Programming Fundamentals  
Tutorial 01 - Introduction to Code::Blocks IDE, debugging, and basic syntax with C/C++.

## Introduction

The purpose of the first part of this document is to familiarise yourself with the basic aspects of **Code::Blocks** projects, and how to create them in the correct format for the programming tutorials in this unit. Also you will learn how to create and run applications using the Code::Blocks **Integrated Development Environment (IDE)**, with an introduction to some basic syntax of the **C/C++ programming language**.

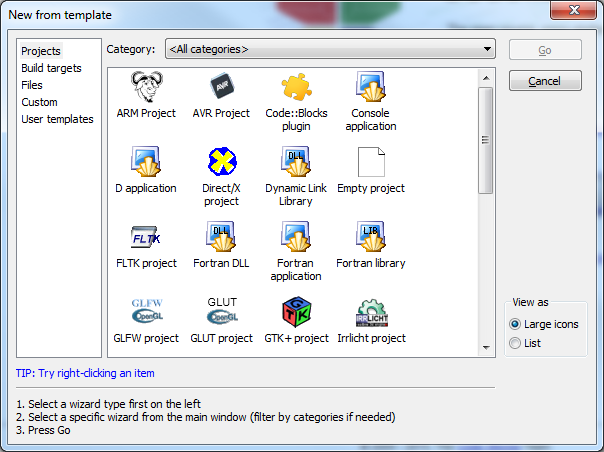
It's crucial that you follow the steps very carefully, if you miss something it can be very hard to find where you went wrong and you may have to start from scratch.

## Creating a New Code::Blocks project

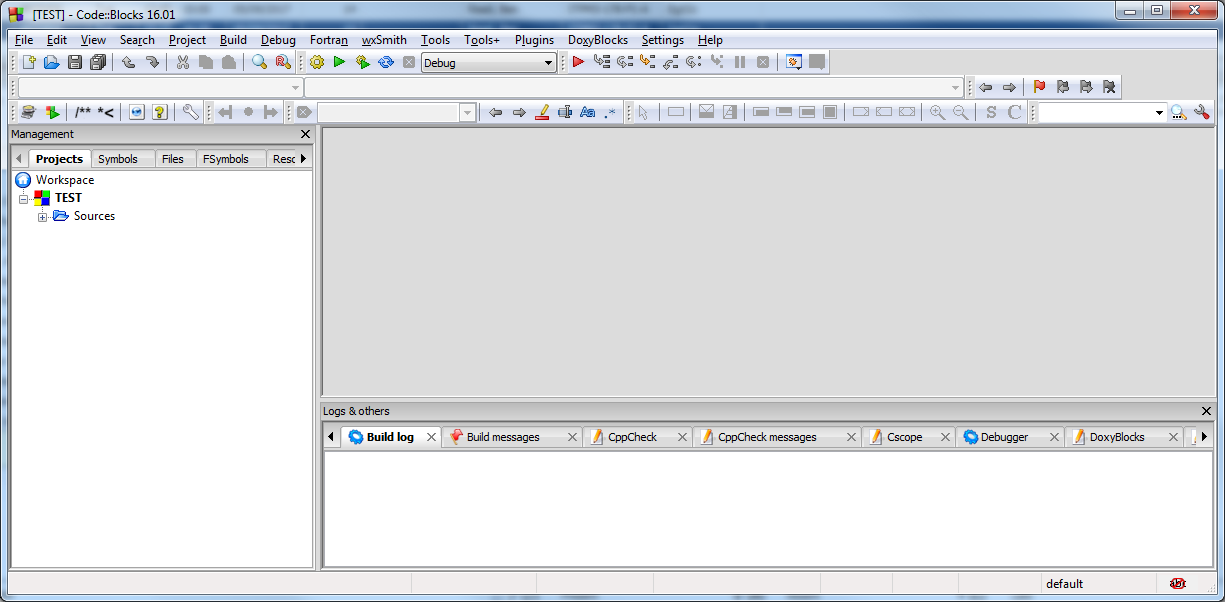
This section will explain how to create projects that you can use for all the coding exercises in this unit. The tutorial exercise below will take you through the process, you should follow these steps for all future coding project exercises.

