Code::Blocks Students' Guide

This document is meant to make it easy for you to install the IDE we will be using this semester, ensure it is working, learn about its most useful features. It is organized as follows;

S1 – Installing Code::Blocks IDE

S2 – Testing the IDE

S3 – Debugging Programs

Feel free to ask any question you might have on the offering's forums. Make sure you work on all steps in this guide, including preparing a .zip file of the entire PA203 project folder you when you are done. This will prepare you for dealing with future graded assignments.

S1 – Installing Code::Blocks IDE

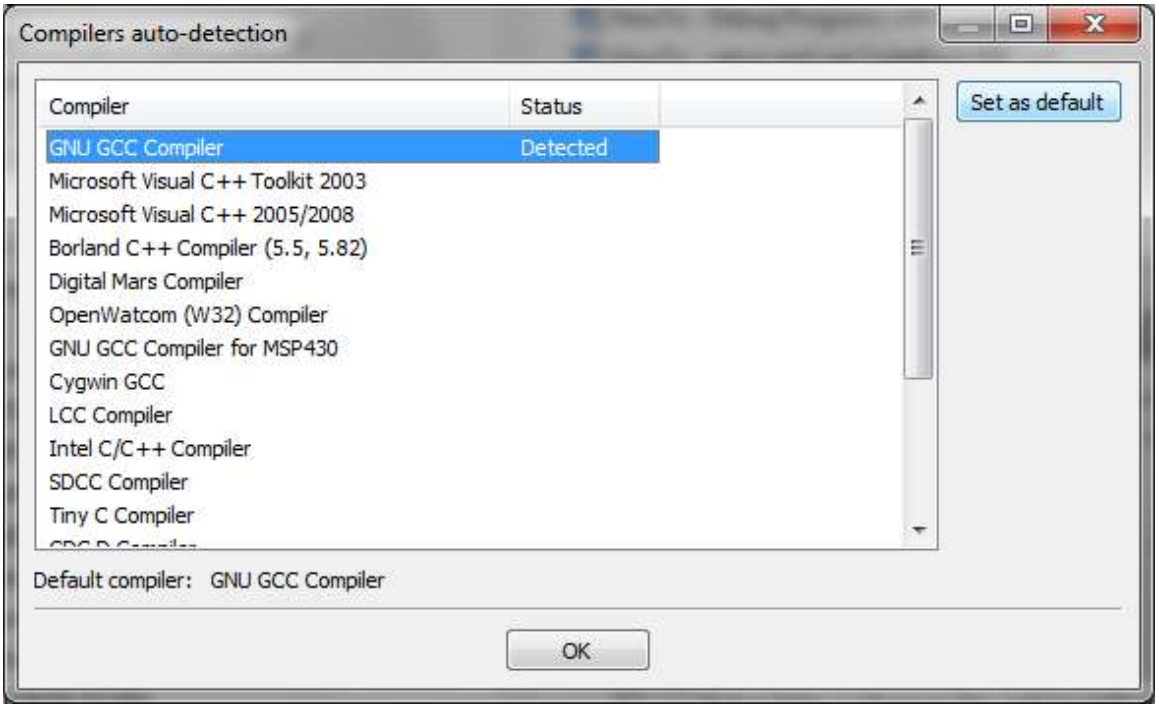
S1.1 – Downloading & Installing the IDE

The installation process is pretty straightforward. Download the **codeblocks-10.05mingw-setup.exe** file from blackboard and run it. Accept all default settings.

Run Code::Block as soon as the installation is finished.

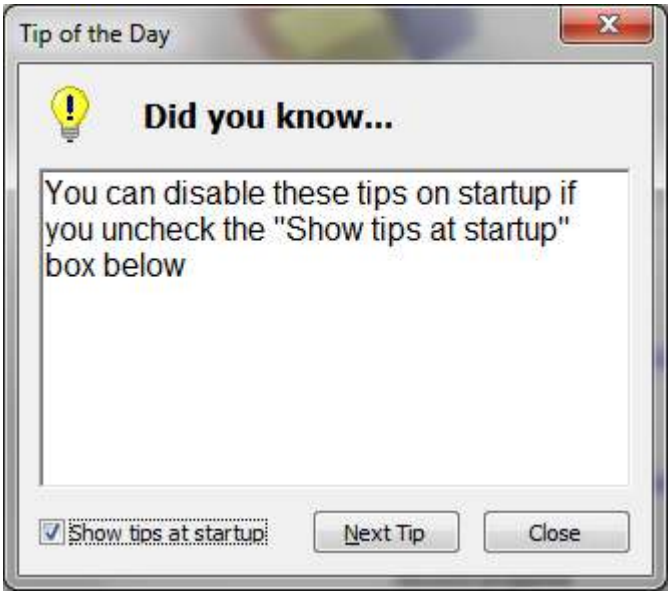
S1.2 – First run

You will see a dialog window showing all available compilers. This installer also installs the GNU GCC Compiler on your machine which is the only one which will be compatible with the systems we will be using to grade your assignments. Select **GNU GCC Compiler** from the list and click on Set as default.

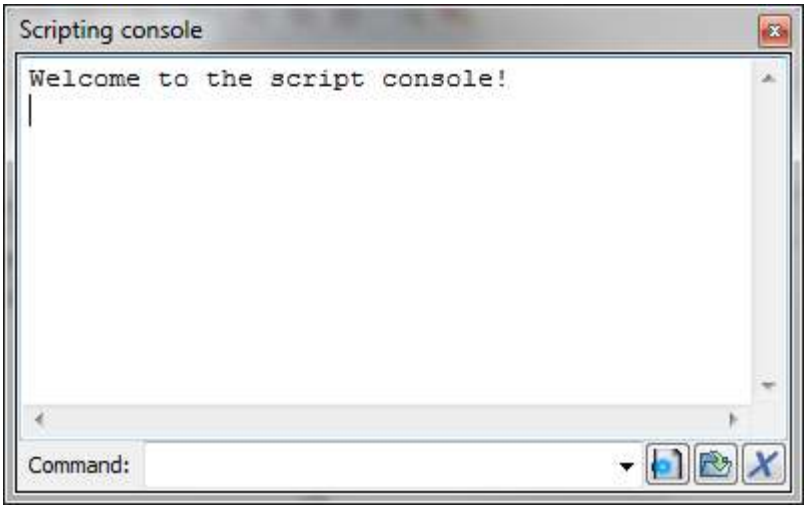


You should now see 3 windows opened;

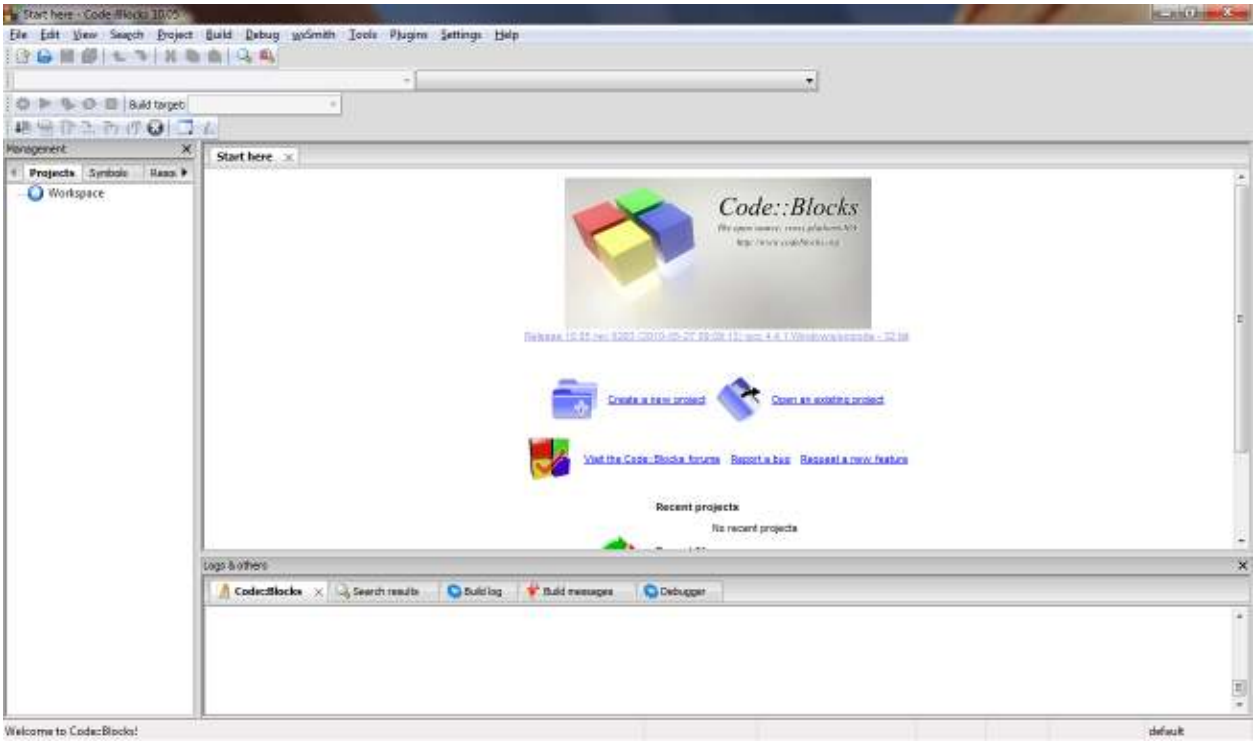
- The Tip of the Day – Feel free to close it for now



- The scripting console – feel free to close it for now.



