

Universidade Federal do Rio Grande do Norte  
Instituto Metr pole Digital

Linguagem de Programac o I • IMD0030  
◁ Implementac o de Listas Encadeadas, v1.0 ▷  
9 de maio de 2017

## 1 Introduc o

O objetivo desta trabalho   implementar uma vers o *inicial* de uma **lista encadeada simples**, sem n  cabe a. *O foco principal do trabalho   nos algoritmos, na manipula o correta de ponteiros e aloca o din mica.* Por este motivo esta vers o de lista segue o paradigma imperativo de programac o e serve apenas para armazenar dados do tipo inteiro. Posteriormente, vamos criar novas vers es da lista utilizando o paradigma orientado   objetos, bem como a generaliza o de tipo de armazenamento atrav s do uso de *template*.

## 2 Especifica o da Lista

Voc  deve desenvolver um conjunto de fun es em C++ para manipular uma **lista encadeada simples**, sem n  cabe a que armazena n meros inteiros em uma ordem qualquer. Por se tratar do paradigma imperativo de programac o, todas as fun es desenvolvidas devem receber como par metro o apontador para o in cio da lista (AIL) sobre a qual a opera o dever  ser realizada.

Assuma que as seguintes declara es devem constar no arquivo de cabe alho da lista:

```
struct SLLNode {           // Struct for a Single Linked List Node.
    int miData;             // List stores integers.
    SLLNode * mpNext;       // Points to next node in list.
};
using SNPtr = SLLNode *; // Defining a new type.
```

Os prot tipos das opera es solicitadas s o:

- `void print( SNPtr _pAIL )`: Imprime os n meros inteiros da lista, na mesma sequ ncia que est o armazenados na lista, no formato " $\{ a_1, a_2, a_3, \dots, a_n \}$ ", onde  $a_i$  quer dizer o  $i$ - simo elemento da lista.
- `int length( SNPtr _pAIL )`: Calcula e retorna o comprimento atual da lista. O comprimento deve ser zero, caso a lista esteja vazia.
- `bool empty( SNPtr _pAIL )`: Retorna verdadeiro caso a lista esteja vazia; ou falso, caso contr rio.
- `void clear( SNPtr & _pAIL )`: Remove todos os elementos da lista, libera a mem ria alocada e faz o apontador de in cio de lista apontar para `nullptr`, indicando lista vazia.

- `bool front( SNPTr _pAIL, int & _retrievedVal )`: Recupera no segundo parâmetro o primeiro elemento da lista (sem removê-lo) e retorna verdadeiro; ou falso, caso a lista esteja vazia.
- `bool back( SNPTr _pAIL, int & _retrievedVal )`: Recupera no segundo parâmetro o último elemento da lista (sem removê-lo) e retorna verdadeiro; ou falso, caso a lista esteja vazia.
- `bool pushFront( SNPTr & _pAIL, int _newVal )`: Cria um novo nó na lista contendo o valor do segundo parâmetro e o insere na frente da lista. Se a operação for bem sucedida, a função deve retornar verdadeiro; ou falso, caso contrário.
- `bool pushBack( SNPTr & _pAIL, int _newVal )`: Cria um novo nó na lista contendo o valor do segundo parâmetro e o insere no final da lista. Se a operação for bem sucedida, a função deve retornar verdadeiro; ou falso, caso contrário.
- `bool popFront( SNPTr & _pAIL, int & _retrievedVal )`: Se a lista possuir um ou mais elementos, a função recupera o primeiro elemento no segundo parâmetro, remove-o da lista e retorna verdadeiro; caso contrário retorna falso.
- `bool popBack( SNPTr & _pAIL, int & _retrievedVal )`: Se a lista possuir um ou mais elementos, a função recupera o último elemento no segundo parâmetro, remove-o da lista e retorna verdadeiro; caso contrário retorna falso.
- `SNPTr find( SNPTr _pAIL, int _targetVal, )`: Busca na lista, a partir do primeiro nó, a primeira ocorrência do valor indicado no segundo parâmetro e retorna um apontador para o nó imediatamente anterior ao nó procurado. Se o nó procurado for (a) o primeiro elemento da lista ou (b) a lista for vazia a função retorna `nullptr`. Para diferenciar os dois casos basta verificar se o AIL da lista é diferente de nulo, caracterizando o caso (a). **Desafio:** E se quiséssemos encontrar a 2ª ocorrência de `_targetVal` ? Ou a 3ª?
- `bool insert( SNPTr & _pAIL, SNPTr _pAnte, int _newVal )`: Insere um novo nó na lista contendo `_newVal`. O `_pAIL` aponta para o início da lista, enquanto que o `_pAnte` aponta para o nó após o qual o novo nó deverá ser inserido. Se o `_pAnte` for nulo, o novo nó deve se tornar o primeiro nó da lista. A função deve retornar verdadeiro se bem sucedida, ou falso, caso contrário.
- `bool remove( SNPTr & _pAIL, SNPTr _pAnte, int & _retrievedVal )`: Remove o nó imediatamente seguinte ao nó apontado por `_pAnte`. O `_pAIL` aponta para o início da lista. Se o `_pAnte` for nulo, o primeiro nó da lista deve ser removido e o AIL deve ser ajustado de acordo. O inteiro removido deverá ser recuperado no parâmetro `_retrievedVal`. A função deve retornar verdadeiro se bem sucedida, ou falso, caso contrário.

O retorno de `find(...)` pode parecer estranho a primeira vista mas ele permite seu uso conjugado, por exemplo, com o método `remove` ou `insert`, como em:

```
int aiVet[] = {8, 2, 3, 9, 10, 5};           // Valores iniciais da lista
auto nVet = sizeof( aiVet ) / sizeof( int ); // Quantidade de elementos no vetor
auto removed(0);                           // Guardará o removido
// Cria uma lista com base nos elementos de aiVet[].
SNPtr ptHead = buildListFromArray( aiVet, nVet );
// Remoção do '9' da lista de forma conjugada.
remove( ptHead, find(ptHead, 9), removed );
std::cout << removed << std::endl; // Deveria imprimir '9'.
print( ptHead ); // Deveria imprimir: { 8, 2, 3, 10, 5 }
```

**Teste seus conhecimentos:** Porque algumas vezes este mesmo ponteiro é passado como referência com o uso do operador `&`?

### 3 Autoria e Política de Colaboração

Este trabalho é individual e o autor deve ser capaz de explicar com desenvoltura o código entregue, caso seja solicitado. O trabalho em cooperação entre alunos da turma é estimulado. Porém, esta interação não deve ser entendida como permissão para utilização de código alheio, o que pode caracterizar a situação de plágio. Trabalhos plagiados receberão nota zero, independente de quem seja o verdadeiro autor dos trabalhos infratores.

### 4 A Tarefa

Você deve implementar as funções solicitadas na Seção 2 em dois arquivos denominados `les_v1.cpp` e `les_v1.h`. O primeiro arquivo (de codificação) deve conter o código propriamente dito das funções solicitadas, enquanto que o segundo (de cabeçalho) deve conter apenas os protótipos das mesmas funções e as declarações de tipo de nó e ponteiro para nó. Além destes dois arquivos, você também deve criar um arquivo denominado `drive_les_v1.cpp` que deve contar a função `main()` e demonstrar o funcionamento de cada uma das funções implementadas. Elabore demonstrações criativas, de maneira que o seu código tenha sua qualidade de funcionamento assegurada.

Todos os arquivos fontes devem ser armazenados em uma pasta com o seguinte nome `Projeto_les_v1`. Dentro desta pasta você deve incluir todos os arquivos fontes, bem como um arquivo texto denominado `README` com explicações sobre o conteúdo de cada arquivo e qual o procedimento de compilação. No `README` você deve também indicar se o trabalho tem alguma restrição ou se alguma parte do trabalho não foi implementada.

A pasta compactada com sua solução deve ser entregue até o prazo especificado no componente Tarefas da Turma Virtual.

◀ FIM ▶