#### Exercise 01 - Creating a project and adding source files

1. Open **Code::Blocks** and select **File > New > Project** from the menu bar. The **New from template** dialog will appear:



1. Click on **Console Application** and click the **Go** button
2. Click **Next** and then choose **C** from the language selector
3. Choose a relevant **Project Title** and **Filename** and ensure you’ve entered a valid project path. If the path does not exist it will create it for you. I suggest you use **H:\CodeBlocks\Projects\PP\_Exercise\_1**
4. Press **OK** and your project will be created. You should end up with something similar to the following screenshot:



1. In order to write code for your project you will need to add a **source file** or edit the default file, main.c. A source file is a text file that contains a program's instructions. To add a file to the project, select **File > New > Empty File (Ctrl + Shift + N)**
2. The ***filename*.c** file will be created and added to the list of **Sources** in your project.
3. Remove the default **main.c** file by right-clicking on it and choosing **Remove File from Project.**
4. Save your project for now, by selecting **File > Save All Files (Ctrl + Shift + S).**

### Adding Source Code

You should now have a project with a single source file but it so far has no code in it. In this part of the exercise you will add some program source code to this file.

To begin, **copy and paste** the following code into your source file:

#include <stdio.h>

int main()

{

printf("Hello World");

printf("\n \nPress Enter to continue...");

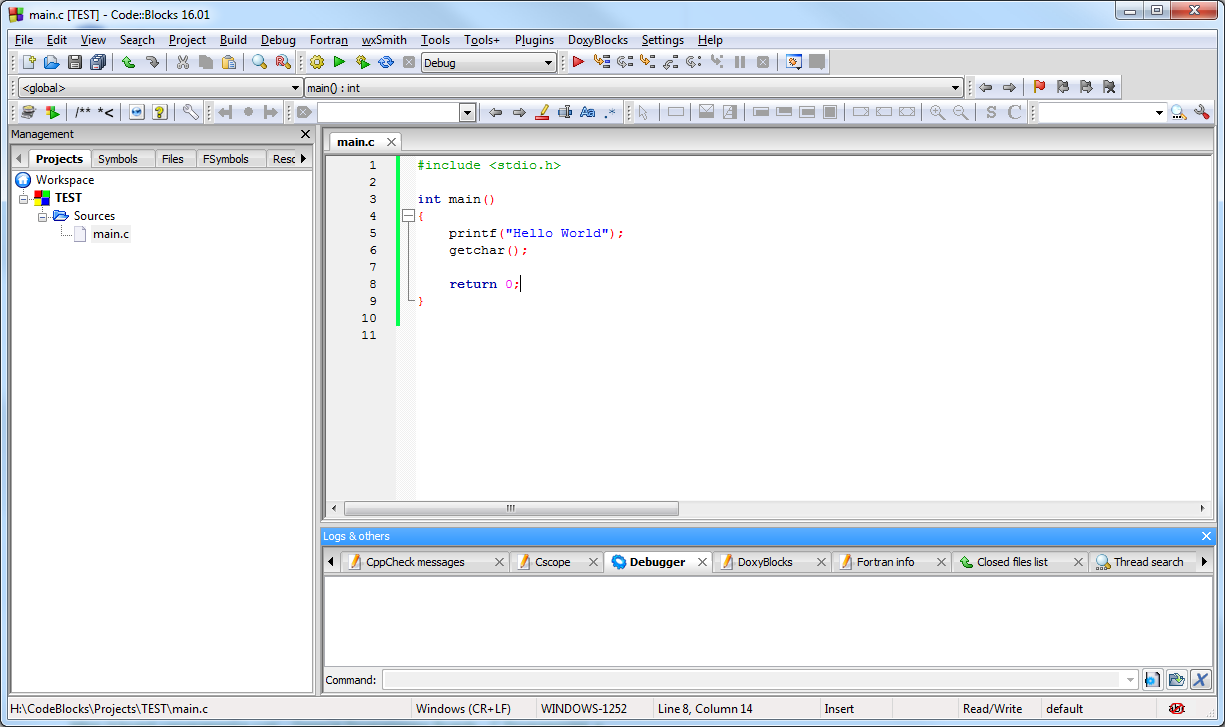
getchar();

return 0;

}

At the moment don't concern yourself with what any of this means, just make sure that it is all correctly copied into your source file.

