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

04/11/2014

Programming in C++
TNG033
Lab 1

Objectives

- To write programs in C++ with arrays and records (**`struct`**).
- To structure programs with functions.
- To divide program's code into header files (**`.h`**) and source code files (**`.cpp`**).
- To define and use overloaded operators.
- To write programs using basic file processing.
- To use redirection of standard input and output.
- To use string objects connected to streams.

Preparation

Review Fö 1 and Fö 2, as well as the examples presented in these lectures.

- Both exercises require reading from streams, i.e. text file or **`cin`**. Thus, pay special attention to slide 20 of Fö 1 and review the examples **`sum_file.cpp`**.
- Exercise 2 requires the use of string stream processing. Thus, review slides 21 to 26 of Fö 1 and the example **`streamstrings_1.cpp`**.

In the appendix of this lab, there is a section about advised compiler settings. Between other things, the advised compiler setting make sure that your compiler can compile **`C++11`**. Make sure you always use them, for every lab 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.

Presenting solutions and deadline

The two exercises are compulsory and you should demonstrate your solution orally during your redovisning lab session on **week 46**.

- Use of global variables is not allowed, but global constants are accepted.
- Exercise 2 is only approved if your code uses string objects connected to streams, when reading the input data.

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 1

Write a program that reads numerical data from a text file, **via redirection of standard input** (see Fö 1), and then computes several statistics. For instance, the mean, median, and standard deviation statistics should be implemented. The standard deviation S of a list of n>0 numbers $x_1, x_2, \ldots, x_n$ is defined below, where $\bar{x}$ is the average of the given list.

$$S = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

Follow the steps given bellow.

1. Download from the course web page the files **`statistics.h`**, **`statistics.cpp`**, and **`main_uppg1.cpp`**.
2. Create a project[1] in Code::Blocks with the files above. You should now be able to compile and execute the program, although it only displays the message "**`No data given!`**".
3. In the header file **`statistics.h`** you find the declaration of several functions to be implemented. Read the comments for each declared function. These functions declaration cannot be modified.
4. Add the definition (implementation) of the functions declared in **`statistics.h`** to the source file **`statistics.cpp`**. Remember to add one function at time, then compile[2] **`statistics.cpp`**, and correct any compilation errors before you proceed to the implementation of another function.
5. Study the **`main`** function given in the source file **`main_uppg1.cpp`**. The **`main`** cannot be modified and is used to test your code.
6. Define the functions **`read_seq`**, **`display`**, **`copy_list`**, and **`sort_list`** declared in **`main_uppg1.cpp`**. These declarations cannot be modified, either. Implement the sorting function with an algorithm different from bubble sort, e.g. selection sort or insertion sort.
7. Compile, link your files, and then execute the program[3].
8. Test your code with the data in **`stat_data_short.txt`** and **`stat_data.txt`**. To this end, you should run the program with redirection of standard input (output). Do not forget to place the input text files in the same folder as the executable file. See also the instructions given in the appendix of this lab. The expected output generated from each of the test files is available in **`stat_data_short_out.txt`** and in **`stat_data_out.txt`**.

---

[1] Instructions of how to create a project in Code::Blocks is given in the appendix of Lab 6, course TND012. Use login = **`TND012`** and password = **`TND012ht2_12`** .

[2] Use **`Ctrl-shift-F9`** for separate compilation of a source code file.

[3] Use **`F9`** to compile all source code files, link, and execute your program.

Exercise 2

The problem

This exercise is about how to calculate the final score for divers, from the scores awarded by seven judges. It is explained below how this calculation is done.

- Seven judges award separately points (a positive real number) to each diver. The highest and lowest points awarded are not considered for the calculation of the final score. The remaining points are averaged and the average is then multiplied by three times the degree of difficulty of the dive.

Each dive's degree of difficulty is also a real number larger than zero.

Your program should start by reading divers data, usually provided in a text file. Correct input data looks as follows.

```
Diver1__name
difficulty1 score11 score12 … score17

Diver2__name
difficulty2 score21 score22 … score27     %some comments

Diver3__name
difficulty3 score31 score32 … score37
…
```

If text appears after the seventh score then it should simply be ignored (but the seven scores are valid). The text file may contain empty lines and white spaces in the beginning of the lines.

For some divers, the data provided in a given text file may not be correct. Specific cases of incorrect divers' data are given below.

