
PRIFYSGOL
BANGOR
UNIVERSITY

# Imperative Programming in C

## Laboratory 1: Getting Started

*Dr. Llyr ap Cenydd*

# Introduction

Welcome to the first laboratory for *ICP1029 Imperative Programming in C*. In these weekly sessions you will be provided with a range of exercises that aim to teach you the concepts and techniques of programming in the C language.

The exercises will come in three types:

- ## Standard Exercises

  They are the foundation of the module and where the majority of learning will take place. There is no substitute to practice when learning to program. These standard exercises will build and compliment on each week's lecture topics and material. **You should aim to complete all these exercises.**

- ## Assessed Work

  At the end of some labs there will be a piece of assessed work. **Each piece of assessed work needs to be completed and submitted for marking.** You will be asked to submit the code for your assessed work to blackboard, and will be assessed in the lab similar to other programming courses in the department. Here you will be expected to show not only your program and code, but also display an understanding of what your code is doing and why.

- ## Challenges

  These exercises are ***optional*** and designed to provide extra challenges for those of you who want to further practice your programming skills. They are for students that have finished the standard and assessed work, and want to be challenged further.

# Setup

The main purpose of today's lab is to set you up ready for the coming Semester.

You are free to use any C/C++ development environment for this course, but we recommend that you use either **Dev-C++** or **Microsoft Visual Studio** as they are available on all University machines.

The aim of this course is to teach imperative programming in C rather than any specific integrated development environment (IDE). However feel free to install another IDE on your home computer or laptop, just don't expect any technical support!

The following lab script is written with Dev-C++ in mind. If you'd prefer to use Microsoft Visual Studio, refer to the guide document on blackboard.

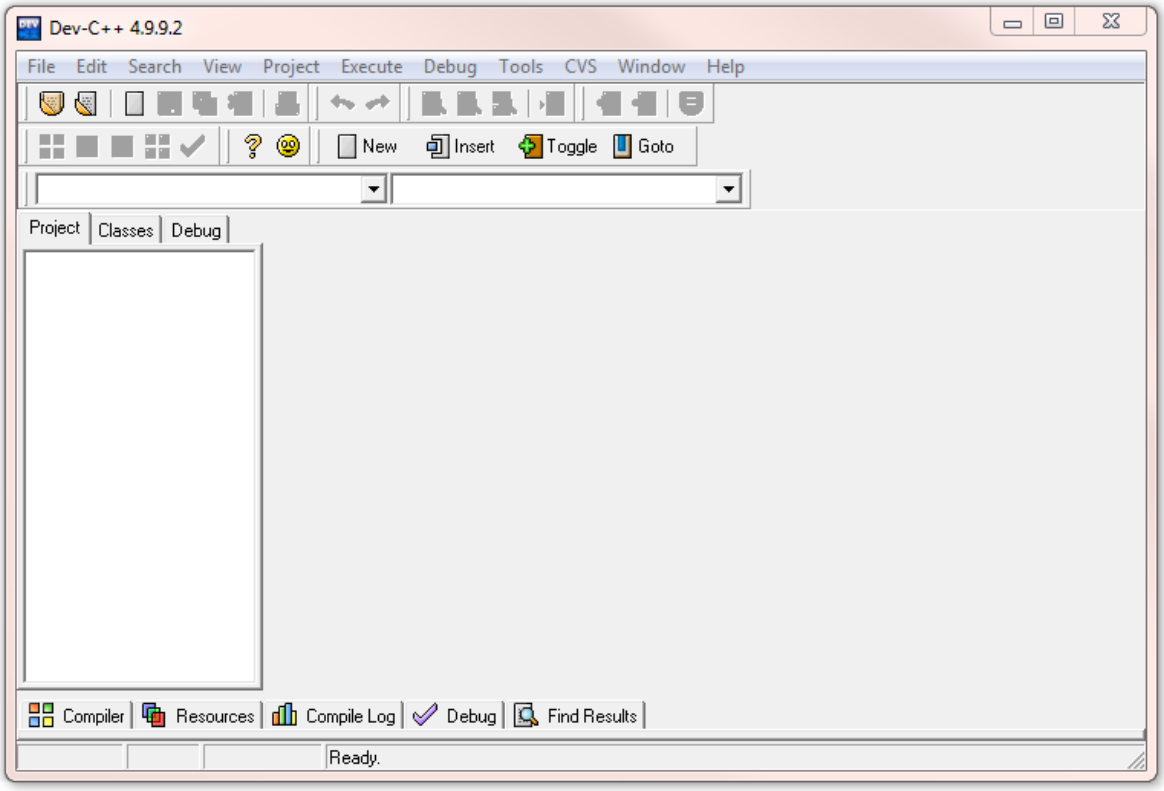
# Introduction to Dev-C++

Dev-C++ is a free, lightweight and portable IDE for programming in C and C++. You can download the software here. We highly recommend version *Dev-C++ 5.0 beta 9.2 (4.9.9.2)* (9.0 MB).

