Source: http://www.fullerton.edu/ecs/cs/resources/epp_exam.php

Sample Exam: A sample exam is attached to this document. Some of the problems indicate that some files are available. They are standard files that you can look up in your texts or create yourself. For example, IntListNode class implements a simple node for a linked list holding integers.

Part I (20 points each)

1. Write a complete C++ program that takes the judges' scores for a gymnastic event and finds each contestant's score. It should satisfy all the following specifications: it first reads in an integer N that gives the number of judges. It then reads in a contestant number followed by N integers which are the scores given the contestant by the N judges. The scores should be read into an array, and then the program calls a function called findAverageScore that takes N and the array of judges' scores as arguments. The function should find and drop the highest score and the lowest score, and then find and return the average of the remaining N-2 scores. The main should then print the contestant number and the average score s/he receives for the event to the screen. It will continue to do this (read in the contestant number, then the N integers, then print the contestant number and the calculated score on a new line) until a negative value is entered for the contestant number. The program will then print the contestant number who scored the highest. (You may assume no ties will occur.) All input will be from the keyboard and you may also assume that N will never be greater than 10 and never less than 3. You may format the output in any way as long as it is readable and correct, and you may assume all inputs are correct. (JUST WRITE CODE; NO DOCUMENTATION REQUIRED except for your NAME and ID at the start of the file!)

Name your program prob1.cpp. Submit it to the EPP - Problem 1 section on the class Titanium site. If you wrote more than one file containing code, make sure to send and copy them as well. Remember to put in your name and ID at the start of each file.

Sample output screen for a session (Bold numbers are printed by the program; normal numbers are typed in by the user.)

Number of Judges: 5

34   7   5   3   7   6

Contestant   34   6

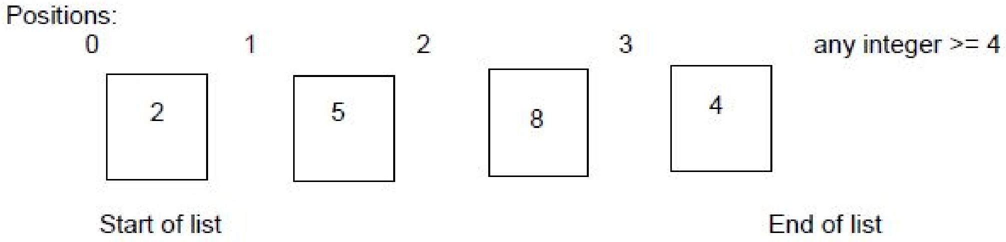
```
28          8    5    8   10   7

Contestant    28   7.666666667

3           5    5     3    4    4

Contestant 3        4.333333333

−1

Contestant 28 had the highest score.

<program ends>
```

2. On the class Titanium site for Problem 2, you are given the following four files defining a simple linked list class called IntegerList: IntegerList.h, IntegerList.cpp, IntListNode.h and IntListNode.cpp. There is also a file called test.cpp that allows testing of this list containing integers. An IntegerList object contains one pointer pointing to the first or front node of the linked list. An IntListNode object contains an integer and a pointer to another IntListNode object following it. The main function in test.cpp allows you to insert integers at the front of the list and to print all integers in it.

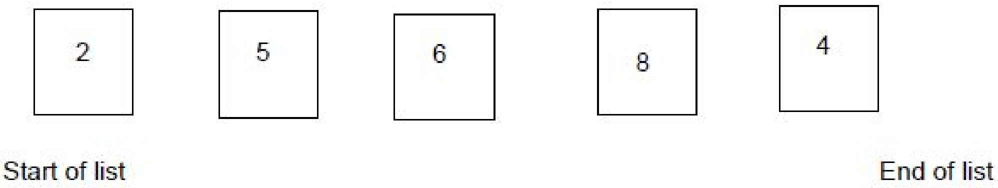
Add a member function to the class IntegerList called AddAtPosition that takes as parameters an integer to be placed in the list and a number indicating the position where the new integer should be inserted. IntegerList.h has the member function heading in the public part of IntegerList. All you need to do is to add the member function declaration (the body of the member function).

