UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Programação e Desenvolvimento de Software 2

Listas encadeadas

Prof. Luiz Chaimowicz
(slides adaptados do Prof. Douglas Macharet)

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

Introdução

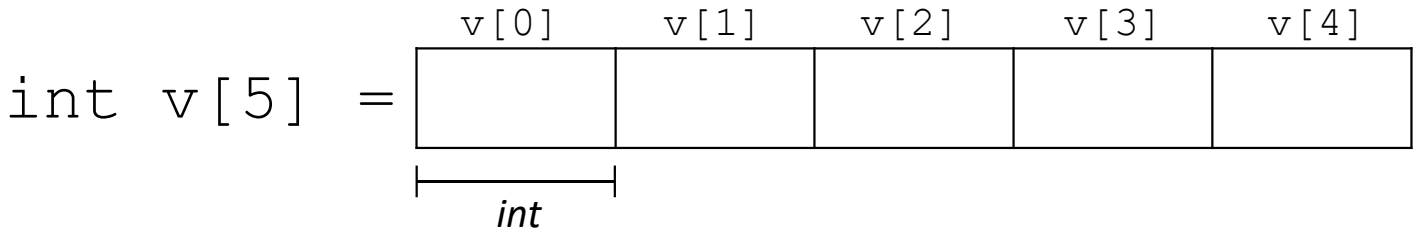
- Tipos Abstratos de Dados (TADs)
  - Conjunto de valores
  - Conjunto de operações sobre esses valores

O que o TAD representa e faz é
mais importante do que **como** ele faz!

- Integridade, manutenção, reutilização, ...

Introdução

- Como guardar uma coleção de elementos?
  - Arrays Tradicionais (vetores)
- Propriedades
  - Tamanho fixo
  - Acesso direto (índice)



Introdução

```
#include <iostream>

using namespace std;

int main() {
  int vetorA[5];
  vetorA[3] = 99;
  for(int i=0; i<5; i++)
    cout << vetorA[i] << "\t";
  cout << endl;

  int vetorB[5] = {};
  for(int i=0; i<5; i++)
    cout << vetorB[i] << "\t";
  cout << endl;

  double vetorC[] = {1.1, 2.2, 3.3};
  cout << vetorC[1] << endl;

  return 0;
}
```

As posições não foram inicializadas!

Posições inicializadas com 0.

Alocação e inicialização.

Introdução

```
int main() {
  int stackMatrixB[3][4] = {
    {0, 1, 2, 3},
    {4, 5, 6, 7},
    {8, 9, 10, 11}
  };
  int rows = 3;
  int cols = 4;
  int** heapMatrix = new int*[rows];
  for (int i=0; i < rows; i++)
    heapMatrix[i] = new int[cols];

  for (int i=0; i < rows; i++)
    for (int j=0; j < cols; j++)
      heapMatrix[i][j] = (rows*i + i) + j;

  for (int i=0; i < rows; i++)
    delete[] heapMatrix[i];

  delete[] heapMatrix;

  return 0;
}
```

STACK

HEAP

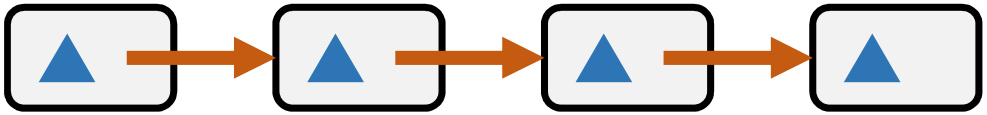
http://www.cplusplus.com/reference/new/operator%20delete[]/

Code

Listas encadeadas

O TAD lista será visto com muito mais detalhes em ED... Aqui vamos usá-lo para praticar os conceitos de TADs e Apontadores

- Forma alternativa de guardar coleções
- Cada **célula / nó** possui duas informações
  - **Conteúdo** (valor) que se deseja armazenar
  - **Referência** para o elemento seguinte na cadeia



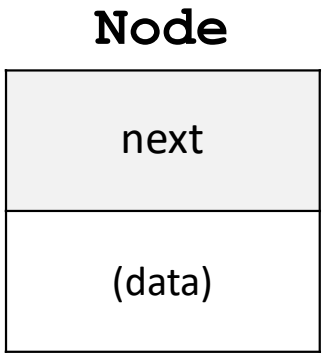
Arrays vs. Listas encadeadas

- Arrays
  - Acesso **direto** (índice)
  - Tamanho fixo e conhecido
    - T = n x sizeof(elemento)
- Listas encadeadas
  - Acesso **sequencial**
  - Tamanho variável (um elemento por vez)
    - T = n x sizeof(elemento) + n x sizeof(referência)

