Variações de Listas Encadeadas

Estrutura de Dados — QXD0010

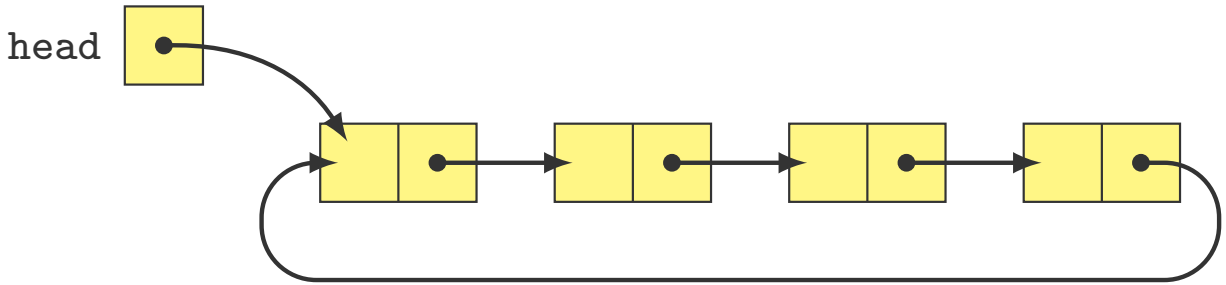
Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará
Campus Quixadá

2º semestre/2023

1. Lista simplesmente encadeada circular

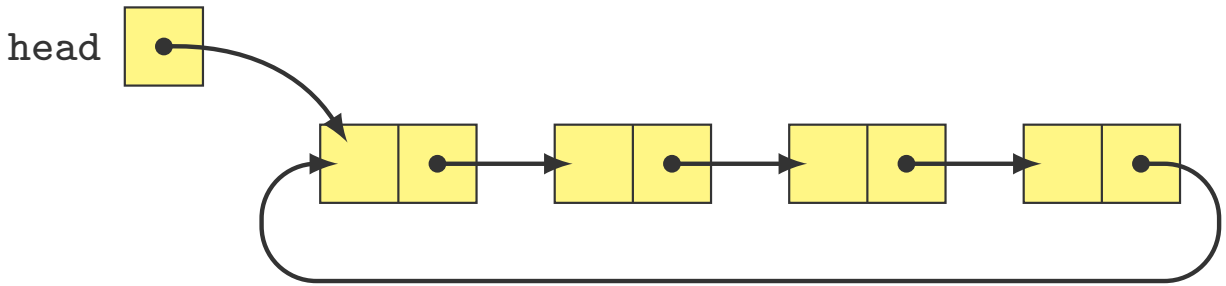
Lista circular (sem nó cabeça):



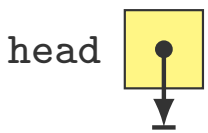
1. Lista simplesmente encadeada circular

Universidade Federal do Ceará
Campus Quixadá

Lista circular (sem nó cabeça):

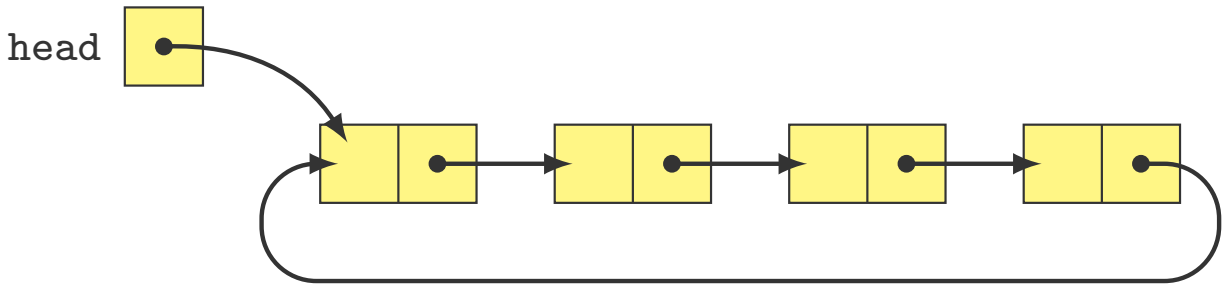


Lista circular vazia: ponteiro `head` é nulo.

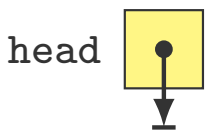


1. Lista simplesmente encadeada circular

Lista circular (sem nó cabeça):



Lista circular vazia: ponteiro `head` é nulo.