- The main IDE panel

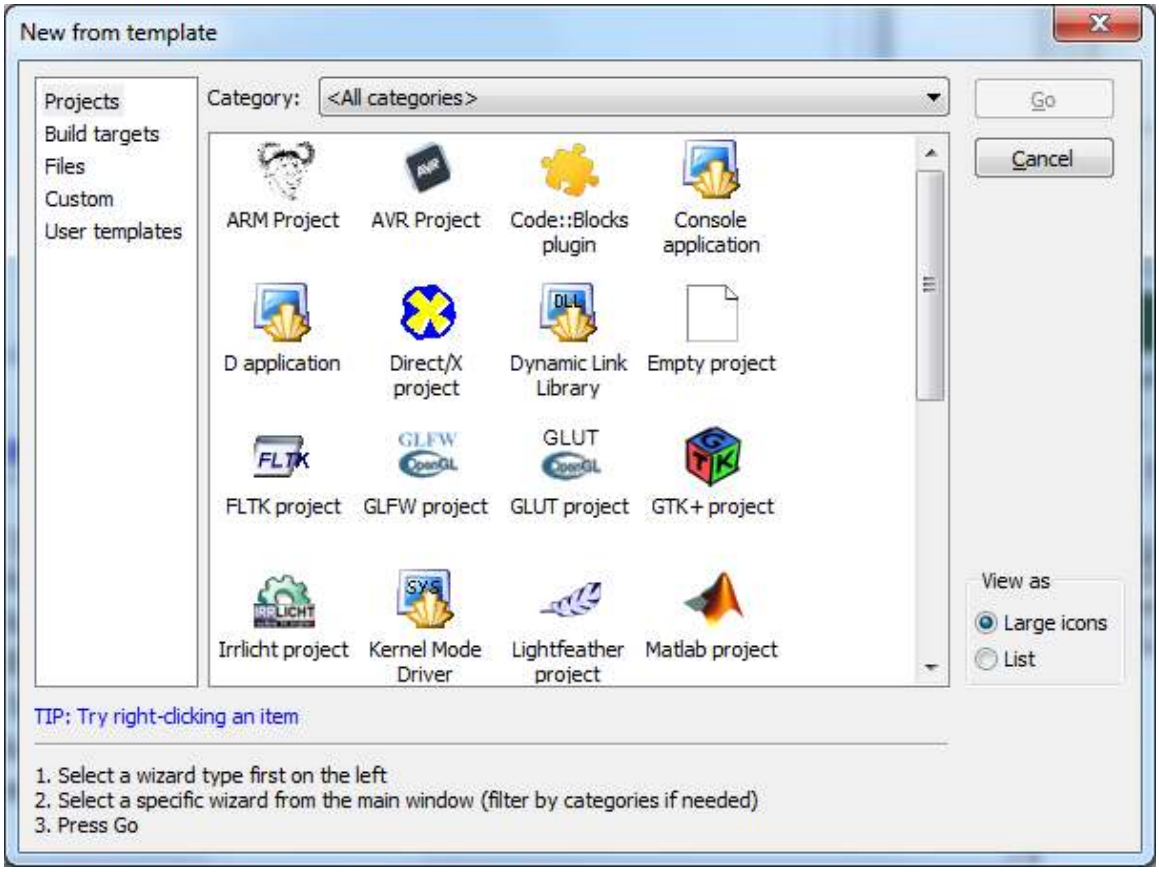


You are now ready to work with this IDE.

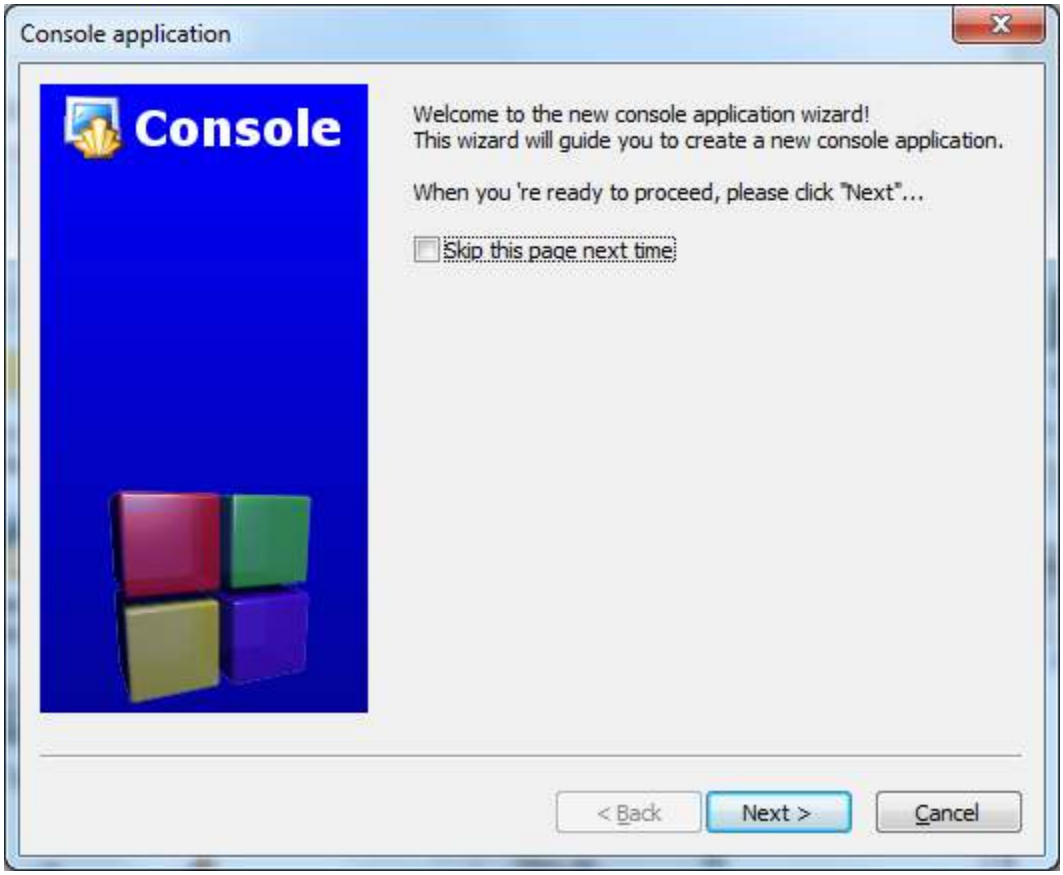
S2 – Testing the IDE

S2.1 – Preparing a new project

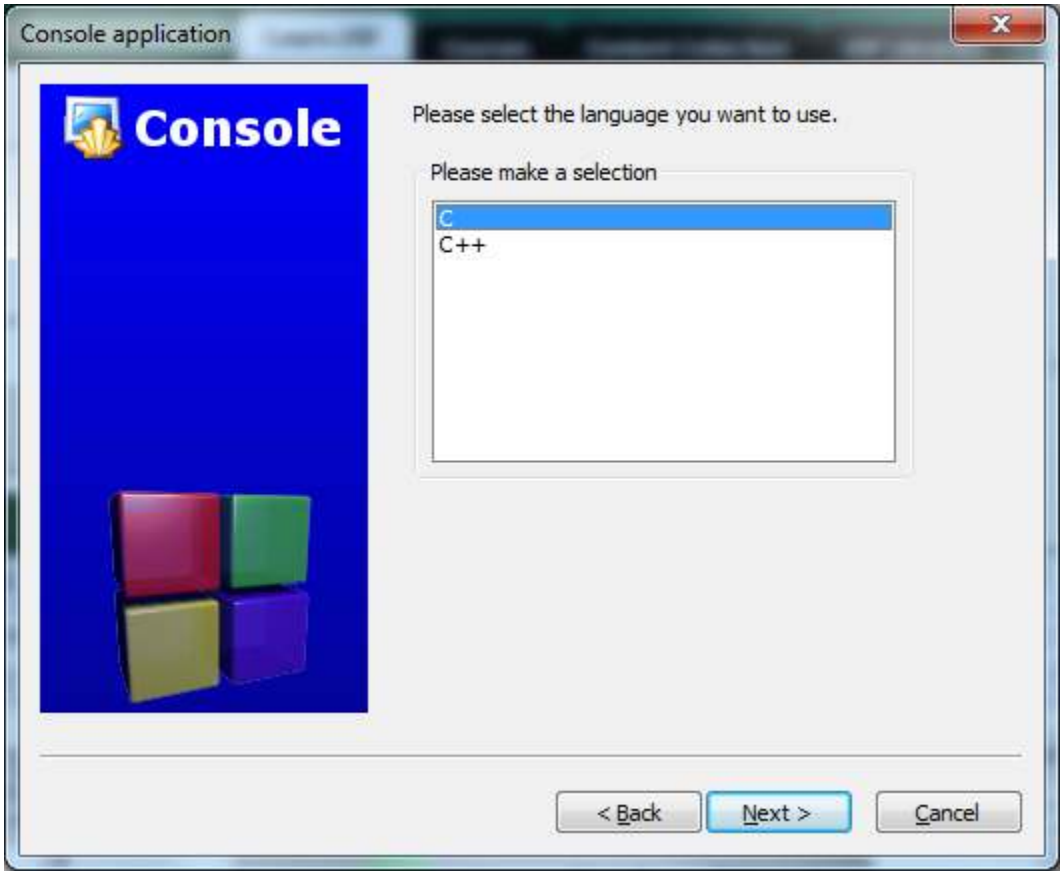
Let us prepare a new project by navigating to the Code::Block menu File → New → Project. This will open the project pane illustrated in the screenshot below.



