Linköping Institute of Technology 04/04/2016
Department of Science and Technology/ITN
Aida Nordman

Data Structures
TND004
Lab 1

Goals

To use

- doubly linked lists to implement a class representing generic sets;
- big-Oh notation to estimate the running time of the set operations;
- C++ classes, templates, and overloaded operators[1], move semantics and smart-pointers.

Prologue

In the TNG033 course, the second lab consisted in implementing a singly-linked list to represent sets of integers. Templates and overloaded operators were also introduced during the TNG033 course. You are going to use these two concepts again in this lab.

In this lab exercise, you go further and implement a **generic** class **`Set`** using instead **doubly-linked lists**. Obviously, templates are used to create a generic class representing sets of objects of a type **`T`**. Moreover, all operations available for **`Set`**s must have linear time complexity, in the worst-case.

Two other very relevant concepts, which have not been considered in previous C++ courses, are also used in the implementation of class **`Set`**.

- **Move semantics**. By equipping the class **`Set`** with a move constructor and a move assignment operator, a performance boost can be provided to client code.
- **Smart-pointers**. Many of the pitfalls (e.g. memory leaks) involved in using built-in pointers (as well as the **`new`** and **`delete`** operations) can be avoided by completely eliminating built-in pointers from the code. Instead, smart-pointers provide by the C++ standard library **`memory`** should be used.

Preparation

Move semantics is nicely discussed in section 13.6 of the book "C++ Primer", 5th edition. Section 12.1 of the book discusses the use of smart-pointers. An introduction to smart-pointers can also be found in "Using C++11's Smart Pointers".

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 exercise and the book's implementation, as summarized below. 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 recommended.

[1] Templates and overloaded operators were introduced in the TNG033 course.

- The book presents an implementation of doubly-linked lists, while in this lab you are requested to implement the concept of set by using a doubly-linked list (though, sets can be implemented in other ways).
- In this lab exercise, the list's nodes are placed in sorted order and there are no repeated values stored in the list. On the contrary, the course book's implementation allows repeated values in the lists and the lists do not need to be sorted.
- Iterator classes are not considered in this exercise, though they are implemented in the course book.
- Smart pointers must be used in the implementation of class **`Set`**, instead of raw pointers. The course book's implementation uses only raw pointers.

Two important similarities between the class **`Set`** and the course book's implementation of class **`List`** (see section 3.5) is that both support the move semantics and they are template classes.

You can find below a list of tasks that you need to do before the **HA** lab session on week 15.

- Review lectures 1 to 3. Big-Oh notation was introduced in Fö 1 and 2, while doubly linked lists where discussed in Fö 3.
- Read sections 2 and 3.2 of the course book.
- Review lesson 2. In this lesson, an introduction to move semantics and smart-pointers was given.
- Read the appendix of this lab.
- Define template class **`Set`**.
    - Add the class **`Set`** definition to a file named **`set.h`** which should contain the headers of all public member functions, declaration of any friend functions (if any), and private data members needed to implement the required functionality. Obviously, it is also needed to define a class **`Node`** to represent the nodes of the list.
    - Implement the constructors of class **`Node`**.
    - Implement each **`Set`** member function with some "*dummy code*". For instance, the comparison operators (e.g. **`operator<=`**) can simply return **`false`**.
    - Finally, test whether your code compiles and runs with the test program given in the file **`main.cpp`** (though the program is not expected to produce the expected output, yet). The file **`main.cpp`** can be downloaded from the course website.

In the beginning of the **HA** lab session on week 15, the lab assistant will give feedback on your class **`Set`** definition. It is very important that the class definition (without yet the member functions implementation) is correct to avoid redoing big parts of the code later on.

Presenting solutions and deadline

You should demonstrate your solution orally during your **RE** lab session on **week 16**. After week 16, if your solution for lab 1 has not been approved then it is considered a late lab. Note that a late lab can be presented provided there is time in a **RE** lab session.

The necessary requirements for approving your lab are given below.

- Use of global variables is not allowed, but global constants are accepted.
- Readable and well-indented code. Note that complicated functions and over-repeated code make code quite unreadable and prone to bugs.
- The use of raw pointers, as well as the **`new`** and **`delete`** operations, are strictly forbidden in the implementation of class **`Set`**.
- There are no memory leaks. We strongly suggest that you use one of the tools listed in the appendix to check whether your program has memory leaks, before "redovisning".
- For a set with n elements, **all operations must have a time complexity of $O(n)$, in the worst-case**[2].
- STL containers cannot be used in this exercise.
- Bring a written answer to exercise 2, with indication of the name plus LiU login of each group member. You'll need to discuss your answers with the lab assistant who in turn will give you feedback. Hand written answers that we cannot understand are simply ignored.

