Listas Simplesmente Encadeadas

Estrutura de Dados — QXD0010

Roberto Cabral
rbcabral@ufc.br

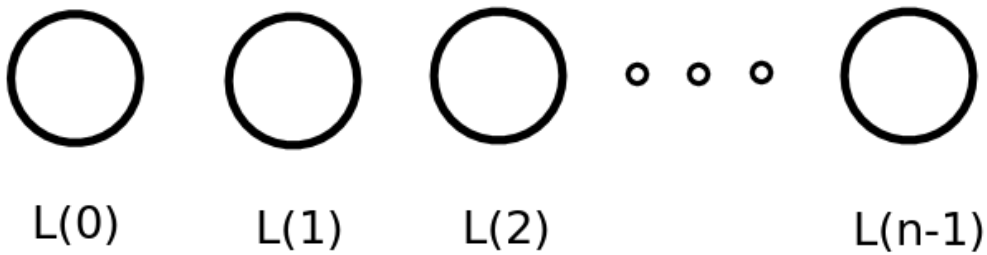
Universidade Federal do Ceará

2º semestre/2022

Introdução

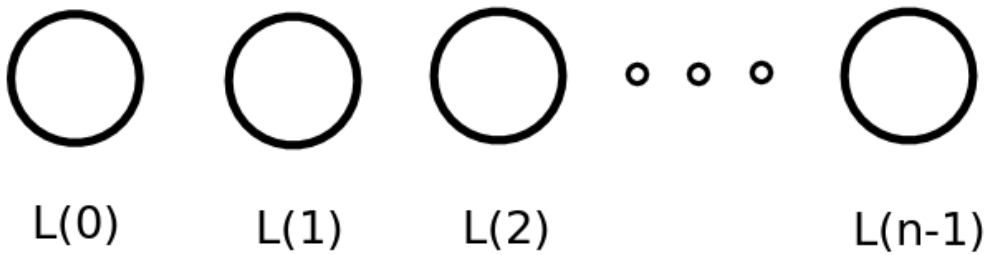
Estrutura de dados: Lista linear

- Uma lista linear L é um conjunto de $n \geq 0$ nós (ou células) $L_0, L_1, \ldots, L_{n-1}$ tais que suas propriedades estruturais decorrem, unicamente, da posição relativa dos nós dentro da sequência linear:

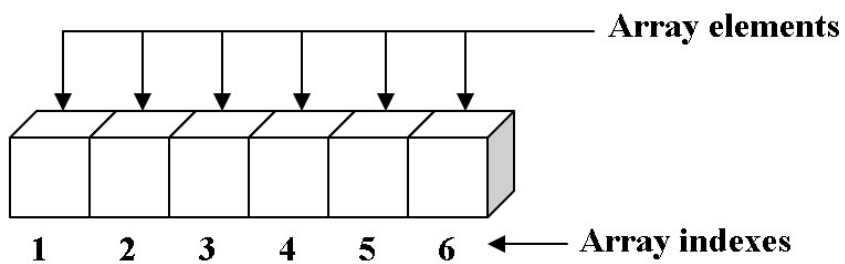


Estrutura de dados: Lista linear

- Uma lista linear L é um conjunto de $n \geq 0$ nós (ou células) $L_0, L_1, \ldots, L_{n-1}$ tais que suas propriedades estruturais decorrem, unicamente, da posição relativa dos nós dentro da sequência linear:



- Vimos que uma lista linear pode ser implementada usando alocação dinâmica de memória por meio de um vetor (alocação sequencial).



