CPSC 1160 SECTION 001/004 INSTRUCTOR: GLADYS MONAGAN

ASSIGNMENT #8: LINKED LISTS

DUE DATE WITH BRIGHTSPACE: MARCH 15 AT 11:50 PM

THE PURPOSE OF THIS ASSIGNMENT IS

- to learn to program using linked lists and to practice programming recursively
- to program functionally (e.g. the beginning of the list is always returned) without instance variables, global variables, nor static variables

READINGS IN THE TEXTBOOK

Read chapter 20 of the textbook (with caveats e.g. no templates and no tail pointers)

- §20.3 is not covered in detail
- §20.5, §20.7, §20.9 not covered in this course

TO IMPLEMENT

We will represent a linked list of integers using the following C++ structure

```
struct Node {
  int value;
  Node *link;
};
```

But we are going to use the functions car and cdr to access the value and link fields of a Node and the functions setcar and setcdr to change the fields of a Node.

You must use these functions in this assignment. Do not access the fields directly.

Implement the following functions in llist_utils.cpp as defined in llist_utils.h. Do this during the lab and yes, you can use class notes.

There will no marks awarded for these 4 functions which are to be implemented in llist_utils.cpp

```
std::ostream& operator << (std::ostream& out, Node* p);
 Node *copyList(Node *p);
Node *reverse(Node *p);


// whatever p was pointing to will not be deallocated
// p is an output parameter and will return the linked list read
std::istream& operator >> (std::istream& in, Node* & p);
```


Implement the functions in **llist.cpp** with the given prototypes of **llist.h.**
Document them properly in llist.cpp. You can **use** helper functions (unless otherwise directed). Put the helper functions in llist.cpp before they are called. Follow the directives 1160 in **llist.h.**
Note that only the first two functions, append and append_it have been started in **llist.cpp**
Continue following those two functions and fill in the other functions..

- Node*append(int x, Node* p);
  recursive version
  Adds a new node with the value of x to the end of the list p. Do not use reverse for append.
- Node* append_it(int x, Node *p);
  iterative version
  same description as above
- bool searchInOrder(int x, Node* p);
  recursive version
  Looks for a value x in the list pointed to by p. If x is in the list, return true, false otherwise.
  Note that the linked list p is sorted in ascending order. Take advantage of this.
- bool searchInOrder_it(int x, Node *p);
  iterative version
  same description as above
- bool isLonger(Node *p, Node *q);
  Return true if p is longer than q, false otherwise
  Do not call the function length in isLonger.
- bool equal(Node *p, Node *q);
  Return true if p and q have the same elements and in the same order, and false otherwise.
- Node* makeDuplicates(Node *p);
  Return a new list with every element repeated twice, in the same order as p.
  p is not modified.
- Node* removeList(Node* p);
  Every node of the list is returned to the heap (freestore).
  Returns the null pointer.
- Node* array2List(int A[], int n);
  The conversion routine converts an array of n integers to a linked list of n integers.
  The order of the elements in the linked list correspond to A[0], A[1], ... A[n-1].
- Node* mergeTwoOrderedLists(Node *p, Node *q);
  Takes two lists that are already sorted in ascending order and returns a new merged list.
- Node* removeIth(int i, Node* p);
  Returns a linked list without the i^{th} node. If the i^{th} node does not exist, do nothing.
  The first node would be the 0^{th}
  Return to the heap the memory no longer needed.
- Node* removeOccurrences (int x, Node *p);
  Removes based on value, not on position, every node that has the value x.
  Return to the heap the memory no longer needed.

- BONUS
  Node* odds(Node* p);
  Return a new list with the odd-positioned values of p (where possible).
  Implement recursively.
  The first node is odd, the third is odd, and so on.
- BONUS
  Node* evens(Node* p);
  Return a new list with the even-positioned values of p (where possible).
  Implement iteratively.

You must use car, cdr, setcar and setcdr, cons, as well as the nullptr in your assignment exclusively.
No marks given for the assignment if you use p->value, p->link or (*p).value and (*p).value or new Node directly. Your code must compile.
Use the file structure given. There is a Makefile provided :

make and make test

You can overload the functions if you needed. Place the helper functions in the file llist.cpp before they are called.

TO SUBMIT WITH BRIGHTSPACE AS A SINGLE ZIP FILE

Before compressing into a single zip file the directory with all the file, do
make clean or make remove

1) **llist.cpp** code with your functions: the functions have been document in llist.h but if you need to explain something about your implementation, add the comments in the llist.cpp file.
   Use helper functions as needed and place them before they are called in the file.
   To reiterate, you **can** use helper functions.
2) **llist_utils.cpp** (with code from class notes if you wish and from the lab)
3) submit but do not modify: **llist_utils.h llist.h doctest.h Makefile**
4) **test_llist.cpp** add tests if you want but they will not be marked
5) **unittest_llist.cpp** will not be marked but make sure that it compiles
6) **README.txt** indicating
   a) whether you did the bonus
   b) any bugs that you know of and/or any functions that you did not finish.

Do not get any code from the Internet nor from a friend (see course outline).
Ask for help if you need help with this assignment during office hours or at the Help Centre.
Start early.