Brandon Crenshaw

CS 3358

Coding Assignment 2:

Implement a stack and a queue class with linked-list Assignment

Learned Note:

The main thing I learned is what is and how to use templates. Templates were not a part of my previous instructions / prerequisite course. The Concept of templates seemed easy enough but I ran into issues when trying to implement them through out the project.

The thing I had to learn about templates is that they are not themselves classes or functions, but merely… templates of class or function. This means that you can't separate them into declarations (prototypes) and definitions. All of that has to be done in one place. So placing prototypes in headers and definition in cpp files when doing templates will not work.

The other thing about about working with templates is understanding when you're dealing with a template and when you're dealing with on of its classes or functions. The <int> suffix (for example) indicated you're dealing with an actual class or function. This knowledge will make error message x100 more understandable when you're told that something you're looking directly at doesn't exist.

Source Code

linkedList.h

```
/* *************************************************
*  Name: Brandon Crenshaw
*  Assignment: #2 - Stack / Queue / Linked Lists
*  Purpose: The linked list class for the project.
*
************************************************* */

#ifndef SNODE_H
#define SNODE_H

template <typename T>
class SNode {                           // basic node class
    protected:

    public:
        SNode() : next(NULL) {}
        SNode<T> * next;
        T data;
};
#endif // SNODE_H


#ifndef SLLIST_H
#define SLLIST_H

template <typename T>
class LinerSinglyLinkedList : public SNode<T> {
    private:
        SNode<T> * head;

    public:
        // CONSTRUCTORS and DESTRUCTOR
        LinerSinglyLinkedList() : head(NULL) { }
        LinerSinglyLinkedList(LinerSinglyLinkedList<T>& lObj) : head(NULL)
```

```
{ copyList(&lObj); }  // copy constructor
        ~LinerSinglyLinkedList() { deleteList(); }

        // METHODS
        bool isEmptyList(){ return head == NULL; }      // simple getter method

        bool addElmAtFront(T *inData){                  // create node make it new head
            bool isAdded = true;
            SNode<T> * newHead = new SNode<T>;
            newHead->data = *inData;
            newHead->next = head;
            head = newHead;
            return isAdded;
        }

        bool insertElmAtEnd(T * inData){                // create node, traverse to tail, add there
            SNode<T> * currNode = head;

            // Make a new node
            SNode<T> * newNode = new SNode<T>;
            newNode->data = *inData;

            //Add to the tail
            if(head == NULL) head = newNode;
            else {
                while(currNode->next != NULL){          // to the end of the linked list.
                    currNode = currNode->next;
                }
            currNode->next = newNode;                   // appends to the end of the linked list.
            }
            return true;
        }


        bool removefromFront(T * inData){       // make 2nd node the head, delete previous head
            bool wasRead = false;
            if (head == NULL){ }
            else {
                wasRead = true;
                SNode<T> * currNode = head;
                *inData = currNode->data;
                head = currNode->next;
                delete currNode;
                currNode = NULL;
            }
            return wasRead;
        }

        bool removefromFront(){        // overridde for delete all that doesn't write node data
            bool wasRead = false;
            if (head == NULL){}
            else {
                wasRead = true;
                SNode<T> * currNode = head;
                if(head->next == NULL){ head = NULL;}
                else { head = head->next; }
                delete currNode;
                currNode = NULL;
            }
            return wasRead;
        }

        bool nextElm(T * inData){     // reports what data is in the head node
            bool wasRead = false;
            if(head == NULL){ }
            else{ *inData = head->data; }
            return wasRead;}

        void deleteList(){            // removes all nodes starting from the head end
            while( head != NULL ) { removefromFront();}
        }

        void copyList(LinerSinglyLinkedList * targetLL) {
        // copies each node one-by-one into another linked-list
            SNode<T> * currNode = targetLL->head;
            T dataCpy;
            do{
                dataCpy = currNode->data;
                insertElmAtEnd(&dataCpy);
                currNode = currNode->next;
            } while( currNode != NULL );
        }
};

#endif // SLLIST_H
```

