Listas Simplesmente Encadeadas

Estrutura de Dados — QXD0010

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

Universidade Federal do Ceará
Campus Quixadá

1º semestre/2021

Introdução

Relembrando: 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:
  - Se $n > 0$, $L[0]$ é o primeiro nó,
  - Para $0 < k \leq n-1$, o nó $L[k]$ é precedido por $L[k-1]$.

Tipos de alocação

O tipo de armazenamento de uma lista linear pode ser classificado de acordo com a posição relativa na memória de dois nós consecutivos na lista.

- **Alocação sequencial:** dois nós consecutivos na lista estão em posições contíguas de memória.

- **Alocação encadeada:** dois nós consecutivos na lista podem estar em posições não contíguas da memória.

Vetores (algumas considerações)

Vetores:

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

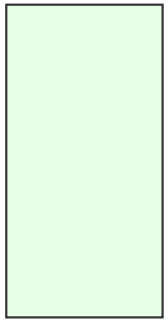
Vetores (algumas considerações)

Vetores:

- estão alocados contiguamente na memória
  - pode ser que tenhamos espaço na memória
  - mas não para alocar um vetor do tamanho desejado
- tem um tamanho fixo
  - ou alocamos um vetor pequeno e o espaço pode acabar
  - ou alocamos um vetor grande e desperdiçamos memória

Listas Encadeadas

Alternativa - Lista Simplesmente Encadeada

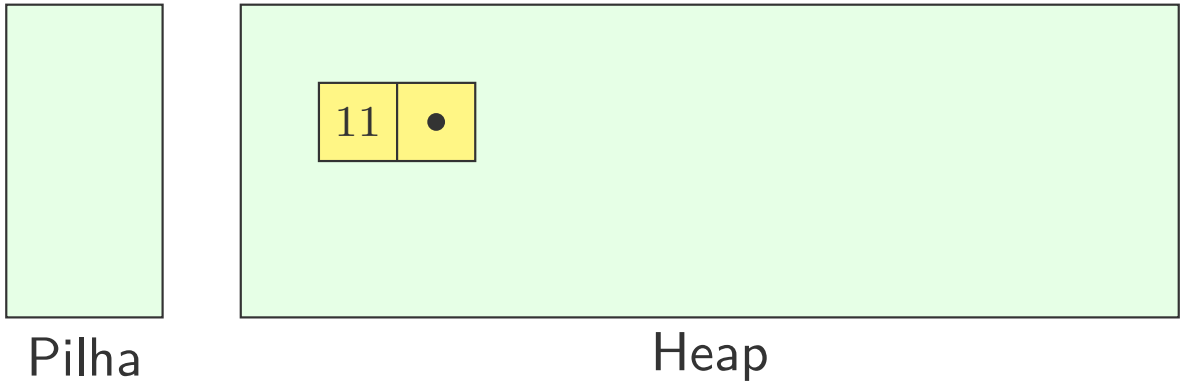


Pilha

Alternativa - Lista Simplesmente Encadeada

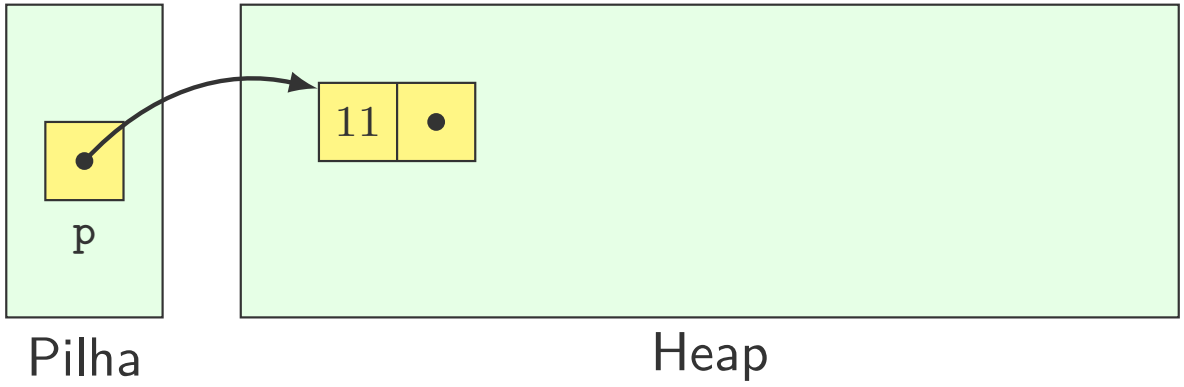
Alternativa - Lista Simplesmente Encadeada

