Linköping University
Department of Science and Technology/ITN
Aida Nordman

Introduction to Programming (C++)

TND012

Laboration 1

Course goals

- To learn the basics of working with an integrated development environment (IDE), **`Code::Blocks`** (**`v 16.01`**).
- To write small programs using assignment, arithmetic expressions, and basic input and output.
- To save, compile, and execute a **`C++`** program.
- To understand, find, and correct compilation errors and warnings.
- To describe the following concepts: source code, object code, and executable code.

Preparation

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

- Read the **`Labs`** section of the course info document. Read also the section about how to contact the staff in the course.
- Review the concepts and code examples presented in Fö 1.
- Create a folder named **`TND012`**. Then, create inside this folder another folder called **`Labs`** which in turn should have inside six other folders named **`Lab1`**, **`Lab2`**, …, **`Lab6`**. Each of the six folders will contain your files with the solutions for the exercises for each of the lab sessions in this course.
- Read the appendix of this lab, "Advised compiler settings". Between other things, the advised compiler setting make sure that your compiler can compile **`C++11`** (a more modern version of C++). Make sure you always use the indicated settings 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.
- Do exercise 1.

Presenting solutions and deadline

The three exercises in this lab are compulsory and you should demonstrate your solutions during your lab session in **week 35**. After week 35, if your solution for lab 1 has not been approved then it is considered a late lab. This lab must be presented latest in your lab session in week 38.

We remind you that in this course a late lab can be presented in another lab session provided there is time.

Before presenting your code make sure that it is readable, well-indented, and that the compiler issues no warnings.

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

1. Download the source file **`uppgift1.cpp`**. Then, double-click on it to start **`Code::Blocks`**.
2. The file **`uppgift1.cpp`** contains a C++ program. Try to understand what the program is supposed to do.
3. Compile and execute the code. To this end, you can select

   `Build` → **`Build and run`**

   or simply press **`F9`** on the keyboard.

   a. Note whether any compilation errors or warnings occur. Compilation errors are indicated through a red square after the line number. Warnings are not automatically visible.

   b. To see a list of warnings and the description of compilation errors, press **`F2`** on your keyboard after compilation.

   c. Identify which instructions generated the compilation errors or warnings. Then, add a comment with the message of the corresponding error/warning obtained.

   d. Correct any warnings or compilation errors.

5. After successful compilation, are there any new files in the same folder where the source file **`uppgift1.cpp`** is located? What are the names of those files and what are they for?
6. Finally, add the necessary code to the program such that it also converts from **`SEK`** to **`Euro`**. As before, the program should ask the user how many Swedish crowns are to be converted to Euro. Then, compile and execute the program.

Exercise 2

You are now going to write your own program to solve a small problem from scratch.

Write a C++ program that displays how much electricity has been used in the last year. Electricity is measured in kilowatt hours (kWh). Your program should start by reading in how many kilowatt hours the electricity meter indicates now (**`int`**) and how much it showed a year ago (**`int`**). Below, you can see an example of the execution of the program (user input is shown in green).

```
Enter electricity meter reading a year ago: 100
Enter electricity meter reading now: 300

You have used 200 kWh during the last year.
```

To solve this exercise, proceed as indicated below.

1. Create a new source file named **`uppgift2.cpp`** (see instructions in the appendix). Then, write in the beginning of the file a comment similar to the one of the previous exercise.
2. Describe, in Swedish or English, each of the steps of your program, before starting the code. Each step should be written within a comment in the **`main()`** function of file **`uppgift2.cpp`** (see the comments in **`uppgift1.cpp`**). Note that these steps correspond to an **algorithm** for the program.
3. Write the **`C++`** code for each of the steps above, **one step at a time**. Then, compile and run the program before you proceed to the next step in the algorithm. Note that is important that you correct any compilation errors/warnings and test if the program is behaving as expected so far, before you proceed to the next step of the algorithm. This way, you develop the program in an incremental way. This avoids ending up with a long list of problems just in the end (which is usually more time consuming).

Exercise 3

Modify the program of the previous exercise such that it also requests the user to enter the regular price of 1 kWh. The energy cost for one year should take into account that 10% of the used energy costs 5% more than the regular price. The other 90% of the used energy is charged at the regular price. Note that the energy costs should be displayed with two digits after the decimal point. To control the number of digits that should be displayed after the decimal point you need to use **manipulators**. Since we have not yet discussed manipulators, we suggest that you read page 23 of the course book or request assistance of your lab assistant.

You can see below an example of the execution of the new program (user input is shown in green).

```
Enter electricity meter reading a year ago: 158
Enter electricity meter reading now: 388
Enter regular price of 1 kWh: 10.5

You have used 230 kWh during the last year.
Total cost = 2427.07 SEK.
10% of the used energy costs 253.57 SEK.
```

To solve this exercise, proceed as indicated below.