If you have any specific question about the exercises, then send us 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:** ...".

Finally, presenting your solution to the lab exercises can only be done in the **RE** sessions.

Exercise 1: class **`Set`**

In this exercise, you need to implement the class **`Set`** that represents a set of elements. **Sorted doubly linked list** is the data structure used to implement sets in this lab. Notice that sets do not have repeated elements.

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). 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 3 and in the course book.

Recall that for a template class, the class definition and the implementation of its member functions need to be provided in the same file (e.g. **`set.cpp`**).

The class **`Set`** provides different set operations like union, difference, subset test, and so on (you can find a description of these operations here). A complete listing of the functionality that must be provided by class **`Set`** is given below.

- Constructor for an empty set,
  e.g. **`Set<int> R;`**

[2] All operations should be performed in linear time, at most.

- Conversion constructor creating a set **S** with a given element **v**,
  e.g. `Set<int> R(v); //create the set R = {v}`
- Copy constructor initializing set **R** with set **S**,
  e.g. `Set<int> R(S);`
- Move constructor.
- Constructor creating a set **R** from **n** integers in a **sorted** array **a**,
  e.g. `Set<string> R(a, 10);`
- Destructor deallocating the nodes in the list, including the dummy nodes.
- Assignment operator,
  e.g. `R = S;`
- Move assignment operator.
- A member function to test whether a set **R** is empty,
  e.g. `R._empty();`
- A member function that returns the number of elements in a set **R**,
  e.g. `R.cardinality();`
- A member function to test whether a given value **v** belongs to a set **R**,
  e.g. `R.is_member(v);`
- A member function to transform a set **R** into the empty set,
  e.g. `R.make_empty();`
- Overloaded `operator+=` such that `R+= S;` should be equivalent to $R = R \cup S$, i.e. the union of **R** and **S** is assigned to set **R**. Recall that the union $R \cup S$ 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 $+S = \{1, 2, 3, 4\}$.

  Union of sets is similar to the problem of merging two sorted sequences. An algorithm to merge sorted sequences efficiently was discussed in the TNG033 course (see lesson 1, exercise 2).
- Overloaded `operator*=` such that `R*= S;` should be equivalent to $R = R \cap S$, i.e. the intersection of **R** and **S** is assigned to set **R**. Recall that the intersection $R \cap S$ is the set of elements in **both** R and S. For instance, if $R = \{1, 3, 4\}$ and $S = \{1, 2, 4\}$ then $* S = \{1, 4\}$.
- Overloaded `operator-=` such that `R-= S;` should be equivalent to $R = R - S$, i.e. the set difference of **R** and **S** is assigned to set **R**. Recall that the set difference $R - S$ 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 $-S = \{3\}$ [3].
- Overloaded `operator+` such that `R+S` should be equivalent to $R \cup S$. Note that this operator can be easily implemented by using `operator+=`.
- Overloaded `operator*` such that `R*S` should be equivalent to $R \cap S$. Note that this operator can be easily implemented by using `operator*=`.

[3] The set difference $R - S$ is also known in the literature as the "*relative complement of S in R*".

- Overloaded **`operator-`** such that **`R-S`** should be equivalent to $R - S$. Note that this operator can be easily implemented by using **`operator-=`**.
- Mixed-mode arithmetic expressions should be supported. Examples of mixed-mode arithmetic expressions are **`R+k`**, **`k+R`**, **`R*k`**, **`R*k`**, **`R-k`**, **`k-R`**, where **`R`** is a set of elements of some type **`T`** and **`k`** is a value of type **`T`**.
- Stream insertion **`operator<<`** outputs all the elements in a set **`R`** to a given output stream, e.g. **`cout << R;`**
- Overloaded **`operator<=`** such that **`R <= S`** returns **`true`** if **`R`** is a subset of **`S`**. Otherwise, **`false`** is returned. Set **`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`**).
- Overloaded **`operator<`** such that **`R < S`** returns **`true`**, if **`R`** is a proper subset of **`S`**. A set **`R`** is a proper subset of a set **`S`** if **`R`** 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.
- Overloaded **`operator==`** such that **`R == S`** 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`** is returned. Use **`operator<=`** to implement this function.
- Overloaded **`operator!=`** such that **`R != S`** returns **`true`**, if **`R`** (**`S`**) has an element that does not belong to set **`S`** (**`R`**), i.e. **`R`** and **`S`** have different elements.