One-dimensional array with six elements

Alocação sequencial

Vantagens do uso de vetores:

- operações de acesso aos elementos são rápidas: $O(1)$
- poucos ponteiros: maior parte do espaço é utilizada para dados

Alocação sequencial

Vantagens do uso de vetores:

- operações de acesso aos elementos são rápidas: $O(1)$
- poucos ponteiros: maior parte do espaço é utilizada para dados

Desvantagens do uso de vetores:

- estão alocados contiguamente na memória
  - pode ser que tenhamos espaço na memória
  - mas não para alocar um vetor do tamanho desejado

Alocação sequencial

Vantagens do uso de vetores:

- operações de acesso aos elementos são rápidas: $O(1)$
- poucos ponteiros: maior parte do espaço é utilizada para dados

Desvantagens do uso de vetores:

- estão alocados contiguamente na memória
  - pode ser que tenhamos espaço na memória
  - mas não para alocar um vetor do tamanho desejado
- têm um tamanho fixo
  - ou alocamos um vetor pequeno e o espaço pode acabar
  - ou alocamos um vetor grande e desperdiçamos memória
  - Solução: criar lista sequencial redimensionável

Alocação sequencial

Vantagens do uso de vetores:

- operações de acesso aos elementos são rápidas: $O(1)$
- poucos ponteiros: maior parte do espaço é utilizada para dados

Desvantagens do uso de vetores:

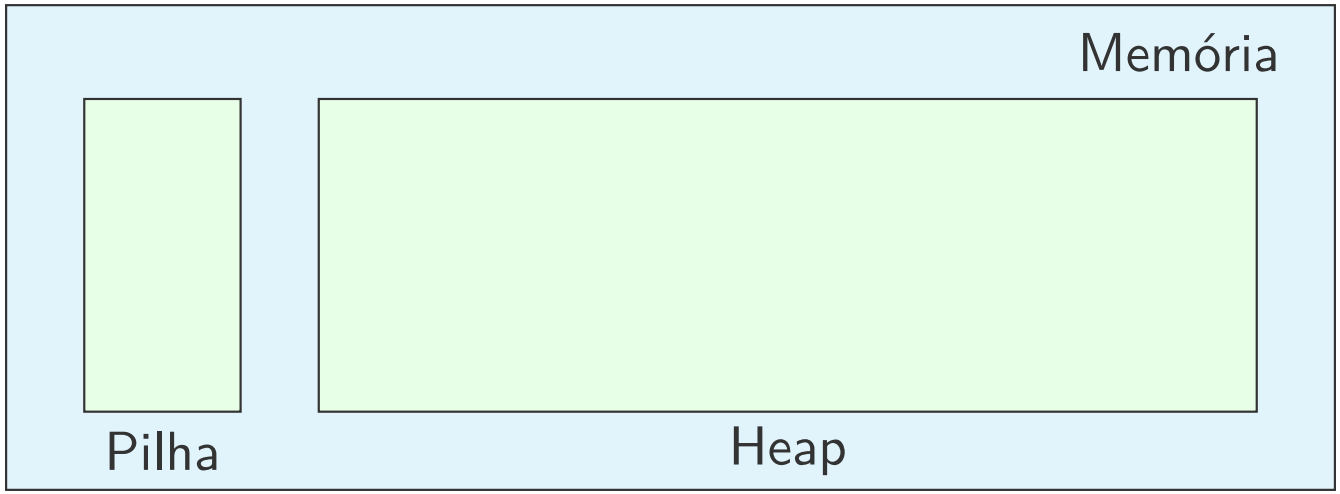
- estão alocados contiguamente na memória
  - pode ser que tenhamos espaço na memória
  - mas não para alocar um vetor do tamanho desejado
- têm um tamanho fixo
  - ou alocamos um vetor pequeno e o espaço pode acabar
  - ou alocamos um vetor grande e desperdiçamos memória
  - Solução: criar lista sequencial redimensionável
- operações de inserção e remoção de elementos são custosas: $O(n)$