1. Create a new source file named **`uppgift3.cpp`**. Then, write in the beginning of the file a comment similar to the one of the previous exercise.
2. Extend the algorithm of exercise 2 with extra steps needed for this exercise, before starting to write C++ code. Each step should be written within a comment in the **`main`** of file **`uppgift3.cpp`**.
3. Write the code for each of the steps of the algorithm. Use a stepwise strategy as for exercise 2 (see point 3 of exercise 2).
4. Avoid the use of *magic numbers* (e.g. **`0.1`**, **`0.05`**) in your program. Instead, declare meaningful constants and use them in the program.

Lycka till ☺

Appendix

How to create a new source code file (`.cpp`) with `Code::Blocks, v16.01`.

1. Start **`Code::Blocks`**.
2. Create a new source code file named, for instance, **`uppgift2.cpp`**. To this end follow the steps below.
   a. Select **`File`** → **`New`** → **`File`**... → **`C/C++ source`** and then click the button **`Go`**.
   b. A wizard will start running.
   c. When requested "*Please select the language for the file*", select **`C++`**.
   d. When requested "*Filename with full path*", then click on the square with "...", choose the correct folder for your file, and enter the file name **`uppgift2.`**
   e. Click on **`Finish`**.
3. You can now write your code, save it (**`Ctrl-S`**), compile and execute it (**`F9`**).
4. Press **`F2`** on your keyboard to see the list of warnings and the description of any compilation errors, if needed.

Advised compiler settings

Compilers can issue **errors** and **warnings** about your code.

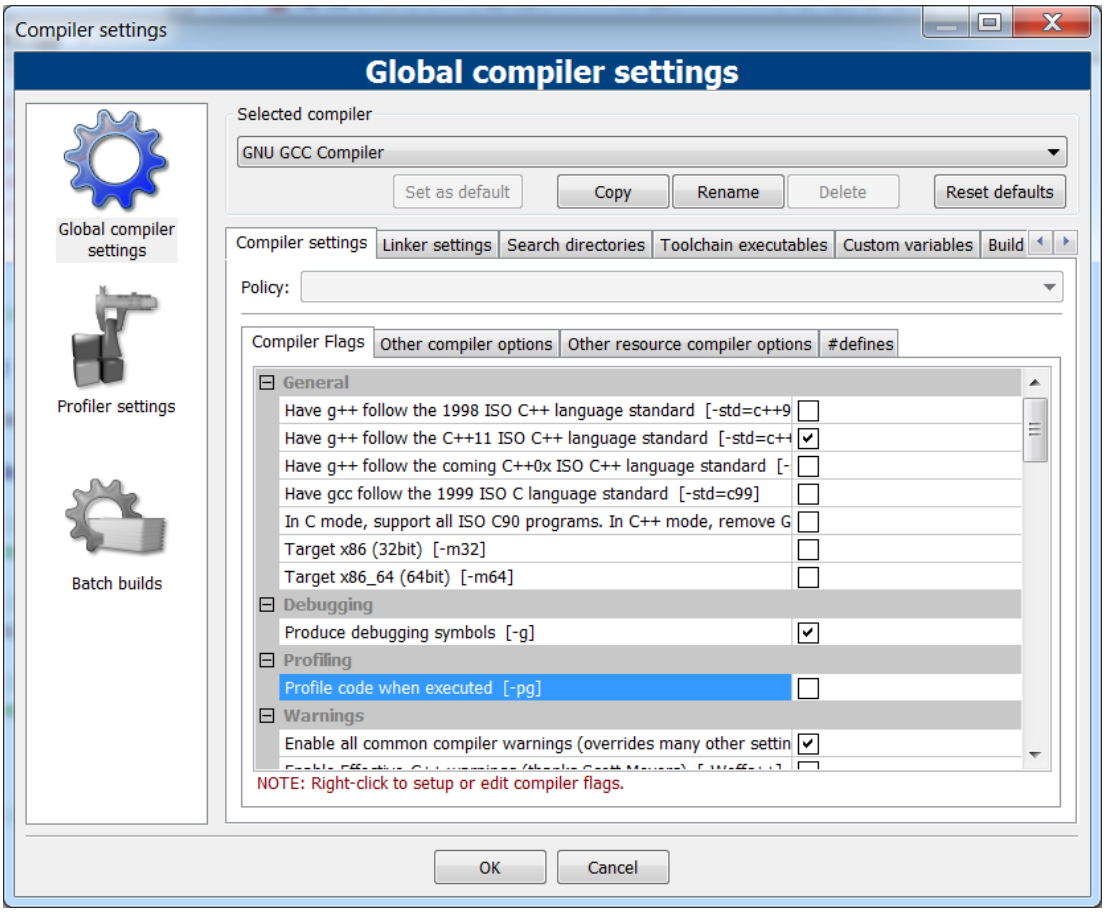
An error implies that your program is not written according the rules of the C++ language (in other words, your program is not a C++ program). Consequently, the program can neither be translated to binary code nor be executed.

Compilers will always warn you about things that might be difficult to find during testing. For example, your compiler can warn you that you are using an uninitialized variable. Uninitialized variables can cause errors while the program is executing and it may be difficult to spot the problem when you are testing the code (e.g. the variable declaration is far from the statement where you first use it).

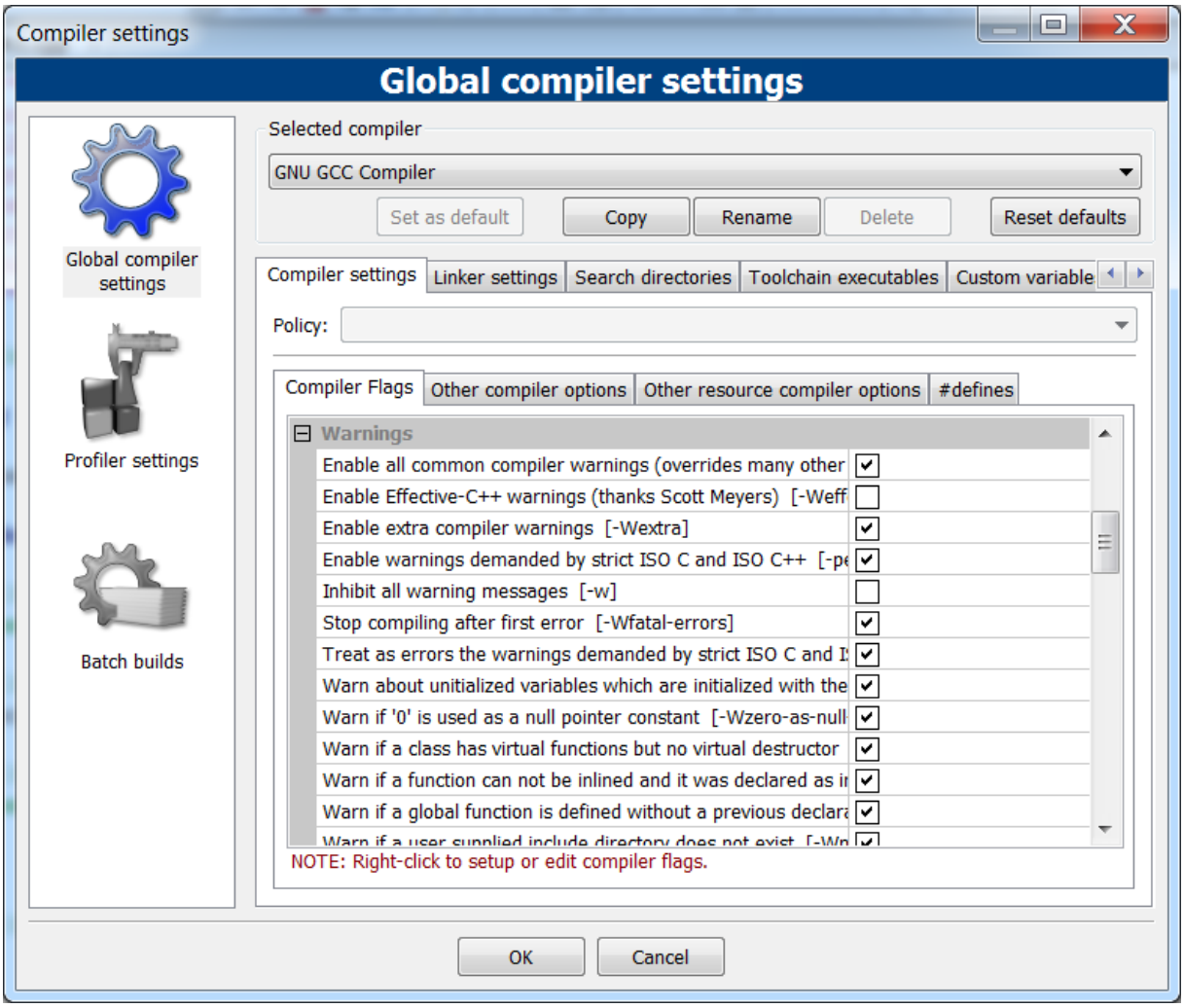
A program can be executed, even if the compiler issues warnings about the code. However, it is very important that you do not ignore the compiler warnings. Although the code may compile and run, compiler warnings are usually a strong indication of a serious problem in the code that may affect badly the program execution at any time.

In Code::Blocks, if you follow the procedure described below then you'll be able to see the gcc compiler warnings. If you use another IDE then you'll need to check how to see all important compiler warnings.