Rename the file prob2IntegerList.cpp. Submit it to the EPP - Problem 2 section on the class Titanium site. If you wrote more than one file containing code, make sure to send and copy them as well. Remember to put in your name and ID at the start of each file you turn in.

See below for the how the position number corresponds to where the new node is placed in the list. If the position number is greater than the length of the list, it will just add the new node at the end.



What above list would look like after **AddAtPosition (6, 2)** was called:



3. Take either your revised IntegerList.cpp or the original IntegerList.cpp and add a recursive member function sum( ) that will return the sum of all numbers stored in the list. If it is called with the list pictured in problem 2, it should return 25 since 2 + 5 + 6 + 8 + 4 = 25.

A non-recursive version will get no credit.

Rename the file **prob3IntegerList.cpp.** Submit it to the EPP - Problem 3 section on the class Titanium site. If you wrote more than one file, make sure to send and copy them as well. Remember to put in your name and ID at the start of each file you turn in.

Part II (40 points) (Resource files described below are available from the Titanium site under EPP-Part II.)

Write a complete C++ program called **inventory.cpp** as described below. **Documentation is required; see next page.**

Write an interactive program for processing inventory information for a parts warehouse. When started, the program should read in from a text file used for storing information between sessions. It is possible that the text file is empty. The program user is then given five choices in the main menu: enter a new part to the system, find and print data for a part when given the part ID number, find and modify data for a part when given the part ID number, copy all information to a binary archive file, and quit. These choices will be explained more fully below.

While the program is running, all parts information should be in the main memory for fast response. You may assume that there will always be enough space in memory to do this.

Specifications:

For each part, you need to store its ID (an integer), a short text description, its price, and a count of how many of that part the warehouse currently has. You are given two files **Part.h** and **Part.cpp** that hold the declarations and definitions for a simple Part class that holds just the ID, and the text description. You can revise the files as needed for this assignment.

When the program first starts, it reads in the information is stored in a text file called **textSave.txt**. You may assume that such a file always exists, but may be empty.

The program should now print a menu and ask the user to enter one of the following letters corresponding to the 4 choices below. The program will do the appropriate task, and keep repeating (get choice, do task) until the option Q is chosen,

N: this stands for a new part. The program should ask the user to enter the part ID and the rest of the part information. The program will then enter the new part information into the system.

F: this stands for finding a particular part. The program will ask for a part ID number, and search the system for a part with that ID. If it is found, the program will display the ID and all other information about the part to the screen. If it is not found, the program will print a message to the screen, telling the user that it could not be found.

A: This stands for archiving the information. Everything is output to a **binary** file called **binarySave.dat**. Any previous file of that name will be overwritten.

Q: this stands for quit. The program will write all the current parts information into the text file textSave.txt, overwriting any previous data before quitting.

Design considerations:

You do not need to be concerned with speed when reading from or writing to the files. However, you should try to make the other tasks run quickly. You may choose any format for the two save files, as long as they contain all the information and can be read and written by your program as text (for **textSave.txt**) and written as binary (for **binarySave.dat**).

Resources:

In addition to the two files Part.h and Part.cpp, you are given files containing templates for a Stack class, a Queue class, and a Tree class (this implements a binary search tree). For each of the three data structures, you are also provided with a test program. All function bodies are given. You may use none, any, or all of them if it will help you. You are also free to revise any of these files if you wish.

You should rename your file inventory.cpp and turn it in through the class Titanium site. Remember to put in your name and ID at the start of the file.

Please also turn in all additional files, old, new, or revised, that your program uses, to the file names.

Required documentation:

Your name and ID at the top for each file you write or revise.

Pseudocode for your **inventory.cpp.** This can be turned in as a separate file (Word or plain text) or as a long comment at the beginning of the C++ file.

If any variable name does not make it immediately clear what its role is, that variable must have documentation whenever it is used.

All functions, **including the main**, must have a brief documentation that states **what the function does**, and **the roles played by each of the parameters, if any**.

Other documentation is optional. Any unusual or tricky code should be documented, but please do not just repeat the code. For example, you do not need to say “for loop” because the code will say “for

(...)”.