Programming Assignment #2
CS 163 Data Structures

Submit your assignment to the D2L Dropbox (sign on via d2l.pdx.edu)

Remember to turn in a design writeup with your program

Problem Statement: Have you ever been using program (such as word, excel, or even vim) and realized that you have made a mistake? Thankfully there is the "undo" and we can reverse as far back as we wanted to go! To perform this type of operation, the programs that we use have data structures that keep track of those operations so that they can in fact be undone.

In our second program, we will be creating a data structure that will allow for such actions. In this program we will be implementing a calculator that does addition, subtraction, multiplication, and division. The client can request a sequence of mathematical operations to take place, undo them if they have made a mistake, or see a history of all of their requests. Once they are satisfied, they can request that the result be calculated. Your program will also handle precedence, where multiplication and division have higher precedence than addition and subtraction.

Programming – Concept: There are three ADTs in this program (A Calculator, Queue, and Stack ADT). The Calculator ADT will accept requests for various mathematical calculations. It will then use the stack and the queue to store the requested calculations. The Queue will keep a history of all of the operations requested, including undo requests. The Stack will keep the operations that will ultimately be performed. Once the request to perform the calculation comes in, the calculator will use the information on the stack to perform the desired calculations.

All of the mathematical operations and the values they will be working on are stored in a queue in the order that they are requested. They are also stored on a stack at the same time. If the client selects to undo a command or set of commands, they are popped off of the stack. The requests to undo are also stored as commands on the queue. To view the history of all requests, we can view the queue.

Unlike a calculator, your program will need to keep track of precedence as well. What this means is that when a request to multiply occurs on the stack, you will need to create an additional data structure to store the commands that will have to come first when performing the evaluation. Be creative in how this is handled!

Programming – ADTs: The three ADTs must be implemented using the following data structures and at a minimum these functions (your can alter the prototypes as needed). Please keep in mind that because we are creating ADTs, the **user** of the program must not be aware that stacks and queues are being used – this is the job of the application program (your test program)! Make sure to thoroughly test each of the stack and queue functions!

1. **Calculator ADT – Uses the stack and the queue to store the requested calculations**

The calculator creates a running total based on the commands requested, once the calculate request takes place.

   a. int add(double value);
   b. int subtract(double value);
   c. int multiply(double value);
   d. int divide(double value);
   e. int calculate(); //perform all of the requested calculations
   f. int last_operation(calculation &); //tells the client what the last calculation was
   g. int undo(int number); //performs an undo request for the number of times
   h. int undo(calculation &); //undos until a particular calculation request
   i. int view_history();

2. **Queue ADT** – Circular Linked List *** *Keeps a history of all calculations*
   a. int enqueue(const calculation &);
   b. int dequeue(calculation &);
   c. int display() const;

3. **Stack ADT** – Linear Linked List of Arrays *** *Allows for undo operations, by popping operations off of the stack and is used when it is time to perform the necessary calculations*
   a. int push(const calculation &);
   b. int pop();
   c. int peek(calculation &) const; //tells the client what the last calculation was
   d. int display() const;

Programming – Data Structures: The stack should be implemented using a linear linked list of arrays that was discussed in class (and Lab 3). Each element of the array will be a calculation. Have the constructor of the stack class receive the size of the array. It should not be hard-coded into the ADT. The array of calculations must be dynamically allocated.

A calculation may be a class or a struct.

The queue should be implemented using a circular linked list, where the rear pointer points to the most recently added item, and rear->next points to the item at the front of the queue.

Things you should know...as part of your program:

1) Lab #3 is on stacks, queues, and these exact data structures.

2) Do not use statically allocated arrays in your classes or structures. All memory must be dynamically allocated and kept to a minimum!

3) All data members in a class must be private

4) Never perform input operations from your class in CS163

5) Global variables are not allowed in CS163

5) Do not use the String class! (use arrays of characters instead!) You may use the cstring library.

6) Use modular design, separating the .h files from the .cpp files. Remember, .h files should contain the class header and any necessary prototypes. The .cpp files should contain function definitions. **Never "#include" .cpp files!**

7) Use the iostream library for all I/O; do not use stdio.h.

8) Make sure to define a constructor and destructor for your class. Your destructor must deallocate all dynamically allocated memory.

9) Remember that 20% of each program's grade is based on a written discussion of the design. *Take a look at the style sheet which gives instruction on the topics that your write-up needs to cover.*