Assignment Program #4: Double Linked List

OBJECTIVES:

At the end of this assignment students will

- Create the classes to implement a double linked list
- Implement a driver for using the double linked list.

PROBLEM STATEMENT

For this project your will implement a double linked circular linked list to simulate a game of musical chairs. You will build the doubly-linked class as well as the simulation.

Your driver will be tested against a series of input files. The input file will include a list of students playing the game. Then a list of numbers for how long the song plays each round. Use this number to go through the list to determine which player is eliminated from this round of the game starting at the player currently at the front of the list. The player after the eliminated player becomes the front of the list. Eliminated players are removed from the list.

Output the following:

A list of the players in the sorted order of input.

For each round until there is a winner

- The ordinal number of the player eliminated and their name
- A space separate list of the players still playing with the current front player first.

The name of who won.

PSEUDOCODE

Submit and get pseudocode approved for the driver software (what runs your game) before coding it. You do NOT need to submit pseudocode for the linked list code.

SUBMIT

- A driver that uses argv to get an input file name
- A DLNode Class for a double linked list
    - data members: contents (int), pointer to previous node, pointer to next node
    - DLNode () : initialize contents to zero, next and previous to NULL
    - DLNode (int newContents) : initialize contents to newContents, next and previous to NULL
    - virtual ~DLNode () : nothing to be done
    - void setContents (int newContents)

- void setNext (DLNode* newNext)
- void setPrevious (DLNode* newPrevious)
- int getContents () const
- DLNode* getNext () const
- DLNode* getPrevious () const

- A DLList class to contain all of the functionality of a doubly-linked list: push front, push back, pop front, pop back, insert (sorted), remove, retrieve, overloaded operator <<, constructor, destructor, clear, size

  - data members: count of nodes, pointer to head node, pointer to tail node
  - DLList () : initialize count to zero, head and tail to NULL
  - ~DLList () : call clear function
  - unsigned int getSize () const : return count
  - void pushFront (int newContents) : create new DLNode with newContents and attach at head
  - void pushBack (int newContents) : create new DLNode with newContents and attach at tail
  - void insert (int newContents) : create new DLNode with newContents and insert in ascending (based on newContents) order
  - int getFront () const: return the value of the contents of the head node; throw an exception (throw "LIST EMPTY") if the list is empty
  - int getBack () const : return the value of the contents of the tail node; throw an exception (throw "LIST EMPTY") if the list is empty
  - bool get (int target) const : return true if target is in list, else return false
  - void popFront () : remove current head node; do nothing if list is empty
  - void popBack () : remove current tail node; do nothing if list is empty
  - bool removeFirst (int target) : remove the first instance of a DLNode containing target; do nothing if target is not found
  - bool removeAll (int target) : remove all instances of DLNode containing target; do nothing if target is not found
  - void clear () : clear all nodes from list, reset count to zero, set head and tail to NULL
  - friend ostream& operator<< (ostream& out, const DLList& src) : display the contents of each node in the list, formatted per the program specification ("NUM1,NUM2,NUM3,...,NUMX"), to the output stream *out*

GRADING – 20 POINTS

	Excellent (100%)	**Satisfactory (80%)**	**Satisfactory (60%)**	**Unsatisfactory (0%)**
Requirements (7 points)	Program meets and exceeds the requirements OR is extremely efficient. Program has no coding, semantic or syntax errors. (7 points)	Program meets the major requirements. May have errors that do not effect program output. (5 – 6 points)	Program meets one major requirement or has significant errors that effect program output. (3 – 4 points)	The program is producing incorrect results at all times. (0 – 2 points)

Readability (5 points)	The code is exceptionally well organized and very easy to follow. Documentation clearly explains what the code is accomplishing and how. Documentation is completed before the program is written (submitted before program submittal). ALL class coding conventions are followed. (5 points)	The code is organized and easy to follow. Documentation is complete and follows class coding conventions. (4 points)	The code is readable. The documentation consists of embedded comments and some simple header documentation. Most coding conventions are followed. (3 points)	The code is difficult to read. The documentation is simple embedded comments (or missing) and does not help the reader understand the code. Coding conventions are not followed. (0 – 2 points)
Current Tasking (5 points)	Program uses current topic as appropriate. (5 points)	Program uses major elements of the current tasking appropriately. (4 points)	Program uses current topic but it may not be used correctly. (3 - 4 points)	Program does not use the current topic appropriately. (0 – 2 points)
Self Assessment (3 points)	Self Assessment is completed demonstrating insight into what you have done and how the topic works. (3 points)	Self assessment is complete. (2 points)	Self Assessment is partially complete (at least 50%). (1 point)	Self Assessment is not completed. (0 points)

Your functions will evaluated against the UNIT TEST for this project and your driver using an input file.

SUBMISSION:

Post your all of your files as a zip folder to Blackboard (you will have to copy and paste your Cloud 9 code into a text editor on your desktop).

- Assignment_4.h
- Assignment_4.cpp

Include a link to your Cloud9 IDE workspace AND a link to your GitHub repository.