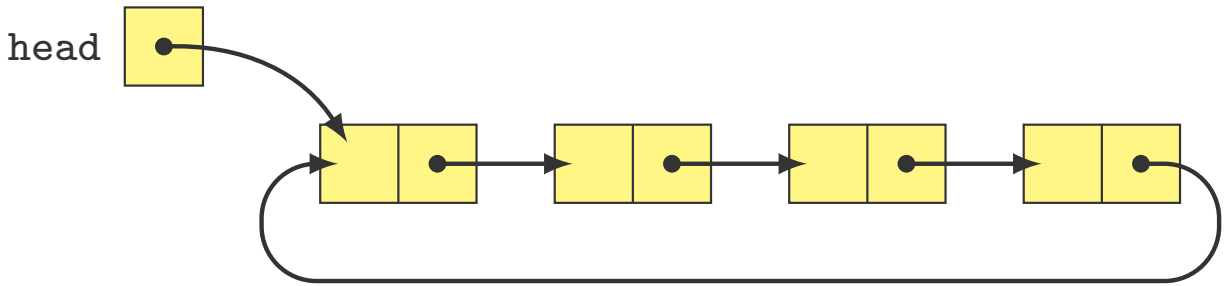


Exemplo de aplicações:

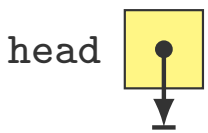
1. Lista simplesmente encadeada circular

Universidade Federal do Ceará
Campus Quixadá

Lista circular (sem nó cabeça):



Lista circular vazia: ponteiro `head` é nulo.



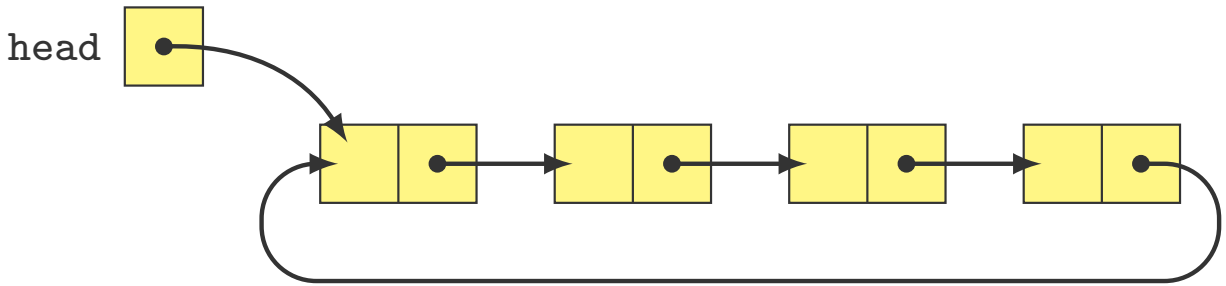
Exemplo de aplicações:

- Execução de processos no sistema operacional

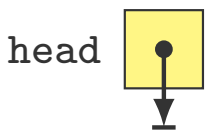
1. Lista simplesmente encadeada circular

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Lista circular (sem nó cabeça):



Lista circular vazia: ponteiro `head` é nulo.

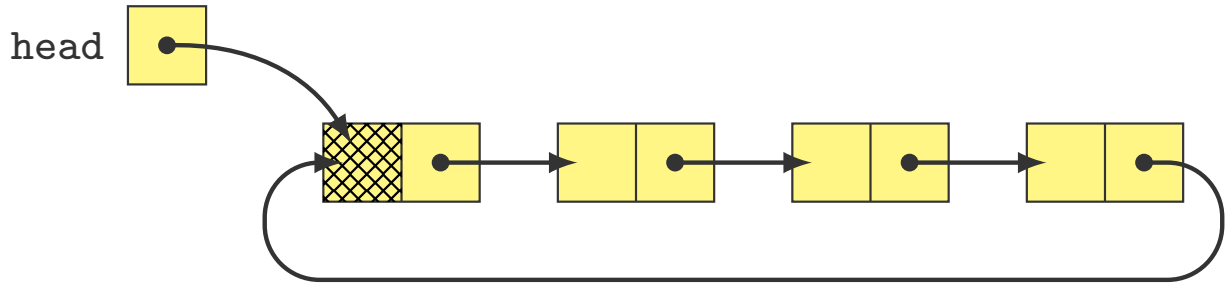


Exemplo de aplicações:

- Execução de processos no sistema operacional
- Controlar de quem é a vez em um jogo de tabuleiro

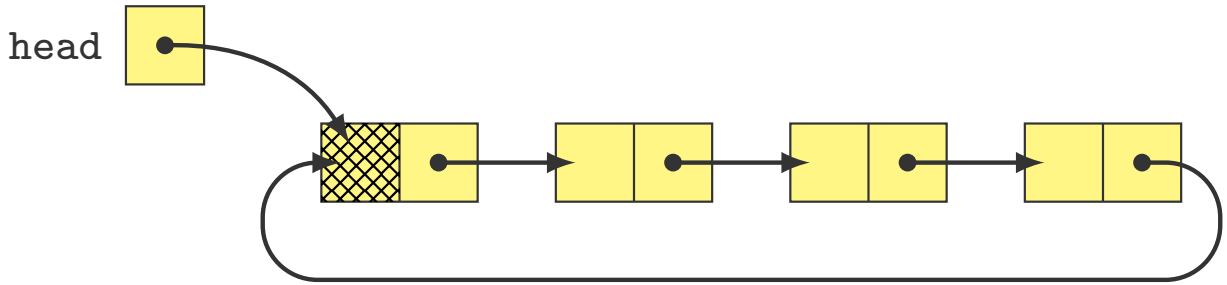
2. Lista circular com nó sentinela

Lista simplesmente encadeada circular com nó sentinela:

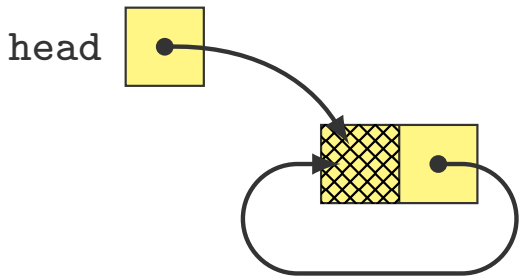


2. Lista circular com nó sentinela

Lista simplesmente encadeada circular com nó sentinela:

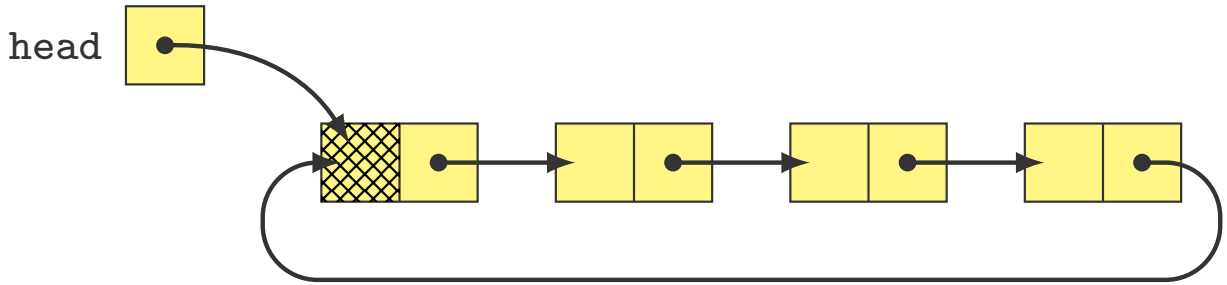


Lista circular vazia com nó sentinela:

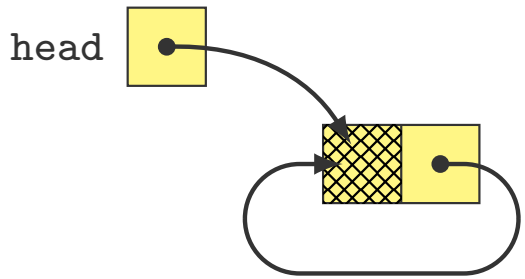


2. Lista circular com nó sentinela

Lista simplesmente encadeada circular com nó sentinela:



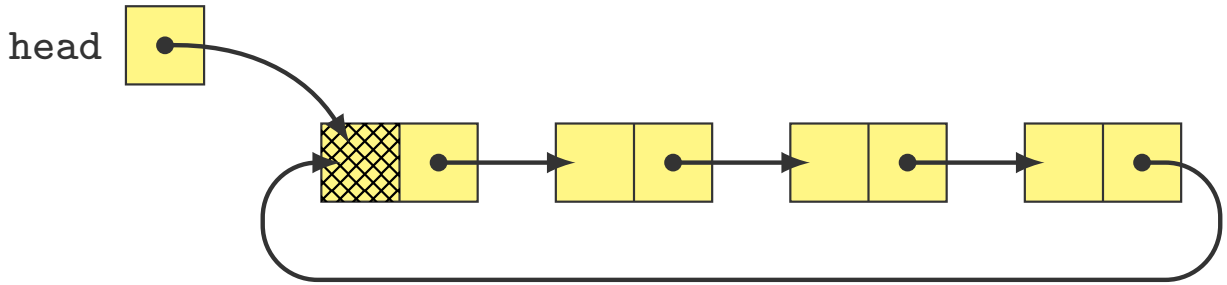
Lista circular vazia com nó sentinela:



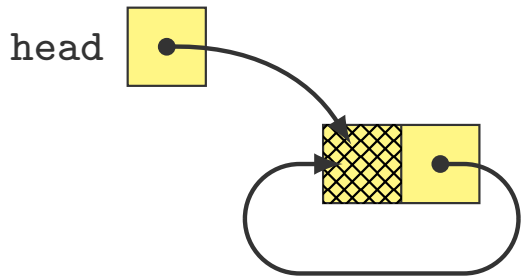
Diferenças para a versão sem nó sentinela:

2. Lista circular com nó sentinela

Lista simplesmente encadeada circular com nó sentinela:



Lista circular vazia com nó sentinela:

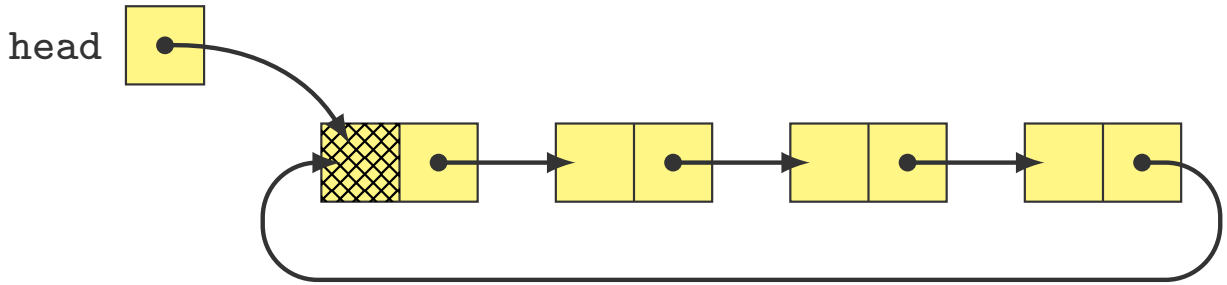


Diferenças para a versão sem nó sentinela:

