CPSC 1160 SECTION 001 INSTRUCTOR: GLADYS MONAGAN

ASSIGNMENT #9: ADT SET

DUE DATE WITH BRIGHTSPACE: WEDNESDAY, NOVEMBER 23 AT 11:50 PM

THE LEARNING OUTCOMES

- to implement the Abstract Data Type "set" (as in a Mathematical Set)
- to see how the implementation of a class can influence the running time complexity of the functionality provided by an ADT
- to use the data structure linked list to implement a class

READINGS IN THE TEXTBOOK

- chapter 9
- chapter 10
- chapter 11 (including the sections 11.11-11.15)
- chapter 20 (skipping sections as listed in the week XI Overview)

FUNCTIONALITY

The functionality provided in the file **Set.h**

- insert an element into the set
- remove an element from the set
- determine if the set is empty
- give the cardinality, the number of elements in the set
- determine if an element is in the set
- print the set
- compute the set union of two sets
- compute the set intersection of two sets
- determine if a set is a subset of another set [1]

Do not change the public part of the class in **Set.h**

Note that the public part of the class does not indicate how the set is implemented (namely with a linked list as required in this assignment).

[1] Subset means containment so the empty set is a subset of any set and a set s is a subset of itself

TO IMPLEMENT

In assignment #8, we used "the C-style approach of using global functions and structs with everything public"

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

with the functions `first` and `rest` used to access the fields `value` and `link` .

For this assignment #9, we are restricting the access to the linked list by embedding Node as a private struct in Set (note that the C++ syntax can be a bit tricky in that you need to give the class name before Node when returning a Node or a pointer to a Node in the implementation of a function). Basically follow the pattern from the lab from November 9.

For this assignment #9, add functions to the private section of **Set.h** as desired.
Read the comments and instructions in Set.h.
The instructions are embedded as C++ comments.
Look for the string CPSC 1160

To speed up the set union and set intersection operations, if we keep the sets' lists sorted, then set union and set intersection can be done by merging two lists. This is actually much easier to program than if you use unsorted lists. So, use a sorted linked list as the base data structure of the set.

The insert operation insert(int x) must insert the value x in order.

Suppose the set S has m elements and the set T has n elements. When you program each of the above operations, **in the comments in the code, state the cost of the operation in terms of m and n**.
For example, the cost of the isEmpty() operation should be O(1) meaning the running time is constant and does not depend on the size of the set S.
We are only looking for the time complexity of those operations/functions that are declared "public" in the header file, namely isEmpty, size, member, insert, remove, union, intersection and subset.

The costs of the operations will likely be O(m) or O(m+n) or O(mn). Of course, you want to program them all as efficiently as you can. If you can do a set operation in O(m+n) instead of O(mn), then do so.

Notes:

- When we compute the union of two sets S and T using U = S + T, we must not destroy the old sets S and T. Why? Because the user may want to keep and then use all three sets S, T, and U. And you must not share nodes in S and T with U so that the user can later insert new elements into S and T without changing U. Therefore, as you construct the new set U, make copies of the nodes from S and T.
- The operation remove(x) should free the memory for the node with x in it. Return it to the heap.
- You can reuse code that you wrote for Assignment #8.
- Do not use code from the Internet or written by anyone else (if you take code from assignment #8 solution given in the class or lab, code from class, or code from the textbook, cite this).

Once you have the above code working (and not before), define and implement the "big three" for the class Set: the copy constructor, the overloaded assignment operator and the destructor.
Write code to test these possibly continuing with what has been started.

TO SUBMIT WITH D2L AS A SINGLE ZIP (COMPRESSSED) FILE

1) The header file s**et.h** which is the interface to the class Set.
   Expand as needed adding more private functions.
2) An implementation file s**et.cpp**.
   Do not forget to give the complexity of the functions in big O notation. Document your code.
3) The test file **unittest_set.cp**
   add tests for a few marks
4) The files provided for you **test_set.cpp Makefile**, **doctest.h**, **set_utils.cpp, set_utils.h**
   Do make clean (or make remove) before compressing the files.
5) A README.txt or README.docx that has information that you might want to give us and that indicates that you have done the BONUS part.

BONUS MARKS

1)
Let P and Q be sets. P is a *proper subset* of Q, written as $P \subset Q$,

- if P has all the elements in Q
- and Q has at least one element that is not in P.

Remember that a set does not have duplicate elements.

Here are some examples

$P \subset Q$ for P = {2, 3} and Q = {2, 3, 5}
$P \subset Q$ for P = {2, 4} and Q = {1, 2, 3, 4}
$P \not\subset Q$ for P = {2, 3, 4} and Q = {2, 3, 4}
$P \not\subset Q$ for P = {2} and Q = {}
$P \not\subset Q$ for P = {} and Q = {}
$P \subset Q$ for P = {} and Q = {2}

Your solution should be linear, not O(nm). Put a comment about the complexity in your code.

2)
Let P and Q be sets. P is equal to Q, written as $P == Q$ if all the elements of P and Q are equal.