1 Instructions

When you are finished submit all your work through the MyClasses page for this class. Create a directory called Homework08, put each programming exercise into its own subdirectory of this directory, zip the entire Homework08 directory up into the file Homework08.zip, and then submit this zip file to Homework #8.

Make sure that you:

- Follow the coding and documentation standards for the course as published in the MyClasses page for the class.
- Check the contents of the zip file before uploading it. Make sure all the files are included.
- Make sure that the file was submitted correctly to MyCLasses.

All non-templated class structures are to have their own guarded specification file (.h) and implementation file (.cpp) that has the same name as the class. All templated class structures are to be guarded and written entirely in their (.h) file. No inline coding in the class specification. In addition you must create a make file that compiles and links the project on a Linux computer with a Debian or Debian branch flavor.

2 Programming Exercises

2.1 Program #1

This exercise is to take our linked list structure and use it as the base storage structure for the stack and queue structures. A complete implementation of the LinkedList class is included in the HW08Files.zip file. Look over it and make sure that you understand all of its functionality.

Finish the implementation of the Stack and Queue classes that are started below. Each of the function bodies can be done in a single line of code. The only exception is the overloaded assignment operator which will take two lines. In fact there are some functions that will take zero lines of code in the function body. As usual, no inline coding for this exercise. Use the LinkedList class to its full potential, there is no reason to reinvent the wheel.

```
1 #ifndef STACK_H
2 #define STACK_H
3
4 #include "LinkedList.h"
5 #include <cstdlib>
6 #include <iostream>
7
8 using namespace std;
9
```

```
10 template <class T> class Stack {
11   private:
12     LinkedList<T> stack;
13
14   public:
15     Stack();
16     ~Stack();
17     void displayStack(bool nl = false) const;
18
19     Stack(const Stack &obj);
20     const Stack operator=(const Stack &right);
21
22     void clear();
23     int size();
24     void push(T);
25     T pop();
26     bool isEmpty();
27 };
28
29 #endif
```

```
1  #ifndef QUEUE_H
2  #define QUEUE_H
3
4  #include "LinkedList.h"
5  #include <cstdlib>
6  #include <iostream>
7
8  using namespace std;
9
10 template <class T> class Queue {
11   private:
12     LinkedList<T> queue;
13
14   public:
15     Queue();
16     ~Queue();
17     void displayQueue(bool nl = false) const;
18
19     Queue(const Queue &obj);
20     const Queue operator=(const Queue &right);
21
22     void clear();
23     int size();
24     void enqueue(T);
25     T dequeue();
26     bool isEmpty();
27 };
28
29 #endif
```