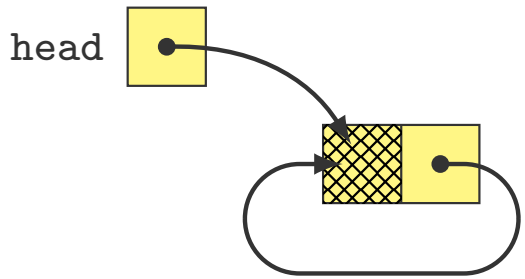
- ponteiro `head` sempre aponta para nó sentinela

2. Lista circular com nó sentinela

Lista simplesmente encadeada circular com nó sentinela:



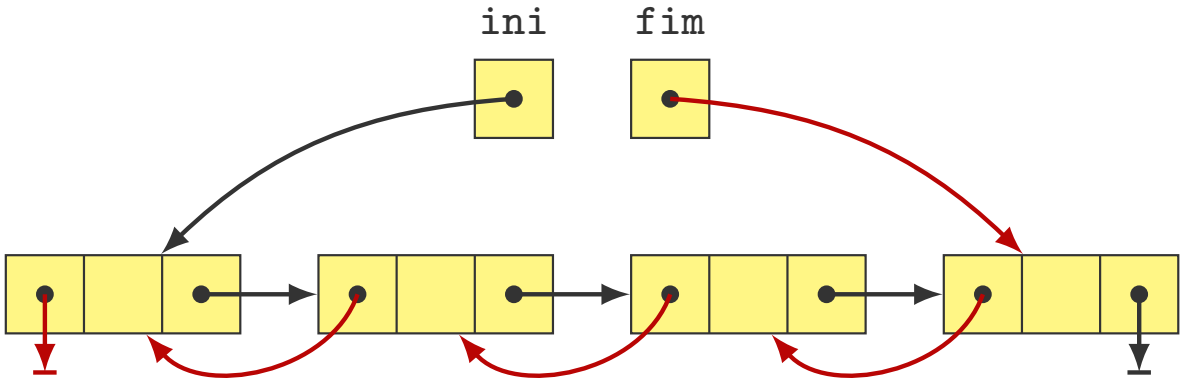
Lista circular vazia com nó sentinela:



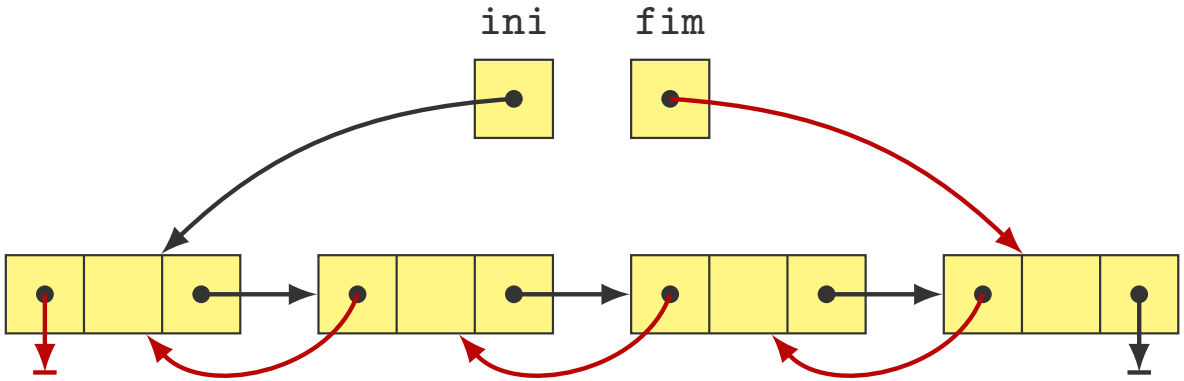
Diferenças para a versão sem nó sentinela:

- ponteiro `head` sempre aponta para nó sentinela
- código de inserção e de remoção mais simples

