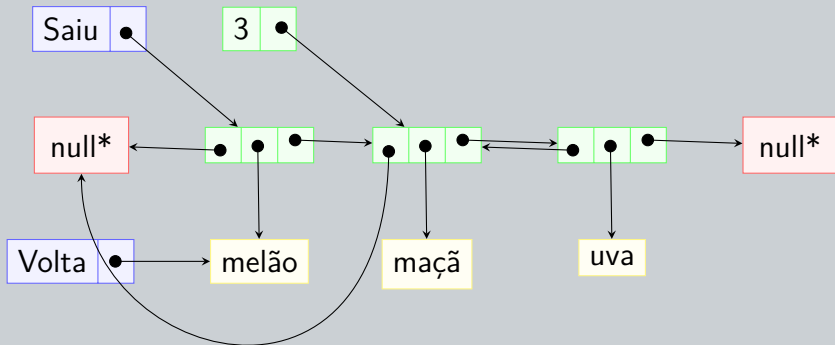
Método *T retiraDolnicioDupla()*

- ▶ Testamos se há elementos;
- ▶ Decrementamos o tamanho;
- ▶ Liberamos a memória do elemento;
- ▶ Devolvemos a informação.



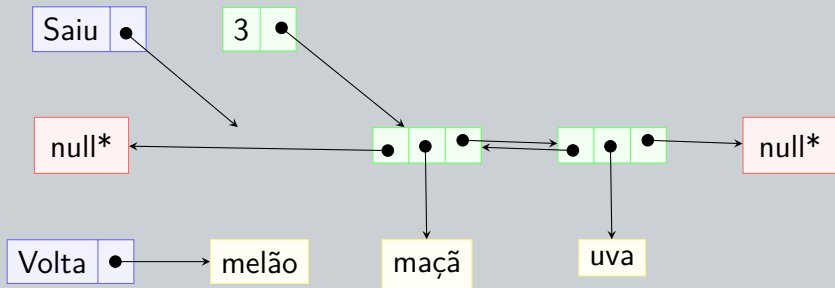
Método *T retiraDolnicioDupla()*

- ▶ Testamos se há elementos;
- ▶ Decrementamos o tamanho;
- ▶ Liberamos a memória do elemento;
- ▶ Devolvemos a informação.



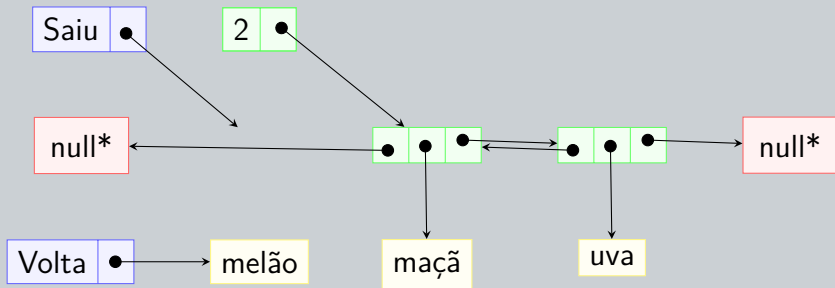
Método *T retiraDoInicioDupla()*

- ▶ Testamos se há elementos;
- ▶ Decrementamos o tamanho;
- ▶ Liberamos a memória do elemento;
- ▶ Devolvemos a informação.



Método *T retiraDoInicioDupla()*

- ▶ Testamos se há elementos;
- ▶ Decrementamos o tamanho;
- ▶ Liberamos a memória do elemento;
- ▶ Devolvemos a informação.



Método *T retiraDoInicioDupla()*

```
T retiraDoInicioDupla()
  ElementoDuplo *saiu; //Variável auxiliar elemento.
  T *volta; //Variável auxiliar tipo T.
  início
    SE (listaVaziaDupla()) ENTAO
      THROW(ERROLISTAVAZIA);
    SENA0
      saiu <- _dados;
      volta <- saiu->_info;
      _dados <- saiu->_proximo;
      SE (_dados ~= NULO) ENTAO
        dados->_anterior <- NULO;
      FIM SE
      _tamanho <- _tamanho - 1;
      LIBERE(saiu);
      RETORNE(volta);
    FIM SE
  fim;
```

Método *adicionaNaPosicaoDupla*(*T* dado, *int* posicao)