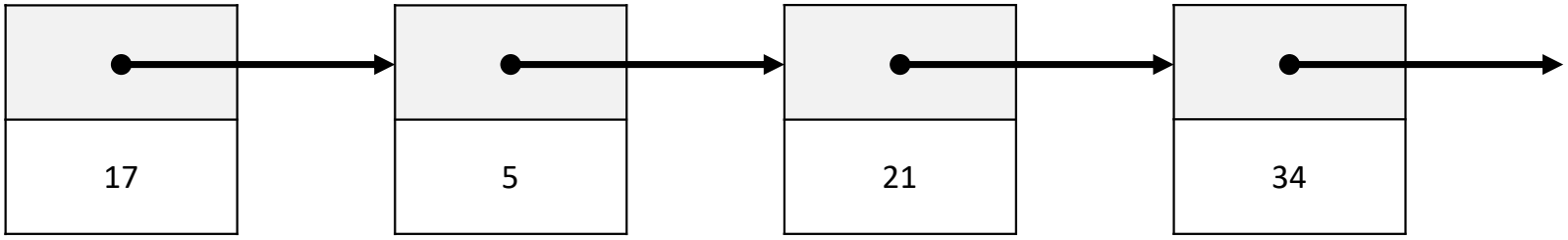
Listas encadeadas

- Célula/Nó
  - Conteúdo
  - Referência

Campo `data` poderia ser um struct com diversos campos ou mesmo outro TAD




```
struct Node {

  int    data;
  Node* next;

};
```



Listas encadeadas

```
#include <iostream>

using namespace std;

int main() {
     Node *a = new Node;
    Node *b = new Node;

     a->data = 99;
     a->next = b;

     b->data = 123;

     cout << a->data << endl;
     cout << a->next->data << endl;

     return 0;
}
```

Listas encadeadas

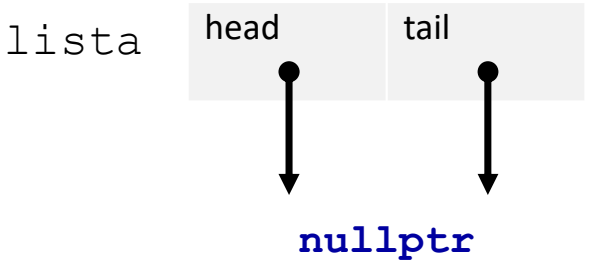
- Possíveis Operações:
  - Criar uma nova lista (inicialização)
  - Inserir elementos
  - Retirar elementos
  - Localizar um elemento
  - Recuperar o valor de um elemento
  - Recuperar o elemento seguinte à um elemento
  - ...

Listas encadeadas

- Todas as operações ficam dentro do Node?
  - Não!
  - Criar um TAD que efetivamente define a **Lista**
- Quais dados / atributos deve possuir?
  - Referência para o primeiro Node (Por quê?)
    - Head, Cabeça
  - Referência para o último Node (Por quê?)
    - Tail, Cauda

Listas encadeadas (Contrato)

```
#include <list.hpp>
int main() {


List lista;
…
}
```



List.hpp

```
#ifndef LIST_H
#define LIST_H

#include <iostream>
using namespace std;

struct Node {
  int   data;
  Node* next;
};

struct List {
  Node* head = nullptr;
  Node* tail = nullptr;

  void insertNode(int data);
  void removeNode(int data);
  void display();
};

#endif
```

Listas encadeadas
Inserção

- Como inserir um novo valor na lista (final)?
  - Criar um Node
  - Se a Lista estiver vazia
    - Novo Node será Cabeça e Cauda
  - Caso contrário
    - Novo Node será inserido após Cauda
    - Esse Node agora é a Cauda

Listas encadeadas

Inserção

List.cpp

```
#include "List.hpp"

void List::insertNode(int data) {

   Node* aux = new Node;
   aux->data = data;
   aux->next = nullptr;

   if (head == nullptr) {
      head = aux;
      tail = aux;
   } else {
      tail->next = aux;
      tail = aux;
   }
}
```