We will be always selecting Console Application in this offering.

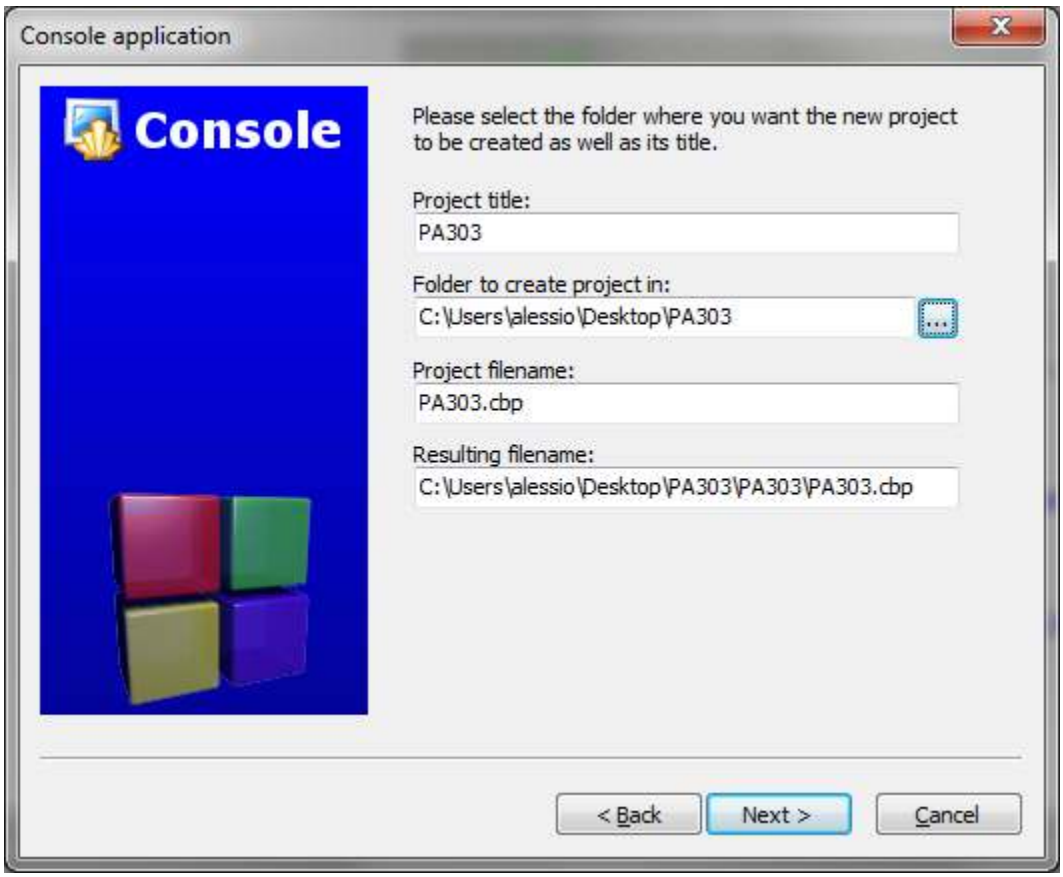


Go for Next and Skip this page next time if you want to.

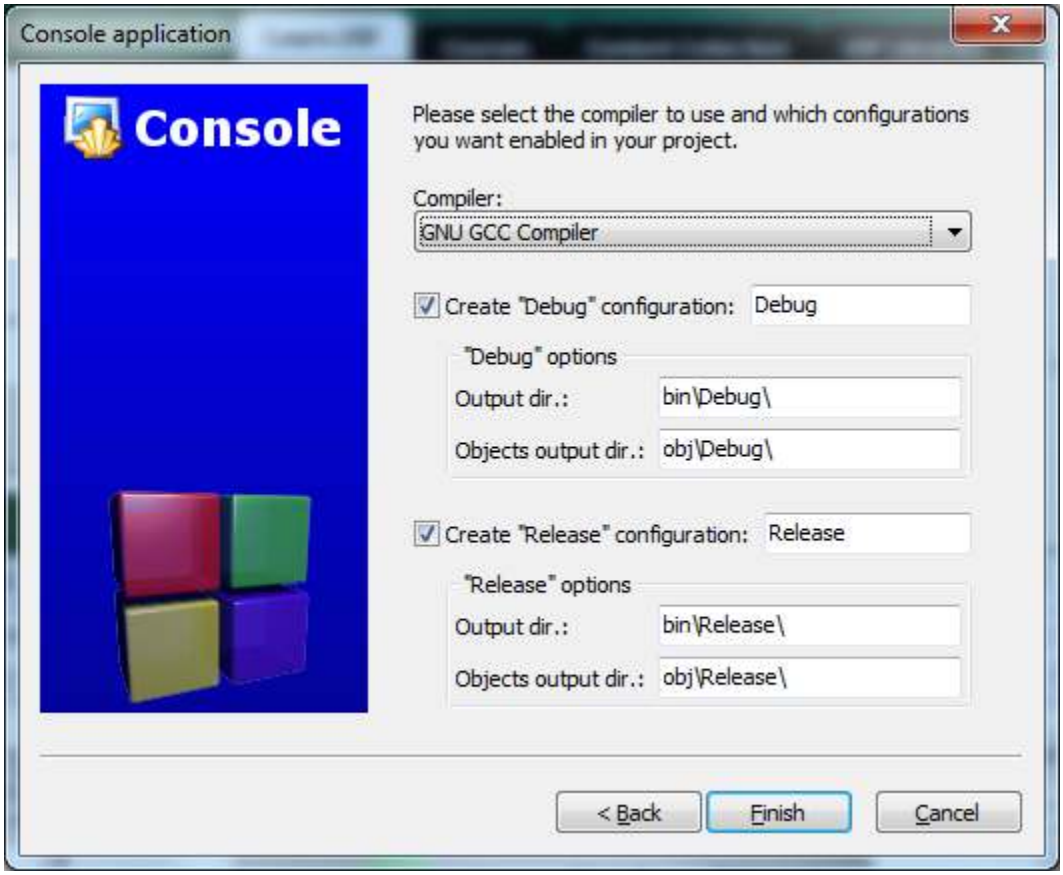


We are going to write C programs in this offering so the selection is fairly obvious.