I have included the following testing program in the HW08Files.zip file.

```
#include "Queue.h"
#include "Stack.h"
#include <iostream>

using namespace std;

template <class T> void checkEmpty(Queue<T
    >);
template <class T> void checkEmpty(Stack<T
    >);

int main() {
```

```
Stack<double> dstack;

checkEmpty(dstack);
cout << endl;

dstack.push(7);
dstack.push(25);
dstack.push(-4);
dstack.push(3);

dstack.displayStack();
cout << endl;

checkEmpty(dstack);
cout << endl;

try {
    cout << dstack.pop() << endl;
    cout << dstack.pop() << endl;
    cout << endl;
} catch (string s) {
    cout << s << endl;
}

dstack.push(74);
dstack.push(2);
dstack.push(14);

dstack.displayStack();
cout << endl;

try {
    cout << dstack.pop() << endl;
    cout << dstack.pop() << endl;
    cout << dstack.pop() << endl;
    cout << dstack.pop() << endl;
    cout << dstack.pop() << endl;
    cout << dstack.pop() << endl;
} catch (string s) {
    cout << s << endl;
}

dstack.displayStack();
cout << endl;

checkEmpty(dstack);
cout << endl;

for (int i = 0; i < 25; i++)
    dstack.push(i);

dstack.displayStack();
cout << endl;

cout << dstack.size() << endl;
dstack.clear();
cout << dstack.size() << endl;
checkEmpty(dstack);
cout << endl;

dstack.push(3);
dstack.push(5);
dstack.push(7);
dstack.push(9);
dstack.push(11);
dstack.displayStack();
cout << endl;

Stack<double> dstack2;
checkEmpty(dstack2);
cout << endl;

dstack2 = dstack;

dstack.displayStack();
dstack2.displayStack();
cout << endl;

dstack.push(3);
dstack.push(5);
dstack.push(7);
dstack.push(9);
dstack.push(11);

dstack2.push(2);
dstack2.push(4);
dstack2.push(6);
dstack2.push(8);
dstack2.push(10);

dstack.displayStack();
dstack2.displayStack();
cout << endl;

Queue<int> iqueue;

iqueue.enqueue(5);
iqueue.enqueue(7);
iqueue.enqueue(12);
iqueue.enqueue(-23);
iqueue.enqueue(17);

checkEmpty(iqueue);
cout << endl;

iqueue.displayQueue();
cout << endl;

try {
    cout << iqueue.dequeue() << endl;
    cout << iqueue.dequeue() << endl;
} catch (string s) {
    cout << s << endl;
}

iqueue.displayQueue();
cout << endl;

iqueue.enqueue(3);
iqueue.enqueue(1);
iqueue.enqueue(-67);
iqueue.enqueue(32);
iqueue.enqueue(7);

iqueue.displayQueue();
cout << endl;

checkEmpty(iqueue);
```

```
    cout << endl;

    iqueue.clear();

    cout << iqueue.size() << endl;
    checkEmpty(iqueue);
    cout << endl;

    for (int i = 0; i < 25; i++)
        iqueue.enqueue(i);

    iqueue.displayQueue();
    cout << endl;

    Queue<int> iqueue2;

    iqueue2 = iqueue;

    iqueue.displayQueue();
    cout << endl;
    iqueue2.displayQueue();
    cout << endl;

    for (int i = 25; i > 0; i--)
        iqueue2.enqueue(i);

    iqueue.displayQueue();
    cout << endl;
    iqueue2.displayQueue();
    cout << endl;

    return 0;
}

template <class T> void checkEmpty(Queue<T
    > q) {
    if (q.isEmpty())
        cout << "Empty" << endl;
    else
        cout << "Not Empty" << endl;
}

template <class T> void checkEmpty(Stack<T
    > s) {
    if (s.isEmpty())
        cout << "Empty" << endl;
    else
        cout << "Not Empty" << endl;
}
```

If you use it you should get the following output.

```
Empty

3 -4 25 7
Not Empty

3
-4

14 2 74 25 7
14
2
74
25
7
Empty List Exception

Empty

24 23 22 21 20 19 18 17 16 15 14 13 12 11
    10 9 8 7 6 5 4 3 2 1 0
25
0
Empty

11 9 7 5 3
Empty

11 9 7 5 3 11 9 7 5 3
11 9 7 5 3 11 9 7 5 3 10 8 6 4 2 11 9 7 5
    3
Not Empty

5 7 12 -23 17
5
7
12 -23 17
12 -23 17 3 1 -67 32 7
Not Empty

0
Empty

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
    17 18 19 20 21 22 23 24
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
    17 18 19 20 21 22 23 24
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
    17 18 19 20 21 22 23 24
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
    17 18 19 20 21 22 23 24 25 24 23 22 21
     20 19 18 17 16 15 14 13 12 11 10 9 8
    7 6 5 4 3 2 1
```

2.2 Program #2

This exercise is to create a postfix expression evaluator. A postfix expression is one where the operator is after the operands. So for example, the infix expression `(2 + 3) * 4` would be written in postfix as `2 3 + 4 *`. One advantage to postfix is that every expression can be written without the use of parentheses. Another advantage is that a simple stack structure can be used to evaluate the expression. Evaluating an infix expression is more complicated and usually requires mutual recursion to parse the expression into postfix form or better yet an expression tree.

The calculator you are to create is to simply handle addition, subtraction, multiplication, and division (+, –, *, /) of double precision numbers. The way a postfix stack driven calculator works is as follows. When a number is encountered it is pushed onto the stack when an operator is encountered two numbers are popped off the stack, the operation is completed (be careful on order), and the result is pushed back onto the stack. When the expression is completed the value of the expression is on the top of the stack. Four example runs are below.

```
Enter a postfix expression: 2 3 - 4 *
Value = -4

Enter a postfix expression: 3 4 5 * +
Value = 23

Enter a postfix expression: 3 4 5 + *
Value = 27

Enter a postfix expression: 5 4 3 + /
Value = 0.714286
```