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

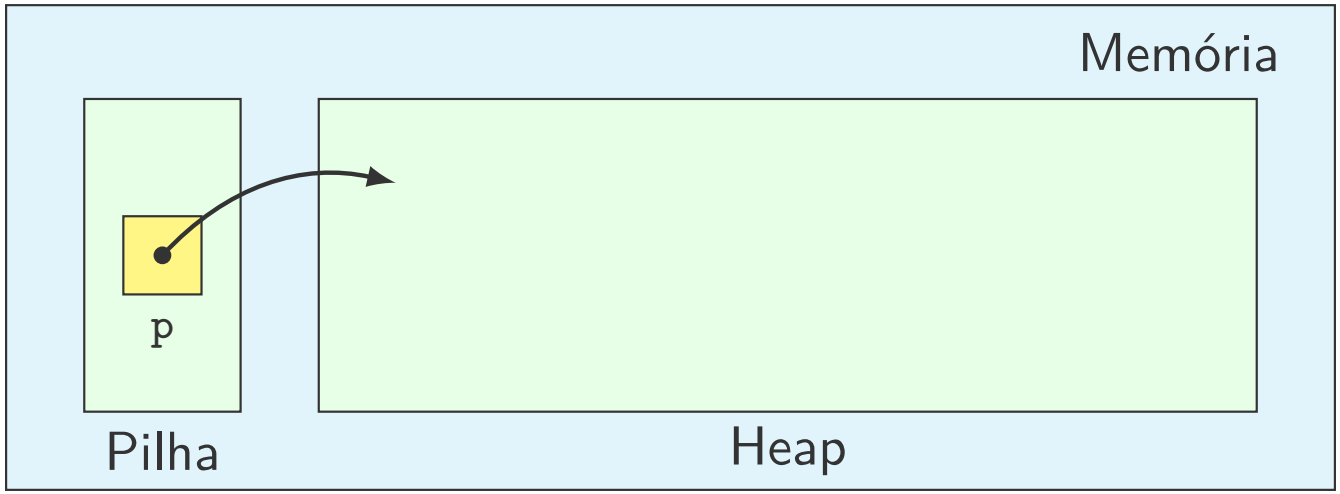
Listas Simplesmente Encadeadas

Alternativa - Lista Simplesmente Encadeada

Alternativa - Lista Simplesmente Encadeada



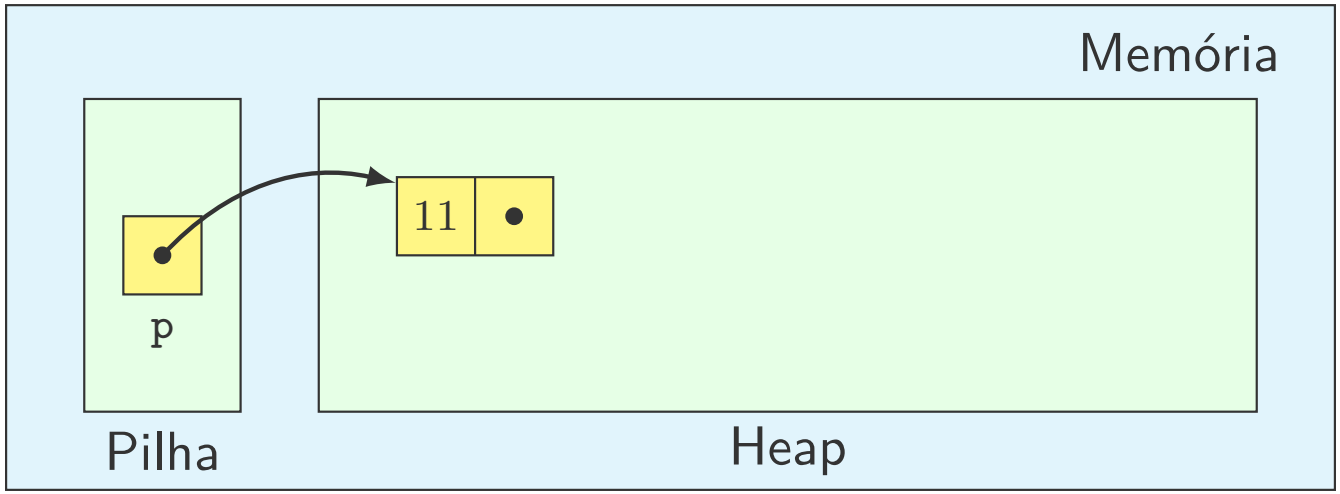
Alternativa - Lista Simplesmente Encadeada