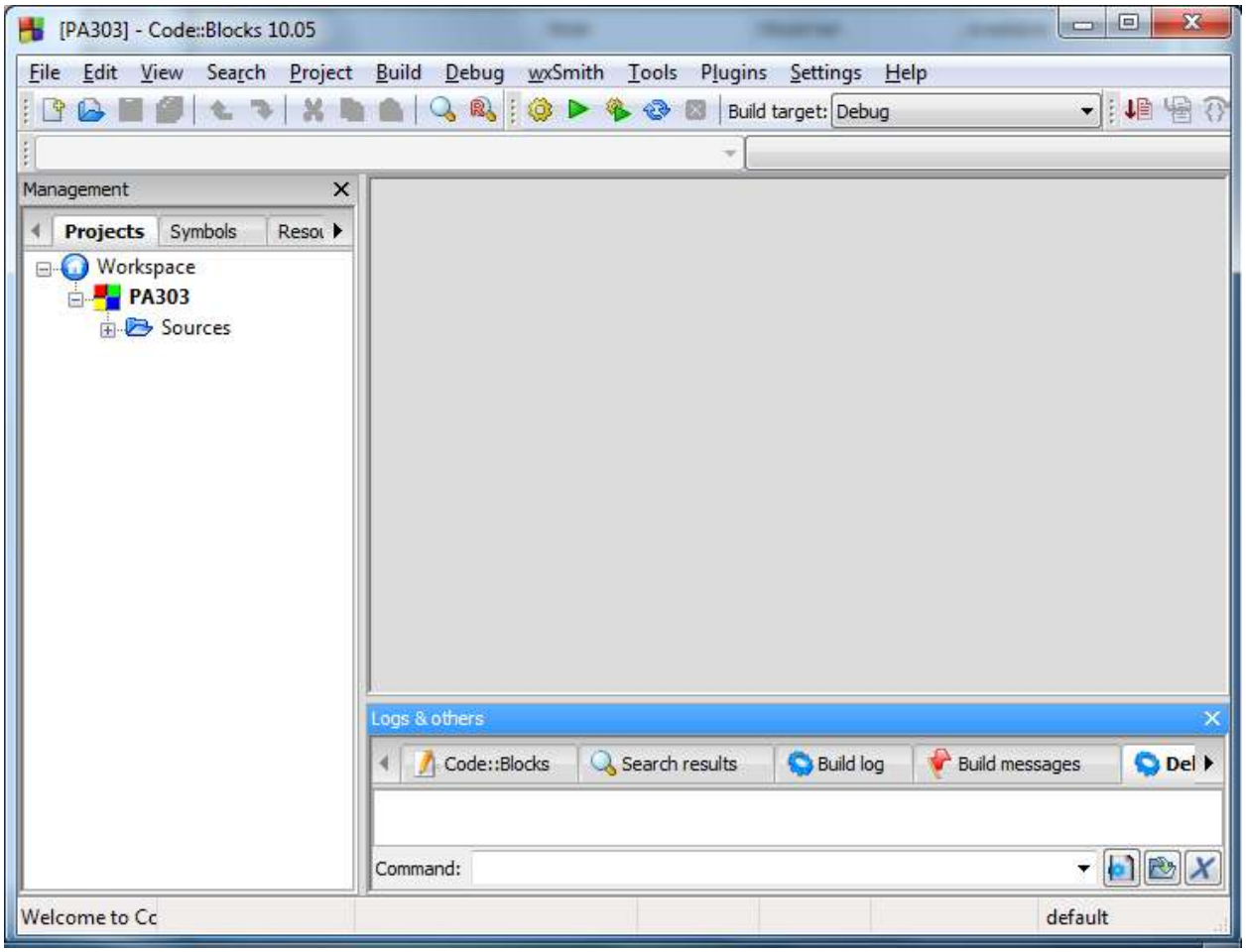
In the next panel, you will have to provide a projec tittle, up to you but I recommend you use the name of the assignment e.g. **PA203**. In this example, I'm selecting to have the project folder on my desktop.



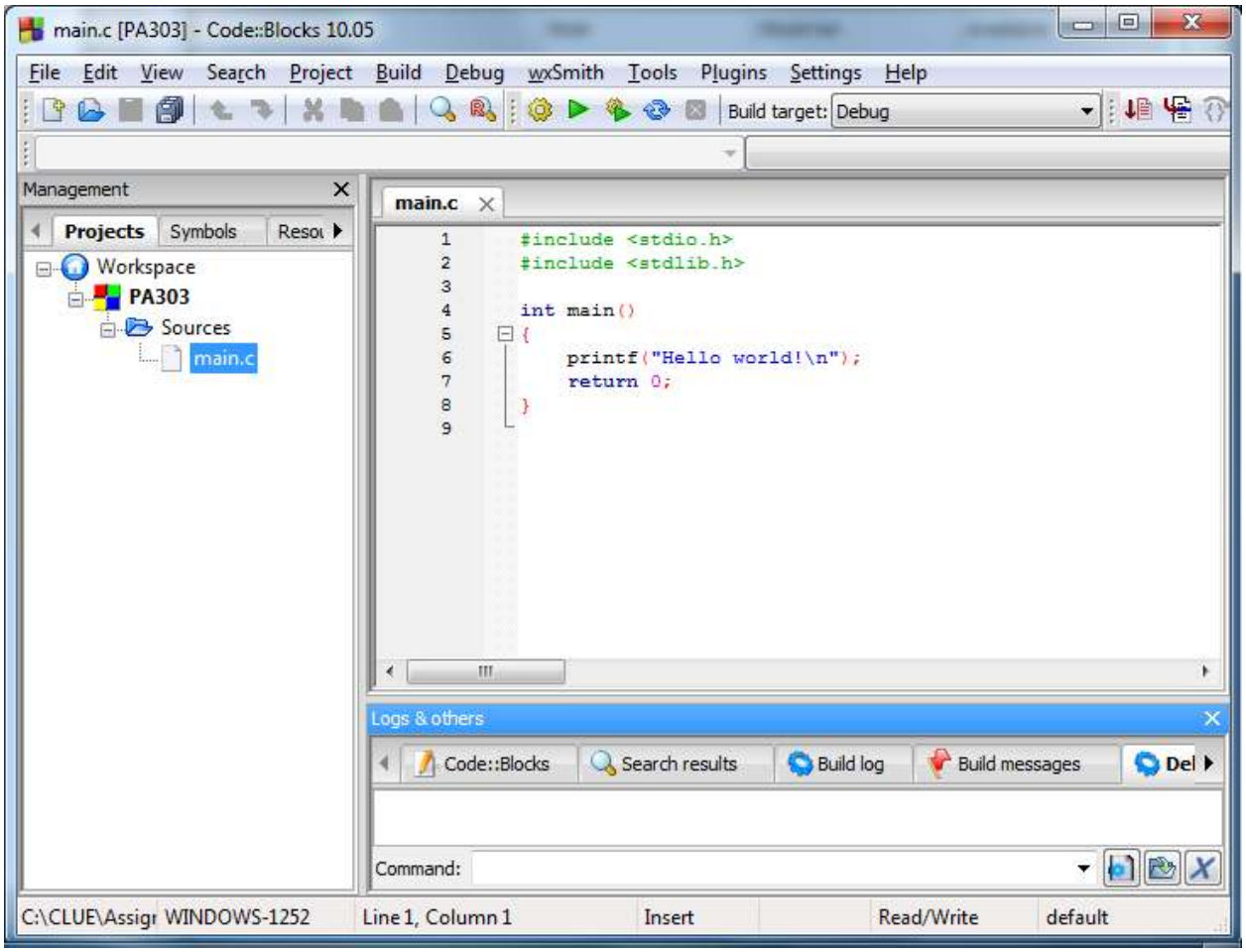
By default, Code::Block will be ready to compile two versions of your project. The **Debug** version is the one you'll work with, troubleshoot and then submit. The **Release** one won't be something we'll even use in this offering.



When you click Finish, you'll be back to the main window with a brand new project opened for you.



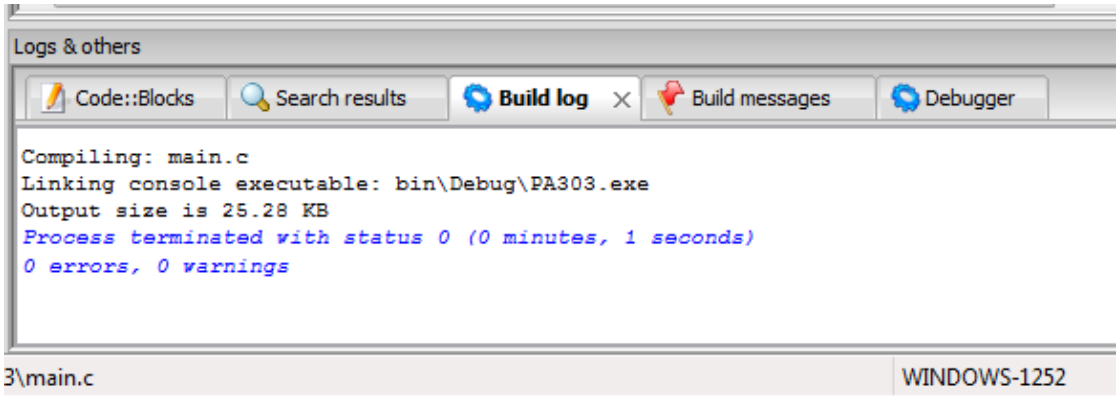
If you open the **Sources** tree on the left panel you will find a main.c already there for you.



S2.2 – Compiling and Executing a project

In order to run your program, you must build it. This entails compiling each of the .c source files which are part of your project, linking them together along with any needed library with a tool named the linker. Refer to the first module's videos lectures for more information about this whole process.

You will achieve this by using the Build→ Build menu or the Ctrl-F9 shortcut. The **Build log** tab in the bottom panel will show you any potential errors.



When your project is built, you may run it with "Build" → "Run or Ctrl-F10 shortcut. The interaction with your program will take place in DOS text Console window which will prompt you to press any key before to close when your program is done executing.