A few things that will come in handy. The first is a simple tokenizer, the program below uses the Queue class you constructed in the previous exercise that uses our linked list to make a list of each operand and operator separately. That way you can tokenize the list and then simply run through the list (dequeue) to extract each operand and operator. Compile and run the program on different inputs to see how the tokenizing works. Notice that you need to have a space between each token. So an input of `2 3 + 4 *` will parse correctly but an input of `2 3+4*` will not. Also, if there is more than one space between the tokens then the parser will produce empty strings in the list. These can, of course, be skipped when reading through the list during the evaluation stage. Be careful with the order of operands, the expression `3 4 -` means $3 - 4 = -1$ not $4 - 3 = 1$.

```
1 #include "Queue.h"
2 #include <iostream>
3 #include <sstream>
4
5 using namespace std;
6
7 int main() {
```

```
8       // string input = "234 32 + 7 *";
9       // Must have spaces between tokens.
10
11      string input;
12      cout << "Enter a postfix expression: ";
13      getline(cin, input);
14
15      Queue<string> tokens;
16      stringstream inputstream(input);
17      string token;
18
19      // Tokenizing w.r.t. space ' '
20      while (getline(inputstream, token, ' '))
21          tokens.enqueue(token);
22
23      // Printing the token queue for testing.
24      try {
25          while (!tokens.isEmpty())
26              cout << tokens.dequeue() << endl;
27      } catch (string s) {
28          cout << s << endl;
29      }
30  }
```

A run of the program is below.

```
Enter a postfix expression: 234 32 + 7 *
234
32
+
7
*
```

Another function that will be useful is the `stod` function. The `stod` function converts a string that looks like a floating point number into a double precision floating point number. So if the string variable `token` holds a string that looks like a number then the line,

```
double val = stod(token);
```

will convert it to a numeric value and store that value into the variable val.

2.3 Program #3

Rewrite the above program using the STL stack and queue classes.

2.4 Program #4

A string of characters has balanced parentheses if each right parenthesis occurring in the string is matched with a preceding left parenthesis, in the same way that each right brace in a C++ program is matched with a preceding left brace. A code string in a computer

language may use more than one type of delimiter to bracket information into "blocks." For example, A string may use braces { }, parentheses ( ), and brackets [ ] as delimiters. A string is properly delimited if each right delimiter is matched with a preceding left delimiter of the same type in such a way that either the resulting blocks of information are disjoint, or one of them is completely nested within the other. These types of functions are obviously used in compilers but they are also found in applications like IDEs, when you put the cursor on a brace or parentheses the IDE will highlight the corresponding brace or parentheses. Write a program that uses an STL stack to check whether a string containing braces, parentheses, and brackets is properly delimited. Three sample runs of the program are below.

```
Enter expression: for (count = 2; count < MAX; count += 2)
Delimiters are balanced.

Enter expression: {[3*(4+x) + 5/(x-1) + ((y^2-2)-(4-x^3))] + [[t+1]-4]/5} * {{4-x^6}+[7*(3+x*y)]}
Delimiters are balanced.

Enter expression: {[3*(4+x) + 5/(x-1) + ((y^2-2)-(4-x^3))]) + [[t+1]-4]/5} * {{4-x^6}+[7*(3+x*y)]}
Delimiters are not balanced.
```

As a hint on how you cam proceed here, when a left delimiter is encountered push it onto a stack. When a right delimiter is encountered pop the stack and make sure the left delimiter that was popped matches the right one being read off the string. If so proceed and if not you know that the delimiters are not balanced. Think about what should happen when the string has been completely read and the delimiters are balanced.

2.5 Program #5

This exercise is to construct a templated priority queue using the STL vector class as the underlying storage structure. The declaration of the class is below. Complete the implementation of this class. The priority of a node will be an integer type and we will use the convention that the higher number will represent the higher priority, so priority 3 will be higher than priority 1.

The way a priority queue works is just like a queue, first in first out, but the items with a higher priority are dequeued first, in order. So if there are items in the queue with priorities 1, 2, and 3, then those of priority 3 are dequeued first, in the order they are in the queue. Then we dequeue those of priority 2, in order, and finally those with priority 1, in order.

```
1  #ifndef PRIORITYQUEUE_H
2  #define PRIORITYQUEUE_H
3
4  #include <iostream>
5  #include <vector>
6
7  /*
8  Templated Priority Queue class that uses the STL vector as the underlying storage
       structure.
9
10 Note: The template class T must
11   1. overloaded =
12   2. Have a default constructor
13   3. Overloaded >>
14   4. Copy Constructor
```

```
 */

using namespace std;

// Node data type that holds the data and priority of the item.

template<class T>
class PQNode {
public:
    T data;
    int priority;

    PQNode(T Data) {
        data = Data;
        priority = 1;
    }

    PQNode(T Data, int Priority) {
        data = Data;
        if (Priority < 1)
            Priority = 1;

        priority = Priority;
    }
};

template<class T>
class PriorityQueue {
private:
    vector<PQNode<T> > queue; // The vector holding all of the items in the queue.

public:
    // Constructors and Destructor
    PriorityQueue();
    PriorityQueue(const PriorityQueue &obj);
    ~PriorityQueue();

    // Acessors and Mutators
    void enqueue(T Data, int Priority = 1);
    T dequeue();

    // Other Functions
    void print();
    bool isEmpty();
    void clear();
    int size();
};

#endif
```