stack.h

```
/* ************************************************
*  Name: Brandon Crenshaw
*  Assignment: #2 - Stack / Queue / Linked Lists
*  Purpose: The class for the stack.
*
************************************************ */

#include "linkedList.h"

#ifndef STACK_H
#define STACK_H

template <typename T>
class Stack : public SNode<T>{
    private:
        LinerSinglyLinkedList<T> stkList;

    public:
        // Constructor & Destructor
        Stack() : stkList() {}
        ~Stack<T>(){ stkList.deleteList(); }

        // No descriptions: These methods only call the linked list methods.
        bool isEmpty(){ return stkList.isEmptyList(); }
        bool push(T * nodeData){ return stkList.addElmAtFront(nodeData); }
        bool top(T * nodeData){return stkList.nextElm(nodeData);}
        bool pop(T * nodeData){ return stkList.removefromFront(nodeData); }
};

#endif // STACK_H
```

queue.h

```
/* ************************************************
*  Name: Brandon Crenshaw
*  Assignment: #2 - Stack / Queue / Linked Lists
*  Purpose: The class for the stack.
*
************************************************ */

#include "linkedList.h"

#ifndef QUEUE_H
#define QUEUE_H

template <typename T>
class Queue : public SNode<T> {
    private:
        LinerSinglyLinkedList<T> qlist;

    public:
        Queue() { }
        ~Queue(){ qlist.deleteList(); }

        // No descriptions: These methods only call the linked list methods.
        bool isEmpty(){ return qlist.isEmptyList(); }
        bool insert(T * nodeData){ return qlist.insertElmAtEnd(nodeData); }
        bool remove(T * nodeData){ return qlist.removefromFront(nodeData); }
        bool next(T * nodeData){ return qlist.nextElm(nodeData);}
};

#endif  // QUEUE_H
```

test.cpp

```
/* **************************************************
*  Name: Brandon Crenshaw
*  Assignment: #2 - Stack / Queue / Linked Lists
*  Purpose: This is ADT Tester of the project.
*
************************************************* */
#include <iostream>
#include"stack.h"
#include "queue.h"


const int NODE_CNT = 5;

void printForEmpty(bool input){
/* **********************************
 * Converts the boolean 0 and 1 to string outputs.
 *
 * @param bool input : used to communicate if ADTs are empty
 * @return    (void) : no return
 * @exception     na : na
 * @note
 * **********************************/
    if(input) std::cout << "   The object is empty\n";
    else{std::cout << "   The object holds data.\n";}
}

//TEMPLATES FOR EACH ADT TEST
template<typename T>
void stackTester(){
/* **********************************
 * This function is tests if stack is empty and what happens when trying to pop an empty
 * stack. It then fills the stack, copies it, and completely depops the copy.
 * It finishes by testing if both stack are empty.
 *
 * @param      na : na
 * @return (void) : no return
 * @exception  na : na
 * @note            designed for numerical and char datatypes; for int, 0 is read from empty entries
 * **********************************/
    Stack<T> testStack;
    T dataHolder;

    printForEmpty(testStack.isEmpty());
    std::cout << "popping an empty stack..."<<std::endl;
    std::cout << "successful pop?: " << testStack.pop(&dataHolder) <<std::endl;      // test pop of empty stack
    std::cout << "data written in the attempt: \"" << dataHolder << "\"" <<std::endl;// make sure no data copied

    std::cout << "\npopulating stack..."<<std::endl;
    for(int i = NODE_CNT; i > 0; i--){                                       // push data onto the stack
        dataHolder = i + 64;
        testStack.push(&dataHolder);
    }
    printForEmpty(testStack.isEmpty());

    std::cout << "copying the stack..."<<std::endl;
    Stack<T> queCopy = testStack;                                            // copy the stack

    std::cout << "popping the stack copy...\n" << std::endl;
    for(int i = 0; i < NODE_CNT; i++){                                       // empty the 2nd stack
        queCopy.top(&dataHolder);
        std::cout << "Top: " << dataHolder;
        queCopy.pop(&dataHolder);
        std::cout << ",  Pop: " << dataHolder << std::endl;
    }

    std::cout << "\nthe original stack" << std::endl;
    printForEmpty(testStack.isEmpty());                                      // check that first stack not empty
    std::cout << "the copied stack" << std::endl;
    printForEmpty(queCopy.isEmpty());                                        // check that 2nd stack is empty
}
```