S3 – Debugging Programs

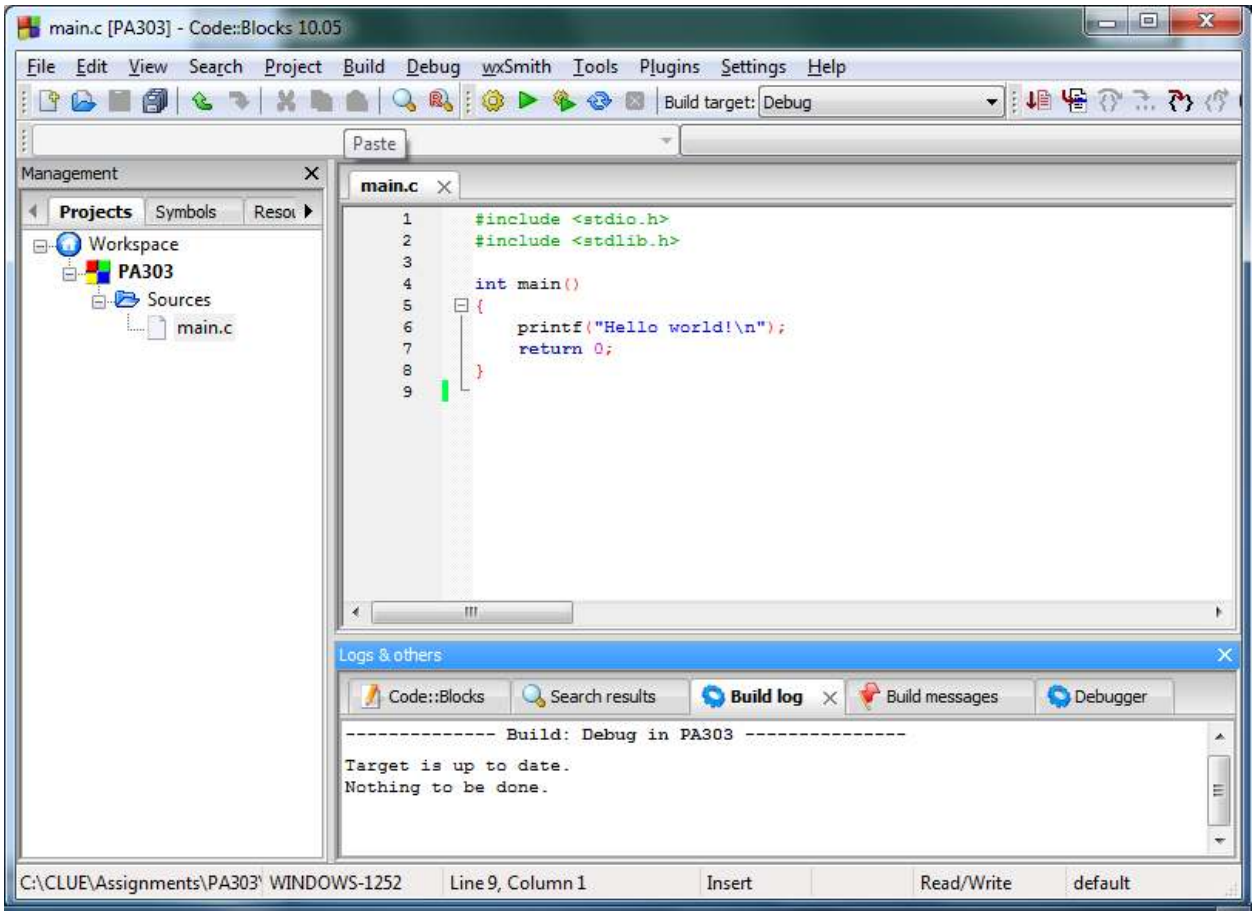
This section supplements the previous ones by showing you how to use Code::Block debugger to troubleshoot your programs.

S3.1 – Why using a Debugger at all?

One of the real benefits of using Code::Block is that you are able to execute step by step a program and display the value of key variables at each step. However, keep in mind you would be able to do the same thing by adding a few instructions to display the values of these same variables as your program execute. So, again, make sure you do not spend hours learning to use the debugger when you should be instead learning to program.

S3.2 – Preparing your program to be ran with the debugger

For the sake of demonstrating how the debugger works, we are going to use a simple "Hello World" program.

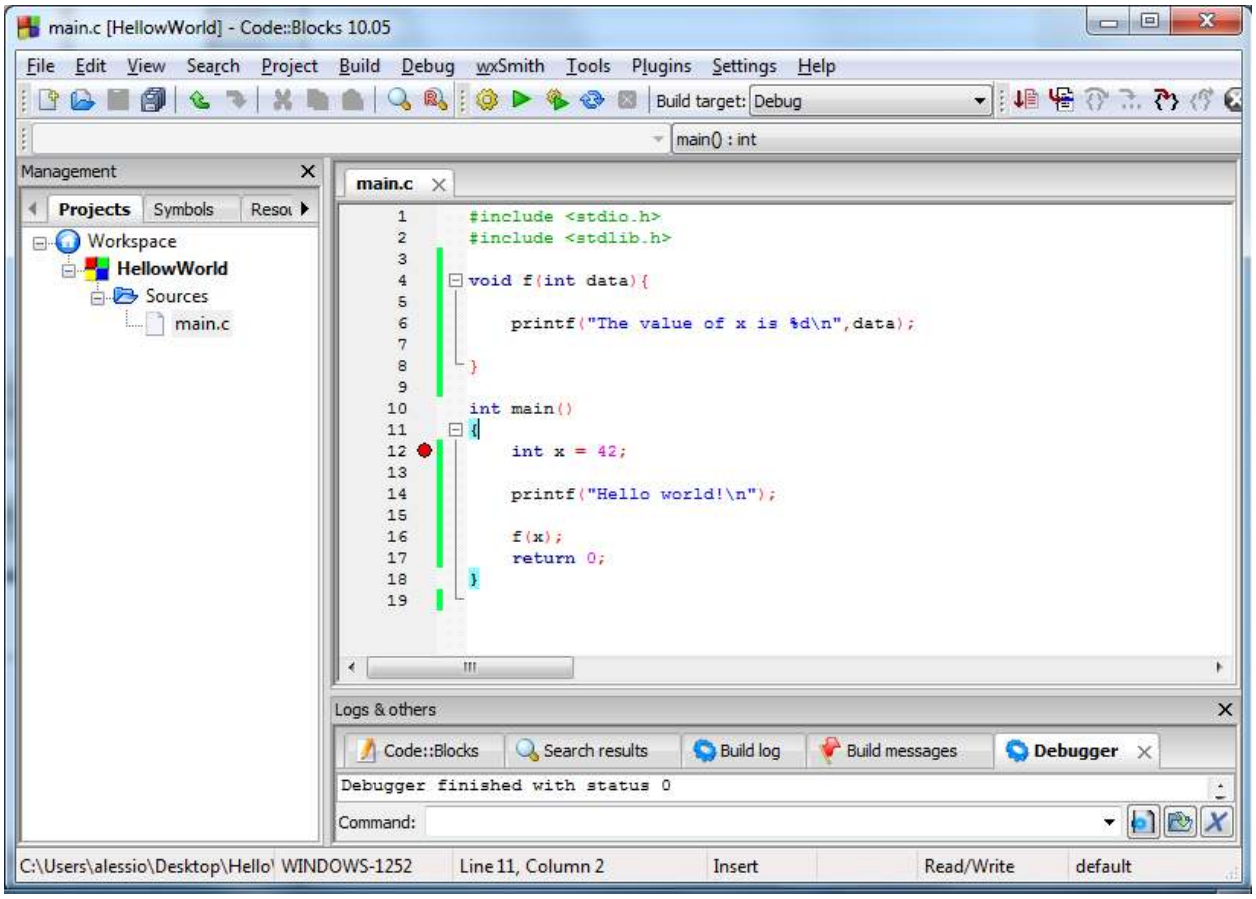


As you may notice in the above screenshot, the project is already built and ready to execute. You may not debug any project which is not already building properly an executing. If you project produces bad results or "crashes" while running, it's ok. This is why we are going to troubleshoot it. But a debugger won't be able to help with syntax errors. So make sure that you have fixed these before to debug.

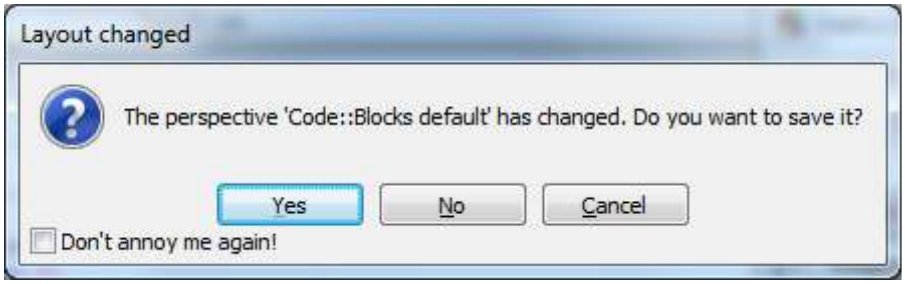
S3.3 – Setting up "BreakPoints"

When you use a debugger, you need to have an idea of where things are going wrong in your program. The best way to do this, is to put a breakpoint so that, when you run your program with the debugger, execution will pause at that line in the code.