- enqueue(T Data, int Priority = 1) — Creates a queue node with the default priority of 1 and loads in the Data parameter to the data of the node. Finally, pushes the node onto the back of the queue.
- dequeue() — removes the first element of the highest priority from the queue and returns the data value. If the queue is empty the templated element's default value is returned. Not the greatest way to handle the situation of an empty queue, in practice we would implement an exception structure for this class to use.
- print() — Prints the contents of the queue to the screen, each item is given its own

line and both the data contents and priority are printed.

- isEmpty() — Returns true if the queue is empty and false otherwise.
- clear() — Removes all contents of the queue.
- size() — Returns the current number of elements in the queue.

As a test, if you create and run the following program,

```
#include <iostream>
#include <vector>

#include "PriorityQueue.h"

using namespace std;

void println(string s = "") { cout << s << endl; }

int main() {
    PriorityQueue<int> queue;

    queue.enqueue(7);
    queue.enqueue(5);
    queue.enqueue(15, 3);

    queue.print();

    int t = queue.dequeue();
    cout << t << endl;
    queue.print();

    t = queue.dequeue();
    cout << t << endl;
    queue.print();

    t = queue.dequeue();
    cout << t << endl;
    queue.print();

    t = queue.dequeue(); //  Too Far
    cout << t << endl;
    queue.print();

    for (int i = 0; i < 10; i++)
        queue.enqueue(i, i);

    queue.print();

    cout << queue.size() << endl;

    queue.clear();

    cout << queue.size() << endl;

    for (int i = 0; i < 10; i++)
        queue.enqueue(i, i % 3 + 1);

    queue.print();

    while (!queue.isEmpty()) {
        t = queue.dequeue();
```

```
53          cout << t << " ";
54      }
55      println();
56
57      PriorityQueue<string> Squeue;
58
59      Squeue.enqueue("Sam", 2);
60      Squeue.enqueue("John");
61      Squeue.enqueue("Jack");
62
63      string strarr[] = {"considerable", "substantial",   "pronounced",
64                         "significant",  "appreciable",   "serious",
65                         "exceptional",  "extraordinary", "tremendous",
66                         "stupendous",   "unlimited",     "boundless",
67                         "cosmic"};
68
69      for (int i = 0; i < 13; i++) {
70          Squeue.enqueue(strarr[i], i % 3 + 1);
71      }
72
73      Squeue.print();
74      println();
75
76      while (!Squeue.isEmpty()) {
77          string c = Squeue.dequeue();
78          cout << c << " // ";
79      }
80      println();
81
82      return 0;
83  }
```

You should get the following output.

```
7  ---  1
5  ---  1
15  ---  3
15
7  ---  1
5  ---  1
7
5  ---  1
5
22003
0  ---  1
1  ---  1
2  ---  2
3  ---  3
4  ---  4
5  ---  5
6  ---  6
7  ---  7
8  ---  8
9  ---  9
10
0
0  ---  1
1  ---  2
2  ---  3
3  ---  1
4  ---  2
5  ---  3
6  ---  1
7  ---  2
```

```
8  ---  3
9  ---  1
2 5 8 1 4 7 0 3 6 9
Sam  ---  2
John  ---  1
Jack  ---  1
considerable  ---  1
substantial  ---  2
pronounced  ---  3
significant  ---  1
appreciable  ---  2
serious  ---  3
exceptional  ---  1
extraordinary  ---  2
tremendous  ---  3
stupendous  ---  1
unlimited  ---  2
boundless  ---  3
cosmic  ---  1

pronounced // serious // tremendous // boundless // Sam // substantial // appreciable
    // extraordinary // unlimited // John // Jack // considerable // significant //
    exceptional // stupendous // cosmic //
```