- Less than seven scores may be provided for a diver.
- Text (e.g. "`????`") may appear between the scores of a diver (see for instance, **`diver_data3.txt`**).

The program should perform the steps indicated below.

- Start by requesting to the user the name of the text file with the divers' data. The program should read the divers' data directly from the text file (i.e. do not use standard input redirection). Thus, **you need to write code to explicitly handle text files**. If the given text file is empty then the program should display an informative text message to the user. A **log file** should be created with the name of the divers with incorrect data in the given input text file, if any.
- Calculate the final score for each diver (with correct data).
- Sort divers decreasingly by the final score.
- Display the divers name and final score, sorted decreasingly by the final score.

You must use string objects connected to streams when reading the data from the text file. More concretely, when reading the data for a diver, read the second line of

data for each diver (i.e. the line with the degree of difficulty and judge scores) into a string and connect the string to a stream. You can then read from this stream and check whether the line has the correct format.

Steps to follow

To write the described program, follow the steps given bellow.

1. Download from the course web page the files **`diver.h`**, **`diver.cpp`**, and **`main_uppg2.cpp`**.
2. Create a project in Code::Blocks with the files above. You should now be able to compile and execute the program, although it does nothing useful.
3. The header file **`diver.h`** declares a new data type named **`Diver`**, together with an overloaded **`operator>>`** (for reading data for a diver), a function for calculating the score of a given diver, between others. Read carefully the comments preceding each function in the source file. These comments describe what the functions are supposed to do. The header file is a specification and, therefore, should not be changed in any way.
4. Add the definition (implementation) of the functions declared in **`diver.h`** to the source file **`diver.cpp`**. Feel free to add other auxiliary functions to this file.
5. Compile separately the source file **`diver.cpp`**.
6. File **`main_uppg2.cpp`** declares the functions **`display`**, **`sort_divers`** and **`read_divers`**. Read the comments preceding these functions that describe what the functions are supposed to do.
7. Add to the **`main`** function in **`main_uppg2.cpp`** the necessary code to perform all steps of the program, as described above. (No, you do not need to implement the functions **`display`**, **`sort_divers`**, and **`read_divers`**, first!). Compile your program and fix any errors.
8. Implement each of the functions **`display`**, **`sort_divers`** and **`read_divers`**. Remember to implement one function at a time, then compile and fix any errors, before implementing the next function.

   Note that the overloaded **`operator>>`** should be used in the function **`read_divers`**, to read each diver's data.
9. Test your program with all the files indicated in the table below. To this end, place the text files in the same folder where is the Code::Blocks project file. The test files cannot be modified.
10. Finally, change your program such that the user can enter the input data for each diver through the keyboard, instead of being given in a text file. How easy is to modify your program? Did it require many changes in the code?!

Fine Name	Description	Expected Output
`diver_data.txt`	No errors	`out_uppg2.txt`
`diver_data1.txt`	Commented line for diver *Anna Andersson*	`out_uppg21.txt`
`diver_data2.txt`	File with empty lines and extra white spaces	`out_uppg22.txt`
`diver_data3.txt`	Data for divers *Anna Andersson* and *Leene Laine* have errors	`out_uppg23.txt`
`diver_data4.txt`	Data for divers *Anna Andersson* and *Gunilla Rosnhoff* have errors Commented lines	`out_uppg24.txt`

Lycka till!!

Appendix

How to redirect the standard input/output

1. Copy all test data files to the same directory where the executable file is located.
2. Open a **DOS** window
3. Change to the directory where the executable file (**.exe**) is located. To this end, use the **DOS** command **cd**. An example is given below.

```
cd C:\TNG033\Labs\Lab1\statistics\bin
```

4. Run the executable file with redirection of standard input/output. Two examples of the commands, you should enter in the **DOS** window, are given below (assume **statistics.exe** is the name of the executable file). See also notes of Fö 1.

