Department of Computer Science
COS110 - Program Design: Introduction
Practical 1

1 Introduction

Deadline: 19th of September, 07:00

1.1 Objectives and Outcomes

The objective of this practical is to test your understanding of the programming concepts covered in the theory classes. In particular, this practical will test your understanding of templates and a new data structure, the linked list, in addition to other previously covered topics.

1.2 Submission

All submissions are to be made to the **assignments.cs.up.ac.za** page under the COS 110 page, and for the correct practical slot. Submit your code to Fitchfork before the closing time. Students are **strongly advised** to submit well before the deadline as **no late submissions will be accepted**.

1.3 Plagiarism

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone elses work without consent, copying a friends work (even with consent) and copying textual material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to **http://www.ais.up.ac.za/plagiarism/index.htm** (from the main page of the University of Pretoria site, follow the *Library* quick link, and then click the *Plagiarism* link). If you have questions regarding this, please ask one of the lecturers, to avoid any misunderstanding.

1.4 Implementation Guidelines

Follow the specifications of the practical precisely. For each practical, you will be required to create your own makefile so pay attention to the names of the files you will be asked to create. If the practical requires you to submit additional files of your own, follow the file structure and format exactly. Incorrect submissions will use up your uploads and no extensions will be given. In terms of C++, unless otherwise stated, the usage of C++11 or additional libraries outside of those indicated in the practical, will not be allowed. Some of the appropriate files that you submit will be overwritten during marking to ensure compliance to these requirements. If the specification makes use of text files, for providing input information, be sure to include blank text files with the specified names.

1.5 Mark Distribution

Activity	Mark
DLL	20
item	10
Total	**30**

2 Practical

2.1 Linked Lists

Linked lists are an example of an unbound data structure. Whereas the array is based around having a fixed size per array creation, there is no logical limit on the ability of a linked list to grow. Physical limitations aside, linked lists are capable of growing largely without any limitations. To achieve this, they trade easier access to their individual elements. This is because linked lists have to be traversed from their root node to any node in question, unlike arrays which are capable of supporting random access to any index without traversing the others.

With regards to linking of the various classes, it is important to know the correct method. Specifically, if a linked list uses another class, such as item, for the nodes it has, and both classes are using templates, then the linked list .cpp should have an include for the item .cpp as well to ensure proper linkages.

Additionally, you will be not be provided with mains. You must create your own main to test that your code works and include it in the submission which will be overwritten during marking.

2.2 Task 1

You are going to implement a variant of the standard linked list, the doubly linked list. Doubly linked lists are because they enable a backwards and forwards traversal of the list through the addition of more pointers. By increasing the memory cost of the list, it enables a better access since the list does not need to be traversed in only one direction. This will consist of implementing two classes: **dLL** and **item**.

2.2.1 dLL

The class is described according to the simple UML diagram below:

```
dLL<T>
-head: item<T>*
-tail: item<T>*
-size: int
----------------------------
+dLL()
+~dLL()
+getHead(): item<T>*
+getTail(): item<T>*
+push(newItem:item<T>*):void
+pop():item<T>*
+getItem(i:int):item<T>*
+minNode():T
+getSize():int
+printList():void
```

The class variables are as follows:

- head: The head pointer of the doubly linked list.
- tail: The tail pointer of the doubly linked list.
- size: The current size of the doubly linked list. This starts at 0 and increases as the list grows in size.

The class methods are as follows:

- dLL: The class constructor. It starts by setting the variables to null and 0 respectively.
- ∼dLL: The class destructor. It will deallocate all of the memory in the class.
- getHead: This returns the head pointer of the doubly linked list.
- getTail: This returns the tail pointer of the doubly linked list.
- push: This adds a new item to the doubly linked list, by adding it to the front of the list.
- pop: This returns the top item of the linked list. The item is returned and removed from the list.
- getItem: This returns the item of the linked list at the index specified by the argument but without removing it from the list. If the index is out of bounds, return null.
- minNode: This returns the value of the item that has the lowest value in the linked list.
- getSize: This returns the current size of the linked list.
- printList: This prints out the entire list in order, from head to tail. Each item's data value is separate by a comma. For example: 3.1,5,26.6,17.3

2.2.2 item

The class is described according to the simple UML diagram below:

```
item <T>
-data:T
--------------------
+item(t:T)
+~item()
+next: item*
+prev: item*
+getData():T
```

The class has the following variables:

- data: A template variable that stores some piece of information.
- next: A pointer of item type to the next item in the linked list.
- prev: A pointer of item type to the previous item in the linked list.

The class has the following methods:

- item: This constructor receives an argument and instantiates the data variable with it.
- ∼item: This is the destructor for the item class. It prints out "Item Deleted" with no quotation marks and a new line at the end.
- getData: This returns the data element stored in the item.

You will be allowed to use the following libraries: **cstdlib,string,iostream**. You will have a maximum of 10 uploads for this task. Your submission must contain **item.h, item.cpp, dLL.cpp, dLL.h, main.cpp** and a makefile.