Data Structures & Algorithms

Lab Exercise 2 – Linked List Project

Due: to be demonstrated by week 7 (wk beginning 22-Oct)

Learning outcomes
At the end of this lab you should be able to:

- Insert nodes into the SLinkedList class
- Demonstrate usage of the SListIterator class
- Explain the difference between a list node (SListNode) and a list iterator (SListIterator)
- Demonstrate usage of the `std::list::splice()` member function.

Q1. Add a new member function to the `SLinkedList` class with the following declaration:

```
void insertBefore(InputIterator & position, T element)
```

This function should insert the specified element before the specified position in the list. Test your function thoroughly. (Hint: carefully consider the `SLinkedList::insertAfter()` member function).

Q2. Add a new member function to the `SLinkedList` class with the following declaration:

```
void reverse()
```

This function should reverse the order of the list elements. For example, if the list contains the sequence A, B, C then the reversed list will contain the sequence C, B, A.
Notes – as part of your solution:

1. You must not delete any list nodes.
2. Consider using a `std::stack` (think about how a stack may help here).

Q3. Add a new member function to the `SLinkedList` class with the following declaration:

```
void makeNewHead(InputIterator position)
```

that exchanges the node pointed to by position with the current head node of the list. For example, if we have a list:

```
head->1->2->3->4->5->6
```

where `head` points to 1 and `position` points to 5, then the re-arranged list looks like:

```
head->5->2->3->4->1->6
```

Do not swap the values stored in each node, you should manipulate the relevant pointers.

Q4. Add a new member function to the `SLinkedList` class with the following declaration:

```
void splice(InputIterator position, list<T> & x);
```

Transfers all elements from list `x` into `*this`. The elements are inserted before the element pointed to by the first argument (`position`). The list container `x` becomes empty after the operation.
This can be achieved quite simply by calling the `insertBefore()` function from Q1 for each element in x.
Note that x is an STL list, so don't forget to `#include <list>`
It is assumed that iterator `position` is always valid (i.e. you do not need to check if `position` is valid inside your function splice). Test your function thoroughly.

Q5. Create a `std::list` of integers and populate it with 20 random numbers between 1 and 10 (don't seed the random number generator to ensure you always have the same sequence of numbers). Create another list of the same type called `dest` but leave it empty.

Using the `list::splice()` member function, move all the nodes in the list starting from:

the last occurrence of 2 (not including 2) to the next occurrence of 4 (not including 4)

to

the start of `dest`.

In the event that 4 does not exist, move all the nodes from the last occurrence of 2 onwards (to the end) to the start of `dest`.
Hint: You will need to use the STL find algorithm and reverse iterators in your solution.