C++ Lab1: Getting started with Code::Blocks

- *The source code needed for this lab is available on Moodle.*
- *A list of useful (English) programming vocabulary is given at the end of this lab.*

Objectives

1. Discover a programming environment.
2. Compile and run simple programs.
3. Respect the C++ syntax to write simple programs.
4. Test and correct a program.

General instructions

- In your home directory on the IUT network, create a folder (or directory, same thing) reserved for your programming exercises (name it **C++** for instance), in which you will create one directory per lab (directory **Lab1** for this lab for instance).
- Download all the **Lab1** files that are in the Moodle workspace and copy them to the **Lab1** folder you just created.
- Do not forget to save your work on the IUT network: it will therefore be accessible from any computer (including at your home).

WORK TO HAND IN: Fill in, as you proceed through the exercises, the file answerSheetLab1.docx and upload it onto Moodle (in "Fiche rendu TP1") at the end of the class. Do not spend time formatting your answers; they should be precise and brief.

1: Using Code::Blocks for the first time

a) If you have not yet done so, create a **C++** directory, and within it, a **Lab1** subdirectory. Reminder: use your home directory on the IUT network.

b) Open the Code::Blocks programming environment from the "Démarrer" menu of your computer. This environment, like many others, is organized into projects. You will therefore need to create one project per exercise.

c) We will start by creating a ready-to-use project. Follow this method each time you have to input all the code. In the next exercise, we will see how to create a project using existing code. Click on "***Create a new project***", then "***Console Application***", "***Go***" (no need to click on "***Go***" if you double-clicked on "***Console Application***"), *"**Next**",* "***C++***", and finally ***Next****"* again.

d) Give a name (***title***) to your project (**ex1** for instance; notice how this name is added in the filename fields), click on **"…"** to check its location (it should be in C++\Lab1, change it if necessary), then click on ***"Next"***. ***A subdirectory having the same name as the project is automatically created.*** Finally, click on "***Finish***" (the default values should be ok).
The project opens with a file main.cpp that has been automatically created. Click on the '+' sign next to the "Sources" directory to see it, and double-click on the file name to see its contents. **Look for its location in your directory and note it on the answer sheet**.

e) Compile the project by clicking on the **Build** icon in the toolbar at the top of the window (the little yellow wheel), or by selecting "***Build***" in the **Build** menu, or by typing **Ctrl-F9**.
Two subdirectories have now been added to your project: one for the object files (obj) and another for executable files (bin**). Look for them and note their location on the answer sheet**.

f) Run the project by clicking on the **Run** icon (green arrow), or by selecting "***Run***" in the **Build** menu, or by typing **Ctrl-F10**.
Note: you can compile and run with a single command: ***Build and run***, or **F9**, or the third icon, combining the arrow and the wheel.

g) Modify the program so that it will display your own message instead of "**Hello world**".

Compile again (this will automatically save the changes) and run again.

Modify the Code::Blocks settings to display line numbers in your code: Settings > Editor > Other editor settings > Show line numbers.

Before going on to the next exercise, do a right-click on the name of your project and select "***Clean***". **This will delete the executable and object files**, freeing up memory space.

h) Finally, you can close the project by right-clicking on its name and selecting "***Close project***". This is not required, but having several projects open at the same time may be confusing.

Exercise 2: Creating a project from an existing file

Sometimes you will need to use an existing code file to start a new project. In this case you do not need the main.cpp file that was automatically created in the preceding exercise, it might even bother you. Creating a project will therefore be slightly different.

a) Create a new project (File→New→Project) and select "***Empty Project***". Give it a name as before (ex2), and continue until "***Finish***". A new directory is created (ex2 this time)

b) Copy the **error.cpp** file from the Lab1 directory on Moodle into this new directory (here: C++\Lab1\ex2).

c) Add this file to your project. To do this, right-click on the project name, select "***Add files***", then choose the file to copy, and then "***Select All***" for it to be added where it is needed.

d) Start by making this file more readable. That is:
- put a single statement per line,
- correct the indentation: offset to the right when opening a bloc and to the left when closing a block.

Next time you need to correct indentation, do it **automatically** by selecting the "***Source code formatter***" in the **Plugins** menu. Careful, if your code contains errors, then the indentation will not be correct. Correct your code and try again.

e) Compile your program. What do you notice?

Read the error messages and correct the errors ***one by*** one. Compile again and go on to the next one until no compilation errors remain. Be sure to correct only compilation errors (those that prevent the compilation from succeeding)

What's the purpose of the compilation process? (Remember to fill in the answer sheet)

f) Run the program entirely. Is the output the one you expected? This is called a run-time error. Explain it on the answer sheet.

Source code can therefore be transformed into an executable file (meaning the compilation was successful), but its execution may not correspond to what was expected. Change the program for it to produce the expected result.

Remember: do not confuse compilation errors and runtime errors!

