Assignment 1

COS 212

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Deadline: 09/04/2021 at 18:00

General instructions:

- This assignment should be completed individually, no group effort is allowed.
- The output of both your C++ and Java programs will be evaluated.
- Be ready to upload your assignment well before the deadline as no extension will be granted.
- Task 1 is divided up into a number of steps to assist you. Follow each step in order to complete the assignment and produce the final deliverables for marking.
- You are NOT allowed to import any Java packages with classes that support linked list-, stack or queue-like functionality (e.g. ArrayList). Doing so you will receive 0.
- The differences between C++ and Java have been deliberately left vague or omitted (e.g. Java and pointers, Java and destructors, etc.) from this specification. It is up to you to discover and investigate these differences and compensate for them.
- Your code may be inspected to ensure that you've followed the instructions.
- If your code does not compile you will be awarded a mark of 0.
- You will be afforded three opportunities per part to upload your submissions for automarking.

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 else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to `http://www.library.up.ac.za/plagiarism/index.htm` (from the main page of the University of Pretoria site, follow the *Library* quick link, and then choose the *Plagiarism* option under the *Services* menu). If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding. Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

Overview:

Upon the successful completion of this assignment, you will have been exposed to and implemented a variety of linear data structures. You will have learnt about the differences and similarities between these structures and your final program will allow users to make their own choices of which implementations to use at runtime. You will also have implemented your own basic data structures in C++ and Java.

Background:

Linear Data Structures

There are a variety of linear structures that one may use to store elements. Some examples include vectors, linked lists, arrays, stacks and queues. Some of these structures are specialized in terms of the order in which they process elements. The choice of which structure is the most appropriate to use depends on your application.

There are various trade-offs between linear data structures. Dynamic arrays for example allow for quick access to elements but managing their sizes causes extra overhead. Linked lists address the size issues of arrays but they are slower when accessing arbitrary elements.

Other linear structures do not allow for access to arbitrary elements and they maintain a certain order amongst the elements themselves. In stacks and queues for example, the next element to which access is allowed is determined by the order in which elements were inserted into these structures. In priority queues on the other hand, access is granted to the current most important element, regardless of the order in which the elements were inserted.

Implementing Linear Data Structures

Linear structures may be implemented in a number of ways. Linked lists, for example, can be implemented as singly linked lists or doubly linked lists. Stacks may be implemented using arrays or linked list-like structures. As discussed previously, each type of implementation comes with its own advantages and disadvantages and which implementations is most suited depends on the situation.

One can easily cater for every type of implementation that one may possibly need and do it in such a way that it's easy and convenient to add additional implementations at a later stage, should it be required, without changing any code that you've already written. For example, you needn't hard code two stack classes where one is implemented in terms on an array and one a linked list. Instead, you can code one stack class and instantiate it with a linked list or array implementation (provided these conform to the same interface). It might seem counter intuitive to code three classes (linked list, array and stack) instead of just two. However, if you need a queue class later on then you can implement it making use of your linked list or array classes as the underlying structures. You can do the same to add a priority queue to your collection of structures. You then have 5 classes which you can mix and match with different implementations as opposed to 6 that you'll need to cater for all these combinations if they were hard coded. If you add an additional type of implementation, say for example a doubly linked list, then you will be able to create 9 combinations of stacks, queues and priority queues with only 6 classes.

Task 1:

Download the archive *assignment1Code.zip* from the course website. This archive contains a number of classes which you will have to implement. The comments in the code describe how each function should be implemented. You are not allowed to modify the header files. The header files will be overwritten for marking purposes. You are provided with the following classes:

LinearStructure

This class is an interface to which linear structure implementations conform. LinkedList, CircularList and DynamicArray inherit from this class.

Node

A singly linked node class to be used for the linked list implementations.

LinkedList

A simple linked list class.

DynamicArray

This class encapsulates a dynamic array. The array grows as more space is needed.

CircularList

This class is also a linked-list like structure and also uses nodes but the implementation should be that of a circular list.

OrderedContainer

This class provides an interface for all types of ordered containers. An ordered container may be implemented in a variety of ways. As an example, a stack may be implemented as either a linked list or an array. The ordered container is provided with an implementation (either linked list, dynamic array or a circular list) and then stores elements in these structures. The subclasses will have to manipulate their underlying implementations to produce the correct results.

Queue

A queue is a linear structure which does not allow access to arbitrary elements stored in it. It prescribes three main operations, namely insertion (enqueueing) and removal (dequeueing) and testing whether the structure is empty. It may have additional operations like seeing the first element in the queue without removing it. Queues are FIFO (first-in-first-out) structures which means that elements are removed and returned from a queue in the order in which they were inserted. If you, for example, enqueue the elements y, x and z in this order into the queue then the elements would be removed in the same order. Queues perform their operations at either the front (removal) or back (insertion) of the structure.

Stack

A stack is similar to a queue in that the order in which elements are inserted determines the order in which the elements are returned or removed. Stacks differ from queues however as they perform all of their operations at only one end of the structure. Stacks should support insertion (push) and deletion (pop) operations. In addition to the aforementioned, stacks should preferably also support tests to determine whether the stack is empty. Stacks are known as LIFO (last-in-first-out) structures. They return elements in the reverse order in which they where inserted. If you, for example, push the elements y, x and z in this order onto the stack then the elements would be removed and returned in reverse order as in z, x and y.

PriorityQueue

Priority queues recognize the fact that some elements might be more important than others and return elements with the highest importance first regardless of the order in which the elements were inserted. For this assignment you will have to implement your priority queue such that largest element is returned from the structure. See the comments in the code for more details.

Implementation Details

You are not allowed to change the header files. You will notice that the linear structures have a very limited interface. You will have to think carefully on how you can use this interface to manipulate the structures appropriately in your container classes.

You may make the following assumptions for any **T**:

- Any T will have at least a default constructor, copy constructor, overloaded assignment operator and a destructor.
- All of the relational operators (==, <, <=, etc) will be overloaded for any **T**.

HINT: Draw a class diagram to depict all of the classes and their relationships. You will also have to create your own makefile.

Task 2

You will have to translate your C++ code into an equivalent Java implementation. Java differs slightly from C++ in some regards and does not directly support the same features but does provide equivalent constructs. Some of these differences have been deliberately omitted or left vague in this specification as your task is to find suitable and standard workarounds for these. Some things to take note of:

- There is no operator overloading in Java. HINT: See the Object class in Java to determine if there is anything you can use from there.
- Exceptions are handled very strictly. All exceptions need to conform to Java's Throwable interface. Methods responsible for throwing exceptions will also have to indicate so in their signatures.
- The generic type T will need to conform to the Comparable interface. See the SkipList class' declaration in the textbook.

Implementation Issues

Do not rename any of the classes or methods in your Java implementation. Create your own main method and place it in a class called `Main` in a file called `Main.java`. Your will also have to include a makefile for your implementation. The makefile's implementation is as follows:

```
Main.class:

javac *.java

clean:
rm -f *.class

run:  Main.class
java -Xmx64m Main
```

Submission instructions

For your submission, you must tar your Java code and C++ code, including the makefiles, into two archives where one will be your C++ submission and the other your Java submission. Your Java archive should be named uXXXJava and your C++ archive should be named uXXXCPLUSPLUS, where XXX is your student number.

Go to the website `https://ff.cs.up.ac.za` and log in with your CS website credentials.

Select COS 212 amongst your modules and find the two assignment links called Assignment1C++ and Assignment1Java and submit the appropriate archives to each of the uploads for automarking before the deadline.