The IDE should now look like this:

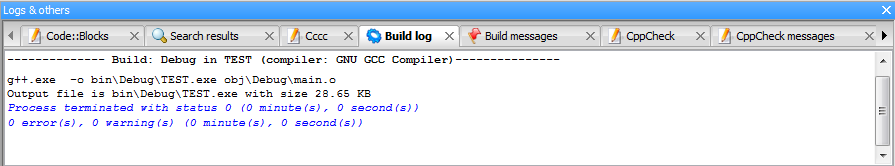


Code::Blocks automatically adds the colours to code, these have specific meanings that will become clear later in the unit, again, don't worry about what it means for now.

### Compiling, Linking and Running a Program

There are two more processes in C that have to occur before this source code actually turns into a program executable. These are called **compilation** and **linking**. Compilation is the process of transforming a source file into machine executable instructions. Linking is essentially the process of combining the compiled executable instructions from several separate source files together in order to create the final program executable. These two stages are almost always done at the same time, and the combination is called **building**.

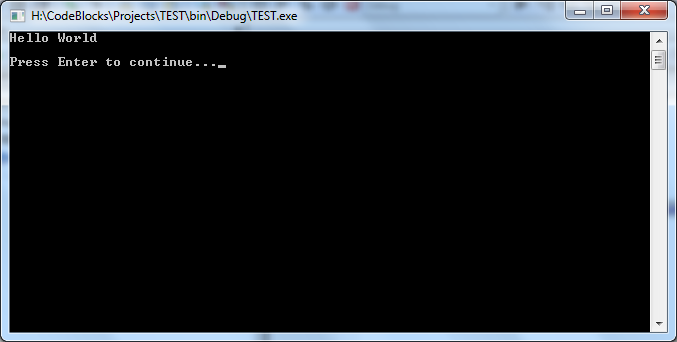
The whole build process is done for you by simply pressing **Ctrl + F9** or by selecting **Build > Build** from the menus, orpressing the cog button on the IDE tool bar. Build your program and you should see a lot of text being written to the **Build log** area at the bottom of the screen. If you can't see the Logs window, select **View > Logs**.It should end up looking something like this:



**Status 0** means that the program has been successfully built and can be executed. It is at this point that any compilation or linking errors will appear, which you would have to fix before the program will work. If you don't end up with a successful build, check that you have followed all the preceding steps carefully.

**NOTE -** You will cover how to correct compiler and linker errors you encounter in later tutorials.

Once built, you can run the code by pressing **Ctrl + F10,** or by clicking on the green **Play** button on the toolbar. You should get a text window that appears like this:



If you haven't programmed in C/C++ before, congratulations on completing your first program!

Press the **Enter** key with the program window in focus and the program will end, just as it has been programmed to do. You can also stop **any** program you started using the IDE by closing the console window.

### The Executable File

Once you have finished developing a program you can find the final executable file in the **…bin\Debug** folder of the project. Double click on this as you would any program and it will run.

**NOTE** - The folder for the executable location can be changed, but I recommend you leave it as the default.

**NOTE** - **Release (optimised)** **builds** go in the **Release** folder by default.

### Saving and Closing a Project

You can save your project at any time, by selecting **File > Save Everything (Alt + Shift + S)**. It's worth saving often to avoid losing work if anything goes wrong, such as a power failure.

You close the IDE as you would any normal Windows application, by clicking on the close window button. Don't worry if you didn't save your work, if not then the IDE will ask you if you want to save before quitting.

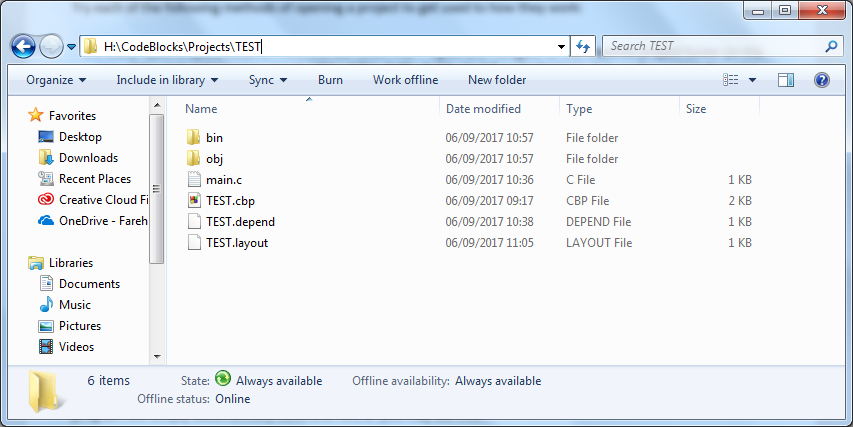
Normally you'll just click **Yes** and all your work will be saved. If you are unsure, you can click **Cancel** to go back to the IDE and check.