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



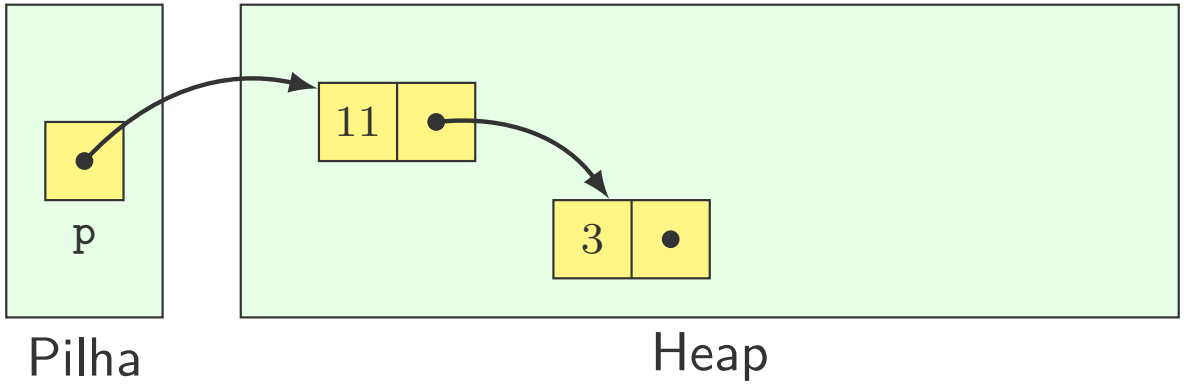
- alocamos memória conforme o necessário

Alternativa - Lista Simplesmente Encadeada



- alocamos memória conforme o necessário
- guardamos um ponteiro para a estrutura em uma variável

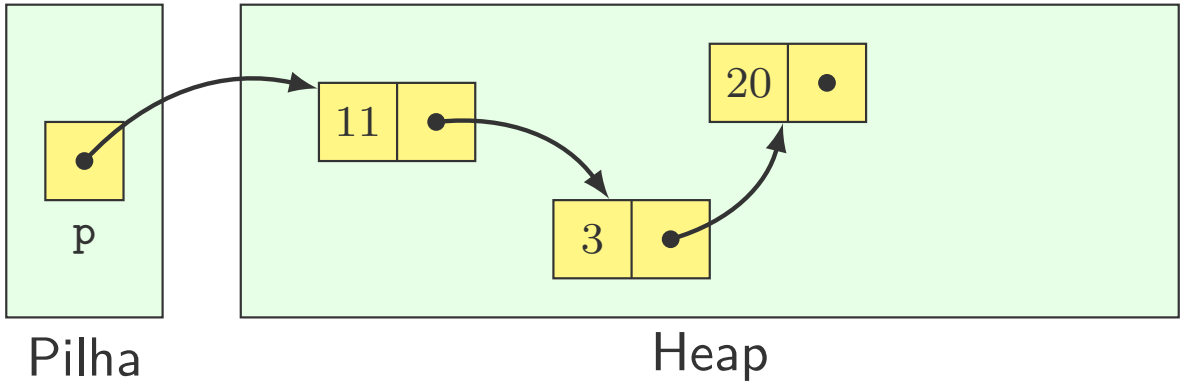
Alternativa - Lista Simplesmente Encadeada



- alocamos memória conforme o necessário
- guardamos um ponteiro para a estrutura em uma variável
- o primeiro nó aponta para o segundo