- declaramos um ponteiro para a lista no nosso programa

Alternativa - Lista Simplesmente Encadeada

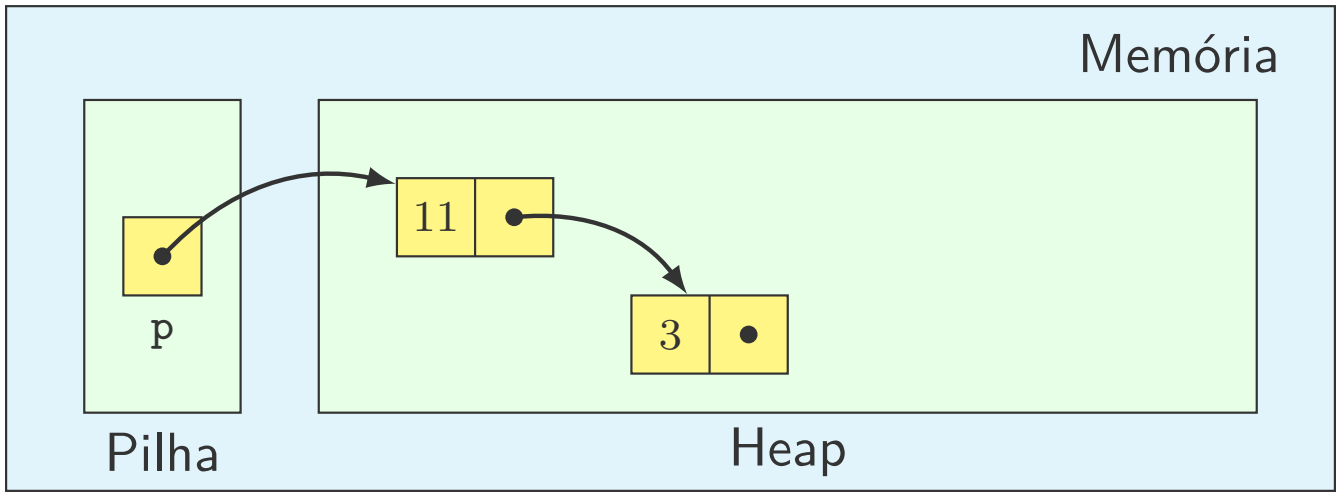
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- declaramos um ponteiro para a lista no nosso programa
- alocamos memória conforme o necessário