3. Lista duplamente encadeada



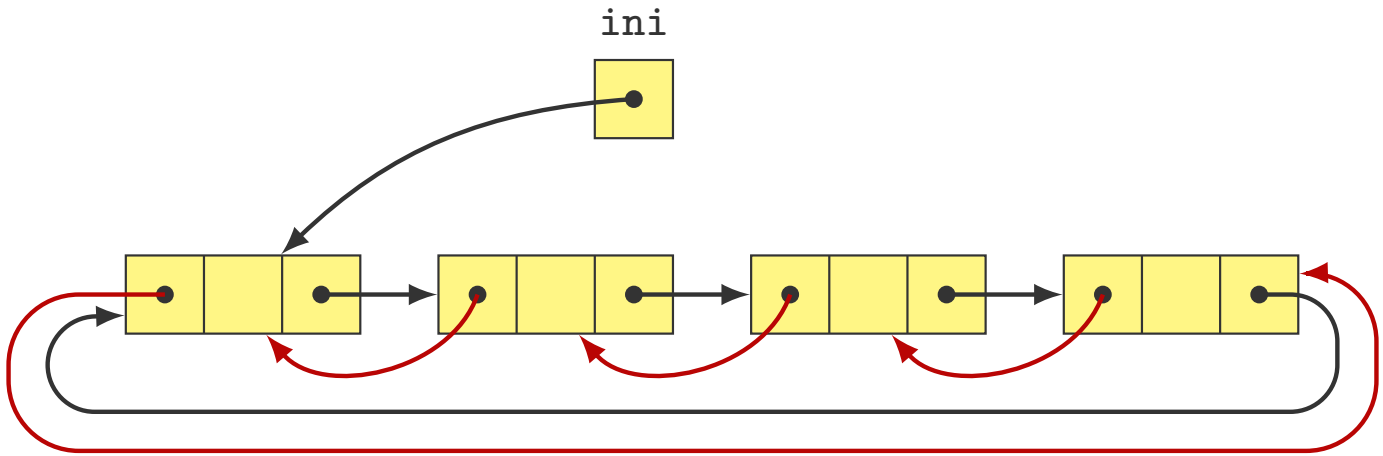
3. Lista duplamente encadeada



- Cada nó tem um ponteiro para o próximo nó e para o nó anterior.
- Se tivermos um ponteiro para o último elemento da lista, podemos percorrer a lista em ordem reversa.

4. Lista duplamente encadeada circular

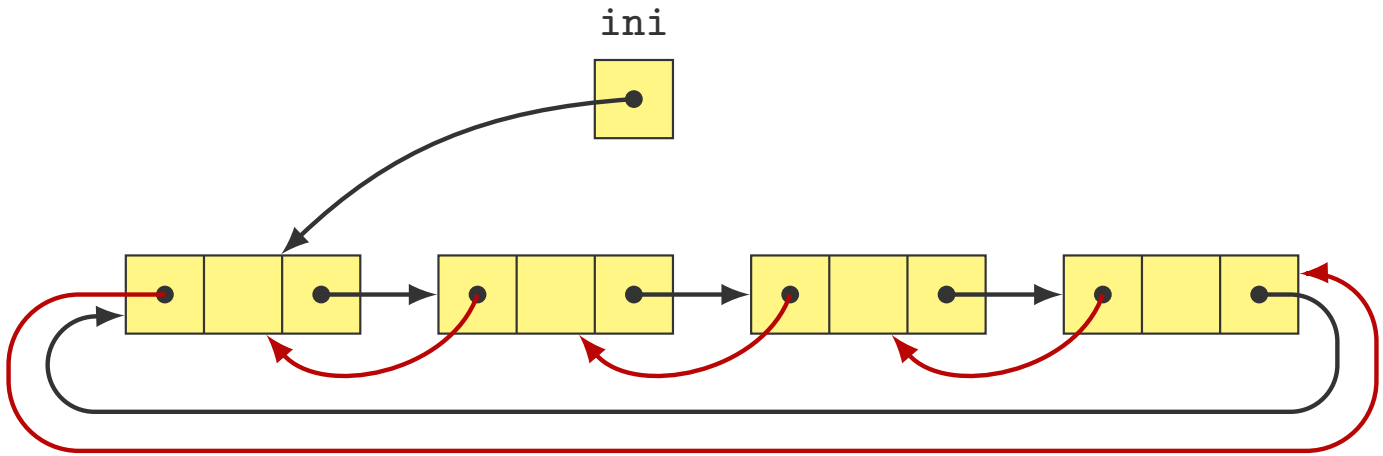
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- Diferença desta para a anterior: o primeiro nó aponta para o último e o último nó aponta para o primeiro.