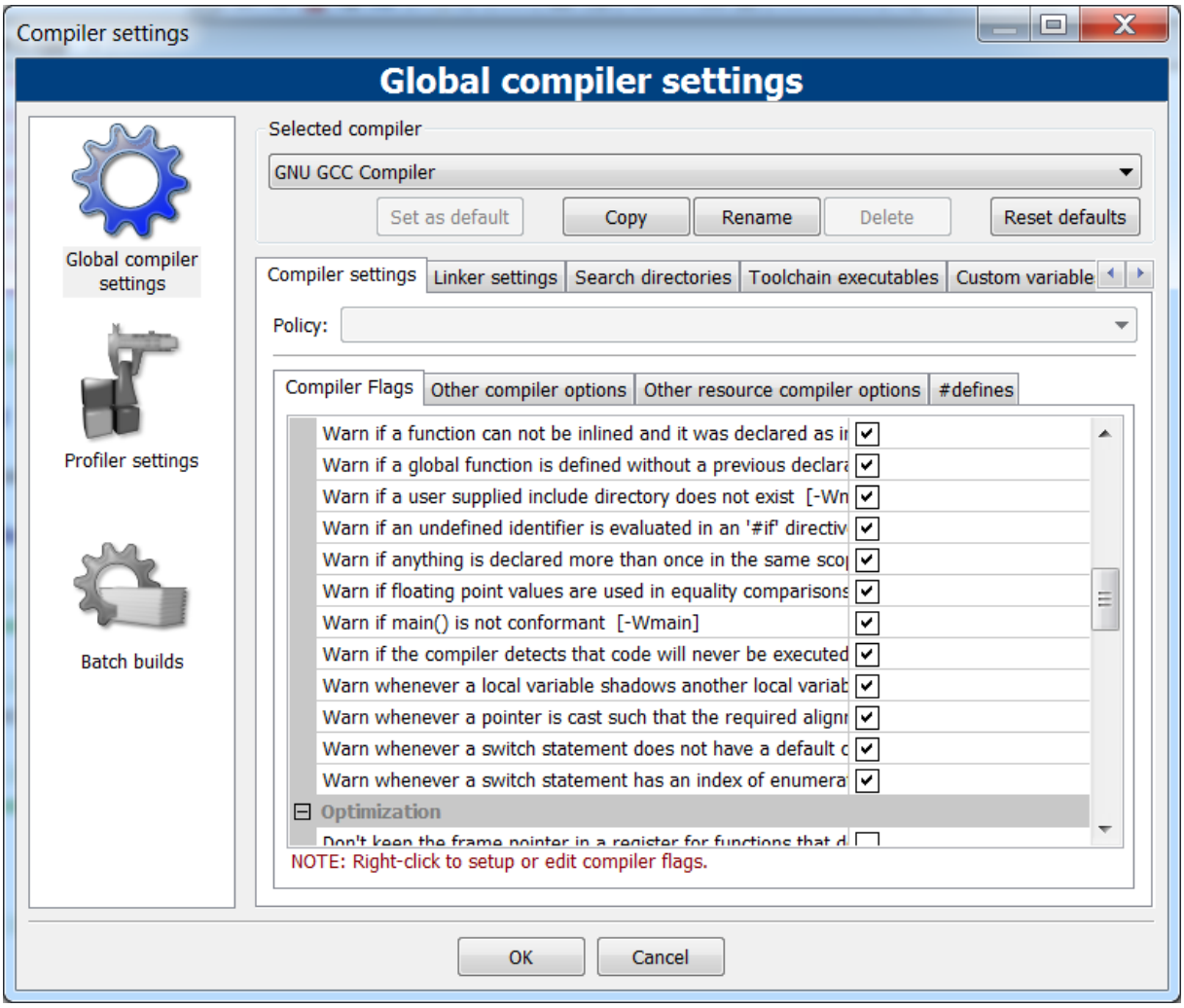
Go to **`Settings`** → **`Compiler`** and click the tab "**`Compiler Flags`**". Then, make sure that the same boxes as in the figures below are selected. Finally, click the **`OK`** button.



(continues next page)



(continues next page)


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 variable
Policy:
Compiler Flags
Other compiler options
Other resource compiler options
#defines
Warn if a function can not be inlined and it was declared as i
Warn if a global function is defined without a previous declara
Warn if a user supplied include directory does not exist [-Wn
Warn if an undefined identifier is evaluated in an '#if' directiv
Warn if anything is declared more than once in the same scop
Warn if floating point values are used in equality comparisons
Warn if main() is not conformant [-Wmain]
Warn if the compiler detects that code will never be executed
Warn whenever a local variable shadows another local variab
Warn whenever a pointer is cast such that the required alignr
Warn whenever a switch statement does not have a default c
Warn whenever a switch statement has an index of enumera
Optimization
NOTE: Right-click to setup or edit compiler flags.
OK
Cancel