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

06/11/2013

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

As preparation for exercise 1, you should review Fö 1 and pay special attention the example code **`sum_array.cpp`**.

For exercise 2, you need to review Fö 2. More concretely, you need review the following.

- Operator overloading, see slides 15-18 of Fö 2.
- The salesman example discussed in Fö 2 (build a project with the files **`salesman.h`**, **`salesman,cpp`**, and **`sales.cpp`**).
- String streams, see slide 24 and the example **`streamStrings_1.cpp`** of Fö 2.

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 shout 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, 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`** and **`copy`** declared in **`main_uppg1.cpp`**. These declarations cannot be modified, either.
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 instruction given in the appendix of this lab. The output generated from each of the test files is available in **`stat_data_short_out.txt`** and from **`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.

[2] Use **`Ctrl-shift-F9`**.

[3] Use **`F9`**.

Exercise 2

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

> .... Hoppen delas in i olika svårighetsgrader, med obligatoriska och fria hopp. De poängbedöms av sju domare på en tiogradig skala. Den högsta och lägsta poängen tas sedan bort och poängsumman divideras med fem. Resultatet multipliceras därefter med tre och sedan med svårighetsgraden, för att ge så rättvis bedömning som möjligt.

Your program should start by reading divers data. 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
…
```

Input data is provided in text files and your program should read this data by using standard input redirection. Thus, **do not write code to explicitly handle text files**. Your program should be able to cope with a garbled text file and ignore any data for a diver that is not as described above. More specifically, the following situations should be considered.

- Less than seven scores may be provided for a diver. Then, this diver should be ignored.
- Any text may appear in the line with the scores. If text appears after the seventh score then it should simply be ignored (but the seven scores are valid). Any text between the scores implies that the data is not valid and the diver should be ignored.
- Your program should ignore any white spaces and empty lines.

You should use string objects connected to streams (see Fö 2) when processing the input. More concretely, when reading the data for a diver, read the line with the 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.

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 only displays the message "**`No data given!`**".

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. Study the **`main`** function given in the source file **`main_uppg2.cpp`**. The **`main`** cannot be modified and it is used to test your code.
6. File **`main_uppg2.cpp`** declares the functions **`display`**, **`sort`** and **`read_divers`** that you need to implement (the comments preceding these functions describe what the functions are supposed to do). Note that the overloaded **`operator>>`** should be used the function **`read_divers,`** to read each divers data.
7. Compile, link your files, and then execute the program. To test your program you should use the files indicated in the table below. Read these files with redirection of standard input. Some of these files are garbled. The test files cannot be modified.

Fine Name	**Description**	**Expected Output**
`diver_data.txt`	No errors	`out_uppg2.txt`
`diver_data1.txt`	Data for diver *Anna Andersson* has errors	`out_uppg21.txt`
`diver_data2.txt`	Data for diver *Anna Andersson* has errors	`out_uppg22.txt`
`diver_data3.txt`	Commented line	`out_uppg23.txt`
`diver_data4.txt`	File with empty lines and extra white spaces	`out_uppg24.txt`
`diver_data5.txt`	Data for diver *Anna Andersson* has errors	`out_uppg25.txt`

Hint: you are advised to first write code to tackle the described problem for the simpler case, i.e. when the input file is not garbled (**`diver_data.txt`**). Then, after having a program that runs for the simpler case, you should make the necessary changes to tackle the case of garbled input data.

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
   ```