Linköping University
Department of Science and Technology/ITN
Aida Nordman

Introduction to Programming (C++)

TND012

Laboration 3

Course goals

- To learn the basics of working with a debugger.
- To write programs with iteration statements (loops).
- To write programs with **nested** loops.
- To write simple programs with arrays.

During the first 15 minutes of the lab session, the lab assistant will demonstrate how to use a debugger. A debugger[1] is a program that allows you to see what is going on "inside" a program while it is executed or what the program was doing at the moment it crashed. Some basic concepts such as breakpoints, step from one instruction to another, inspecting the value of a variable are demonstrated.

Preparation

You must perform the tasks listed below before the start of the lab session on week 37. In each lab session, your lab assistant has the possibility to discuss about three questions with each group. To make the best use of your time during the lab session, it is important that you read the description of all exercises in this lab and do the indicated preparation steps.

1. Download `deb.cpp` and create a project in Code::Blocks (follow the preparation instructions given in the appendix of this lab). You must create the project directly in your student account before the lab session starts. Creating a project on your private machines and then copying it to a different machine is not guaranteed to work. `deb.cpp` will only be used for the debugger demo.
   The debugger in Code::Blocks is only available, if a project has been created.
2. Review the concepts introduced in Fö 3 and Fö 4.
3. Review the examples presented in lesson 2.
4. Do exercise 1 before your lab session in week 37.
5. Do exercise 2 before your lab session in week 37. Pay special attention to the temperature table example of Fö 3 (`temperature_table.cpp`) before starting this exercise.

[1] The debugger used by Code::Blocks is called GDB (GNU project debugger).

6. Since exercise 3 uses arrays, you should also review the concepts introduced in Fö 5 before starting exercise 3.
7. Often students think that exercise 4 is more difficult than the others in this lab. Review the nested loops examples given in Fö 4.

Your source files should be placed in the folder `TND012\Labs\Lab3.`

Remember that you should start each exercise by writing an algorithm for the problem that indicates the major steps of the program. As usual, these steps should be written in Swedish or English, within comments, in the `main`. Then, encode each of the steps in your algorithm in C++, **one step at a time**. Always compile and run the program before you proceed to the next step in the algorithm.

Presenting solutions and deadline

The four exercises in this lab are compulsory and you should demonstrate your solutions during your lab session on **week 37**. After week 37, if your solution for lab 3 has not been approved then it is considered a late lab. This lab must be presented latest on your lab session of week 38. We also remind you that your code for the lab exercises cannot be sent by email to the staff and that at most two labs can be presented in a lab session

Before presenting your code make sure that it is readable, well-indented, and that the compiler issues no warnings. Otherwise, your lab won't be approved.

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 and your study programme to the e-mail's subject, e.g. "`TND012/ED: …`"..

Exercise 1

Write a program that calculates the total price for several football match tickets. Each ticket price depends on the age of the ticket's owner[2].

- If the owner is over 15 years old then the ticket costs 80 SEK.
- Otherwise, if the owner is at least 8 years old then he pays 30 SEK.
- Children younger than 8 years can get a ticket for free.

Your program should perform the following task by the given order.

1. Ask the number of tickets to be bought. If the number of tickets given is a negative integer then the program should display an error message and request the user to enter the number of tickets again.
2. Request the age of each ticket owner. Assume that the user always gives a non-negative integer for each age, i.e. no need to validate the age.
3. Display the total price to be paid.

A running example is shown below (user input shown in green).

[2] Recall exercise 2 of Lab 2.

```
Welcome to our Football Arena.
------------------------------

Number of tickets: -4
Invalid number of tickets!!

Number of tickets: 4

Enter age for person 1: 5
Enter age for person 2: 12
Enter age for person 3: 20
Enter age for person 4: 25

Total price = 190 SEK
```

Exercise 2

Write a program that displays a table of prices, with and without taxes (*momstabell*). The program requests the user to enter the following data.

- The first and last prices (without taxes) to be displayed in the table.
- The step amount from one price to another.
- The tax in percentage.

Do not use arrays to solve this problem. A running example is shown below (user input shown in green).

```
Enter first price: 10.00
Enter last price: 15.00
Enter price step: 0.5
Enter tax percentage: 25

Taxes Table
Tax = 25.00%
Price tax free      Tax       Price with tax
-------------------------------------------
10.00               2.50      12.50
10.50               2.63      13.13
…                   …         …
15.00               3.75      18.75
```

Your program should test the following possible user input errors and display an error message if needed. The user should be able to re-enter the data in case of error.

- First price or last price or step or tax is a negative number.
- First price larger than the last price.
- Price step is zero.

Exercise 3

Write another program that performs the following tasks, by the given order. No user input validation is needed in this exercise.