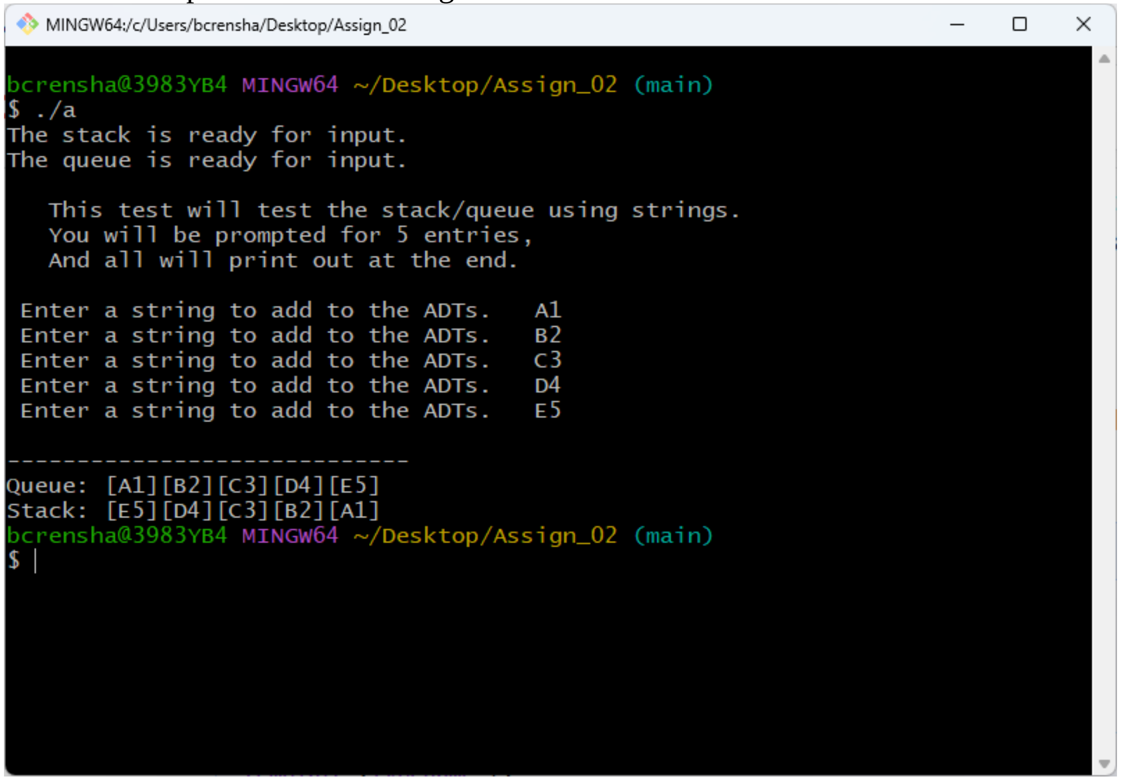
```
template<typename T>
void queueTester(){
/* **********************************
 * This function is tests if queue is empty and what happens when trying to remove from
 * the empty queue. It then fills the queue, copies it, and completely depops the copy.
 * It finishes by testing if both queues are empty.
 *
 * @param      na : na
 * @return (void) : no return
 * @exception  na : na
 * @note            designed for numerical and char datatypes; for int, 0 is read from empty entries
 * **********************************/
    Queue<T> testQueue;
    T dataHolder;

    printForEmpty(testQueue.isEmpty());
    std::cout << "removing from an empty queue..."<<std::endl;
    std::cout << "successful removal?: " << testQueue.remove(&dataHolder) <<std::endl;
    std::cout << "data written in the attempt: \"" << dataHolder << "\"" <<std::endl;

    std::cout << "\npopulating queue..."<<std::endl;
    for(int i = NODE_CNT; i > 0; i--){                                              // filling queue
        dataHolder = i + 64;
        testQueue.insert(&dataHolder);
    }
    printForEmpty(testQueue.isEmpty());

    std::cout << "copying the queue..."<<std::endl;
    Queue<T> queCopy = testQueue;                                                   // copying queue

    std::cout << "removing from the copied queue...\n" << std::endl;
    for(int i = 0; i < NODE_CNT; i++){                                              // emptying 2nd queue
        queCopy.next(&dataHolder);
        std::cout << "Next: " << dataHolder;
        queCopy.remove(&dataHolder);
        std::cout << ",  Removed: " << dataHolder << std::endl;
    }

    std::cout << "\nthe original queue" << std::endl;
    printForEmpty(testQueue.isEmpty());                                             // make sure 1st queue not
empty
    std::cout << "the copied queue" << std::endl;
    printForEmpty(queCopy.isEmpty());                                               // make sure 2nd queue is
empty
}


int not_main(){                                        //changed to 'not_main' so the other code could compile
/* ****************************************************************************************************************
 * This function is the application driver. It tests the stack ADT with integers and
 * then with chars. It then tests the queue ADT with integers and then chars as well.
 * vehicle instances stored in memory.
 *
 * @param na : na
 * @return (int) : application exit code 0
 * @exception na : na
 * @note na
 * ****************************************************************************************************************/
    std::cout << "-------------------------------------------------------- STACK TESTING (w/ INTEGERS)" << std::endl;
    stackTester<int>();
    std::cout << "-------------------------------------------------------- QUEUE TESTING (w/ INTEGERS)" << std::endl;
    queueTester<int>();
    std::cout << "----------------------------------------------------------- STACK TESTING (w/ CHARS)" << std::endl;
    stackTester<char>();
    std::cout << "----------------------------------------------------------- QUEUE TESTING (w/ CHARS)" << std::endl;
    queueTester<char>();
    std::cout << "--------------------------------------------------------------------- END OF TESTING" << std::endl;
    return 0;
}
```

