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

Requirements:

* In files called kvDLL.h and kvDLL.cpp, do the following.
* Use the same type definition of KEY\_VALUE as was used for the circular key-value list.

typedef struct key\_value {

int key;

int value;

} KEY\_VALUE;

* Create a class called **node** with the following members:
  + Private members:
    - KEY\_VALUE kv;
    - node \*next, \*prev;
  + Public:
    - constructor … node(KEY\_VALUE kv, node \*next = NULL, node \*prev = NULL)
  + Friend kvDoublyLinkedList
* Create a class called kvDoublyLinkedList with the following members
  + Private members:
    - node \*first, \*last;
    - int nCount // to record the number of entries in the list
    - int getIndex(int key, node \*pn, int index) const; // recursion helper
    - void printIt(node \*pn, int index, int limit) const; // recursion helper
    - void printItBackwards(node \*pn, int index, int limit) const; // recursion helper
    - void clear(node \*pn); // recursion helper
  + Public members:
    - kvDoublyLinkedList() // causes the object to be initialized (first=last= NULL and nCount = 0).
    - ~kvDoublyLinkedList() // deletes the dynamically allocated nodes in the list.

// MODIFIERS

* + - bool insert(KEY\_VALUE kv) // inserts kv at the beginning of the list. Returns whether the value was inserted.
    - bool add(KEY\_VALUE kv) // appends kv to the end of the list. Returns whether the value was added.
    - bool insertAt(int index, KEY\_VALUE kv) – inserts kv at the index, presuming 0 <= index <= getCount(); values at that position need seem as if they have shifted right. Returns true if the value was inserted; false otherwise.
    - bool deleteAt(int index, KEY\_VALUE &kv) – deletes the key-value pair at the given index. Returns true if the index was within 0 <= index < getCount(); returns false otherwise. If the delete takes place, it returns the key-value of the deleted node via the reference parameter “kv”
    - bool deleteFirst(KEY\_VALUE &kv) – deletes the first item on the list, returning true its kv. If the list is empty, it returns false and does not change kv.
    - bool deleteLast(KEY\_VALUE &kv) – deletes the last item on the list, returning true its kv. If the list is empty, it returns false and does not change kv.
    - void clear() // causes the list to be emptied; all dynamically allocated memory should be deleted. **This must be implemented with recursion, no loop.**

// NON-MODIFIERS

* + - bool readAt(int index, KEY\_VALUE &kv) const; // same behavior as the prior assignments.
    - int getIndex(int key) const // returns the first position at which the key was found; otherwise returns -1. **This must be implemented with recursion, no loop.**
    - 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 getCount() const – returns the number of entries in the list
    - int getCapacity() const – returns -1
* Demonstrate that the kvDoublyLinkedList data structure works the provided kvDLLmain.cpp driver file, capturing the output in kvDLLoutput.txt. To do this, create a Makefile that is consistent for the assignment. Then run:

make

./kvDLL > kvDLLoutput.txt

* The kvDLLoutput.txt file should be identical to the output found in kvDLLcorrectOutput.txt. Furthermore, it should be **very** similar to the cListOutput.txt file from the Circular Key-Value List assignment.
* Deliverables:
  + Into D2L a **zip file** containing kvDLL.cpp, kvDLL.h, kvDLLoutput.txt, and the Makefile.
  + In class: a printing of kvDLL.h, kvDLL.cpp, and kvDLLoutput.txt, in that order. Do not print the kvDLLMain.cpp file.