Save and close your project now before going on to the next section.

### Reopening a Project

Try each of the following methods of opening a project to get used to how they work:

1. If the project is open, close it. From the IDE select **File > Open** and navigate to the required project folder (in the case of this tutorial exercise case it will be **H:\Code::Blocks\Projects\*projectname****),* click on the project filename ending in **.cbp** and press **Open.**
2. Close the project again. If a project was closed quite recently it will be in the **Recent Projects** list, and since you've just closed your project it should definitely be there. To open from the recently used project, go to **File > Recent Projects** and select the project.
3. Close the project. You can also open the project from File Explorer. Navigate to the project folder, and double click on the **.cbp** file



Your Explorer list may look different depending on your settings. You may not be able to see the file extension, if not the icon for the **.cbp** file is shown here >

### Automatically Compiling and Running.

Instead of pressing **Ctrl + F9** to build and then **Ctrl + F10** to run, you can actually just press **F9** to build and run the program at the same time.

Change the **"Hello World"** text to **"Hello [YOUR NAME]"**, and press **F9**. Your program should automatically be built and executed with a window showing the new text, assuming there are no errors.

**HINT** - you can copy an **entire project folder** anywhere (e.g. to memory stick), and if you double click on the **.cbp** file it will open normally. This is useful if you want to take the project to work away from college.

**HINT** - If you are pushed for space, for instance you want to zip up and email the entire project, use **Build > Clean,** save and close the project, and delete the **obj** and **bin** folders.This will delete all the temporary files that the project uses, which results in a much smaller folder size. Be aware that this deletes the **.exe** file also, so you will need to rebuild the project in order to run it again.

### Introduction to Basic C/C++ Syntax

Now that you are able to create projects and build executable programs the remainder of the tutorial will introduce and explain some of the basic C/C++ **structure, operations** and **syntax** that are required in order to correctly write a program. To help you examine these ideas and understand how programs work, the basic functions of the Code::Blocks **debugger** will be shown to allow you to step through programs one operation at a time.

#### Exercise 2a - Creating a simple example

One of the easiest ways to understand how programs work is to take examples and step through them operation by operation in order to see what is happening. In this exercise you will create a very simple program, and then step through what it is doing using the **debugger**. Don't worry if you don't know what the program code means just yet, or what the debugger is and what it can do, you will discover this using detailed step by step instructions during the exercise.

1. Create a new project as detailed in the last tutorial called **PP\_Exercise\_2**.
2. **main.c** should have been created for you. If not, click **File > New > Empty File** and add it to the project.
3. Copy and paste the following code into **main.c**:

/\*Ben Read

Tutorial 01 Exercise 02

Main.c

01/09/2017

\*/

#include <stdio.h>

// All programs start in main()

int main()

{

// DECLARE variables

int number1; // Declare number1 as an integer variable

int number2;

int result;

// DEFINE variables

number1 = 10; // Assign variable called number1 to the integer value 10

number2 = 7;

// DEFINE result as ADDITION of number1 and number 2

result = number1 + number2;

//PAUSES program and waits for user input

printf("Press Enter to continue");

getchar();

// EXIT program

return 0;

}

1. Save your project

Before you run the code for the first time, there are a few basic concepts that appear in this code that are worth discussing.

### Comments

The first part of the program has the following lines:

/\* Ben Read

Tutorial 01 Exercise 02

Main.c

01/09/2017

\*/

Any text that is between a pair of /\* and \*/ (even over several lines of text) is **completely ignored by the complier**. This is called a **comment** and comments are used to explain and describe what code is doing in plain English to make it easier to understand. In the above case it is used to describe who created the file, what project it belongs to, what the filename is, and what date it was created - you should get into the habit of doing this for all of your source files. You should also always use comments extensively to document your code; it is usually impossible to remember what a section of code does after a few weeks. Additionally, comments help inform others of the intent of the code. This is especially crucial in situations where you are working with others, or where your work will be assessed or marked.