4. Lista duplamente encadeada circular

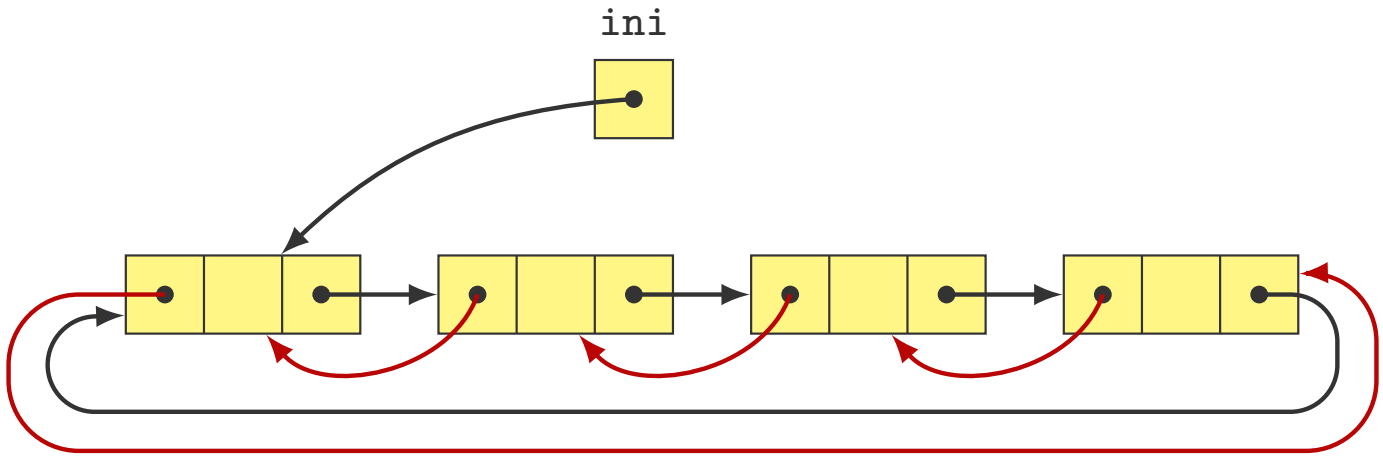
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- Diferença desta para a anterior: o primeiro nó aponta para o último e o último nó aponta para o primeiro.

- Vantagens: perimite percorrer a lista em duas direções e permite inserção e remoção em tempo $O(1)$, uma vez que você tenha um ponteiro para o nó em questão

4. Lista duplamente encadeada circular



- Diferença desta para a anterior: o primeiro nó aponta para o último e o último nó aponta para o primeiro.

- Vantagens: perimite percorrer a lista em duas direções e permite inserção e remoção em tempo $O(1)$, uma vez que você tenha um ponteiro para o nó em questão

- Essa lista também pode ser implementada com nó cabeça.

Exercícios

Exercícios

- Implemente uma lista duplamente encadeada com as operações:
  - inserir nó
  - remover nó
  - saber se há nó com dado valor
  - tamanho da lista
  - concatenar duas listas
  - imprimir lista de frente para trás ou reversamente

- Implemente uma lista circular duplamente encadeada com as operações:
  - inserir nó
  - remover nó
  - saber se há nó com dado valor
  - tamanho da lista
  - concatenar duas listas
  - imprimir lista de frente para trás ou reversamente

Iterador para a classe List

Iterador

- Na classe `List` não sobrecarregamos o operador de indexação pois esse operador deixa os algoritmos em listas encadeadas ineficientes.
  - percorrer a lista consumiria $O(n^2)$ passos.

Iterador

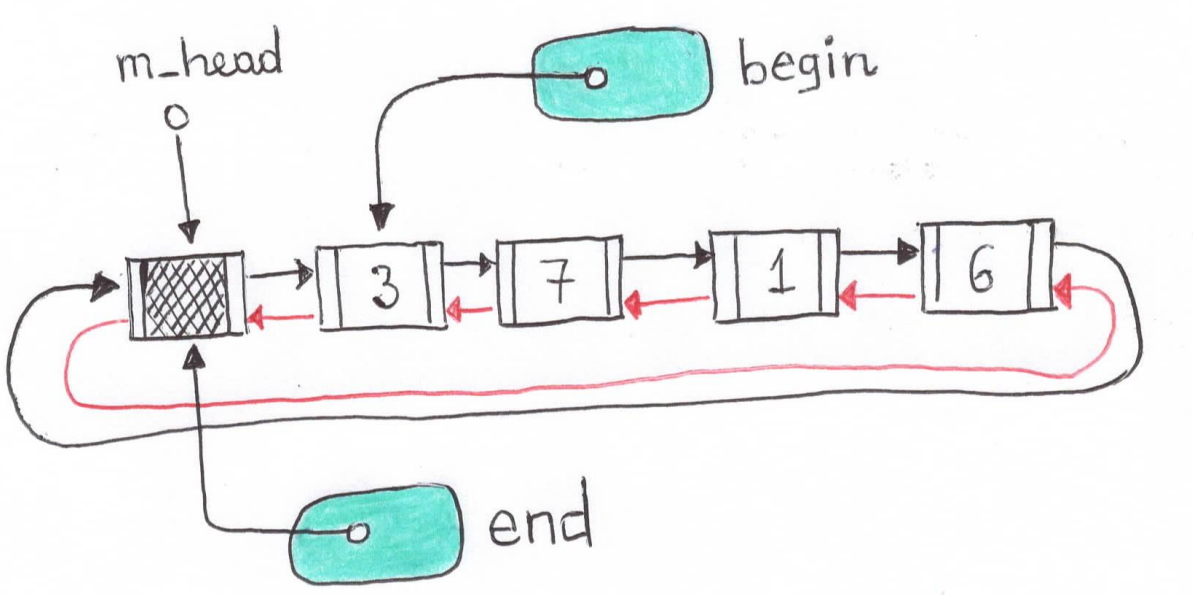
- Na classe `List` não sobrecarregamos o operador de indexação pois esse operador deixa os algoritmos em listas encadeadas ineficientes.
  - percorrer a lista consumiria $O(n^2)$ passos.