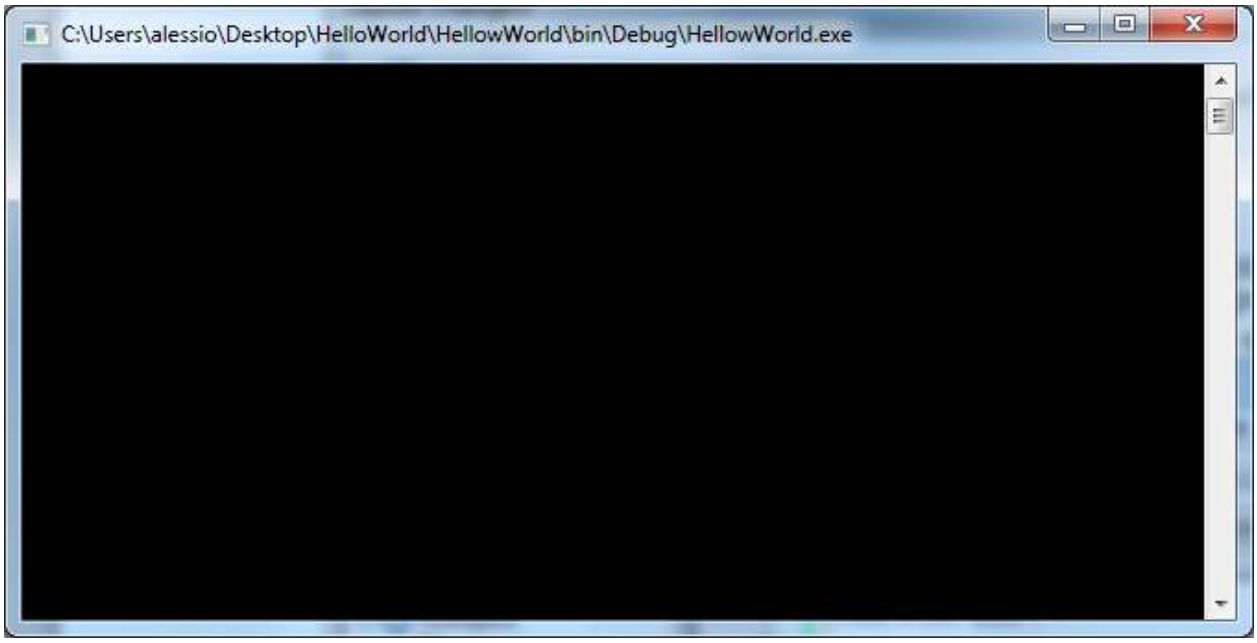
Select a line of code and use Debug → Toggle Breakpoint to set or unset a breakpoint at this line. You may also use the F5 shortcut instead. Notice the red dot on the break point line below;



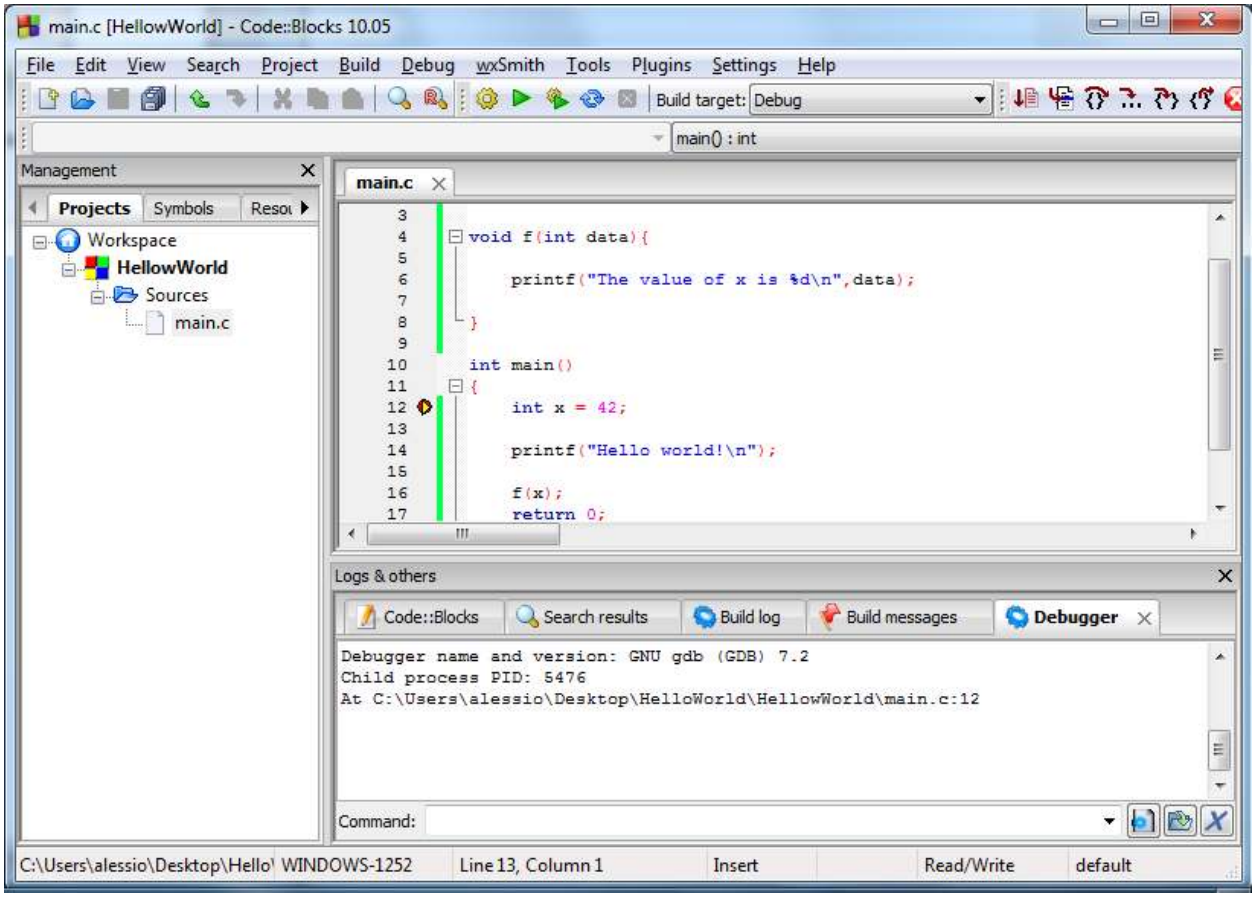
Now, we may run our program with the debugger by selecting Debug → Start. A popup will inform you that the IDE will now display your project in Debug Mode.



The, the program will start executing until it reaches your breakpoint. The display Console will be blank since we inserted a breakpoint before any of our printf.



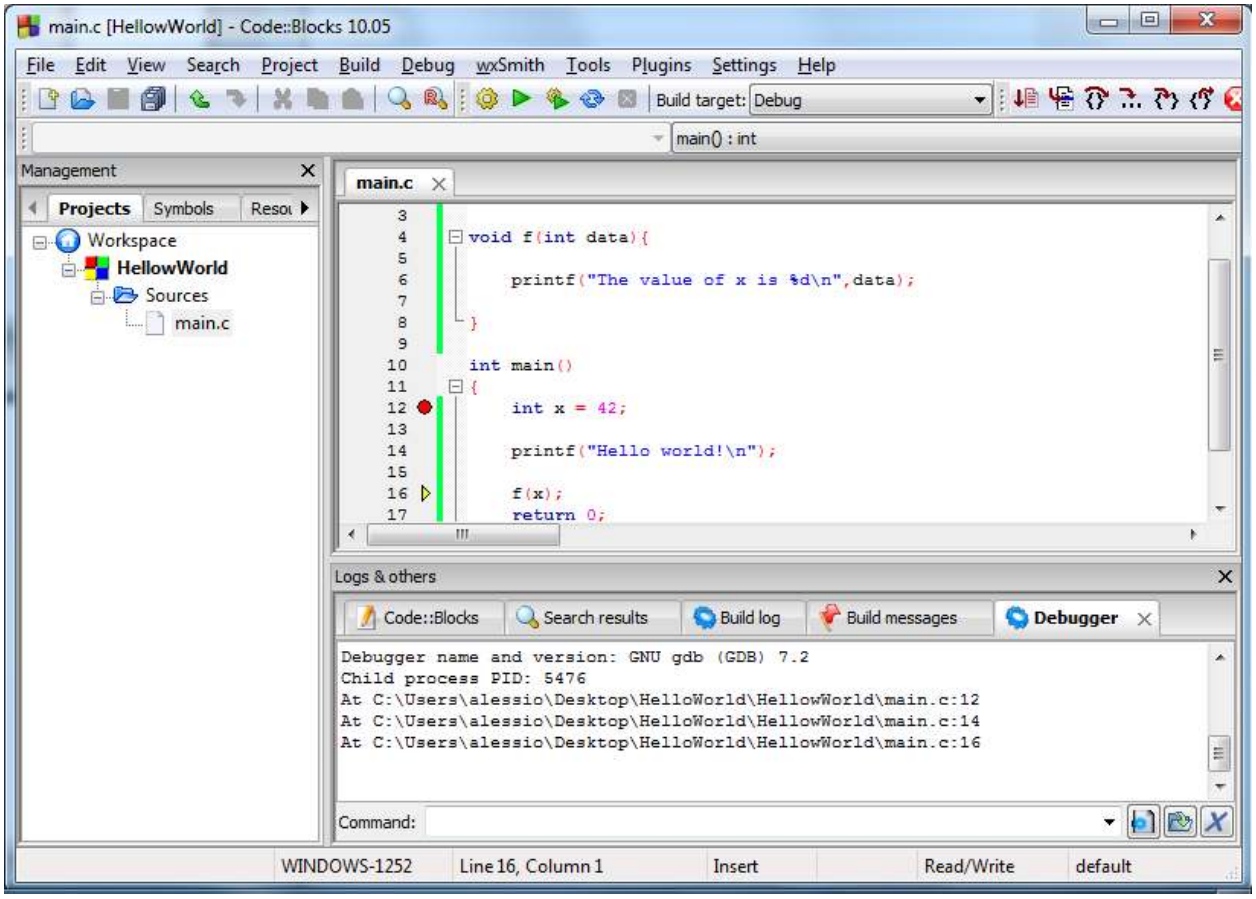
Meanwhile, the IDE window will mark that we are at the break point with a right pointing triangle.