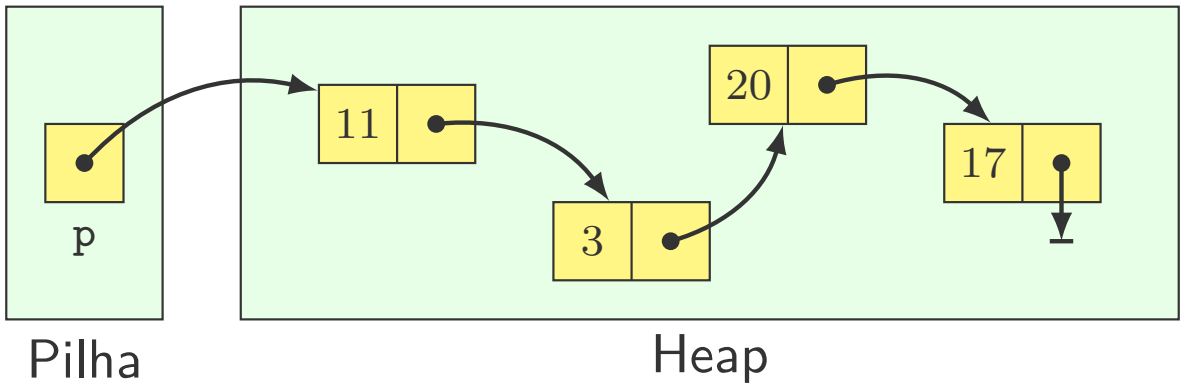
Alternativa - Lista Simplesmente Encadeada

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- alocamos memória conforme o necessário
- guardamos um ponteiro para a estrutura em uma variável
- o primeiro nó aponta para o segundo
- o segundo nó aponta para o terceiro

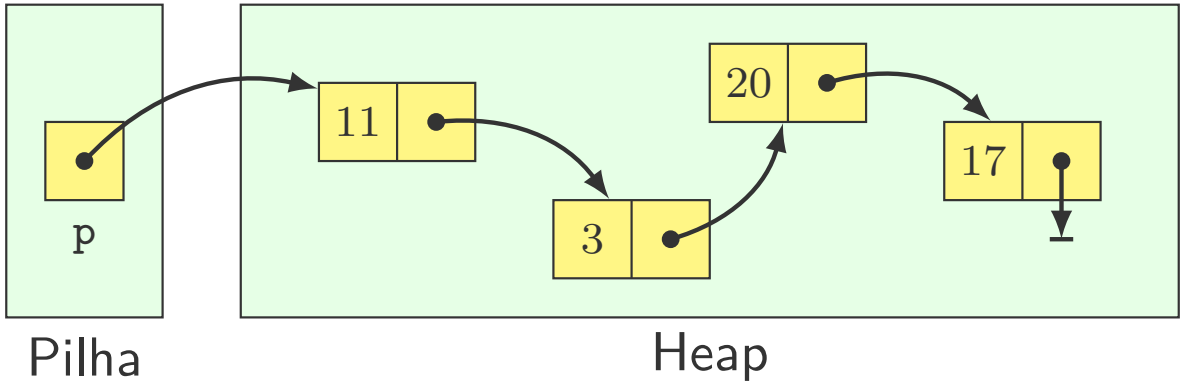
Alternativa - Lista Simplesmente Encadeada



- alocamos memória conforme o necessário
- guardamos um ponteiro para a estrutura em uma variável
- o primeiro nó aponta para o segundo
- o segundo nó aponta para o terceiro

Alternativa - Lista Simplesmente Encadeada

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- alocamos memória conforme o necessário
- guardamos um ponteiro para a estrutura em uma variável
- 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 tem os seguintes dados:
  - um ponteiro para o primeiro nó.
  - um ponteiro para o último nó.
  - um ponteiro para marcar o elemento atual.
  - o tamanho da lista.

Lista Simplesmente Encadeada

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

- A Lista Simplesmente Encadeada tem os seguintes dados:
  - um ponteiro para o primeiro nó.
  - um ponteiro para o último nó.
  - um ponteiro para marcar o elemento atual.
  - o tamanho da lista.

- Operações possíveis são:
  - Criar lista.
  - Liberar lista.
  - Adicionar um elemento na lista.
  - Remover um elemento da lista.
  - Buscar um elemento na lista.
  - Consultar o tamanho atual da lista.
  - Saber se lista está vazia.