- ▶ Praticamente idêntico à lista encadeada;
- ▶ Procedimento:
 - ▶ Caminhamos até a posição;
 - ▶ Adicionamos o novo dado na posição;
 - ▶ Tratamos o caso especial;
 - ▶ Incrementamos o tamanho.
- ▶ Parâmetros:
 - ▶ O dado a ser inserido;
 - ▶ A posição onde inserir;

Método *adicionaNaPosicaoDupla*(*T* dado, *int* posicao)

```
adicionaNaPosicaoDupla(T dado, int posicao)
  ElementoDuplo *novo, *anterior; // auxiliares.
  inicio
  SE (posicao > _tamanho + 1) ENTAO THROW(ERROPOSICAO);
  SENA0
  SE (posicao = 1) ENTAO RETORNE(adicionaNoInicioDupla(info));
  SENA0
  novo <- aloque(ElementoDuplo);
  SE (novo = NULO) ENTÃO THROW(ERROLISTACHEIA);
  SENA0
  anterior <- _dados;
  REPITA (posicao - 2) VEZES anterior <- anterior->_proximo;
  novo->_proximo <- anterior->_proximo;
  SE (novo->_proximo ~= NULO) ENTAO
    novo->_proximo->_anterior <- novo;
  novo->_info <- info;
  anterior->_proximo <- novo;
  _tamanho <- _tamanho + 1;
  FIM SE
  FIM SE
  FIM SE
fim;
```


Método *T retiraDaPosicaoDupla(int posicao)*

- ▶ Praticamente idêntico à lista encadeada;
- ▶ Procedimento:
 - ▶ Caminhamos até a posição;
 - ▶ Retiramos o novo dado na posição;
 - ▶ Tratamos o caso especial;
 - ▶ Decrementamos o tamanho.
- ▶ Parâmetros:
 - ▶ A posição onde retirar;

Método *T* *retiraDaPosicaoDupla*(*int* *posicao*)

```
T retiraDaPosicaoDupla(int posicao)
    ElementoDuplo *anterior, *eliminar; //Variáveis elemento.
    T *volta; //Variável tipo T.
    inicio
        SE (posicao > _tamanho ) ENTAO THROW(ERROPOSICAO);
        SENA0
            SE (posicao = 1) ENTAO RETORNE(retiraDoInicioDupla());
            SENA0
                anterior <- _dados;
                REPITA (posicao - 2) VEZES
                    anterior <- anterior->_proximo;
                eliminar <- anterior->_proximo;
                volta <- eliminar->_info;
                anterior->_proximo <- eliminar->_proximo;
            SE eliminar->_proximo ~= NULO ENTAO
                eliminar->_proximo->_anterior <- anterior;
            _tamanho <- _tamanho - 1;
            LIBERE(eliminar);  RETORNE(volta);
        FIM SE
    FIM SE
fim;
```

Método *adicionaEmOrdemDupla*(*T dado*)

- ▶ Idêntico à lista encadeada;
- ▶ Procedimento:
 - ▶ Necessitamos de uma função para comparar os dados “>;
 - ▶ Procuramos pela posição onde inserir comparando dados;
 - ▶ Chamamos *adicionaNaPosiçãoDupla*().
- ▶ Parâmetros:
 - ▶ O dado a ser inserido;

Por conta do aluno:

- ▶ Operações de inclusão e exclusão:
 - ▶ AdicionaDupla(dado);
 - ▶ RetiraDupla();
 - ▶ RetiraEspecíficoDupla(dado);
- ▶ Operações - inicializar ou limpar:
 - ▶ DestróiListaDupla();

Trabalho Lista Duplamente Encadeada

- ▶ Implemente uma classe ListaDupla todas as operações vistas;
- ▶ Implemente a lista usando Templates;
- ▶ Use as melhores práticas de orientação a objetos;
- ▶ Documente todas as classes, métodos e atributos;
- ▶ Aplique os testes unitários disponíveis no moodle da disciplina para validar sua estrutura de dados;
- ▶ Entregue até a data definida no moodle.

Perguntas????



UNIVERSIDADE FEDERAL
DE SANTA CATARINA



Este trabalho está licenciado sob uma Licença Creative Commons Atribuição 4.0 Internacional. Para ver uma cópia desta licença, visite

<http://creativecommons.org/licenses/by/4.0/>.



UNIVERSIDADE FEDERAL
DE SANTA CATARINA