S3.4 – Stepping through your programs

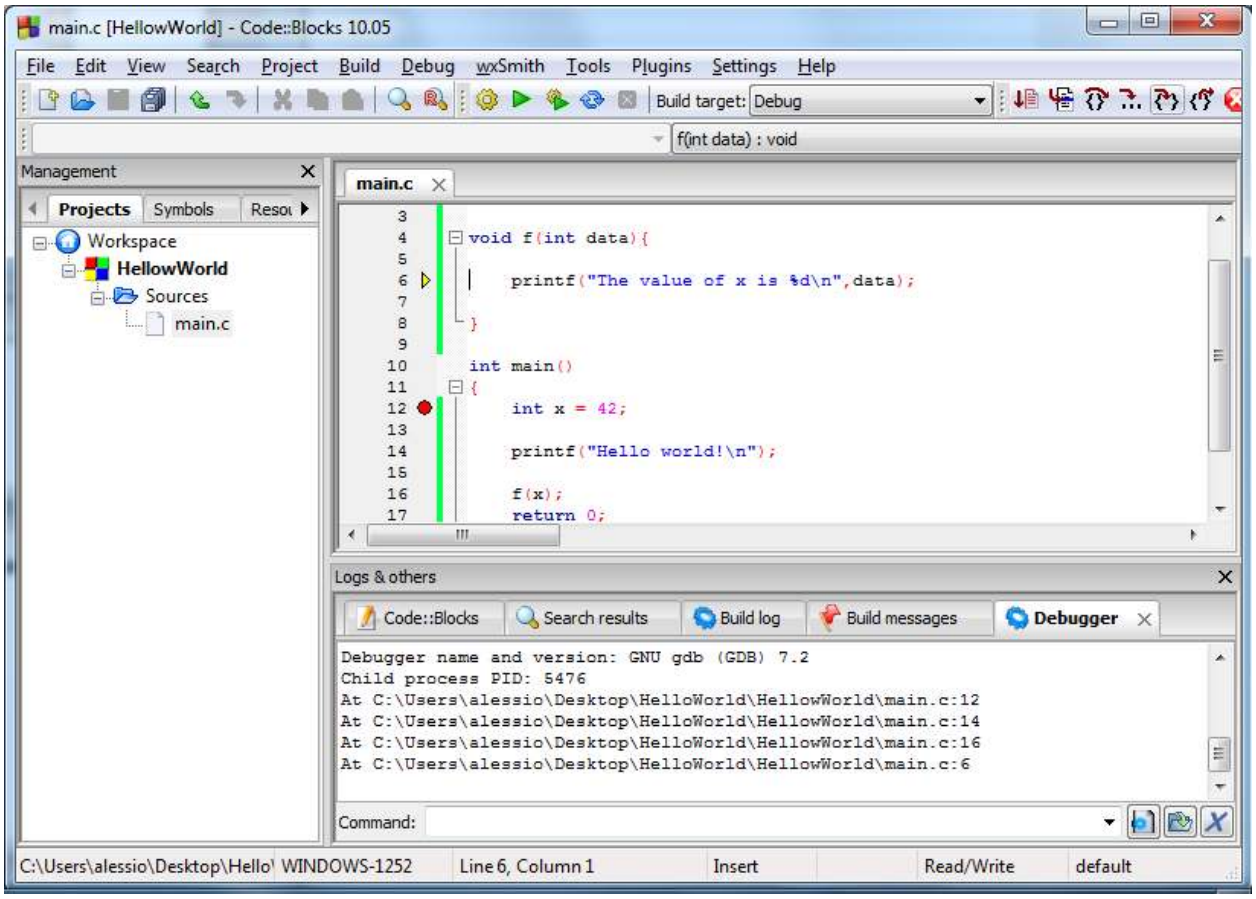
Right now, we might just select Debug → Continue and the debugger would just run until the program is done. Generally though, we don't want to do this. We want to "step" line per line throught the program. There are two ways of stepping

Next Line or Next Instruction will do pretty much what it sounds it'd do. Here is the result after using it twice;

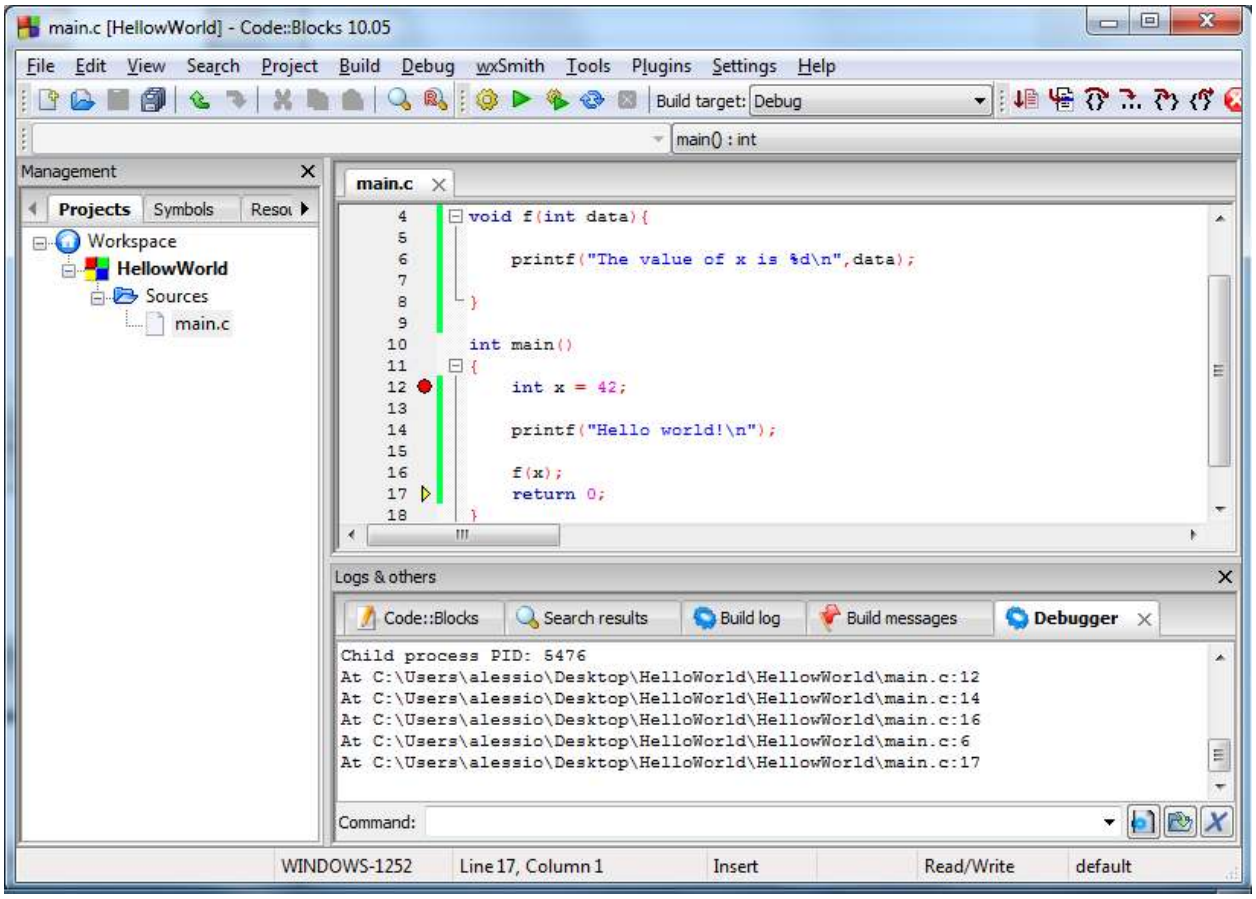


Now that we are on a function call, things are different. If we use the same way of stepping, the whole function call will be executed and we'll find ourselves on line #17 without having a possibility to step through the code of the function.

This is when you might want to use Step into instead which will allow you to get into the function's code and step through it.