Alternativa - Lista Simplesmente Encadeada

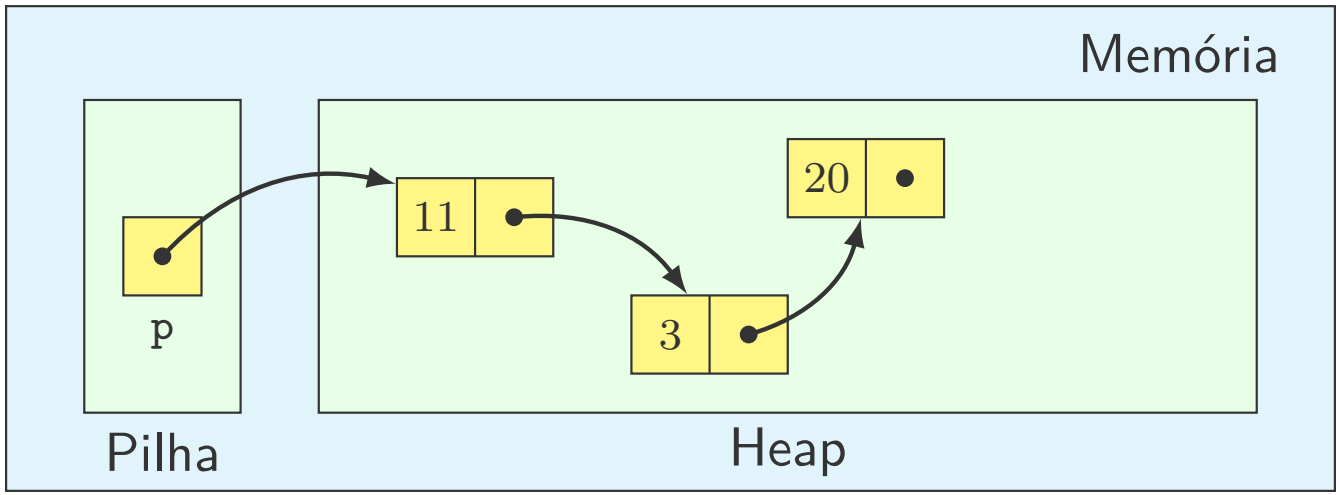
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- declaramos um ponteiro para a lista no nosso programa
- alocamos memória conforme o necessário
- o primeiro nó aponta para o segundo

Alternativa - Lista Simplesmente Encadeada

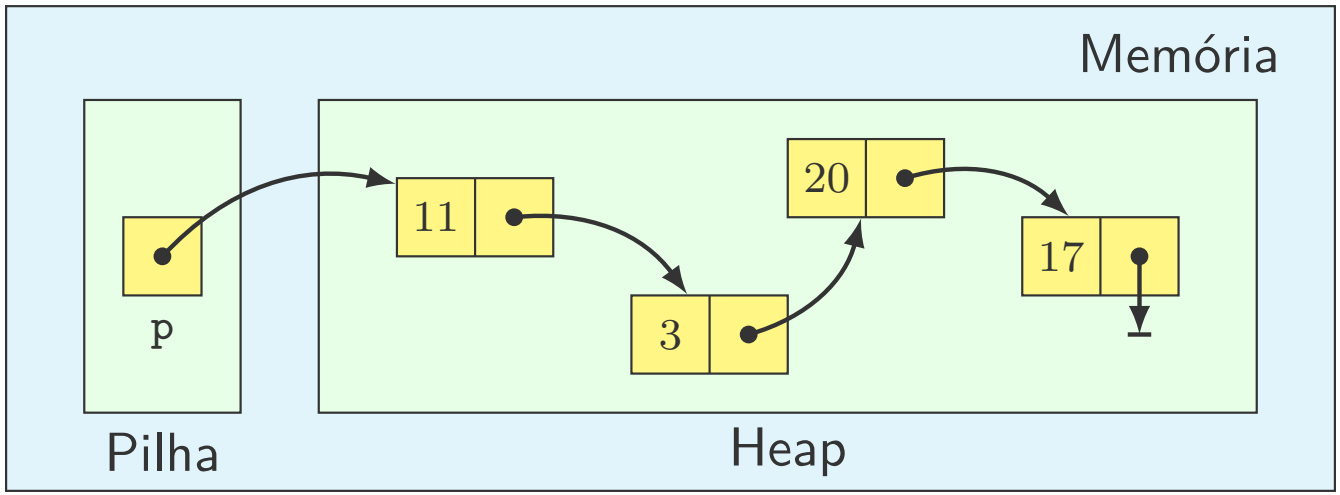
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- declaramos um ponteiro para a lista no nosso programa
- alocamos memória conforme o necessário
- o primeiro nó aponta para o segundo
- o segundo nó aponta para o terceiro

Alternativa - Lista Simplesmente Encadeada

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- declaramos um ponteiro para a lista no nosso programa
- alocamos memória conforme o necessário
- o primeiro nó aponta para o segundo
- o segundo nó aponta para o terceiro
- o último nó aponta para `nullptr`

Lista Simplesmente Encadeada

- O TAD Lista Linear pode ser implementado usando **alocação encadeada** como uma lista simplesmente encadeada.