Another form of comment is as follows:

// All programs start in main()

Anything after the two forward slashes, //, is also ignored by the compiler until the **end of the line** that it is on. You can continue to comment on the next line by ending the current line with a backslash, **\**, however, it is often neater to use **//** on every line. Also notice that in the IDE comment text is turned **grey**. This is to give you a visual indication that this text is a comment.

### Variable Declarations

A **variable** is an area of computer memory that is used to store a value. In C/C++ there are various different basic **types** of variable that are used to store different sorts of values, but for this simple example you will just be using **integer** variables. Integers are whole number values that can be positive, negative or zero.

In order to use a variable in a program you need to **declare** that variable. A variable is declared using the following syntax:

**type\_name** ***variable\_name*;**

and the type\_name for an integer is int so for example the three integers declared in the example are:

// DECLARE variables

int number1; // Declare number1 as an integer variable

int number2;

int result;

which reserves memory for three integer variables, number1, number2 and result. Variable names in C/C++ **must be unique** and **must begin** **with an alphabetic character or an underscore**. There are **key words** that C/C++ uses itself that cannot be used as variable names, for instance you can't use int or main as a variable name. Variable names are case-sensitive so you could create three variables called number1, e.g. number1, Number1, NUMBER1, and they would all be considered individually by the compiler.

The variable name is the word used to **refer to the value the variable contains**. At the point of declaration variables have not been given a value, so their **value is undefined**. Variables should be **assigned** **a value** before use, this will be discussed next.

You might notice at the end of each variable declaration there is a **semi-colon** - **;**. The semi-colon in C/C++ is used to separate all statements and declarations. It's important to be very precise in C/C++, so take the time to check that you have correctly inserted semi-colons as this is a common cause of errors, especially for beginners.

### Statements

The next concept to understand is the **statement**. A statement is simply an operation in source code that performs some action. These are otherwise known as procedures. Below are the statements in the example that define the variables, these are all called **assignment statements**:

// DEFINE variables

number1 = 10; // Assign variable called number1 to the integer value 10

number2 = 7;

// DEFINE result as ADDITION of number1 and number 2

result = number1 + number2;

The first two statements simply assign specific integer values to the two variables. The third statement uses an **arithmetic operator**, in this case **addition** (**+**) to add the values of number1 and number2 together, then the result of this addition is assigned to the result variable. Again, note the use of **semi-colons at the end of each statement**.

Another common source of errors to those new to C/C++ is when using arithmetic operators and assignment. When learning arithmetic, the questions are often in the form:

103 + 27 = ??

However, if you tried to use this format in your code:

103 + 27 = another\_result;

then the compiler **will not accept it**. The result of an arithmetic operation must **always be on the left hand side** of the equals (**=**) sign, like so:

another\_result = 103 + 27;

The final statement in the program is:

// EXIT program

return 0;

Again, don't concern yourself with what this does just yet. It will be discussed alongside functions later in the unit. All you need to know for now is that in this case it essentially ends the program.

### The **main()** Function

The following line is used in all standard C/C++ applications and it is essentially where every program begins:

int main()

This is called the main() **function**. Don't worry for now exactly what this means, you will go over it later in the unit. As mentioned previously, C/C++ is **case-sensitive** - it matters if you use upper case or not. So, for instance, if you used:

int MAIN()

or:

int Main()

or anything other than **'main'**, the program **will not compile**.

### Statement Blocks

All the operations that a program executes are called **statements** and in C/C++ all statements must be inside **statement blocks:**

**{**

// program operations (statements) go between **braces**

**}**

A statement block consists of a pair of **braces** and inside the braces are program statements. There are several uses for statement blocks that you will be introduced to during the unit, but for now all you need to know is that a function **requires** a statement block to **contain the function's statements**. Therefore as main() is a function all the statements that belong to it need to be inside a pair of braces:

int main()

