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

Programming in C++
TNG033
Lab 2

Course goals

- To implement a dynamic data structure: singly linked list.
- To write programs using pointers and dynamic memory allocation/deallocation.
- To create classes with overloaded operators.

Preparation

To solve the exercises in this lab, you need to understand concepts such as pointers, dynamic memory allocation and deallocation, and how singly linked lists can be implemented. These topics were discussed in Fö 3 to Fö 5, and lesson 1. Since the lists used in this lab are implemented with a dummy node, you should review the singly linked lists implementation given in the code folder **`Lists2`** for Fö 5.

Classes, constructors, destructors, **`const`** member functions, and friend functions are also used in the exercises of this lab. These concepts were discussed in Fö 7 and Fö 8.

Recall also how two sorted sequences can be merged into one sorted sequence. This concrete problem was discussed in lesson 1 and you should follow a similar algorithm for implementing union of two sets (**`operator+`**). Thus, review also the solution for exercise 2 of Lesson 1.

Sections 7 (skip sections 7.3.5 and 7.3.6), 8 (skip sections 8.4.5 -8.4.10, 8.5-8.7), and 13.1.1 of the course book discuss also the concepts used in this lab.

Presenting solutions and deadline

All exercises in this lab are compulsory and you should demonstrate your solution orally during your redovisning lab session on **week 48**. 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. "**`TNG033:`** …".

Exercise

Implement a class, named **`Set`**, representing a set of integers. A set is internally implemented by a **sorted singly linked list**. Every node of the list stores an element in the set (an integer) and a pointer to the next node. To simplify the implementation of the operations that insert (remove) an element in (from) the list, an empty set has one dummy node (see slide 19 of Fö 5).

Note that sets do not have repeated elements (according to the mathematical definition of set).

Download from the course web page the files **`node.h`**, **`node.cpp`**, **`set.h`**, **`set.cpp`**, **`main.cpp`**, and **`out_uppg2.txt`**. Then, create a project with all source and header files. You should be able to compile, link, and execute the program, although it does not produce any useful result, yet. A brief description of the files is given below.

- The interface of class **`Node`** is available in the file **`node.h`** and its implementation is in the file **`node.cpp`**. This class is already implemented and it cannot be modified.
- The header file **`set.h`** contains the interface of class **`Set`**, i.e. the class definition. The public interface of the class cannot be modified, i.e. you cannot add new public member functions neither change the ones already given. Private (auxiliary) member functions can be added.
- You should add the implementation of each member function of class Set to the source file **`set.cpp`**.
- The source file **`main.cpp`** contains the **`main`** function to test your code. It creates several objects of class **`Set`** and it also calls the member functions for class **`Set`**. You cannot change the code in **`main.cpp`**.
- The main function includes several test phases to test your code incrementally. Thus, you should implement those member functions needed by each test phase, then compile, link and run the program. If it passes the tests for the current test phase then you can proceed to the next phase.
- The file **`out_uppg2.txt`** contains the expected output of the program.

To be approved in this lab you need also to bring a written answer to the following question.

- **Question:** does the class **`Set`** provide set union and set difference operations that are commutative? Motivate your answer.

A brief description of the Set member functions is given below.

- **`Set ();`**
  Constructor for an empty set,
  e.g. **`Set R;`**
- **`Set (const Set& S);`**
  Copy constructor initializing **`R`** with set **`S`**,
  e.g. **`Set R(S);`**
- **`Set (int a[], int n);`**
  Constructor creating a set **`R`** from **`n`** integers in a **non-sorted** array **`a`**,
  e.g. **`Set R(array, 10);`**
- **`~Set();`**
  Destructor deallocating the nodes in the list, including the dummy node.
- **`const Set& operator=(const Set& S);`**
  Overloaded assignment operator,
  e.g. **`R = S;`**
- **`bool member (int x) const;`**
  If the element **`x`** is in the set **`R`** then **`true`** is returned, otherwise **`false`**,
  e.g. **`R.member(5);`**
- **`int cardinality() const;`**
  Returns the number of elements in the set **`R`**,
  e.g. **`R.cardinality();`**
- **`bool empty() const;`**
  Returns **`true`** if the set **`R`** is empty, otherwise **`false`**,
  e.g. **`R.empty();`**
- **`Set operator+(const Set& S) const;`**
  Returns a new set with the set union of **`R`** and **`S`**. The union is the set of elements in set `R` or in set `S` (without repeated elements). For instance, if $R = \{1, 3, 4\}$ and $S = \{1, 2, 4\}$ then $R + S = \{1, 2, 3, 4\}$.
- **`Set operator*(const Set& S) const;`**
  Returns a new set with the intersection of **`R`** and **`S`**. The intersection is the set of elements in **both** **`R`** and **`S`**. For instance, if $R = \{1, 3, 4\}$ and $S = \{1, 2, 4\}$ then $R * S = \{1, 4\}$.
- **`Set operator-(const Set& S) const;`**
  Returns a new set with the difference of **`R`** and **`S`**. The difference is the set of elements that belong to **`R`** but do not belong to **`S`**. For instance, if $R = \{1, 3, 4\}$ and $S = \{1, 2, 4\}$ then $R - S = \{3\}$.
- **`Set operator+(int x) const;`**
  Returns a new set with the union of **`R`** and set **`{x}`**,
  e.g. **`S = R + 5;`**
- **`Set operator-(int x) const;`**

Returns a new set with the difference of **R** and **{x}**,
e.g. **S = R - 5;**

- **bool operator<=(const Set& S) const;**
  Returns **true** if **R** is a subset of **S**, otherwise **false**. **R** is a subset of **S** if and only if every member of **R** is a member of **S**. For instance, if $R = \{1, 8\}$ and $S = \{1, 2, 8, 10\}$ then R is a subset of S (i.e. **R <= S** is **true**), while S is not a subset of R, (i.e. **S <= R** is **false**).

- **bool operator<(const Set& S) const;**
  Returns **true** if **R** is a proper subset of **S**, otherwise **false**. A proper subset **R** of a set **S** is a subset that is strictly contained in **S** and so necessarily excludes at least one member of **S**. For instance, **S < S** always evaluates to **false**, for any set S. Use **operator<=** to implement this function.

  - **bool operator==(const Set& S) const;**
    Returns **true** if **R** is a subset of **S** and **S** is a subset of **R** (i.e. both sets have the same elements), otherwise **false**. Use function **operator<=** to write this function,
    e.g. **R == S;**

- **ostream& operator<<(ostream &os, const Set& S);**
  Stream insertion operator **operator<<** outputs all the elements in S into the output stream **os**,
  e.g. **cout << R;**

Lycka till!!

Appendix: advised compiler settings

In Code::Blocks, go to **Settings > Compiler** and make sure that only the following compiler flags are on.

- Produce debugging symbols
- Enable extra compiler warnings
- **Have g++ follows C++11 ISO C++ language standard**
- Warn if the compiler detects that code will never be executed
- Warn if floating point values are used in equality comparisons
- Warn if anything is declared more than once in the same scope
- Warn about uninitialized variables which are initialized with themselves
- Warn whenever a local variable shadows another local variable, parameter, or …