- Uma solução eficiente para ter acesso individual aos elementos da lista é por meio de um iterador.
  - percorrer a lista com um iterador consome $O(n)$ passos.

Iterador

- Na classe `List` não sobrecarregamos o operador de indexação pois esse operador deixa os algoritmos em listas encadeadas ineficientes.
  - percorrer a lista consumiria $O(n^2)$ passos.
- Uma solução eficiente para ter acesso individual aos elementos da lista é por meio de um iterador.
  - percorrer a lista com um iterador consome $O(n)$ passos.
- Precisamos implementar um iterador para a nossa `List`!

Iterador



Esquema de uma lista encadeada e dois iteradores

- A classe `List` fornecerá duas funções-membro públicas que retornam iteradores para duas posições distintas da lista:
  - `list.begin()`: iterador para o primeiro elemento
  - `list.end()`: iterador para após o último elemento

Iterador

- Nosso iterador pertencerá à categoria `Bidirectional Iterator`,
  - permite avançar para frente e voltar atrás; permite acessar o valor do elemento apontado e fazer alterações nele.

Iterador

- Nosso iterador pertencerá à categoria `Bidirectional Iterator`,
  - permite avançar para frente e voltar atrás; permite acessar o valor do elemento apontado e fazer alterações nele.

- As operações suportadas pelo nosso iterador serão:
  `++`, `*`, `==`, `!=`, `--`

Iterador

- Nosso iterador pertencerá à categoria `Bidirectional Iterator`,
  - permite avançar para frente e voltar atrás; permite acessar o valor do elemento apontado e fazer alterações nele.

- As operações suportadas pelo nosso iterador serão:
  `++`, `*`, `==`, `!=`, `--`

- Esse tipo de iterador sobrecarrega os operadores:
  - `operator++()`
  - `operator++(int)`
  - `operator*()`
  - `operator==()`
  - `operator!=()`
  - `operator--()`
  - `operator--(int)`

Exercício

- Implementar uma classe chamada `iterator_list`, que implementa a lógica de um bidirectional iterator para a nossa lista duplamente encadeada.

Funções de Inserção e Remoção

Inserção e Remoção em Listas Encadeadas

- É muito comum querer inserir ou remover um elemento em algum ponto da lista que não seja o seu início ou o seu fim.

Inserção e Remoção em Listas Encadeadas

- É muito comum querer inserir ou remover um elemento em algum ponto da lista que não seja o seu início ou o seu fim.

- Por questão de eficiência, listas encadeadas não fornecem o operador de indexação.
  - Logo não é possível acessar um nó pelo seu índice. Na verdade, não existem índices numa lista encadeada.

Inserção e Remoção em Listas Encadeadas

- É muito comum querer inserir ou remover um elemento em algum ponto da lista que não seja o seu início ou o seu fim.

- Por questão de eficiência, listas encadeadas não fornecem o operador de indexação.
  - Logo não é possível acessar um nó pelo seu índice. Na verdade, não existem índices numa lista encadeada.

- Assim, as listas do C++ não fornecem a possibilidade de inserir ou remover usando indexação.
  - Nas listas da STL do C++, para inserir ou remover, é necessário usar um iterador.

Inserção em Lista Duplamente Encadeada

- Numa lista duplamente encadeada, se quisermos inserir um elemento, precisamos ter um iterador.

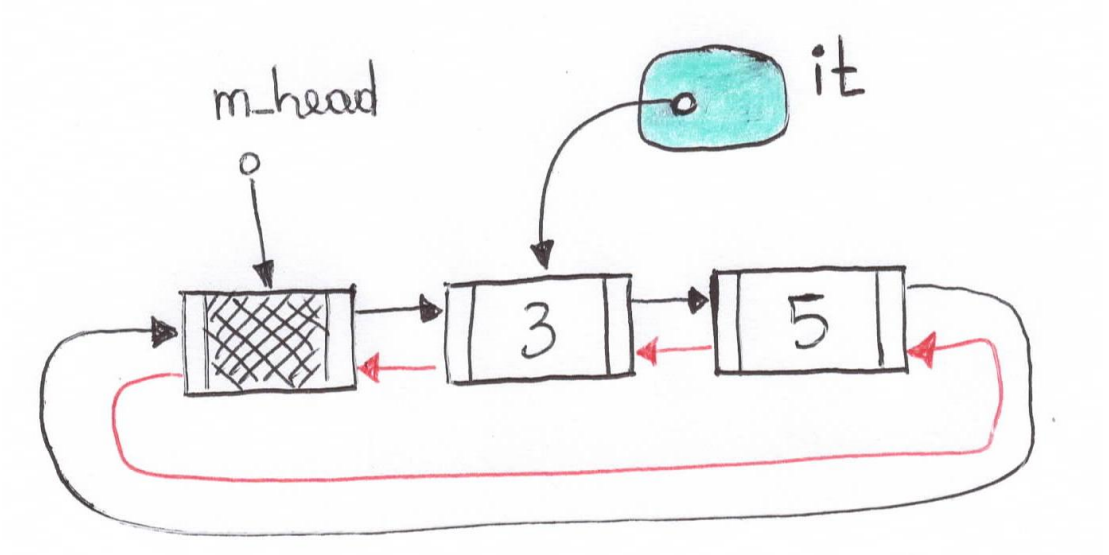
- O novo elemento será inserido antes do nó apontado pelo iterador.