Cria o Node

"liga" o nó e ajusta Head/Tail

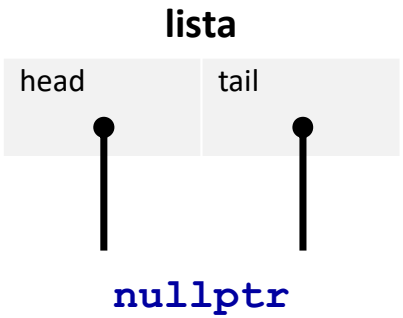
Listas encadeadas
Inserção

main.cpp

```
#include "List.hpp"

int main() {
  List lista;

  lista.insertNode(111);
  lista.insertNode(222);

  return 0;
}
```

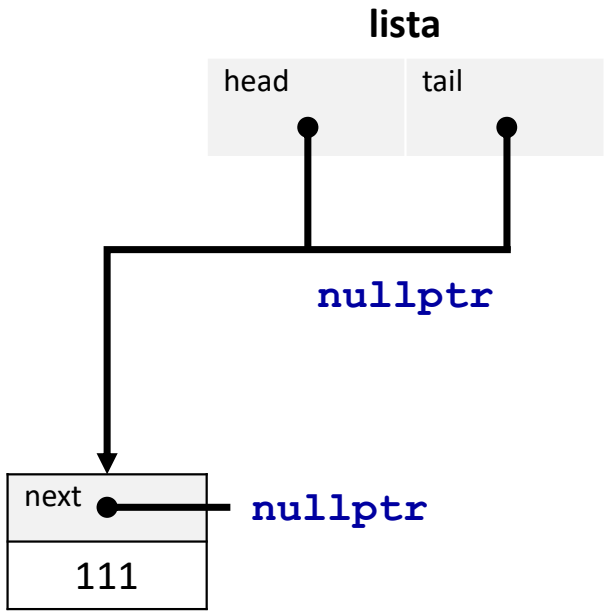
Listas encadeadas
Inserção

```
#include "List.hpp"

int main() {

  List lista;

  lista.insertNode(111);
  lista.insertNode(222);

  return 0;
}
```



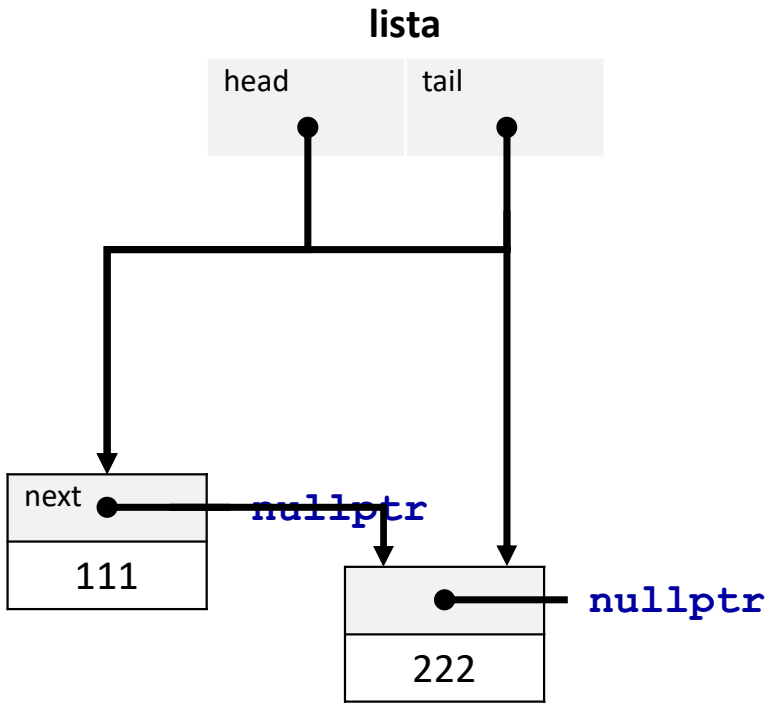
Listas encadeadas
Inserção

```
#include "List.hpp"

int main() {

  List lista;

  lista.insertNode(111);
  lista.insertNode(222);

  return 0;
}
```



Listas encadeadas
Inserção

```
#include "List.hpp"

int main() {

  List lista;

  lista.insertNode(111);
  lista.insertNode(222);

  return 0;
}
```



Listas encadeadas
Remoção

- Como remover um determinado Node?
- Quantos casos?
  - Cabeça
    - Cabeça aponta para o próximo do Node removido
    - Teste especial se a lista ficar vazia após a remoção
  - Cauda
    - Node anterior não aponta mais para ninguém
    - Node anterior vira Cauda
  - Outros
    - Node anterior aponta para o próximo do removido