Note that all operations should have a time complexity of $O(n)$, in the worst-case, where n is the number of elements in the given sets. Moreover, raw pointers, as well as the **`new`** and **`delete`** operations, cannot be used in the implementation of class **`Set`**. Instead, smart-pointers should be used.

The file **`main.cpp`** contains a test program for class **`Set`** and it can be downloaded from the course website. Feel free to add other tests to this file, though the original file must be used when presenting the lab. Recall that for a template class, the class definition and the implementation of its member functions need to be provided in the same file (e.g. **`set.h`**).

Other member functions can be added to class **`Set`**, besides the ones described above. In this case, the extra added functions should not belong to the public interface of the class.

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. You may assume that all sets below are sets of integers.

- `Set<int> S1(S2);`
- `S1 = S2;`
- `S.cardinality();`
- `S1 + S2`
- `k+S;`

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. Deliver your written answers to the lab assistant, when you present the lab.

Appendix

Sets

You can find information about mathematical sets and their operations here.

Checking for memory leaks

Memory leaks are a serious problem threat in programs that allocate memory dynamically. Moreover, it is often difficult to discover whether a program is leaking memory.

Specific memory monitoring software tools can help the programmers to find out if their programs leak memory. Most of these tools are commercial, but there are a few which you can install and try for free.

- Dr. Memory available for Windows, Linus, and Mac. My experience of using Dr. Memory is that it is easy to install[4] and easy to use. Just make sure that you have added the path to the directory where Dr. Memory is installed to the **`PATH`** environment variable.

  A scientific paper presenting Dr. Memory is available in the proceedings of the "*IEEE/ACM International Symposium on Code Generation and Optimization*" (but, it's not a must to read it to be able to use the tool).

- Valgrind only available for Linux. I have heard good reviews about it.

- Visual leak detector for Visual C++ (guess ... for Visual Studio!). Dr. Memory detects a larger range of memory-related problems when compared to "visual leak detector".

Advised compiler settings

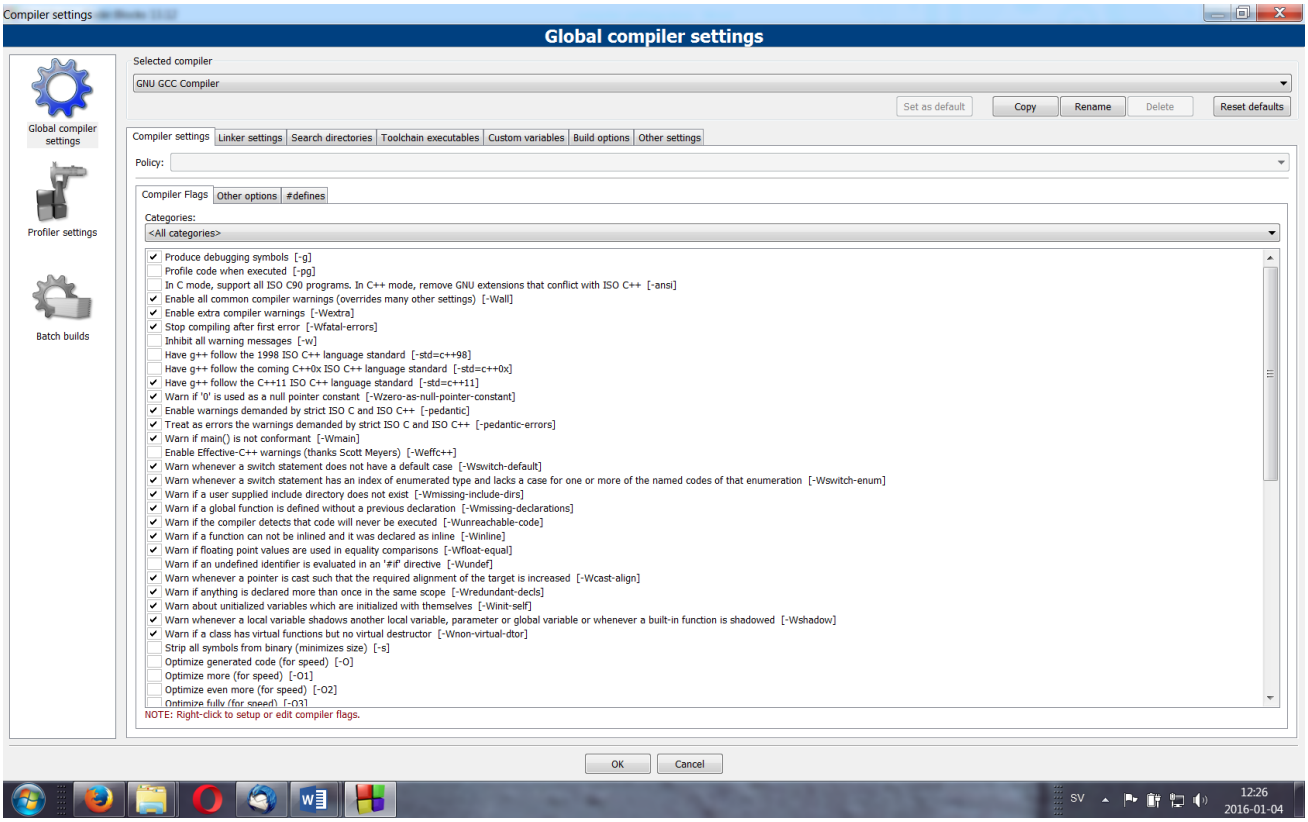
It is very important that you do not ignore the compiler warnings. Although the code may compile and run, compiler warnings are usually a strong indication of a serious problem in the code that may affect badly the program execution at any time.

