**C++ Workshop – 150018**

**Homework Assignment #5**

**Linked Lists**

**Before you begin, make sure that your program follows the following rules:**

a. Your program should be easy to read. This means to make sure that you are using correct indentation and blank lines to make your code more readable

b. Make sure that you use constants whenever possible.

c. Make sure that the names that you give your identifiers are meaningful.

d. Break your program down into smaller sub-problems and use functions to slof each sub-problem.

e. Document your code as you were taught in the lecture. You are to write an abstract at the beginning of the program, comments before each method and function, and comment complicated sections of code.

f. Make sure that you understand the assignment and write a program that does exactly what the assignment requires.

g. As a general rule, it is better to pass object as cbr (and not cbv). When passing it cbr – it does not call the copy constructor which saves both time and memory.

For example, in question 1, the signature of the method operator= should be

List & List::operator=(const List& l)

h. Make sure that the function/method heading uses the right value return type. If the function is not supposed to calculate a single value then have your function return void. If your function/method is supposed to return an object then make sure that it is actually returning the object and not a reference to the object (Details for this type of error will be explained in further detail in question 2.)

i. Add an example of the output of the program in comments at the end of your program.

j. All solutions must be submitted in to Moodle according to the date that your instructor gave you which may not be the date listed in Moodle. You are given exactly one week to complete the assignment. You may work in teams of two. If you choose to do so, then one student must upload the program. The comments on the program must containe the names of both students. The other student must upload a Word document that lists the names of both students who worked on the program.

**Question 1**

Complete the List class definition discussed in lecture to handle lists whose elements are sorted in a non increasing order (every element is less than or equals to its predecessor). You may find it helpful to use the class defined in the course handbook. Add the following methods:

* 1. operator=: deep assignment of elements of a list to another list retaining same order of elements
  2. operator<<: prints all elements of the sorted list.
  3. operator>>: inputs sorted elements. Note, the input is terminated when it receives a value that is greater than its predecessor.(You can have duplicate entries.)
  4. insert(int key): inserts the given integer into its appropriate place in the sorted list. At this point, the list is in non-ascending order (this allows duplicate entries).
  5. remove(int key): removes the element whose value matches the given key. The method preserves sorted order. If more than one element matches the key, then only the first one is removed. If the key value is not found, the function throws the exception "value not found".

Use the following main program to check your functions:

#include <iostream>

using namespace std;

#include "List.h"

enum CHOICES { EXIT, INSERT, REMOVE, ASSIGN, PRINT };

int main()

{

List lst;

List lst2;

int choice, val;

cout << "enter the list values\n";

cin >> lst;

cout << "choose 0-4\n";

cin >> choice;

while (choice != EXIT)

{

switch (choice)

{

case INSERT:

cout << "enter a value to insert\n";

cin >> val;

lst.insert(val);

break;

case REMOVE:

cout << "enter a value to remove\n";

cin >> val;

try {

lst.remove(val);

}

catch (char\* msg)

{

cout << msg << endl;

}

break;

case ASSIGN:

lst2 = lst;

cout << "list2: " << lst2;

cout <<"list 1: ";

case PRINT:

cout << lst << endl;

break;

default:cout << "ERROR\n";

}

cout << "choose 0-4\n";

cin >> choice;

}

return 0;

}

Example:

enter the list values

25 20 20 15 8 6 10

choose 0-4

4

25 20 20 15 8 6

choose 0-4

1

enter a value to insert

15

choose 0-4

1

enter a value to insert

7

choose 0-4

4

25 20 20 15 15 8 7 6

choose 0-4

2

enter a value to remove

15

choose 0-4

2

enter a value to remove

8

choose 0-4

3

list2: 25 20 20 15 7 6

list1: 25 20 20 15 7 6

choose 0-4

0

**Question 2:**

Using the sorted List class you defined above, write the following three **global** functions. (Note, you should **not** define these functions as friends of the class, but rather have them use public helper methods defined in the class.)

**Part A – Function merge**

**merge** that merges two lists. The function receives two sorted lists, lst1 and lst2, each of type List, and returns a new list that combines the two lists in non-ascending order. Note: the newly created list may have duplicate entries.  
  


**Part B – Function makeSet**

**makeSet** that converts a list into a set with no duplicate entries. The function receives a list of integers in non-ascending order and converts it into an ordered set having no duplicate values. The set should be in strictly ascending order.

7

5

4

3

1

**Part C – Function reverse**

**reverse** that reverses the elements of a list. In other words, the last element is now the first element, the second to last element is now the last element, etc. The first element (which is now the last element) will point to null

The list will now be a non-descending list of elements

1

3

4

5

7

Use the following main program to check your functions:

#include <iostream>

#include "List.h"

using namespace std;

int main()

{

List lst1, lst2, mergedList;

cout<<"enter sorted values for the first list:"<< endl;

cin>>lst1;

cout<<"enter sorted values for the second list:"<< endl;

cin>>lst2;

mergedList = merge(lst1,lst2);

cout <<"the new merged list: " << mergedList <<endl;

makeSet(mergedList);

cout<<"the new merged set: " << mergedList << endl;

reverse(mergedList);  
cout<<"the new merged reverse: " << mergedList << endl;

return 0;

}

Some general comments about question 2:

1. You can use the class List that you wrote for question 1

2. According to the rules of the question – you are to write the code for the three functions. **You may not change** the lcass List in order to answer the question

3. You may not access the private fields of List (aka you cannot change what head points to)

4. Your code can only use List and the methods that belong to List

**Hints for functions that do not return values (**in this homework: makesset and reverse)

* the function gets an object that is an instance of a class. The function is to change the value of this object
* One way to do so is :
  + To create a new list (a new object)
  + Execute the appropriate methods of List on the new object and/or on the object that was passed as a parameter
  + Set the parameter to be equal to the new object that was created

**Hints for functions that return an object** (in this homework merge)

* Reminder:
  + When a function returns an object, it automatically calls the copy constructor to copy over the returned object – that object is passed to the program that called the function. The object that was defined in the function is destroyed by the destructor.
  + When a function returns a reference to an object, then there is no call to the copy constructor. Which means that the address of the object is returned to where the function was called. However, when it leaves the function then the object is destroyed, which means that there is garbage at the address which is returned
* For this assignment
  + If you function merge returns a reference to a List, then the List will be destroyed right before it leaves the function and therefore the function will pass to the main program an address and at that address there will be garbage. If your main program tries to print out the list that was returned from the function there will be an error
  + If your function returns an object of type List then there will be no problem (assuming of course that your function correctly built the merged list from two other lists) This is because your function will automatically call the copy constructor which will copy the list over to a new list, destroy the temporary list in the function, and return the new list to the main program.
  + Therefore, it is recommended that you use the following signature for your function

List merge(const List& lst1, const  List& lst2);

enter sorted values for the first list :

6 5 4 3 2 1 9

enter sorted values for the second list :

7 5 4 3 9

the new merged list : 7 6 5 5 4 4 3 3 2 1

the new merged set : 7 6 5 4 3 2 1

the new merged reverse : 1 2 3 4 5 6 7

Press any key to continue . . .

Example: