CS3610 Project 1

Due: January 28, Tuesday, 11:59 pm

In this project, you will implement a linked list class that includes a member function to reverse the encapsulated list in-place. The "in-place" constraint means that you are prohibited from copying the encapsulated list into any additional storage containers, such as a stack, during the course of the function's run. Although using additional storage containers can help simplify the reversal process, they unnecessarily waste space. Instead, you will use pointer manipulation to reorder the existing list. Consequently, the overall space complexity of your reversal algorithm will be constant $O(1)$.

The header for both the linked list class and the list node structure is outlined below.

```
template <typename T>
class LinkedList {

public:
  LinkedList() : head(NULL) {}
  ~LinkedList();

  void push_front(const T data);
  void pop_front();
  void reverse();
  void print() const;

private:
  struct ListNode {
    ListNode(const T data) : data(data), next(NULL) {}

    T data;
    ListNode* next;
  };

  ListNode* head;
};
```

You must write the definitions for the functions `push_front`, `pop_front`, `reverse`, `print`, and `~LinkedList`. The linked list constructor is already written as an initializer list. For those of you not familiar with initializer lists, `head(NULL)` is equivalent to writing `head = NULL` inside the brackets of the constructor definition. An initializer list is also used for the `ListNode` struct to set the member variable `data` to a user defined input and the `next` pointer to `NULL`. An example of its use is shown in the following code snippet:

```
ListNode* node = new ListNode(2);
cout << node->data << ", " << node->next << endl;
delete node;
```

The `new` command calls the `ListNode` constructor, which initializes `data` to the integer 2 (assuming the template type `T` is set to `int`) and `next` to `NULL`. Thus, the output of the code above will be:

```
2, 0
```

Input

Input commands are read from the keyboard. Your program must continue to check for input until the quit command is issued. The accepted input commands are listed as follows:

`a i` : Add the integer i to the front of the list. (`push_front`)

`d` : Delete the first element of the list. (`pop_front`)

`r` : Reverse the list.

`p` : Print the data value of each node in the list.

`q` : Quit the program.

Although your linked list class will be templated, your program will only be tested on integer data. The test suite used for grading will not contain data of any other type, so it is not necessary for you to verify the type of the input data.

Output

Print the results of the `p` command on one line using space as a delimiter. If the list is empty when issuing any of the commands `d`, `p`, or `r`, output the message `Empty`. Do not worry about verifying the input format.

Sample Test Case

Use input redirection to redirect commands written in a file to the standard input, e.g. `$ ./a.out < input1.dat`.

Input 1

```
a 1
a 2
a 3
p
r
d
```

```
p
q
```

Output 1

```
3 2 1
2 3
```

Turn In

Submit your source code under blackboard. If you have multiple files, package them into a zip file.

An Implementation Issue

The `LinkedList` class is declared in linked_list.h. You may ask "should I put the definitions of the functions into a separate source file, say, linked_list.cc?" The answer would be complicated when a template class is involved, as in this project. Some explanations can be found in `https://www.codeproject.com/Articles/48575/How-to-define-a-template-cla` and `https://www.experts-exchange.com/articles/1199/Separating-C-template-declaration-html`. An easy solution would be just putting your implementations in the same header file linked_list.h.

Grading

Total: 100 pts.

- **10**/100 - Code style, commenting, general readability.
- **05**/100 - Compiles.
- **05**/100 - Follows provided input and output format.
- **80**/100 - Successful implementation of the 5 requested linked list functions.