Lista Simplesmente Encadeada

- O TAD Lista Linear pode ser implementado usando **alocação encadeada** como uma lista simplesmente encadeada.

- A Lista Simplesmente Encadeada contém dois dados:
  - um ponteiro para o primeiro nó (`head`).
  - o número de elementos atualmente na lista (`size`).

Lista Simplesmente Encadeada

- O TAD Lista Linear pode ser implementado usando **alocação encadeada** como uma lista simplesmente encadeada.

- A Lista Simplesmente Encadeada contém dois dados:
  - um ponteiro para o primeiro nó (`head`).
  - o número de elementos atualmente na lista (`size`).

- Operações que podemos querer realizar numa lista:
  - Criar uma nova lista vazia.
  - Deixar a lista vazia.
  - Destruir a lista.
  - Adicionar um elemento em qualquer posição da lista.
  - Remover da lista um elemento em certa posição.
  - Acessar um elemento em uma dada posição.
  - Buscar um elemento.
  - Consultar o tamanho atual da lista.
  - Saber se lista está vazia.
  - Imprimir a lista

Detalhes de Implementação

Listas Encadeadas – Detalhes de Implementação

É formada por um conjunto de objetos chamados nós.

Nó é um elemento alocado dinamicamente que contém:

- o dado armazenado
- um ponteiro para o nó seguinte na lista

Listas Encadeadas – Detalhes de Implementação

É formada por um conjunto de objetos chamados nós.

Nó é um elemento alocado dinamicamente que contém:

- o dado armazenado
- um ponteiro para o nó seguinte na lista

- Um nó pode ser implementado como um `struct` ou como uma `class`.

Listas Encadeadas – Detalhes de Implementação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

É formada por um conjunto de objetos chamados nós.

Nó é um elemento alocado dinamicamente que contém:

- o dado armazenado
- um ponteiro para o nó seguinte na lista

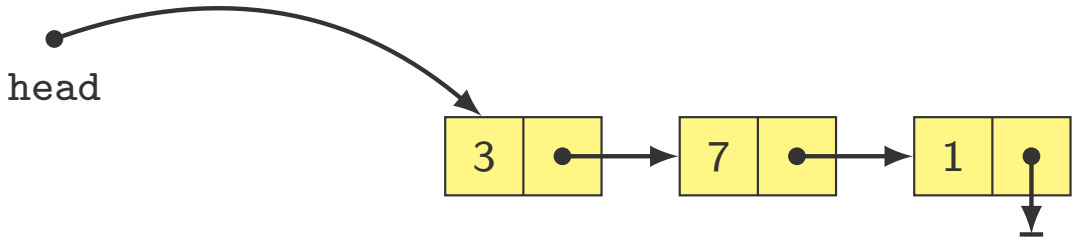
- Um nó pode ser implementado como um `struct` ou como uma `class`.
- Vou implementar como um `struct`

Arquivo Node.h


```
1  #ifndef NODE_H
2  #define NODE_H
3
4  typedef int Item;
5
6  struct Node {
7      Item data;    // data
8      Node *next;   // pointer to the next node
9
10     // Constructor: initializes node's data
11     Node(const Item& k, Node *nextnode) {
12         data = k;
13         next = nextnode;
14     }
15 };
16
17 #endif
```