Detalhes de Implementação

Listas Encadeadas – Detalhes de Implementação

- É formada por um conjunto de objetos chamados nós.

Nó: elemento alocado dinamicamente que contém:

- um conjunto de dados
- um ponteiro para o nó seguinte na lista

Listas Encadeadas – Detalhes de Implementação

- É formada por um conjunto de objetos chamados nós.

Nó: elemento alocado dinamicamente que contém:

- um conjunto de dados
- um ponteiro para o nó seguinte na lista

- Um nó pode ser implementado com 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: elemento alocado dinamicamente que contém:

- um conjunto de dados
- um ponteiro para o nó seguinte na lista

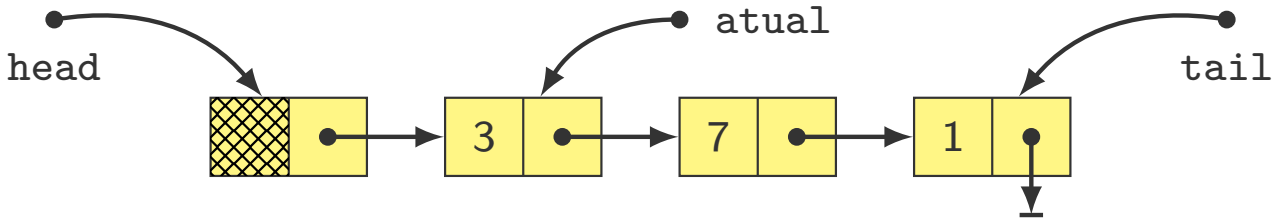
- Um nó pode ser implementado com um `struct` ou como uma `class`.
- Vamos implementar como um `struct`

```
1 struct Node {
2     int key;
3     Node *next;
4
5     Node(const int& k = 0, Node *nextval = nullptr) {
6         key = k;
7         next = nextval;
8     }
9 };
```

Listas Encadeadas – Detalhes da Implementação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

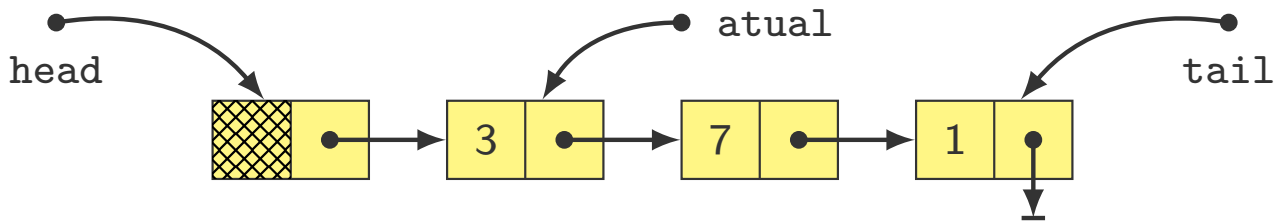
- Conjunto de nós ligados entre si de maneira sequencial



Listas Encadeadas – Detalhes da Implementação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Conjunto de nós ligados entre si de maneira sequencial

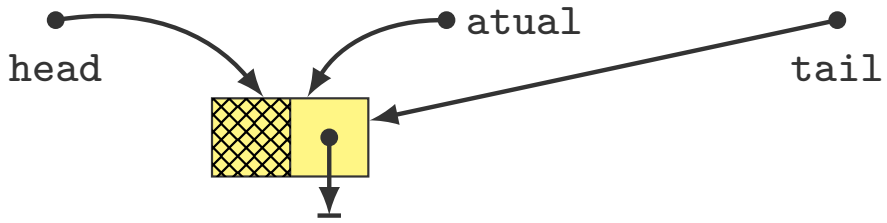


Observações:

- a lista encadeada é acessada a partir de um ponteiro (head)
- o primeiro nó da lista é um nó auxiliar que serve apenas para marcar o início da lista.
- para facilitar inserções no final da lista, incluímos um ponteiro para o último elemento da lista (tail)
- a lista possui um ponteiro apontando para o nó atual da lista (atual)
- o campo `next` do último nó aponta para `nullptr`

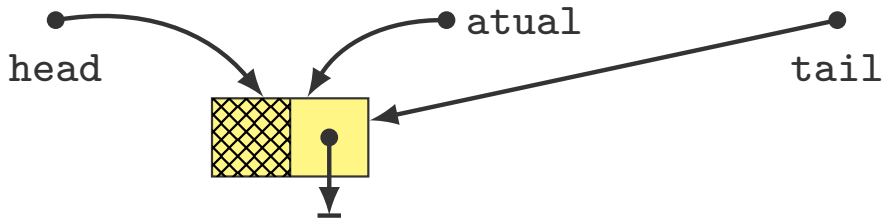
Listas Encadeadas – Detalhes da Implementação

- Estado inicial de uma lista simplesmente encadeada que possui um nó cabeça auxiliar



Listas Encadeadas – Detalhes da Implementação

- Estado inicial de uma lista simplesmente encadeada que possui um nó cabeça auxiliar

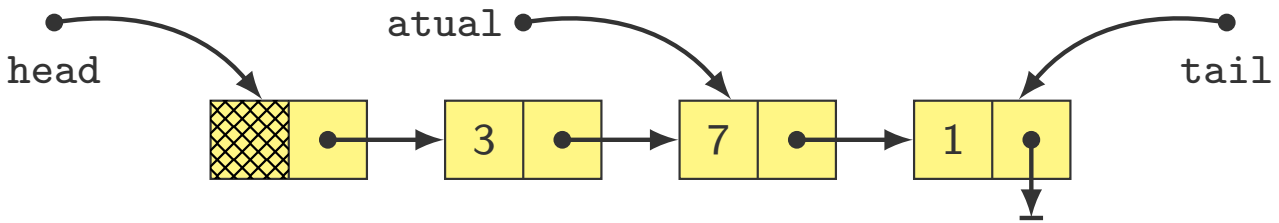


Observações:

- usamos o nó cabeça para facilitar a codificação de algumas funções, como a inserção e a remoção.
- o uso do nó cabeça também evita termos que considerar casos especiais para quando a lista está vazia ou quando a posição atual está no início da lista.

Detalhes da Implementação – Posição atual

- Nossa lista encadeada possuirá como atributo um ponteiro para marcar o elemento atual.
- Abaixo, o elemento atual é o 7 e ele está na posição 1 da lista.
- Assim como na lista sequencial, aqui os nós são indexados do 0 ao $n - 1$.

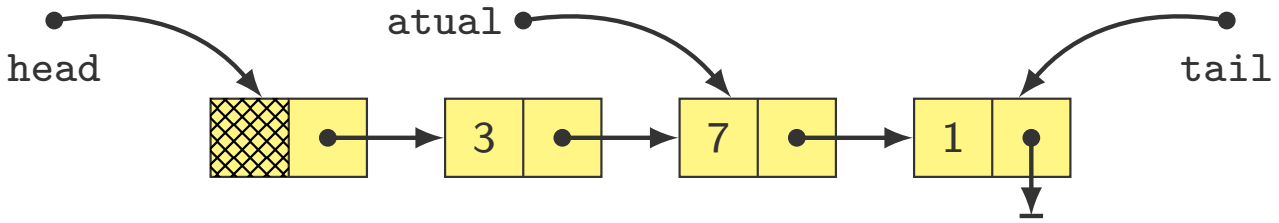


- Pergunta: Se quisermos inserir um elemento na posição atual? Como fazer? Qual a complexidade do algoritmo?

Detalhes da Implementação – Posição atual

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

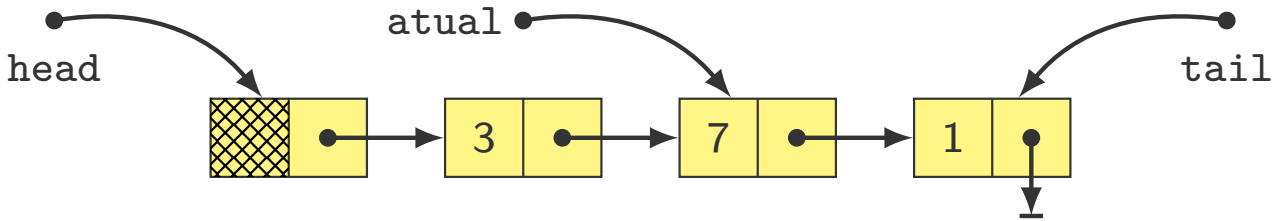
- Nossa lista encadeada possuirá como atributo um ponteiro para marcar o elemento atual.
- Abaixo, o elemento atual é o 7 e ele está na posição 1 da lista.
- Assim como na lista sequencial, aqui os nós são indexados do 0 ao $n - 1$.



