COP 4530: Data Structures
Project 3

You are not allowed to use the Internet. You may only consult approved references*.
This is an individual project.
This policy is strictly enforced.

You must submit a **hard copy** of all of the items requested below. You must also submit your *code*† to Canvas.

For full credit, the code that is submitted must:

- Use the specified signature, if applicable.
- Be implemented in a file using the specified file name, if applicable.
- Be correct (i.e., it must always return the correct result).
- Be efficient (i.e., it must use the minimum amount of time and the minimum amount of space necessary to be a correct implementation).
- Be readable and easy to understand. You should include comments to explain when needed, but you should not include excessive comments that makes the code difficult to read.
  - Every class definition should have an accompanying comment that describes what is for and how it should be used.
  - Every function should have declarative comments which describe the purpose, preconditions, and postconditions for the function.
  - In your implementation, you should have comments in tricky, non-obvious, interesting, or important parts of your code.
  - Pay attention to punctuation, spelling, and grammar.
- Follows ALL coding guidelines from section 1.3 of the textbook. Additional coding guidelines:
  - No magic numbers. Use constants in place of hard-coded numbers. Names of constants must be descriptive.
  - No line of the text of your source code file may have more than 80 characters (including whitespace).
  - All header files should have #define guards to prevent multiple file inclusion. The form of the symbol name should be <FILENAME>_H_
  - Do not copy and paste code. If you need to reuse a section of code, then write a function that performs that code.
  - Define functions inline only when they are simple and small, say, 5 lines or less
  - Function names, variable names, and filenames must be descriptive. Avoid abbreviation.
  - Use only spaces (no tabs), and indent 3 spaces at a time.
- Compile and run on the C4 Linux Lab machines (g++ compiler, version 4.8.2). *The shell script and makefile that I will use to compile and run your code will be posted on Canvas. Please note that I may use my own* `main.cpp` *file to test the code you submit.*
- Have no memory leaks.

*The list of approved references is posted on Canvas. You must cite all references used.
†Your code must compile and run on the C4 Linux Lab machines

Project Description

In the *Josephus problem*, a group of soldiers is surrounded by the enemy, and one soldier is to be selected to ride for help. The selection is made in the following manner: An integer n and a soldier are selected randomly. The soldiers are arranged in a circle and they count off beginning with the randomly selected soldier. When the count reaches n, that soldier is removed from the circle, and the counting begins again with the next soldier. This process continues until only one soldier remains, who is the (un)fortunate one selected to ride for help.

Project Tasks

Create a file named `circularLinkedList.h` with the following class definition:

```
template <class T>
class List {
   private:
      class Node {
         public:
            T data;
            Node * next;
      };
   public:
      List();
      List(const List&);

      ~List();

      int getSize() const;

      bool isEmpty() const;
      bool insert(const T&, const int&);
      bool remove(const int&);

      T runJospehusAlgorithm();

      const List& operator=(const List&);

      void display(ostream&) const;
      friend ostream& operator<< <>(ostream&, const List<T>&);
   private:
      Node * _first; // pointer to the first element of the linked list
};
```

To earn full credit you are not permitted to use a size member variable.
Hint: Start with the pointer based linked list implementation that we developed in lecture and modify the class so that it holds a circular linked list.

1. Implement each of the functions of the list class definition to create a singly linked circular list:
(*Implementation of the functions should also be located in the* `circularLinkedList.h` *file*)
   (a) [1 point] Implement the default constructor to create an empty linked list.
   (b) [5 points] Implement the copy constructor. *For full credit you are not permitted to call the* `getSize()` *function - you must check if you have reached the end of the list while you are traversing the list to copy elements.*
   (c) [5 points] Implement the destructor. *For full credit you are not permitted to call the* `getSize()` *function - you must check if you have reached the end of the list while you are traversing the list to delete elements.*
   (d) [10 points] Implement the `getSize()` function.
   (e) [5 points] Implement the `isEmpty()` function. *For full credit you are not permitted to call the* `getSize()` *function.*
   (f) [10 points] Implement the `insert()` function. *For full credit you are not permitted to call the* `getSize()` *function - you must check if you have reached the end of the list while you are traversing to the insertion position.*
   (g) [10 points] Implement the `remove()` function. *For full credit you are not permitted to call the* `getSize()` *function - you must check if you have reached the end of the list while you are traversing to the deletion position.*
   (h) [5 points] Implement the assignment operator. *For full credit you are not permitted to call the* `getSize()` *function - you must check if you have reached the end of the list while you are traversing the list to copy elements.*
   (i) [10 points] Implement the `display()` function. *For full credit you are not permitted to call the* `getSize()` *function - you must check if you have reached the end of the list while you are traversing the list to display the elements.*
   (j) [1 point] Implement << operator. *Hint: refer to the pointer based linked list code*

   You should implement additional (private) helper functions as needed. Helper functions must also be located in the `circularLinkedList.h` *file.*

2. [30 points] Implement the function to determine the node selected by the Josephus Algorithm.
For full credit:
   1. Your function cannot modify the contents of the list object calling the function (*Hint: make a copy of the list and run the algorithm on the copied list.*)
   2. For each iteration of the algorithm (*i.e., each time you select an item to remove from the list*) you must
      - Randomly select a start node, from the remaining nodes, such that each of the nodes of the remaining linked list are equally likely to be selected.
      - Randomly select a count between 0 and two times the size of the original list that you will walk to determine which node will be removed.
   3. Implement the algorithm as described in the project description.

3. Write a main function to test each of the list class functions.
   (a) [3 points] Your circular list must work with the test program that we developed for our other list implementations. (*You should add a public* `getCapacity()` *function that always returns a negative value*)
   (b) [5 points] Write your own function(s) in `main.cpp` to test your list class and your `runJospehusAlgorithm()` function.

   Note that I will be creating my own `main.cpp` *file to test your code for the above tasks.*