Listas Encadeadas – Detalhes da Implementação

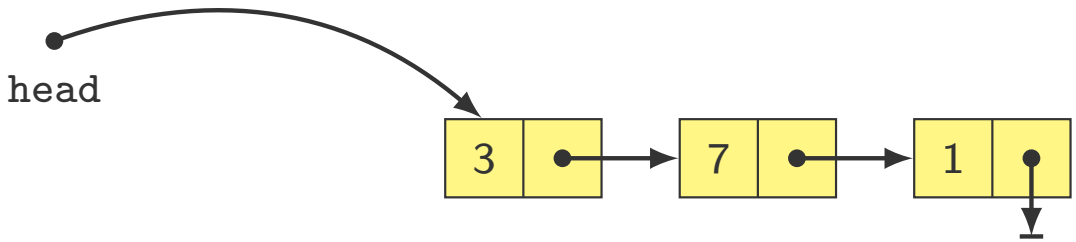
- Conjunto de nós ligados entre si de maneira sequencial



Listas Encadeadas – Detalhes da Implementação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Conjunto de nós ligados entre si de maneira sequencial



Observações:

- a lista encadeada é acessada a partir de um ponteiro (`head`)
- o campo `next` do último nó aponta para `nullptr`
- Assim que a lista é criada, ela está vazia e não tem nenhum elemento. Logo, o ponteiro `head` inicia apontando para `nullptr`.

Arquivo `LinkedList.h`

- Este arquivo contém a declaração da classe `LinkedList`, que contém a lógica da lista simplesmente encadeada discutida anteriormente.

Arquivo LinkedList.h


```
1  #include <iostream>
2  #include "Node.h"
3
4  class LinkedList {
5  private:
6      Node* m_head; // ponteiro para o primeiro elemento
7      int m_size;   // número de elementos na lista
8  public:
9      LinkedList();
10     LinkedList(const LinkedList& l);
11     int size();
12     bool empty();
13     void push_back(const Item& element);
14     Item pop_back();
15     Item& get(int index);
16     std::string toString();
17     void insert(int index, const Item& data);
18     void remove(const Item& element);
19     void removeAt(int index);
20     void clear();
21     ~LinkedList();
22 };
```