In **Code::Blocks**, if you follow the procedure described below then you'll be able to see the compiler warnings. The advised compiler settings also make sure that g++ can compile **`C++11`**. Use always the settings indicated below when writing C++ programs in this course. For your personal machines, you only need to set the compiler flags (i.e. options) once. But for the computers in the lab rooms, we cannot promise that the set compiler flags are not changed when you log out and, consequently, you need to set the correct flags every time you log in.

Go[5] to **`Settings`** → **`Compiler`** and click the tab **"`Compiler Flags`**". Then, make sure that the following boxes on the left are selected (see figure below). Finally, click the **`OK`** button.

[4] Dr. Memory is installed in the computers of the lab rooms.

[5] Note that these instructions are valid for Code::Blocks.


Compiler settings
Global compiler settings
Global compiler settings
Profiler settings
Batch builds
Selected compiler
GNU GCC Compiler
Set as default
Copy
Rename
Delete
Reset defaults
Compiler settings
Linker settings
Search directories
Toolchain executables
Custom variables
Build options
Other settings
Policy:
Compiler Flags
Other options
#defines
Categories:
<All categories>
Produce debugging symbols [-g]
Profile code when executed [-pg]
In C mode, support all ISO C90 programs. In C++ mode, remove GNU extensions that conflict with ISO C++ [-ansi]
Enable all common compiler warnings (overrides many other settings) [-Wall]
Enable extra compiler warnings [-Wextra]
Stop compiling after first error [-Wfatal-errors]
Inhibit all warning messages [-w]
Have g++ follow the 1998 ISO C++ language standard [-std=c++98]
Have g++ follow the coming C++0x ISO C++ language standard [-std=c++0x]
Have g++ follow the C++11 ISO C++ language standard [-std=c++11]
Warn if '0' is used as a null pointer constant [-Wzero-as-null-pointer-constant]
Enable warnings demanded by strict ISO C and ISO C++ [-pedantic]
Treat as errors the warnings demanded by strict ISO C and ISO C++ [-pedantic-errors]
Warn if main() is not conformant [-Wmain]
Enable Effective-C++ warnings (thanks Scott Meyers) [-Weffc++]
Warn whenever a switch statement does not have a default case [-Wswitch-default]
Warn whenever a switch statement has an index of enumerated type and lacks a case for one or more of the named codes of that enumeration [-Wswitch-enum]
Warn if a user supplied include directory does not exist [-Wmissing-include-dirs]
Warn if a global function is defined without a previous declaration [-Wmissing-declarations]
Warn if the compiler detects that code will never be executed [-Wunreachable-code]
Warn if a function can not be inlined and it was declared as inline [-Winline]
Warn if floating point values are used in equality comparisons [-Wfloat-equal]
Warn if an undefined identifier is evaluated in an '#if' directive [-Wundef]
Warn whenever a pointer is cast such that the required alignment of the target is increased [-Wcast-align]
Warn if anything is declared more than once in the same scope [-Wredundant-decls]
Warn about unitialized variables which are initialized with themselves [-Winit-self]
Warn whenever a local variable shadows another local variable, parameter or global variable or whenever a built-in function is shadowed [-Wshadow]
Warn if a class has virtual functions but no virtual destructor [-Wnon-virtual-dtor]
Strip all symbols from binary (minimizes size) [-s]
Optimize generated code (for speed) [-O]
Optimize more (for speed) [-O1]
Optimize even more (for speed) [-O2]
NOTE: Right-click to setup or edit compiler flags.
OK
Cancel
SV
12:26
2016-01-04

Lycka till !