main.cpp (uses class Test)

```
/* ************************************************
*  Name: Brandon Crenshaw
*  Assignment: #2 - Stack / Queue / Linked Lists
*  Purpose: This is ADT Tester of the project.
*
************************************************* */
#include <iostream>
#include <string>
#include "stack.h"
#include "queue.h"

template<typename T>
class Test : public Stack<T>, public Queue<T> {
    private:
             const int MAX_ENTRIES = 5;
             int count;
             Stack<T> stack;
             Queue<T> queue;

    public:
             Test() : count(0){
                   if(stack.isEmpty()){ std::cout << "The stack is ready for input." << std::endl;}
                   else { std::cout << "Error: The stack is not empty." << std::endl;}
                   if(stack.isEmpty()){ std::cout << "The queue is ready for input." << std::endl;}
                   else { std::cout << "Error: The queue is not empty." << std::endl; }
                   std::cout << "\n   This test will test the stack/queue using strings.\n";
                   std::cout << "   You will be prompted for "<< MAX_ENTRIES << " entries,\n";
                   std::cout << "   And all will print out at the end.\n" << std::endl;
             }

             bool StoreData();
             void printOut();
             void runTest();
};

template<typename T> bool Test<T>::StoreData(){
/* *********************************
 * Stores data from user in the stack/queue.
 *
 * @param      na : none
 * @return (bool) : indicates successful storage
 * @exception  na : na
 * @note
 * *********************************/
       T userInput;

       std::cout << " Enter a string to add to the ADTs.   ";
       std::cin >> userInput;
       return (stack.push(&userInput) && queue.insert(&userInput));
}

template<typename T> void Test<T>::printOut(){
/* *********************************
 * Prints out all data from stack/queue (empties the ADT)
 *
 * @param      na : none
 * @return (void) : no return
 * @exception  na : na
 * @note
 * *********************************/
       T strBuffer;

       std::cout << "\n-----------------------------" << std::endl;
       std::cout << "Queue: ";
       while ( ! queue.isEmpty() ){
             queue.remove(&strBuffer);
             std::cout << "[" << strBuffer<< "]";
       }
	std::cout << std::endl;
	std::cout << "Stack: ";
	while ( ! stack.isEmpty() ){
		stack.pop(&strBuffer);
		std::cout << "[" << strBuffer<< "]";
	}
}

template<typename T> void Test<T>::runTest (){
	for (int i = 0; i < MAX_ENTRIES; i++){ Test<T>::StoreData(); }
	Test<T>::printOut();
}


int main(){
/* ********************************************************************************
 * This function is the application driver. It tests the stack ADT with user inputs
 *
 * @param     na : na
 * @return (int) : application exit code 0
 * @exception na : na
 * @note na
 * ********************************************************************************/
	Test<std::string> tester1;
	tester1.runTest();
    return 0;
}
```

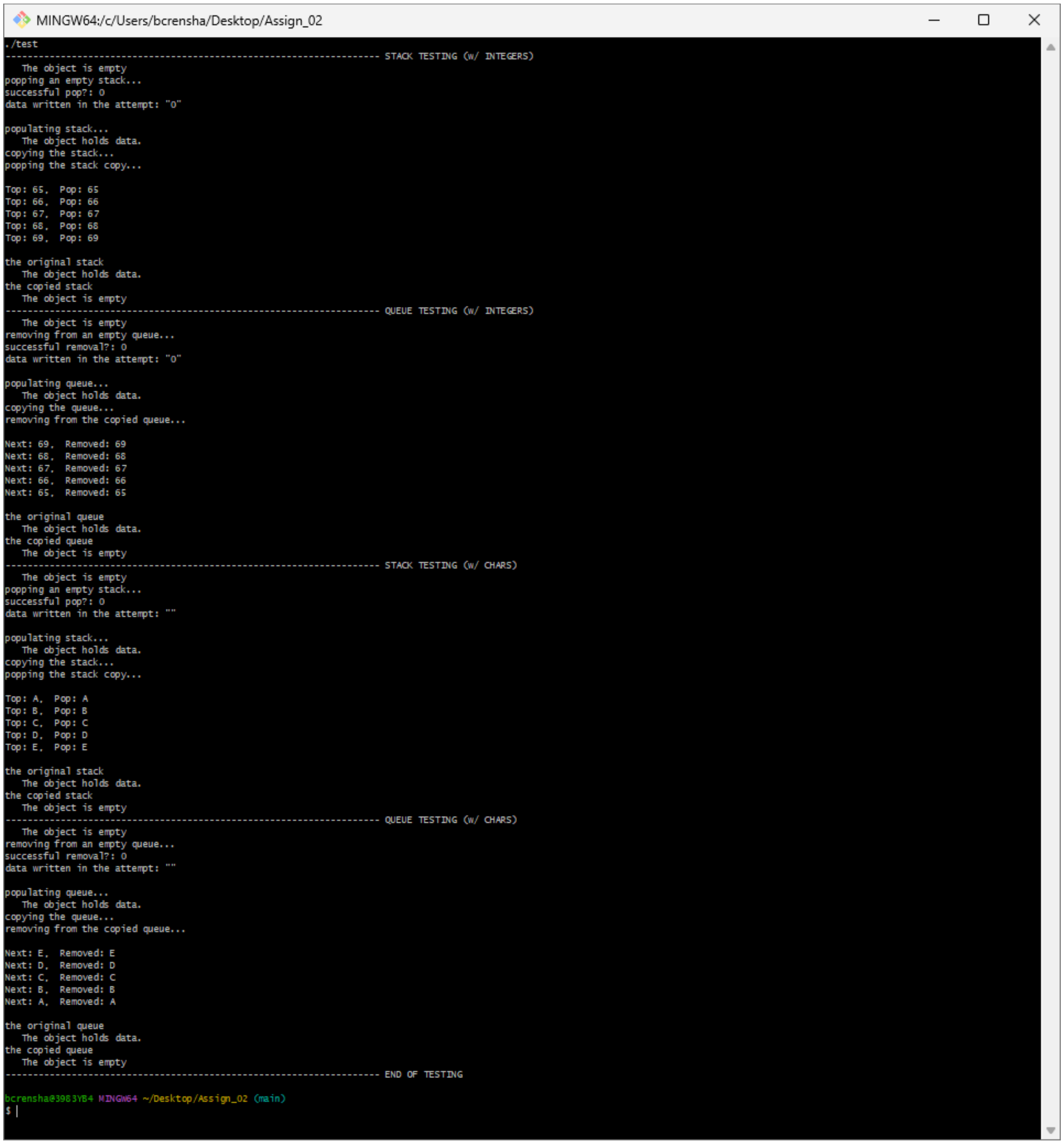
Screenshots

This is the output of the code using the Test Class



```
bcrensha@3983YB4 MINGW64 ~/Desktop/Assign_02 (main)
$ ./a
The stack is ready for input.
The queue is ready for input.

   This test will test the stack/queue using strings.
   You will be prompted for 5 entries,
   And all will print out at the end.

 Enter a string to add to the ADTs.   A1
 Enter a string to add to the ADTs.   B2
 Enter a string to add to the ADTs.   C3
 Enter a string to add to the ADTs.   D4
 Enter a string to add to the ADTs.   E5

-----------------------------
Queue: [A1][B2][C3][D4][E5]
Stack: [E5][D4][C3][B2][A1]
bcrensha@3983YB4 MINGW64 ~/Desktop/Assign_02 (main)
$ |
```

This is the output of my original test code.



MINGW64:/c/Users/bcrensha/Desktop/Assign_02

```
./test
-------------------------------------------------------------------- STACK TESTING (w/ INTEGERS)
   The object is empty
popping an empty stack...
successful pop?: 0
data written in the attempt: "0"

populating stack...
   The object holds data.
copying the stack...
popping the stack copy...

Top: 65,  Pop: 65
Top: 66,  Pop: 66
Top: 67,  Pop: 67
Top: 68,  Pop: 68
Top: 69,  Pop: 69

the original stack
   The object holds data.
the copied stack
   The object is empty
-------------------------------------------------------------------- QUEUE TESTING (w/ INTEGERS)
   The object is empty
removing from an empty queue...