Exercise 3: variable scoping

a) By proceeding as in exercise 2, create a new empty project. Copy into it the **variables.cpp** file from the Moodle workspace.

b) Compile it: what happens? Why?

c) In which bloc(s) can the variable ***n*** be declared? Why?

d) Declare ***n*** in one of these blocs, with an **int** type, then compile and run the program. How do you explain the value that is displayed?

e) Declare ***n*** in the bloc b1, then assign to it the value 3 in this same bloc. Compile and run the program. Everything should now be ok!

Exercise 4: Your turn

Create a new project, as in exercise 1, and write a program that asks the user to enter their name, and then prints "`Hello` …" using the name entered by the user (Use a string variable). Run and test your program.

It is important to know how to run your program outside of the Code::Blocks environment. In which directory has the executable file been saved? What is its name? Test its execution **outside Code::Blocks.**

Note: To do this, you might need to select the 2 static libraries libgcc and libstdc++ in the compilation parameters (**Settings** menu). Once you've tested the execution outside Code::Blocks, remove these library settings as they slow down compilation enormously.

Note: Under Windows, the execution window closes as soon as the program ends (you can hardly see the window!). To keep it open, add the statement `system("PAUSE");` just before the return statement (this requires adding `#include <cstdlib>`). Careful: not compatible with Unix.

Exercise 5: Your turn again: first program to hand in

Write a program that asks the user to input two variables of type **double**, displays them, exchanges their content and displays them again (without using the swap function of course!). Copy your code onto the answer sheet and state what values you used to test it.

Exercise 6: Strange divisions …

As you noticed in exercise 2, a program that compiles is not necessarily a program that is correct. There can remain logic errors that a syntactic test cannot detect. (Remember the difference between compilation errors and runtime errors).

a) By proceeding as in exercise 2, create a new empty project, and copy into it the file **bizarre.cpp** from the Moodle workspace.
b) Compile and run the program with the values 10 and 3, then 10 and 0. Do not run the debugger. What do you get?
c) Change the type of the variable **x** from **int** to **double** and run the program again with the same values. What do you now get?

Exercise 7: Average temperature

a) Write a program that asks the user to enter 2 temperatures (use **int** variables) recorded during a day (for instance one for the morning and another for the evening) and then computes and displays the average temperature of the day. Careful: the average should be a **float**.
b) Change your code to consider a 3rd temperature.

c) Change your code to use the same variable for inputting all 3 temperatures. You can of course keep a separate variable for calculating the average. For those of you who have already been programming: no loops and no arrays here!

d) Add a comment at the end of your program to list the tests you ran to verify your code.

e) Copy your code (with the test cases) onto the answer sheet **AND** upload the code file (only the main.cpp file) onto Moodle.

Programming Vocabulary Lab1

- **directories, files and paths** — *répertoires, fichiers et chemins d'accès*
    - C++/Lab1/ex1/main.cpp is a path to the main.cpp file.
    - C++ is a directory, also called a folder (so are Lab1 and ex1).
    - All directories are subdirectories of your root directory.
- **to run a program** — *exécuter un programme*
    - to execute a program
- **a run-time error** — *erreur d'exécution*
    - an error that occurs during the execution of a program. Not to be confused with a compilation error.
- **the program crashed** — *le programme a planté*
    - an unexpected termination of a program. May result in an error message, or a "froozen" screen …
- **a statement** — *instruction*
    - a block of code that does something
- **scope of a variable** — *portée d'une variable*
    - that part of the source code where the variable is declared, and can therefore be accessed from.
- **to display data** — *afficher des données*
    - to output information to a screen (for a user)
- **to enter data** — *saisir des données*
    - to input information from a user (on a screen)
- **input / output** — *entrée / sortie*
    - << is the output operator, >> is the input operator. They are also called the extraction (>>) or insertion (<<) operators.
    - cin and cout (these are not statements) are C++ iostream library objects that allow access to "standard input" and "standard output", usually the terminal.
    - here is a statement outputting (or displaying) information to the screen:
      `cout << "Hi there ! Please enter your name: ";`
    - here is a statement inputting (or entering) a value from the user:
      `cin >> myName;`
- **truncated versus rounded date** — *données tronquées/arrondies*
    - if the number 9.876 is truncated, it becomes 9, if it is rounded to the nearest integer, it becomes 10.

FYI *(for your information)*
The C++ reference book: The C++ Programming Language by Bjarne Stroustrup.
And from his **FAQ** *(frequently asked question)* pages: http://www.stroustrup.com/bs_faq2.html

- **How do you pronounce "cout"?**
  "cout" is pronounced "see-out". The "c" stands for "character" because iostreams map values to and from byte (char) representations.
- **How do you pronounce "char"?**
  "char" is usually pronounced "tchar", not "kar". This may seem illogical because "character" is pronounced "ka-rak-ter", but nobody ever accused English pronunciation (not "pronounciation" :-) and spelling of being logical.

For more information: There are many websites listing programming terminology, just google them. There are also many websites offering C++ programming tutorials (here for instance), and more and more universities are offering online C++ programming courses (here for instance). If you have the time and motivation, there's a lot out there to improve your English … and programming skills!