   Redirection of standard input

```
statistics.exe < stat_data.txt
```

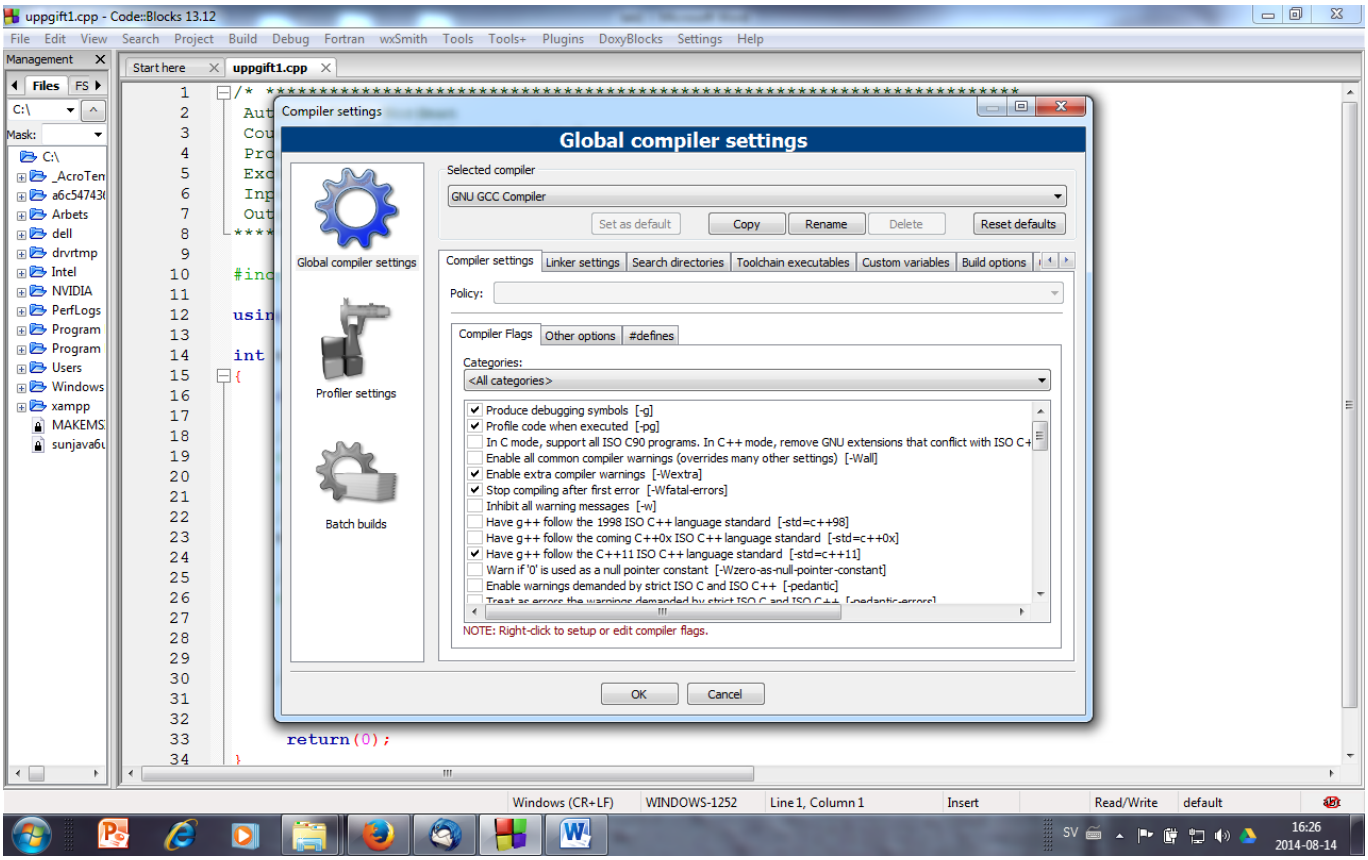
   Redirection of both standard input and standard output

```
statistics.exe < stat_data.txt > stat_data_out.txt
```

Advised compiler settings

In Code::Blocks, go to **Settings** → **Compiler** and click the tab "**Compiler Flags**". Then, make sure that only the following boxes on the left are selected (see figure below). Finally, click the **OK** button.

- Produce debugging symbols
- Enable extra compiler warnings
- Stop compiling after first error
- **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 …


uppgift1.cpp - Code::Blocks 13.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Files FS
Start here
uppgift1.cpp
Compiler settings
Global compiler settings
Selected compiler
GNU GCC Compiler
Set as default
Copy
Rename
Delete
Reset defaults
Global compiler settings
Profiler settings
Batch builds
Compiler settings
Linker settings
Search directories
Toolchain executables
Custom variables
Build options
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++
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]
NOTE: Right-click to setup or edit compiler flags.
OK
Cancel
return(0);
Windows (CR+LF)
WINDOWS-1252
Line 1, Column 1
Insert
Read/Write
default
16:26
2014-08-14