CSC 2110 – Winter 2016
Project 02
Total 120 points
(100 points + 20 bonus points)

Project Description:

Goal: explore C++ programmings advanced features and learn how to use basic types of data structures. In this project, there are three tasks. The first two tasks are mandatory tasks (50 points each, total 100 points. The third one is a bonus task of 20 points. You could choose to finish it or not. However, for highly capable students, you are strongly recommended to treat it as an extra exercise and build a high quality, error-free program.

- Task I: Write a program to sort elements using the quick sort algorithm.
- Task II: Write a program to keep track of a hardware store inventory in order to become familiar with the vector type and discover how to sort records using recursive functions.
- Task III (bonus task): Write a program to convert an infix expression into an equivalent postfix expression.

Basic features make up most of the programs, but you can add more features and make your programs more effective and better overall.

Grading:

The grade of each program will be based on the creation of a program that works correctly (50%), clear problem analysis and algorithm design (20%), the appropriate use of concepts and techniques such as classes, exception handing, functions, arrays, linked list, vectors, queues, and stacks (20%), and the readability of the functions and programs with informative comments (10%).

Submissions:

Include a README file (.txt), with some basic instructions on how to use your program. Also include in this README what the contributions in your project are. Given the instructions, we would test your project accordingly.

Pack everything into a single .zip file (i.e. your code, the README file, and the project description file). Upload your project to the blackboard satisfying the requirements by the due date. No email or hard copy would be acceptable.

Restrictions:
You must work individually. Do not share your code or copy from external resources.

Task I (50 points)

Sort an array of 10,000 elements using the quick sort algorithm as follows:

a. Sort the array using pivot as the middle element of the array.

b. Sort the array using pivot as the median of the first, last, and middle elements of the array.

c. Sort the array using pivot as the middle element of the array. However, when the size of any sublist reduces to less than 20, sort the sublist using an insertion sort.

d. Sort the array using pivot as the median of the first, last, and middle elements of the array. When the size of any sublist reduces to less than 20, sort the sublist using an insertion sort.

e. Calculate and print the CPU time for each of the preceding four steps. To find the current CPU time, declare a variable, say, x, of type *clock_t*. The statement $x = clock()$; stores the current CPU time in x. You can check the CPU time before and after a particular phase of a program. Then, to find the CPU time for that particular phase of the program, subtract the before time from the after time. Moreover, you must include the header file ctime to use the data type *clock_t* and the function clock. Use a random number generator to initially fill the array.

Task II (50 points)

Write a program to keep track of a hardware store inventory. The store sells various items. For each item in the store, the following information is kept: item ID, item name, number of pieces ordered, number of pieces currently in the store, number of pieces sold, manufacturers price for the item, and the stores selling price. At the end of each week, the store manager would like to see a report in the following form:

```
                      Friendly Hardware Store

itemID itemName       pOrdered pInStore pSold manufPrice sellingPrice
4444   Circular Saw        150      150    40      45.00       125.00
3333   Cooking Range        50       50    20     450.00       850.00
.
.
.

Total Inventory: $#########.##
Total number of items in the store: ___________
```

The total inventory is the total selling value of all of the items currently in the store. The total number of items is the sum of the number of pieces of all of the items in the store.

Your program must be **menu driven**, giving the user various choices, such as checking whether an item is in the store, selling an item, and printing the report. After inputting the data, sort it according to the items names. Also, after an item is sold, update the appropriate counts.

Initially, the number of pieces (of an item) in the store is the same as the number of pieces ordered, and the number of pieces of an item sold is zero. Input to the program is a file consisting of data in the following form:

itemID
itemName
pOrdered manufPrice sellingPrice

Use seven parallel vectors to store the information. The program must contain at least the following functions: one to input data into the vectors, one to display the menu, one to sell an item, and one to print the report for the manager.

Task III (20 points)

(Infix to Postfix) Write a program that converts an infix expression into an equivalent postfix expression.

The rules to convert an infix expression into an equivalent postfix expression are as follows:

Suppose **infx** represents the infix expression and **pfx** represents the postfix expression. The rules to convert **infx** into **pfx** are as follows:

a. Initialize **pfx** to an empty expression and also initialize the stack.

b. Get the next symbol, **sym**, from **infx**.

   - If **sym** is an operand, append sym to **pfx**.
   - If **sym** is (, push sym into the stack.
   - If **sym** is), pop and append all of the symbols from the stack until the most recent left parentheses. Pop and discard the left parentheses.
   - If **sym** is an operator:
     - Pop and append all of the operators from the stack to **pfx** that are above the most recent left parentheses and have precedence greater than or equal to **sym**.
     - Push **sym** onto the stack.

c. After processing **infx** , some operators might be left in the stack. Pop and append to **pfx** everything from the stack.

In this program, you will consider the following (binary) arithmetic operators: +, -, *, and /. You may assume that the expressions you will process are error free.

Design a class that stores the infix and postfix strings. The class must include the following operations:

- **getInfix**: Stores the infix expression.
- **showInfix**: Outputs the infix expression.
- **showPostfix**: Outputs the postfix expression.

Some other operations that you might need are:

- **convertToPostfix**: Converts the infix expression into a postfix expression. The resulting postfix expression is stored in **pfx**.
- **precedence**: Determines the precedence between two operators. If the first operator is of higher or equal precedence than the second operator, it returns the value true; otherwise, it returns the value false.

Include the constructors and destructors for automatic initialization and dynamic memory deallocation.

Test your program on the following expressions:

a. A + B - C;

b. (A + B) * C;

c. (A + B) * (C - D);

d. A + ((B + C) * (E - F) - G) / (H - I);

e. A + B * (C + D) - E / F * G + H;

For each expression, your answer must be in the following form:

Infix Expression: A + B - C;
Postfix Expression: A B + C -

————x———— **End** ————x————