Arquivo main.cpp

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #include <iostream>
2  #include "LinkedList.h"
3  using namespace std;
4
5  int main() {
6      LinkedList list; // cria lista vazia
7
8      for(int i = 1; i <= 10; ++i) // insere 1..10
9          list.push_back(i);
10
11     LinkedList list2(list);
12
13     cout << list.toString() << endl; // imprime lista na tela
14
15     cout << list2.toString() << endl; // imprime lista na tela
16
17     for(int i = 0; i < list.size(); ++i)
18         list.get(i) *= 2; // dobra cada valor
19
20     cout << list.toString() << endl; // imprime lista na tela
21 }
```

Exercício

- Implementar as funções-membro da classe LinkedList.

Listas Sequenciais × Encadeadas

Listas sequenciais $\times$ Listas encadeadas

- Acesso à posição k:
  - Vetor: $O(1)$
  - Lista: $O(k)$ (precisa percorrer a lista)

Listas sequenciais $\times$ Listas encadeadas

- Acesso à posição k:
  - Vetor: $O(1)$
  - Lista: $O(k)$ (precisa percorrer a lista)
- Inserção na posição 0:
  - Vetor: $O(n)$ (precisa mover itens para a direita)
  - Lista: $O(1)$

Listas sequenciais $\times$ Listas encadeadas

- Acesso à posição k:
  - Vetor: $O(1)$
  - Lista: $O(k)$ (precisa percorrer a lista)
- Inserção na posição 0:
  - Vetor: $O(n)$ (precisa mover itens para a direita)
  - Lista: $O(1)$
- Remoção da posição 0:
  - Vetor: $O(n)$ (precisa mover itens para a esquerda)
  - Lista: $O(1)$

Listas sequenciais $\times$ Listas encadeadas

- Acesso à posição k:
  - Vetor: $O(1)$
  - Lista: $O(k)$ (precisa percorrer a lista)
- Inserção na posição 0:
  - Vetor: $O(n)$ (precisa mover itens para a direita)
  - Lista: $O(1)$
- Remoção da posição 0:
  - Vetor: $O(n)$ (precisa mover itens para a esquerda)
  - Lista: $O(1)$
- Uso de espaço:
  - Vetor: provavelmente desperdiçará memória
  - Lista: não desperdiça memória, mas cada elemento consome mais memória por causa do ponteiro

Listas sequenciais $\times$ Listas encadeadas

- Acesso à posição k:
  - Vetor: $O(1)$
  - Lista: $O(k)$ (precisa percorrer a lista)
- Inserção na posição 0:
  - Vetor: $O(n)$ (precisa mover itens para a direita)
  - Lista: $O(1)$
- Remoção da posição 0:
  - Vetor: $O(n)$ (precisa mover itens para a esquerda)
  - Lista: $O(1)$
- Uso de espaço:
  - Vetor: provavelmente desperdiçará memória
  - Lista: não desperdiça memória, mas cada elemento consome mais memória por causa do ponteiro

Qual é melhor?

- depende do problema, do algoritmo e da implementação

Exercício 1

Funções Adicionais

Exercício: Implemente as seguintes funções adicionais na LinkedList.

- `LinkedList(int v[], int n)`
  Construtor que recebe um array `v` com n inteiros e inicializa a lista com os n elementos do array `v`.

- `LinkedList(const LinkedList& list)`
  Construtor de cópia, que recebe uma referência para uma LinkedList `list` e inicializa a nova lista com os elementos de `list`.

- `const LinkedList& operator=(const LinkedList& l)`
  Implemente uma versão sobrecarregada do operador de atribuição para a LinkedList. O operador de atribuição permite atribuir uma lista a outra.
  Exemplo: `list2 = list1;`
  Após esta atribuição, `list2` e `list1` são duas listas distintas que possuem **o mesmo** conteúdo.

Funções Adicionais

- `bool equals(const LinkedList& lst)`
  Determina se a lista `lst`, passada por parâmetro, é igual a lista em questão. Duas listas são iguais se têm o mesmo tamanho e o valor do k-ésimo elemento da primeira lista é igual ao k-ésimo valor da segunda.

- `void concat(LinkedList& lst)`
  Concatena a lista atual com a lista `lst`. A lista `lst` não é modificada nessa operação.

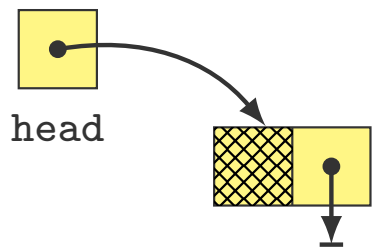
- `void reverse()`: Inverte a ordem dos nós (o primeiro nó passa a ser o último, o segundo passa a ser o penúltimo, etc.) Essa operação faz isso sem criar novos nós, apenas altera os ponteiros.

Exercício 2

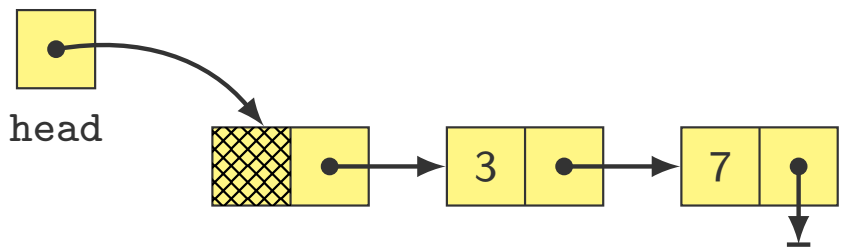
`LinkedList` usando nó sentinela

Exercício: Implemente a lista simplesmente encadeada `LinkedList` usando um nó sentinela, que é um nó auxiliar (sem conteúdo de valor) que serve apenas para marcar o início da lista.

- O ponteiro `head` **sempre** aponta para o nó sentinela.
- Quando a lista está vazia, o nó sentinela é o único nó na lista e seu campo `next` aponta para `nullptr`.



Lista vazia



Lista com 2 elementos

- Neste contexto, **reimplemente todas as operações vistas nesta aula**.

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM