Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

Estruturas de Dados Básica I • IMD0029
◁ Implementação de Listas Encadeadas, v1.0 ▷
6 de novembro de 2017

1 Introdução

O objetivo desta trabalho é implementar uma versão *inicial* de uma **lista encadeada simples**, sem nó cabeça. *O foco principal do trabalho é nos algoritmos, na manipulação correta de ponteiros e alocação dinâmica.* Por este motivo esta versão de lista segue o paradigma imperativo de programação e serve apenas para armazenar dados do tipo inteiro. Posteriormente, vamos criar novas versões da lista utilizando o paradigma orientado à objetos, bem como a generalização de tipo de armazenamento através do uso de *template*.

2 Especificação da Lista

Você deve desenvolver um conjunto de funções em C++ para manipular uma **lista encadeada simples**, sem nó cabeça que armazena números inteiros em uma ordem qualquer. Por se tratar do paradigma imperativo de programação, todas as funções desenvolvidas devem receber como parâmetro o apontador para o início da lista (AIL) sobre a qual a operação deverá ser realizada.

Assuma que as seguintes declarações devem constar no arquivo de cabeçalho da lista:

```
struct Node {     // Struct for a singly linked list node.
    int data;     // List stores integers.
    Node * next;  // Points to next node in list.
};
```

Os protótipos das operações solicitadas são:

- `void print( Node * head )`: Imprime os números inteiros da lista, na mesma sequência que estão armazenados na lista, no formato "$\{\ a_1,\ a_2,\ a_3,\ \ldots,\ a_n\ \}$", onde a_i quer dizer o i-ésimo elemento da lista.
- `size_t length( Node * head )`: Calcula e retorna o comprimento atual da lista. O comprimento deve ser zero, caso a lista esteja vazia.
- `bool empty( Node * head )`: Retorna verdadeiro caso a lista esteja vazia; ou falso, caso contrário.
- `void clear( Node * & head )`: Remove todos os elementos da lista, libera a memória alocada e faz o apontador de início de lista apontar para `nullptr`, indicando lista vazia.

- `int front( Node * head )`: Retorna o primeiro elemento da lista (sem removê-lo); ou uma exceção `std::runtime_error`, caso a lista esteja vazia.
- `int back( Node * head )`: Retorna o último elemento da lista (sem removê-lo) ou uma exceção `std::runtime_error`, caso a lista esteja vazia.
- `void push_front( Node * & head, int value )`: Cria um novo nó na lista contendo o valor do segundo parâmetro e o insere na frente da lista.
- `void push_back( Node * & head, int value )`: Cria um novo nó na lista contendo o valor do segundo parâmetro e o insere no final da lista.
- `int pop_front( Node * & head )`: Remove e retorna o primeiro elemento da lista. Se a lista estiver vazia a função gera a exceção `std::runtime_error`.
- `int pop_back( Node * & head )`: Remove e retorna o último elemento da lista. Se a lista estiver vazia a função gera a exceção `std::runtime_error`.
- `Node * find( Node * head, int target )`: Busca na lista, a partir do primeiro nó, a primeira ocorrência de `target` 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 a lista é diferente de nulo, caracterizando o caso (a).
- `void insert( Node * & head, Node * prev, int value )`: Insere um novo nó na lista contendo `value`. O `head` aponta para o início da lista, enquanto que o `prev` aponta para o nó imediatamente anterior ao local de inserção do novo nó. Se o `prev` for nulo, o novo nó deve se tornar o primeiro nó da lista.
- `void remove( Node * & head, Node * prev )`: Remove o nó imediatamente seguinte ao nó apontado por `prev`. O `head` aponta para o início da lista. Se o `prev` for nulo, o primeiro nó da lista deve ser removido e o AIL deve ser ajustado de acordo.

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 A[] = {8, 2, 3, 9, 10, 5};          // Valores iniciais da lista
auto n = sizeof( A ) / sizeof( int );   // Quantidade de elementos no vetor
// Cria uma lista com base nos elementos de A[].
Node * head = build_list( A, n );
// Remoção do '9'  da lista de forma conjugada.
remove( head, find(head, 9) );
print( head ); // 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 arquivo 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 ▶