Inserção em Lista Duplamente Encadeada

- Numa lista duplamente encadeada, se quisermos inserir um elemento, precisamos ter um iterador.

- O novo elemento será inserido antes do nó apontado pelo iterador.

- Exemplo: Inserir o nó 2 antes do nó 3.



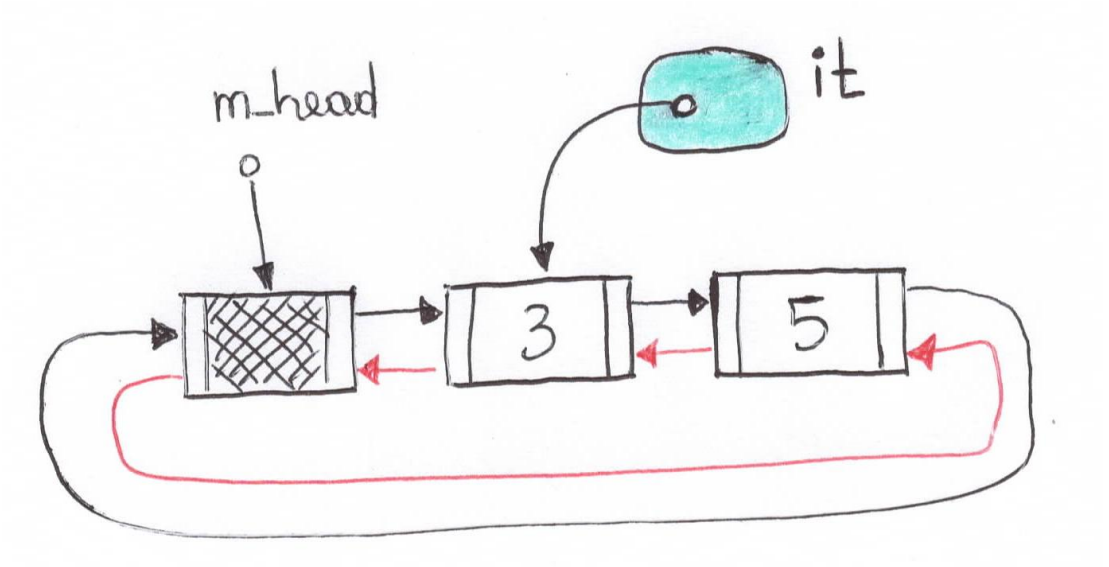
Remoção em Lista Duplamente Encadeada

- Numa lista duplamente encadeada, se quisermos remover um nó, precisamos ter um iterador apontando para ele.

Remoção em Lista Duplamente Encadeada

- Numa lista duplamente encadeada, se quisermos remover um nó, precisamos ter um iterador apontando para ele.

- Exemplo: Remover o nó com valor 3.



Como inserir e/ou remover numa lista??

- Até agora vimos apenas como inserir ou remover no início ou no final de uma lista encadeada.

- Atenção: Precisamos de operações que perimitam inserir e/ou remover em qualquer posição da lista!!

Exercício

Exercício: Implementar as novas funções abaixo:

- `iterator insert(iterator position, const int& val)`
  Insere um novo elemento antes do iterador position. Retorna um iterador apontando para o elemento recém-inserido.

Exercício

Exercício: Implementar as novas funções abaixo:

- `iterator insert(iterator position, const int& val)`
  Insere um novo elemento antes do iterador position. Retorna um iterador apontando para o elemento recém-inserido.

- `iterator erase(iterator position)`
  Remove o elemento apontado pelo iterador position.
  Retorna um iterador apontando para o elemento logo após o elemento que foi removido. Se a função apagou o último elemento, o valor retornado é o iterador `end()`.

Exercício

- `iterator insert(iterator position, unsigned n, const int& val)`
  Insere um valor `val` um número `n` de vezes após o iterador `position`.
  Retorna um iterador apontando para o primeiro dos elementos recém-inseridos.

Exercício

- `iterator insert(iterator position, unsigned n, const int& val)`
  Insere um valor `val` um número `n` de vezes após o iterador `position`.
  Retorna um iterador apontando para o primeiro dos elementos recém-inseridos.

- `iterator insert(iterator position, std::initializer_list<int> l)`
  Insere uma lista de elementos após o iterador position.
  Retorna um iterador apontando para o primeiro dos elementos recém-inseridos

Exercícios Adicionais

Implementar as seguintes funções adicionais:

- inserir no final da fila
- remover do final da fila
- consultar elemento no final da fila
- consultar elemento no início da fila
- criar construtor que recebe uma lista inicializadora (std::initializer_list)
- criar versões `const` das funções begin() e end() e implementar uma versão nova do iterador que possa ser utilizada neste caso.

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM