Assignment Objective: Build skills on C class creation while implementing a circular list of key-value pairs.

Requirements:

* The class properties:
  + Semantically, to the user, this class will behave the same as the **list** class in the list assignment. It has all the same member functions as **list**.
  + It has two new private member functions, virtualToPhysical() and physicalToVirtual(); see below.
  + It has two C Preprocessor macros: cListInc() and cListDec(); see below.
  + However, this will be implemented in a manner such that insert() will be O(1) as well as add() is O(1).
    - To do this, two new integer member variables will be defined: first and last.
    - first will be the index of the first item on the list.
    - last will be the index of the last item on the list.
    - For a list that is not full, when something is inserted, first will be decremented, and the value will be put at that index. Care must be taken that if first becomes negative, it should be set to the value of capacity-1. I.e., first will wrap around to the end of the array.
    - Similarly, when a value is added to the list, last will be incremented by 1, and the value will go there. If last is incremented to capacity, it will be set to 0 and the value will be inserted there.
    - Note, insertAt() and deleteAt() become more complicated. And performance is still O(n).
* Into two files, cList.h and cList.cpp, create a class called cKeyValueList with the following members
  + Macros to be added to cList.h:
    - #define cListInc(x) (x = (x + 1) % capacity)
    - #define cListDec(x) (x = (x + capacity - 1) % capacity)
  + Typedef to be added to cList.h:

typedef struct key\_value {

int key;

int value;

} KEY\_VALUE;

* + Private members:
    - KEY\_VALUE \*keyValues // A pointer to the array into which the list values will be stored.
    - int listCapacity; // memorialize the capacity of the list
    - int listCount; An integer to record the number of entries in the list
    - int first; // to track the first entry of the list
    - int last; // to track the last entry of the list
    - int virtualToPhysical(int virtualIndex) const // returns the list’s physical index for the given virtual index.
    - Int physicalToVirtual(int physicalIndex) const // returns the list’s virtual index for the given physical index.
  + Public members:
    - constructor cKeyValueList(int theCapacity) that causes the array to have theCapacity entries, with a default of 100, setting listCapacity to theCapacity, setting listCount to 0, and first and last to 0.
    - destructor ~ cKeyValueList () that deletes the dynamically allocated array “a”
    - bool insert(KEY\_VALUE kv) – inserts the value v at the beginning of the list; all other entries shift right. Returns true if v was inserted; otherwise, it returns false.
    - bool add(KEY\_VALUE kv) – inserts the value v at the end of the list. Returns true if the value was inserted; otherwise, it returns false.
    - bool insertAt(int index, KEY\_VALUE kv) – inserts the value v at the index “index”; values at that position need to shift right. Returns true if the value is inserted. Returns false if the list was full or if the index was greater than listCount.
    - bool deleteAt(int index, KEY\_VALUE &kv) – deletes the value at the given index; shifts the entries right of that index to the left. If the index is within the range of the list, it sets “value” to the value of the item deleted and returns true. Otherwise, it does not change “kv” and returns false.
    - bool readAt(int index, KEY\_VALUE &kv) const – Same as deleteAt(), save that it does not delete the entry; it just returns the appropriate key-value pair.
    - bool deleteFirst(KEY\_VALUE &kv) – If the list is not empty, the function deletes the value at the first of the list, updating where first indexes (no copying is required here). The value deleted is returned via the referenced parameter and the function returns true. If the list was empty, the referenced value does not change value and the function returns false.
    - bool deleteLast(KEY\_VALUE &kv) – If the list is not empty, the function deletes the value at the last of the list, updating where last indexes (no copying is required here). The value deleted is returned via the referenced parameter and the function returns true. If the list was empty, the referenced value does not change value and the function returns false.
    - void clear() – causes the list to be emptied
    - void printIt(int limit) const – causes limit items of the list to be printed, one key per line; for each line, print the index and the key at that index. If limit < 0 or limit > listCount, then print the whole list.
    - void printItBackwards(int limit) const – same as printIt() but prints the list from the “bottom” going “up”.
    - int getIndex(int key) const – returns the first position at which the key was found; otherwise returns -1.
    - int getCount() const – returns the number of entries in the list
    - int getCapacity() const – returns the capacity of the list
* Demonstrate the correctness of the implementation by doing the following, using the cListMain.cpp that is given to you on D2L:
  + Create an appropriate Makefile, consistent with classroom instruction.
  + Compile the code into the executable file cList.exe (or cList)
    - make
    - Execute the code, inspect the results, and compare them with cListCorrectOutput.txt.
* **Submission process:**
  + Create the file cListOutput.txt using the following command in your environment:
    - ./cList > cListOutput.txt
    - cat cListOutput.txt (this step ensures that you created the file correctly)
  + **Printing:**
    - In-class students:
      * Print a combined listing of only these files, in the following order: cList.h, cList.cpp, and cListOutput.txt.
      * Bring the listing to class when due.
  + **D2L submission:**
    - Submit a zip file to D2L containing the Makefile, cList.h, cList.cpp, and cListOutput.txt files.