- Pergunta: Se quisermos inserir um elemento na posição atual? Como fazer? Qual a complexidade do algoritmo?
- Podemos fazer melhor do que está acima?

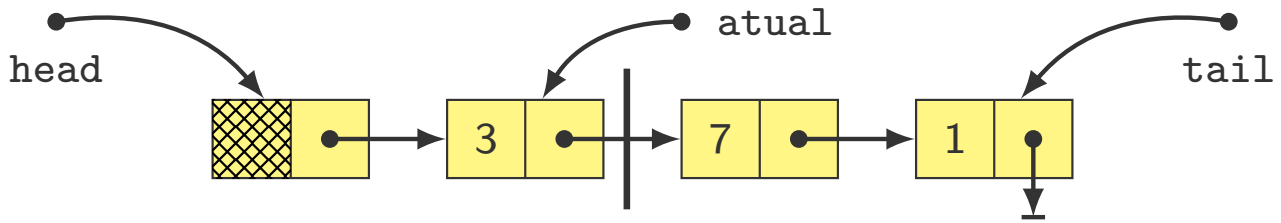
Detalhes da Implementação – Posição atual

- Nossa lista encadeada possuirá como atributo um ponteiro para marcar o elemento atual.
- Abaixo, o elemento atual é o 7 e ele está na posição 1 da lista.
- Assim como na lista sequencial, aqui os nós são indexados do 0 ao $n - 1$.



- Pergunta: Se quisermos inserir um elemento na posição atual? Como fazer? Qual a complexidade do algoritmo?
- Podemos fazer melhor do que está acima?
  Resposta: SIM

Detalhes da Implementação – Posição atual



- Vamos considerar que o ponteiro `atual` aponta uma posição anterior ao nó da posição atual.
- No exemplo acima, o nó da posição atual tem valor 7 e está na posição 1 da lista.
- Agora, dá para inserir e remover na posição atual em tempo $O(1)$.

Arquivo LList.h

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

```
1  #ifndef LLIST_H
2  #define LLIST_H
3  #include <cassert>
4
5  struct Node {
6      int key;
7      Node *next;
8
9      Node(const int& k = 0, Node *nextval = nullptr) {
10         key = k;
11         next = nextval;
12     }
13 };
```

Arquivo LList.h (Continuação)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  class LList {
2  private:
3      Node* head;    // Ponteiro para a cabeca da lista
4      Node *tail;    // Ponteiro para o ultimo elemento da lista
5      Node *atual;   // Ponteiro para o elemento atual
6      int size;      // Tamanho da lista
7
8      // Funcao auxiliar de inicializacao dos
9      // atributos privados
10     void init();
11
12     // Libera todos os nos alocados, inclusive o no cabeca
13     void removeAll();
14
15 public:
16     LList();  // Construtor
17
18     ~LList(); // Destrutor
```

Arquivo LList.h (Continuação)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1       // Deixa a lista vazia, com zero elementos
2       void clear();
3
4       // Insere um elemento na posicao atual
5       // item: o elemento a ser inserido
6       void insert(const int& item);
7
8       // adiciona um elemento ao final da lista
9       // item: o elemento a ser inserido
10      void append(const int& item);
11
12      // Remove o elemento atual e devolve o seu valor
13      int remove();
14
15      // Remove sempre o ultimo elemento da lista
16      // e devolve o seu valor
17      int remove_back();
18
19      // Configura a posicao atual para o inicio da lista
20      void moveToStart();
```