successful removal?: 0
data written in the attempt: "0"

populating queue...
   The object holds data.
copying the queue...
removing from the copied queue...

Next: 69,  Removed: 69
Next: 68,  Removed: 68
Next: 67,  Removed: 67
Next: 66,  Removed: 66
Next: 65,  Removed: 65

the original queue
   The object holds data.
the copied queue
   The object is empty
-------------------------------------------------------------------- STACK TESTING (w/ CHARS)
   The object is empty
popping an empty stack...
successful pop?: 0
data written in the attempt: ""

populating stack...
   The object holds data.
copying the stack...
popping the stack copy...

Top: A,  Pop: A
Top: B,  Pop: B
Top: C,  Pop: C
Top: D,  Pop: D
Top: E,  Pop: E

the original stack
   The object holds data.
the copied stack
   The object is empty
-------------------------------------------------------------------- QUEUE TESTING (w/ CHARS)
   The object is empty
removing from an empty queue...
successful removal?: 0
data written in the attempt: ""

populating queue...
   The object holds data.
copying the queue...
removing from the copied queue...

Next: E,  Removed: E
Next: D,  Removed: D
Next: C,  Removed: C
Next: B,  Removed: B
Next: A,  Removed: A

the original queue
   The object holds data.
the copied queue
   The object is empty
-------------------------------------------------------------------- END OF TESTING

bcrensha@3983YB4 MINGW64 ~/Desktop/Assign_02 (main)
$ |
```