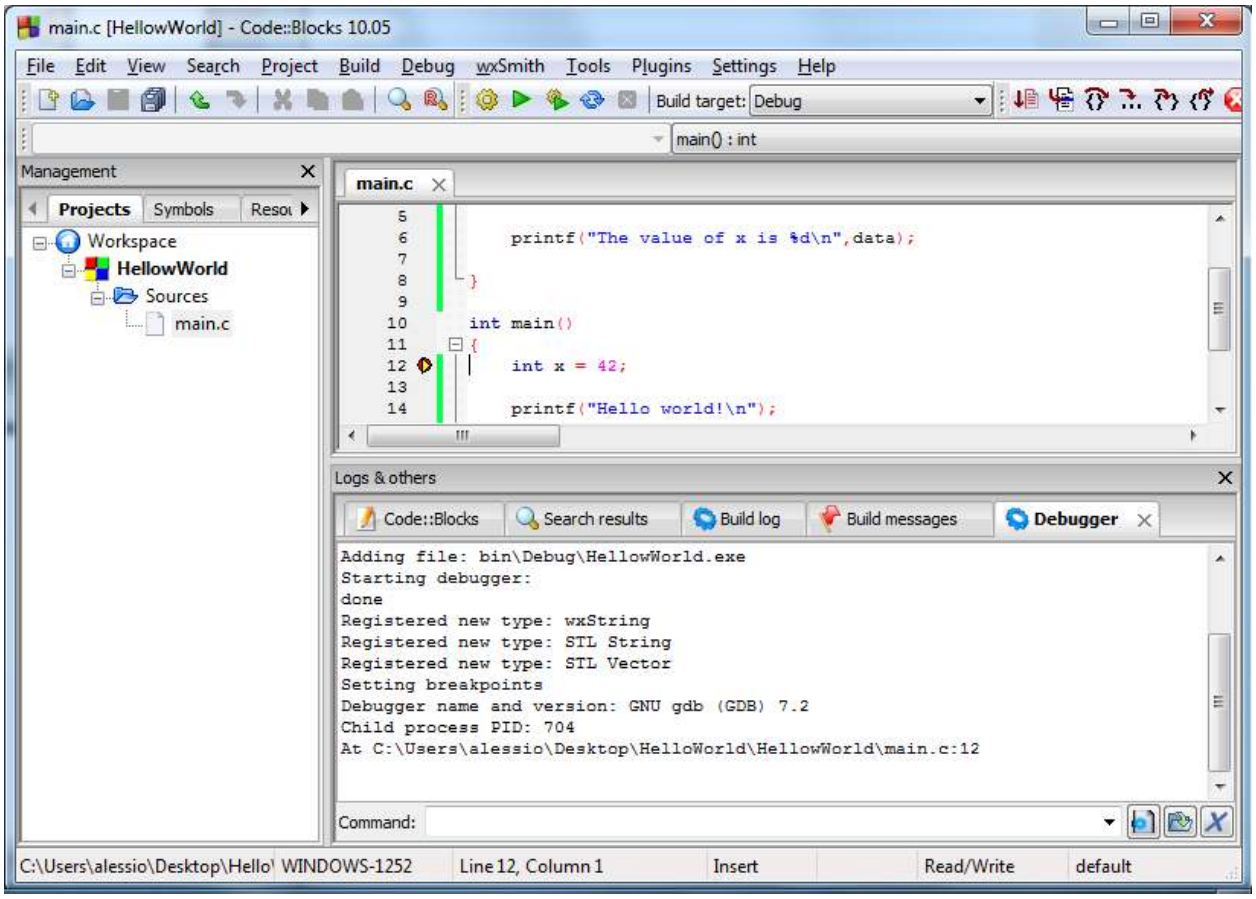


If you want to step through all its code, go for it. If at some point you just want to run until the function returns, then use the Step Out option.



S3.5 – Inspecting Variables Values on the go

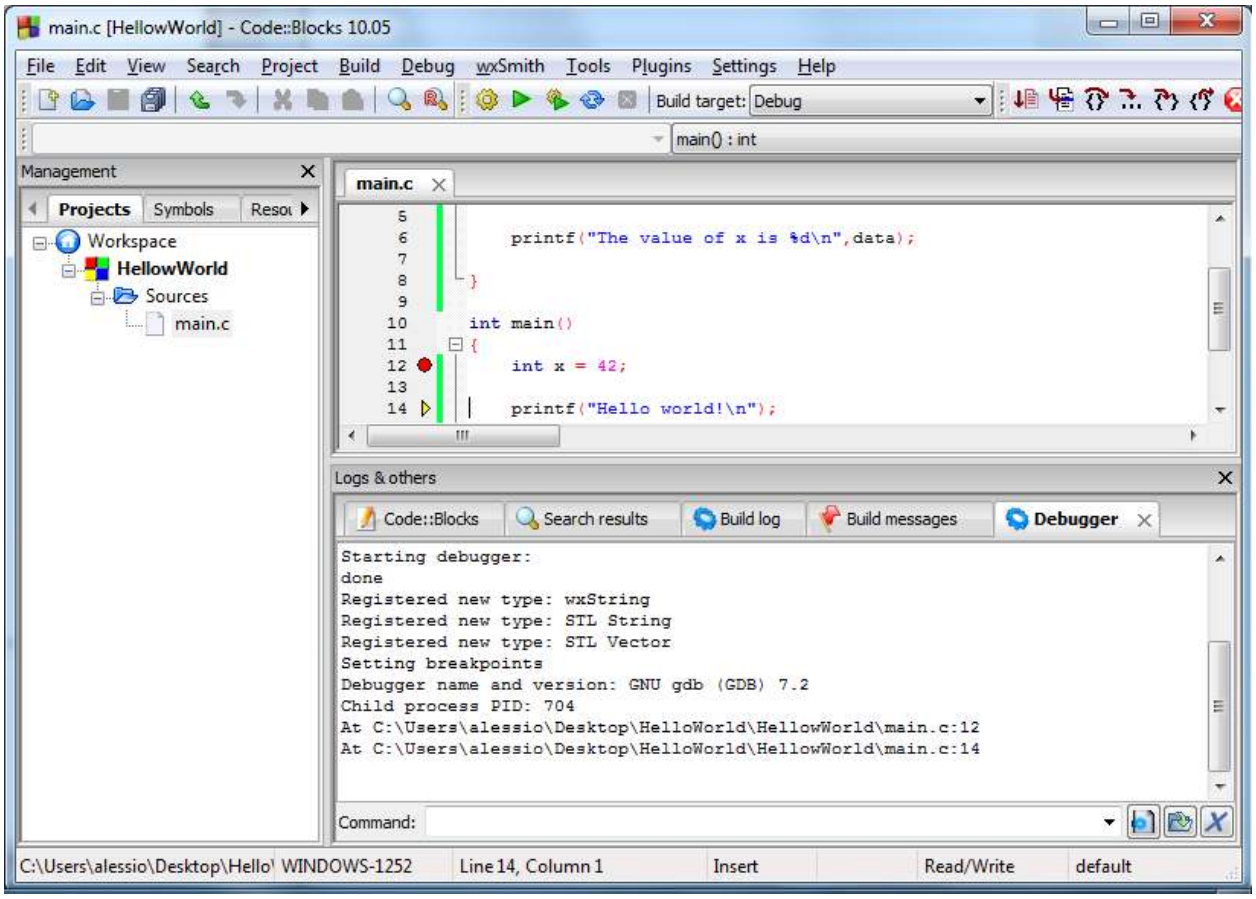
The other main feature of a debugger is to allow you to keep an eye on a list of variables as you step through the program. Let's finish the previous run and start a new one with Debug → Start.



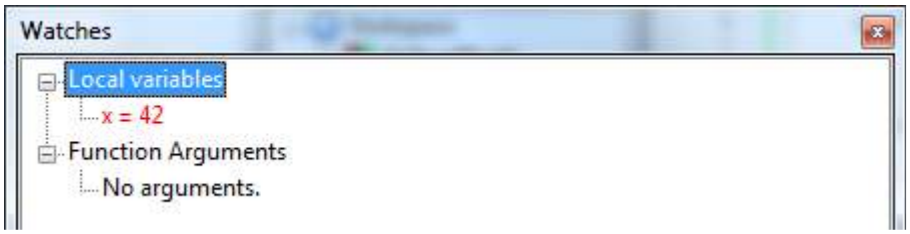
We are going to open the window holding all our variables monitoring by selecting Debug → Debugging Windows → Watches.

This panel shows all the local variables, here x only, and functions arguments if we are in a function accepting parameters, here none. We haven't stepped through the line of our program initializing x so it is right now holding a random value.

Note for version 12.11 or later; please note that the Watches window has been updated starting in version 12.11. The same information is displayed but the layout is a bit different. Do not let that confuse you.



Now we've stepped through the initialization, look at your watch window now.



The red indicates what was modified since the last line which was executed. In addition to watching the variables the debugger finds for you, you may also add your own watches by selecting Debug → Edit Watches.

Note for version 12.11 or later; You may now just type the name of the variable you want to watch on the last line of the watches window.

The variables we specify by hand are listed at the bottom of our watch window