Arquivo LList.h (Continuação)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1       // Configura a posicao atual para o final da lista
2       void moveToEnd();
3
4       // Move a posicao atual uma posicao para tras.
5       // Nada acontece se a posicao ja estiver no inicio da
        lista
6       void prev();
7
8       // Move a posicao atual uma posicao a frente.
9       // Nada acontece se a posicao ja estiver no final da lista
10      void next();
11
12      // Devolve o numero de elementos da lista
13      int length() const;
14
15      // Devolve a posicao do elemento atual
16      int posAtual() const;
```

Arquivo LList.h (Final)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1       // Configura a posicao atual
2       void moveToPos(int newpos);
3
4       // Devolve o elemento atual
5       int& getValue() const;
6
7       // Devolve true se cheia e false caso contrario
8       bool empty() const;
9
10      // operador[] sobrecarregado. Devolve uma
11      // referencia para o elemento na posicao i da lista
12      int& operator[](const int& index);
13  };
```

Implementação da classe LList

Exercício: Implemente as funções-membro da classe LList.

Programa cliente main.cpp


```
1  #include <iostream>
2  #include "LList.h"
3
4  void print(LList& l) {
5    for(l.moveToPos(0); l.posAtual() < l.length(); l.next())
6      std::cout << l.getValue() << " ";
7    std::cout << std::endl;
8  }
9
10 int main() {
11   LList lst;
12
13   for(int i = 1; i <= 15; i++) // insere ints na lista
14     lst.append(i);
15
16   print(lst); // imprime na tela
17
18   for(int i = 0; i < lst.length(); i++)
19     lst[i] *= 2; // dobra o valor de cada elemento
20
21   print(lst); // imprime na tela
22   return 0;
23 }
```

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

Outras operações em lista encadeada

Exercícios: Implemente as demais operações sobre listas:

- `bool empty() const`: Retorna se lista está ou não vazia
- `int length() const`: Retorna número de nós.
- `void removeAll(int x)`: Remove da lista todas as ocorrências do inteiro x.
- `int removeAt(int k)`: Remove a k-ésima célula da lista encadeada e retorna o seu valor. Note que deve-se ter $1 \leq$ `k` $\leq$ `size()`; caso contrário, retorna-se o menor inteiro possível.

Outras operações em lista encadeada

- `List *copy()`: Retorna um ponteiro para uma cópia desta lista.
- `void copyArray(int v[], int n)`: Copia os elementos do array `v` para a lista. O array tem `n` elementos. Todos os elementos anteriores da lista são apagados.

Outras operações em lista encadeada

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

- `void concat(LList& lst)`: Concatena a lista atual com a lista `lst` passada por referência. Após essa operação, `lst` será uma lista vazia, ou seja, o único nó de `lst` sera o nó cabeça.

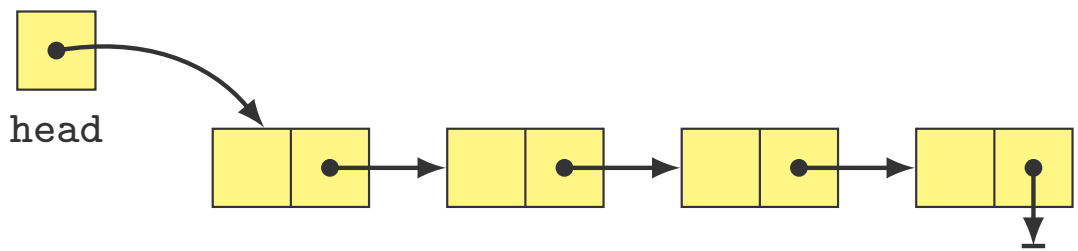
- `void reverse()`: Inverte a ordem dos nós (o primeiro nó passa a ser o último, o segundo passa a ser o penúltimo, etc.) Essa operação faz isso sem criar novos nós, apenas altera os ponteiros.

Exercício 2

Implementação de lista encadeada sem nó cabeça

Exercício: Implemente a lista simplesmente encadeada sem usar nó cabeça auxiliar.

- Quando a lista estiver vazia, `head` e `tail` apontam para `nullptr`.
- Caso contrário, `head` deve apontar diretamente para o primeiro nó da lista e `tail` para o último nó.



- Implemente todas as operações vistas nesta aula.

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM