UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

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

1 Introduction

Deadline: 17th of October, 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 data structures, specifically the stack and the linked list.

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**
Stack	30
Total	**30**

2 Practical

2.1 Stacks

A stack is a data structure that that is a container. It follows the strategy of LIFO (Last In First Out) which means that items added, chronologically sooner, than later items are taken out after items that have been added later. This can be likened to what happens when making a pile of textbooks. Starting from no books, you place one book after another, each on top of the other. After a few books, getting to the books you first placed down, requires picking up the books you placed later.

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

Imagine that you are an engineer on a distant planet. You were tasked with mining the resources of the planet and have done so through largely primitive means. However you have managed to set up an automated train system to gather and collect the resources from your distant operations. Your train system collects boxes full of resources. A box might contain ores or wood or even liquid fuel. These boxes are placed into the train using drones who stack them on each other to save space. You now need to implement a management system using the newly researched data structures that will prove useful. In particular, the rapid pace of your development means that resources are accumulated quickly and fixed sized structures would be unwise. Therefore you will be implementing the stack through the use of a linked list.

2.2.1 resrcStack

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

```
resrcStack
-top: stackNode*
```

```
---------------------------
+resrcStack()
+~resrcStack()
+push(t: stackNode*):void
+pop():void
+peek():stackNode *
+print():void
+determineTransportRequirement():string
```

The class variables are defined below:

- top: The current top of the stack. It will start as null but should refer to the top of the stack.

The class methods are defined below:

- resrcStack: The class constructor. It will start by initialising the variables to null.
- ∼resrcStack: The class destructor. It will deallocate all of the memory assigned by the class.
- push: This will receive a stackNode to add onto the current stack. The node is added from the front.
- pop: This will remove the top stackNode from the stack. If it is empty, print out "EMPTY" with a newline at the end and no quotation marks. When removing the node, it should be deleted.
- peek: This will return the top node of the stack but without removing it from the stack.
- print: This will print the entire contents of the stack. For each resrc node in the stack, print the following information out sequentially, line by line. The format of this is as follows:

  ```
  Resource: Unrefined Ores
  Quantity: 3500
  Resource: Refined Alloys
  Quantity: 30000
  ```

- determineTransportRequirement: This function will be used to determine what sort of transportation will be required to move the resources contained within the stack to their required destinations. The result of this function is a string returned for the following conditions:

  1. drone: If the number of nodes is less than or equal to 5, drones will required.
  2. car: If the quantity of the items in total, is greater than 50 and the items number at no more than 10, a car will be required. Quantity of items refers to the sum of the quantity variables in each of the stack nodes.

3. train: If the quantity of the items in total of the items is over 100, a train will be required. Quantity of items refers to the sum of the quantity variables in each of the stack nodes.

Return either drone, car or train depending on the conditions present. If none of the conditions are met, return "LOGISTICS ERROR" without the quotation marks. The train takes precedence over the car which takes precedence over the drones for what to return.

2.2.2 stackNode

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

```
stackNode<T>
-resrc:T
+next:stackNode *
-quantity:int
------------------------
+stackNode(i:T,q:int)
+~stackNode()
+getResrc():T
+getQuantity():int
```

The class variables are defined below:

- resrc: This is the basic resource unit. It will describe the resource contained within the box. It might be a code that describes the contents or just a description and so must be a template type to accommodate for a variety of manifests.
- next: A pointer to the next node of the stack.
- quantity: A variable which describes the number of a resource contained with the stackNode box. The quantity is an absolute measurement and applies regardless if the resource is ore, wood or even crude oil.

The class methods are defined below:

- stackNode: A class constructor. It receives the template type and the weight for that item.
- ∼stackNode: The class destructor. It should print out "Resource Unit Destroyed" with no quotation marks and a new line at the end.
- getResrc: This returns the template variable stored within the class.
- getQuantity: This returns the quantity of the resources.

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