Linköping Institute of Technology
Department of Science and Technology/ITN
Aida Nordman

04/03/2014

Data Structures
TND004
Lab 1

Course goals

To use

- doubly linked lists to implement a class representing sets;
- Big-Oh notation to estimate the running time of the set operations;
- C++ classes, templates, and overloaded operators.

Preparation

Doubly linked lists where discussed in Fö 2, while Big-Oh notation was introduced in Fö 1. Thus, you need to review these two lectures. Moreover, it is advisable that you read sections 2, 3.2, and 3.5 of the course book.

Section 3.5 of the course book presents a possible implementation for doubly linked lists. Although in this lab you also need to implement doubly linked lists, there are some important differences between the lists in this exercises and the book's implementation. For instance in this exercise, the list's nodes are placed in sorted order, there are no repeated values stored in the lists, iterator classes and move constructors are not considered[1]. Most of the differences are motivated by the fact that we are going to use doubly-linked lists to implement the concept of (mathematical) set. Nevertheless, studying the implementation presented in section 3.5 of the course book is useful.

Presenting solutions

You should demonstrate your solution orally during your second lab session. You also need to deliver a written paper with the estimate of the running time (using Big-Oh), for the requested set operations. Hand written answers that we cannot understand are simply ignored. Do not forget to put the name and LiU login of the group members in the delivered paper.

Use of global variables is not allowed, but global constants are accepted.

If you have any specific question about the exercises, then drop me an e-mail. Be short and concrete, otherwise you won't get a quick answer. You can write your e-mail in Swedish. Add the course code to the e-mail's subject, i.e. "**TND004:** …".

Exercise 1: code implementation

In this exercise, you need to implement a class `Set` that represents a set of elements. Use a **sorted doubly linked list** to implement this class. Notice that sets do not have repeated elements. The class should offer different set operations like union, difference, subset test,

[1] Iterators and move constructor could also be added to the class `Set` of this lab. However, this is left as a future extension.

and so on (you can find a description of these operations here). For a set with n elements, **all operations should have a time complexity of $O(n)$[2], in the worst-case**. STL should not be used in this exercise.

The idea is that class **Set** represents a generic set of elements of a type **T**, e.g. set of **int**egers, or set of **string**s. Thus, class **Set** is implemented as a template class with a type parameter **T**. Every node of the list stores a value of type **T** (i.e. a set's element). The class **Set** defines a nested class **Node** to represent a node of the doubly linked list. To make it easier to remove and insert an element from the list, the list's implementation uses "dummy" nodes at the head and tail of the list, as discussed in Fö 2 and in the book.

File **set.cpp** contains the definition of the template class **Set**, while the file **main.cpp** contains a test program that cannot be modified. Recall that for a template class, the class definition and the implementation of its member functions need to be provided in the same file (**set.cpp**). The public interface of the class **Set** cannot be modified (i.e. you cannot modify the public members nor add new public members).

Some of the **Set** member functions are not yet implemented, e.g. copy constructor, assignment operator, **_union**. Your job is to implement these functions.

Four overloaded operators are provided, in the template class **Se**t, to represent set union, set intersection, set difference, and the stream insertion operator. The overloaded operators are implemented by simply calling private member function of class **Se**t that do the necessary tasks.

All given code is documented. The documentation, in the form of html pages, was generated automatically with doxygen[3]. You should start by studying the documentation of the given code.

More concretely, you should follow the steps below.

1. Download the files **set.cpp**, **main.cpp**, and **out.txt**.
2. Create a project with the source files above and create an executable. The program should run, although it does not do any useful task, yet.
3. Read the documentation provided for class **Set**. This documentation is accessible through a web browser. For instance, if you follow the links "*Classes* > *Set*" then you get a list of all public and private member functions of the class, a list of all private data members (private attributes), and a list of all class's friends. For each member there is a link in the text, named "*More…*", that leads to a detailed description of the member, including its code in the source file.
4. Take a look at the test program in the file **main.cpp** (it contains the **main()** function). You should test your code in a stepwise manner. See the test phases in the **main()**. You can compare the output of your program with the expected output provided in the file **out.txt**.
5. Implement the **Set** member functions whose body includes the comment "**//ADD CODE**".

[2] All operations should be performed in linear time, at most.
[3] Doxygen is considered the *de facto* standard tool for generating documentation from annotated C++ sources.

a. First, implement the private member functions **`print`**, **`init`**, **`insert`**, and **`erase`**. These functions are useful in the implementation of several other member functions.

b. Second, implement the following public member functions: constructors, destructor, assignment operator, **`is_empty`**, **`cardinality`**, **`is_member`**, and **`clear`**. Obviously, you should start with the constructors (the default constructor is already implemented).

c. Third, you need to implement the private member functions **`_union`**, **`_intersection`**, and **`_difference`**. These functions are called by the friend overloaded operators (**`operator+`**, **`operator*`**, and **`operator-`**) to represent the set union, intersection, and difference, respectively.

d. Finally, implement the comparison operators, i.e. **`operator<=`**, **`operator<`**, and **`operator==`**.

Remember to implement one function at a time, then compile and correct any (compilation) errors, before proceeding to the next function. Moreover, code for memory (nodes) allocation (i.e. use of **`operator new`**) should be placed in the functions **`init`** and **`insert`**, while code for memory deallocation (i.e. use of **`operator delete`**) should be placed in the destructor and in function **`erase`**. Calls to **`operator new`** and to **`operator delete`** should not appear anywhere else.

Exercise 2: time complexity analysis

You are requested to estimate the running time of the following statements. Use Big-Oh notation and **motivate clearly** your answers.

- **`Set<T> S(V, n);`** -- where **`V`** is an array with **`n`** objects of type **`T`**.
- **`S1 = S2;`** -- where **`S1`** and **`S2`** are two **`Set`**s.
- **`S.cardinality();`** -- where **`S`** is a **`Set`**.
- **`S1 <= S2;`** -- where **`S1`** and **`S2`** are two **`Set`**s.
- **`S1 + S2;`** -- where **`S1`** and **`S2`** are two **`Set`**s.
- **`k+S;`** -- where **`S`** is a **`Set`** of elements of a type **`T`** (e.g. **`int`**) and **`k`** is a value of type **`T`** (e.g. constant **`5`**).

Remember to write your answers in paper, preferably computer typed, and do not forget to indicate the name plus LiU login of each group member.

Lycka till !