{

// program statements for main function

}

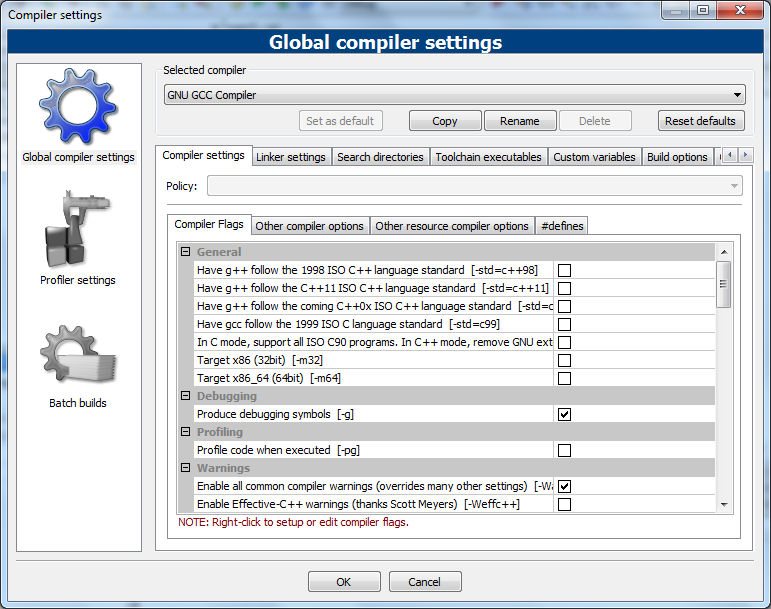
Ensure that you always correctly pair braces. Failure to do this is a common cause of program errors.

#### Exercise 2b - Stepping through the simple example using the Debugger

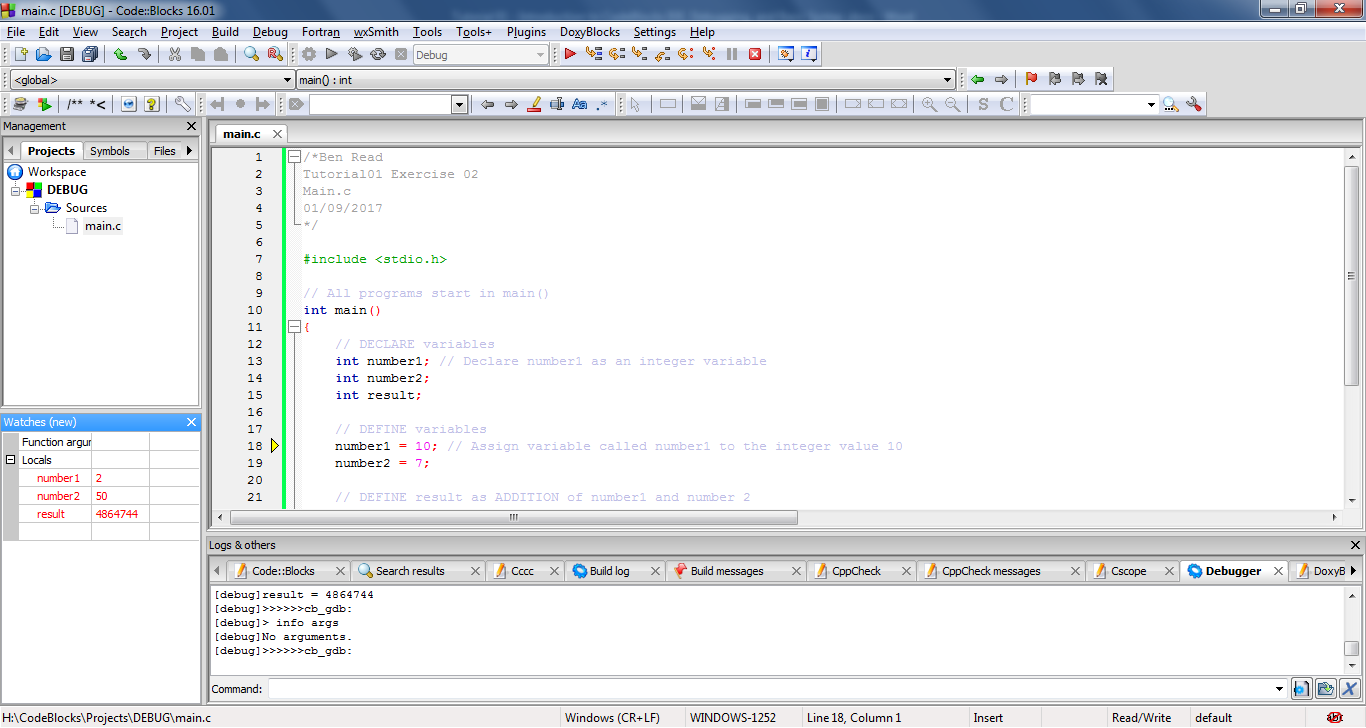
In the last section you created a new project with a simple example and the syntax of the example was discussed. In this part you will learn to use some basic functions of the Code::Blocks debugger in order to step through the program code. The debugger allows you to move through code and examine how it works in fine detail, including what order the operations are performed and the value of variables after each statement is executed. The debugger is extremely useful for understanding code and finding errors.

***Configuring the Debugger to watch variables***

1. Open your project using any of the methods discussed previously.
2. Click **Settings > Compiler**
3. Ensure **Produce debugging symbols** and **Enable all common compiler warnings** are ticked:



1. Open the Debugger watches window via **Debug > Debugging Windows > Watches**
2. Drag the **Watches** window into the main IDE area and attach it under the Projects window.
3. Click **View >Perspectives > Save Current**. This will ensure the **Watches** window remains where you put it next time you open the IDE.
4. Go to the IDE and press **Shift + F7**. The program should start, and the IDE should look something like the following:

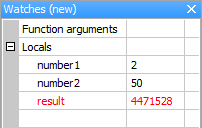


You should see a **small yellow arrow** in the left hand grey column of the source code window (next to line 18, above). This represents **'the next operation to be executed'** in the program. Since this is the start of the program, the first operation is to start at the statement block that contains all of the program statements for main(). The first operation in main() is to assign the integer variable **number1** with a value of **10**.

You may wonder why it missed out the variable declarations. This is because they are **not program statements** - they do not actually perform any kind of operation. The memory for the variable is reserved during compilation, and then statements use the variable name labels to access the variable data.

1. Press **F7** again. You should see the yellow arrow jump to the next line:

At this stage the **Watches** window will also have changed to something like:



For the basic code you will be developing during this unit this will be the most useful of the debug windows. Essentially, it will show you the names of all of the variables that are being used in main() and what their current values are. At the moment they are all random numbers which they picked up from the memory address they were allocated. This is because the assignment statements haven't been executed yet, and the values are **undefined**, as mentioned before.

1. **Press F7 several more times. Note down what happens in your notepad at each step, and explain what you think is happening. Take note of any colour changes, and what it might mean.**
2. If at any time you want to stop the debugger, press **Shift + F8** or on the Debugger menu bar of the IDE.

#### Exercise 2c - Breakpoints

One of the most useful features that a debugger provides are **breakpoints**. Breakpoints are user defined places in code that the debugger will stop at when execution reaches them. This can save a lot of effort instead of having to step through line by line. You can set and unset them by clicking on a line of source code and pressing **F5** or just by clicking on the grey bar to the left of the source code - a red dot will appear. This is a breakpoint.

1. Set a breakpoint on the following line:



1. What do you expect to happen if you run the program by pressing **F8**? Predict what values will be in the **Watches** window and where the yellow arrow will end up.
2. Press **F8** and compare what happens to your predictions.
3. You can combine both types of debugging, try pressing **F7** once the breakpoint has been reached to see what happens.

#### Exercise 03

1. Create a new project as detailed in the last tutorial called **PP\_Exercise\_3**.
2. Copy and paste the same code from exercise 02 into **main.c.**
3. Save and close the project.
4. Change the code to **multiply** the two numbers instead of adding them. You will need to investigate to find out what the multiplication operator is in C/C++
5. Use the debugger to step through the code to see what happens.

#### Exercise 04

Create a program with six integer variables initialised to different values, then calculate the average and put it into another integer variable. Describe how you decided on what new arithmetic operator is needed to calculate the average. Is the average always correct? If not, why?

#### Exercise 05 and up

Find some short and simple programs that you don't fully understand, create projects for them and step through them to improve your understanding of what they are doing. The more you practice the better.

## Conclusion

These are the basic syntax and debugging tools that you need to get by for very simple programs. Start using the breakpoints and stepping through code to more fully understand the code you are writing, and make sure you understand how to correctly use the debugger when it stops execution due to some sort of error. I fully expect you to be able to demonstrate and explain your code to me line by line using the debugger. Additionally, if you are having a problem and want my help, I expect you to be able to show that you have attempted to solve the problem by analysing it with the debugger.