COSC220 Spring 2021 – Project # 2 Due: 4/11/2021

PROJECT SPECIFICATIONS:

You will be developing a mini-STL library contains the following containers.

- miniStackVT (a miniStack implemented using STL <vector>)
- miniStackDA (a miniStack implemented using dynamic array)
- miniStackLT (a miniStack implemented using STL <list>)
- miniStackDL (a miniStack implemented using doubly linked list)
- miniQueueVT (a miniQueue implemented using STL <vector>)
- miniQueueDA (a miniQueue implemented using dynamic array)
- miniQueueLT (a miniQueue implemented using STL <list>)
- miniQueueDL (a miniQueue implemented using doubly linked list)

Exception handling needs to be incorporated in your project. You need to implement a driver program with menu options to allow users to test all your containers by:

- adding/remove elements to/from a container,
- displaying the content AND size of a container,
- displaying the value of the front OR top element in a container,

1. The header file "miniStackXX.h" should have the following (note: XX should be replaced by VT, DA, LT, or DL. All the implementation should be in file "miniStackXX.cpp".

```
#ifndef MINISTACKXX_H
#define MINISTACKXX_H

template <class DataType>
class miniStackXX{
    private:
        /* ONE of the following depending on XX
            vector<DataType> content; // for VT
            list<DataType> content; // for LT
            DataType *content; // for DA
            DNode<T>* head; // DL
            You may also add extra data members if needed
         */
        int elementCount;              // number of elements in the stack


    public:
        miniStackXX();              // constructor create a stack with size zero
        ~miniStackXX();                 // destructor
        int size() const;       // return the number of elements in the stack
        bool IsEmpty() const;       // check empty stack
        void Push (const DataType & ); // push a node in the stack
        void PrintStack() const;    // print all stack content (TOP to BOTTOM)
        void  Pop();            // Pop an element from the top of stack.
                                // Issue exception if the stack is empty
        DataType& TopStack(); // Return Data from the top of the stack.
                                // Issue exception if the stack is empty
        const DataType& TopStack() const; // Return Data from the top of the stack.
                                // Issue exception if the stack is empty


};
#include "miniStackXX.cpp"
#endif
```

2. The header file “miniQueue.h” should have the following (note: XX should be replaced by VT, DA, LT, or DL. All the implementation should be in file “miniQueue.cpp”.

```
#ifndef MINIQUEUE_H
#include MINIQUEUE_H

template <class DataType>
class miniQueueXX
{
    private:

       /* ONE of the following depending on XX
           vector<DataType> content; // for VT
           list<DataType> content; // for LT
           DataType *content; // for DA
           DNode<T>* head; // DL
           You may also add extra data members if needed
        */
      int elementCount;;  // element count in a Queue
    public:
       miniQueueXX(); //class constructor - queue size set to zero
       ~miniQueueXX(); //class destructor - return memory used by queue elements
       void enqueue(const DataType &); //add an item to the back of the queue
       void dequeue(); //remove the first item from the queue and return its value
                          // Issue exception if the queue is empty
       DataType& front(); //return the value of the front item in the queue
                          // Issue exception if the queue is empty
       const DataType& front() cosnt; //return the value of the front item in the
    queue                 // Issue exception if the queue is empty

       void PrintQueue() const;    // print all queue content (front to back)
       bool IsEmpty() const; //returns true if there are no elements in the queue
       int size() const;       // return the number of elements in the queue
};
#include "miniQueue.cpp"
#endif
```

OTHER REQUIREMENTS

1. Make sure to start working on your project early and make steady progress.
2. Make sure to thoroughly test your classes using a driver program before you hand in your program.
3. Make sure that you give proper names to your variables, program files.
4. Make sure that your program is nicely indented and has meaningful header and inline comments.

WHAT TO TURN IN

Upload your source code (.cpp and .h files) to myClass, put your testing plan and all the output generated from your testing in a pdf file and upload it to myClass as well. The instructor will test your program in the Linux lab during grading. So make sure to develop your project in the Linux environment.