Listas encadeadas
Remoção

Avaliar os casos

Desalocar a memória

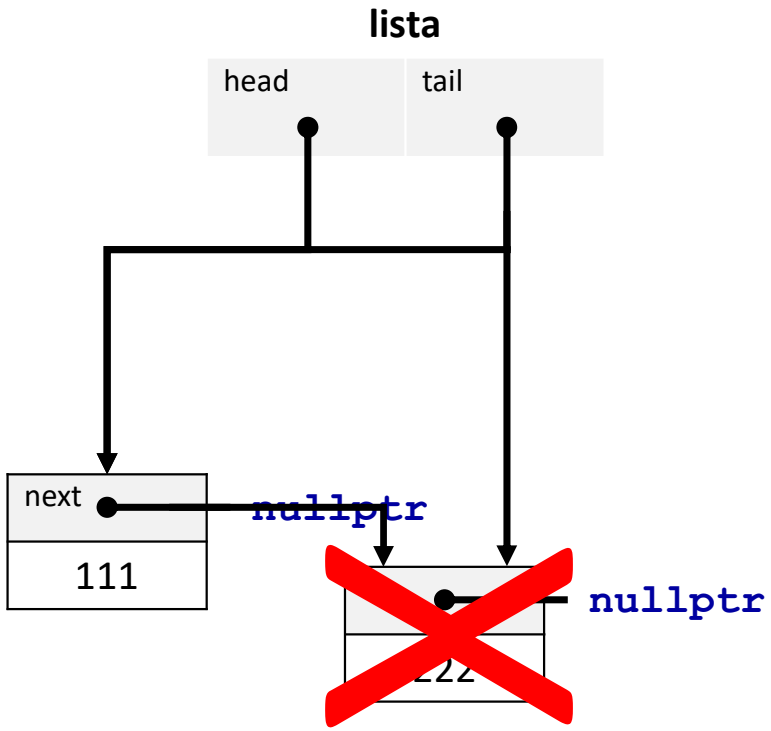
Caminha na lista

List.cpp

```
void List::removeNode(int data) {

  Node *current = head;
  Node *previous = nullptr;

  while (current != nullptr) {
    if (current->data == data) {
      if (previous == nullptr) { // HEAD
        head = current->next;
        if (current == tail)
            tail = nullptr; //vazia
      } else if (current->next == nullptr) { //TAIL
        previous->next = nullptr;
        tail = previous;
      } else {
        previous->next = current->next;
      }
      delete current;
      return;
    }
    previous = current;
    current = current->next;
  }
}
```

Listas encadeadas
Remoção

```
#include "List.hpp"

int main() {

  List lista;

  lista.insertNode(111);
  lista.insertNode(222);

  lista.removeNode(222);

  return 0;
}
```



Listas encadeadas
Enumeração

- Como exibir o estado atual da Lista?
  - Percorrer a lista até chegar ao último elemento
  - Como determinar o último elemento?
    - Comparar com a Cauda
    - Aponta para `nullptr`

Listas encadeadas

Enumeração

List.cpp

```
void List::display() {

  Node *aux = head;
  while (aux != nullptr) {
    cout << aux->data << "\t";
    aux = aux->next;
  }
  cout << endl;
}
```

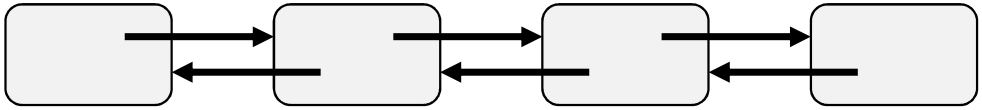
Listas encadeadas
Enumeração

```
#include "List.hpp"

int main() {
  List lista;

  lista.insertNode(111);
  lista.insertNode(222);
  lista.display();

  return 0;
}
```

Code

Listas duplamente encadeadas

- Referência também para o elemento anterior



```
struct Node {

  int    data;
  Node* next;
  Node* previous;

};
```

Listas duplamente encadeadas

- Vantagens
  - Percorrer em ambas as direções
  - Inserção/Remoção diretas (fornecido ponteiro)
- Desvantagens
  - Espaço extra para a nova referência
- Qual devo utilizar?
  - Depende da sua aplicação!

Listas encadeadas
Exercício

- Listas Simples e Duplamente encadeadas
- Implemente as seguintes operações
  - Atributo que guarda o número de elementos
  - Método que verifica se um elemento está na lista

- Memória está sendo liberada corretamente? [Pythontutor]
  - Como destruir corretamente a Lista? (liberar os Nodes)