Practical 7: Linked lists

Due date: 11:59pm, 5 October 2018

1 General Instructions

All submissions for the practical assignments should be under version control. Submission procedure remains the same with the first practical assignment.

The directory under version control for this assignment should be named as

```
https://version-control.adelaide.edu.au/svn/aXXXXXXX/2018/s2/adds/assignment7/
```

where aXXXXXXX is your student ID.

If you get stuck on one of the hidden test cases and really cannot resolve it yourself, please feel free to ask the practical tutors for hints.

We encourage you to finish your work before the practical session and take the session as consulting time.

2 Problem Description

2.1 Objective

This practical will test your capability of implementing a linked list.

2.2 Design

In a file named `design.pdf`, describe how you are going to solve the problem and test your implementation with the test cases you designed based on the stages below.
Testing Hint: it's easier if you test things as a small piece of code, rather than building a giant lump of code that doesn't compile or run correctly. As part of your design, you should also sketch out how you are going to build and test the code.

2.3 Problem

Linked lists are dynamic data structures that allow storing a list of items in separate places in the memory. Each item of the list, together with the address of the next item, is placed in one object of type Node (find the description below). The last node of the list should include the last item and a null pointer (*NULL* or *nullptr*).

Lists, as abstract data types, can be implemented by different structures, including arrays and linked lists. Please do not use arrays for this practical.

In this practical, you should write the code for two classes named LinkedList, and Node. You must include separate header and implementation files for both classes. The class Node should consist of two member variables, an integer "data" and a pointer to Node "next", and 5 functions, a constructor, and getter and setter for the two member variables.

The class LinkedList should have only one member variable: head, which is of type pointer to Node. If the list is empty, head should contain *NULL* or *nullptr*. It should also have at least the following member functions. Note that add functions should construct nodes from the heap, and delete functions should manually delete the one that is removed from the list. Don't forget to write test cases for each function and ensure that the tests pass before progressing to the next function.

- `void addFront(int newItem)`: The function inserts a new node, containing the newItem, at the beginning of the list.

- `void addEnd(int newItem)`: The function inserts a new node, containing the newItem, at the end of the list.

- `void addAtPosition(int position, int newItem)`: The function inserts a new node, containing the newItem, such that it is the position-th member of the list. i.e. we assume the first element of the list is in position 1. If position is larger than the size of the list, the new item is added to the end of the list. If position< 1, the new item is added at the beginning of the list.

- `int search(int item)`: The function searched the list for the item, and if found, both prints the position of the of the item (followed by a space) and returns the position of the item in the list (positions start from 1). If not found, both prints 0 (followed by a space) and returns 0. Note that the returning type is different from what was explained in the search function in the lecture.

- `void deleteFront()`: The function deletes the first element of the list.

- `void deleteEnd()`: The function deletes the last element of the list.

- `void deletePosition(int position)`: The function deletes the element at the given position of the list. If the position¡1 or it is larger than the size of the list, only print "outside range".

- `int getItem(int position)`: The function both prints the value of the item (followed by a space) and returns the value of the item at the given position of the list, If beyond the size of the array, both prints $std :: numeric\_limits < int >:: max()$

(followed by a space) and returns $std :: numeric\_limits < int >:: max()$. You should include <limits> for this. Take a look at
$http://www.cplusplus.com/reference/limits/numeric\_limits/$
if you need.

- `void printItems()`: The function prints the value of the items of the list from head to tail. In case of an empty list, it does not print anything

- A constructor with no parameters, which makes an empty list.
- A constructor that takes an array of integers and makes a linked list, containing all the elements of the array, in the same order. As the second parameter, it takes the size of the array.
- A destructor that manually deletes all the elements that are still in the list.

Note that the printing in the functions search and getItem is for the purpose of easy testing.

2.4 Main function

The test script will compile your code using `g++ -o main.out -std=c++11 -O2 -Wall *.cpp`. It is your responsibility to ensure that your code compiles on the university system. g++ has too many versions, so being able to compile on your laptop does not guarantee that it compiles on the university system. You are encouraged to debug your code on a lab computer (or use SSH).

You are asked to create a main function (`main.cpp`). It takes in one line of input.
`int1 int2 ... intn  FUNCTIONINITIAL param1 param2`

int1, until intn are integers, separated by space, which should be placed in an integer array, and passed to the linked list constructor. For simplicity, we assume that the size of this array never exceeds 100; therefore, you can take an static array with the size of 100.

After the elements of the list, the input consists of an string, denoting a function, followed by its parameters. The string is one of these:

- AF standing for addFront
- AE standing for addEnd
- AP standing for addAtPosition
- S standing for search
- DF standing for deleteFront
- DE standing for deleteEnd

- DP standing for deletePosition
- GI standing for getItem

Call the indicated function with its parameter. If the function has only one parameter, then the last integer value from the input is not used. At the end, call the function *printItems*, to produce the required output.

Sample input: 5 2 7 10 AP 3 9
expected output: 5 2 9 7 10

Sample input: 3 4 2 1 DP 3 0
expected output: 3 4 1

Sample input: 45 20 2 10 GI 3 0
expected output: 2 45 20 2 10

3 Marking Scheme

Each practical assignment is worth 3% of your final mark.

You may submit your work before your practical session in Week 9 to get marked by your tutors so that you can improve your answer based on the feedback and resubmit before the due date.

Every practical assignment is marked out of 6.

- Design (2 marks):
  - UML diagram of central classes and explanation of core functions (1 mark)
  - Details of your own test cases/schemes (1 mark)
- Style (2 marks):
  - Proper usage of C++ language elements and paradigm (1 mark)
  - Comments on non-trivial code blocks and functions (1 mark)
- Functionality (2 marks):
  - Passing public test cases (1 mark)
  - Passing hidden test cases (1 mark)

Your final mark for this assignment will be based on your latest submission in the web-submission system. If you have already got a higher mark, it will not be replaced with a lower mark. With delays of upto 1, 2 and 3 days, your mark will be capped at 25%, 50% and 75%, respectively.