- Read the price of 20 products and stored them in an array.
- Read the tax percentage.
- Display a *momstabell* as shown below.

```
Enter products price: 10 12.5 13 8.5 1120 155 200 220 44 150
                      112 75 109 80 2300 157 66 47 449 154
Enter tax: 25
Price tax free     Tax      Price with tax
=========================================
   10.00            2.50          12.50
   12.50            3.12          15.62
   13.00            3.25          16.25
    8.50            2.12          10.62
 1120.00          280.00        1400.00
  155.00           38.75         193.75
  200.00           50.00         250.00
  220.00           55.00         275.00
   44.00           11.00          55.00
  150.00           37.50         187.50
  112.00           28.00         140.00
   75.00           18.75          93.75
  109.00           27.25         136.25
   80.00           20.00         100.00
 2300.00          575.00        2875.00
  157.00           39.25         196.25
   66.00           16.50          82.50
   47.00           11.75          58.75
  449.00          112.25         561.25

  154.00           38.50         192.50
```

Exercise 4

In cryptarithmetic puzzles, mathematical equations are written using letters. An example is shown below.

```
SEND + MORE = MONEY
```

To find a solution to the puzzle, i.e. the numbers corresponding to `SEND`, `MORE`, and `MONEY`, the following rules must be followed.

- Each letter (e.g. ´`S`´, ´`E`´) represents a digit from `0` to `9`.
- No two different letters can represent the same digit.

Thus, making `S = 9`, `R = 8`, `O = 0`, `M = 1`, `Y = 2`, `E = 5`, `N = 6`, `D = 7`, we get the solution `SEND = 9567`, `MORE = 1085`, and `MONEY = 10652` (indeed, `9567 + 1085 = 10652`) for the puzzle above.

Write a program that finds solutions to the puzzle

```
TOO + TOO + TOO + TOO = GOOD
```

Your program should display which numbers are represented by `TOO` and by `GOOD`.

Note that the puzzle above has two solutions, indicated below. The program should find and display both solutions.

```
TOO = 166 and GOOD = 664
TOO = 499 and GOOD = 1996
```

Hint: Follow the steps described below.

1. Write a program that assigns every possible digit, from `0` to `9`, to each letter (`T`, `O`, `G`, and `D`) and displays the long sequence in the `uppgift4_out.txt`[3] (with 10^4 cases). Use **nested loops** to generate this sequence.
2. Modify the program above such that only the sequences where different letters represent different digits. The displayed sequence is available in the file `uppgift4b_out.txt`.
3. Modify the program from the previous step that only the cases corresponding to solutions of the puzzle are displayed.

Lycka till ☺

[3] Both files `uppgift4_out.txt` and `uppgift4b_out.txt` can be downloaded from the course web site.

Appendix: 15 Minute Intro into Using a Debugger

Create a project with the source file **deb.cpp**. In Code::Blocks, the debugger only works, if there is a project. In addition, go to "**Settings → Compiler**" and make sure the box "**Produce debugging symbols**" is checked.

Preparation

Create a project in Code::Blocks

Select **File → New → Project → Console application**

Follow the wizard (choose C++ option, set the name of the project to "**Debugging**", and place it in the folder TND012\Labs\Lab3\).

After the wizard has terminated, click on **Sources** (in the left pane). You should be able to see a file named **main.cpp**.

Add your own source file (deb.cpp)

1. Right click on main.cpp → **Remove file from project**
2. Right click on project name → **Add files… → select deb.cpp → click OK**
3. **deb.cpp** should now be listed under the project name in left side pane.

Debugging in Code::Blocks

The file **deb.cpp** contains a single program for debugging.

Start Debugging, Option 1

1. Position cursor at the first statement of the main function

Debug → Run to cursor (or press F4)

The program will be executed until it reaches the cursor position, then it will pause.

Start Debugging, Option 2 using breakpoints

1. Position cursor at a statement,

   set a breakpoint (**Debug → Toggle Breakpoint**, or press F5)

Debug → Start / Continue (or press F8)

The execution will pause at the first breakpoint it encounters. A list of all set breakpoints is accessible via **Debug → Debugging Windows Debug → Breakpoints**.

Stepping through the program

To continue with the next statement, select **Debug → Next Line** or press F7.

Using **Debug → Start / Continue** (or press F8) will continue till the next breakpoint or the end of the program.

Demo 1

1. Try to debug the example given in **deb.cpp**.

   Start the debugger from the first line of the main function.

   Inspect the variable **n** with a right click on **n** and select **Watch 'n'**.

A list of all inspected variables and watches is accessible via **Debug → Debugging Windows Debug → Watches**.

Continue to the next line (F7) and see how **n** changes.

Continue to the loop and inspect sum for two or three iterations

In order to leave the while loop, place the cursor on the line where the sum is printed.

Then either add a breakpoint (F5) and continue (F8) **or** run to cursor (F4).

Inspect the sum and compare it to what is printed in the console window after stepping to the next line (F7).

The idea of this demo is to show that when loops iterate many times, one can inspect some iterations and then just exit the loop (without having to step through each iteration).

Demo 2

Stop the debugger if it is still running (**Debug → Stop debugger** or Shift+F8)

Uncomment the last line in the main function.

Remove all breakpoints (**Debug → Remove all breakpoints**)

Start the debugger (F8)

The idea of this demo is just to show what happens when there is a run time error or an exception.