Dev C++ should be available for you to use on all departmental machines. To launch the software, simply click on the Windows Start button and type in Dev-C++ which should bring up the shortcut.

***It must be stressed here that while Dev-C++ is a full-featured IDE, it is not recommended for use in larger projects for a variety of reasons. So while it is more than adequate for our needs in the lab, you should use a more substantial IDE (such as Visual Studio) for real projects!***

Once you've installed/run Dev-C++, you should see something like the screenshot below. This is the main window for the IDE.



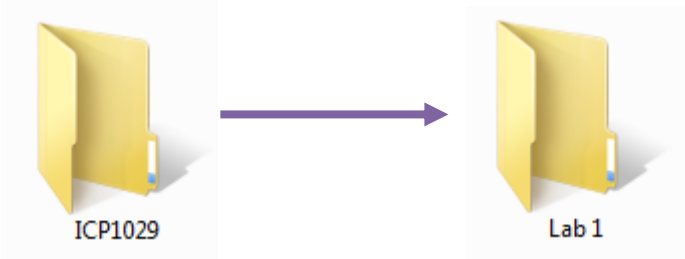
# Exercise 1 - Hello World!

## Setting up

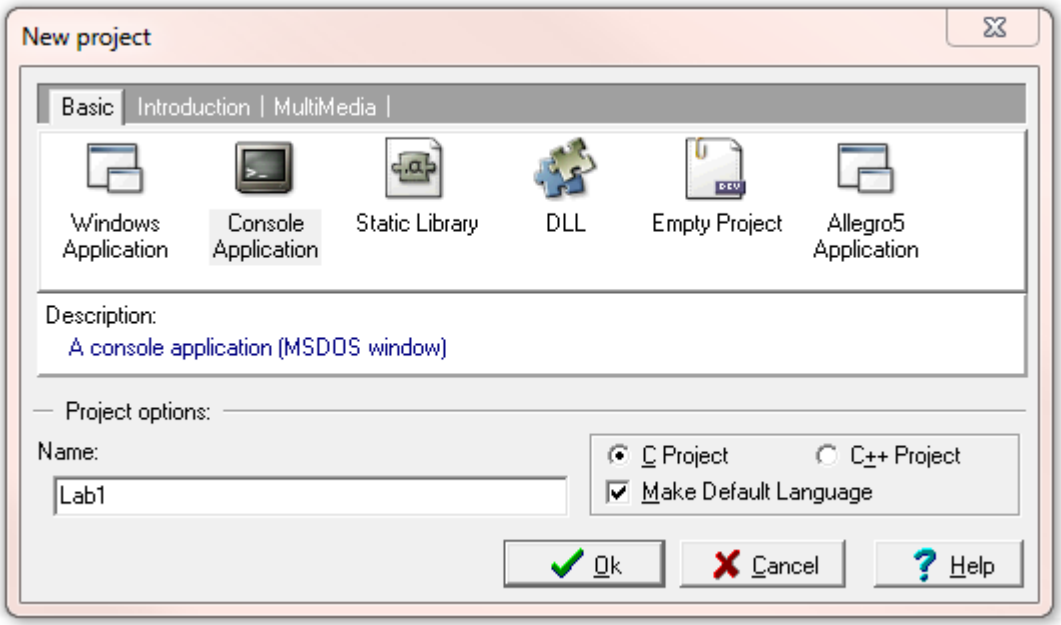
The first thing you should do is create a folder to store the course contents on your University M: drive. Create a new folder called ICP1029 or something similar.

The idea is that all the code you write for this course is stored here, with each Lab being in its own folder within. The IDE we are using for this module works in terms of projects and source files, so it's good practice to create an organised file system.

Create a new folder within the ICP1029 folder called Lab 1.




Let's start by creating a new project. In Dev-C++, go to File->New Project. You should see the following window:



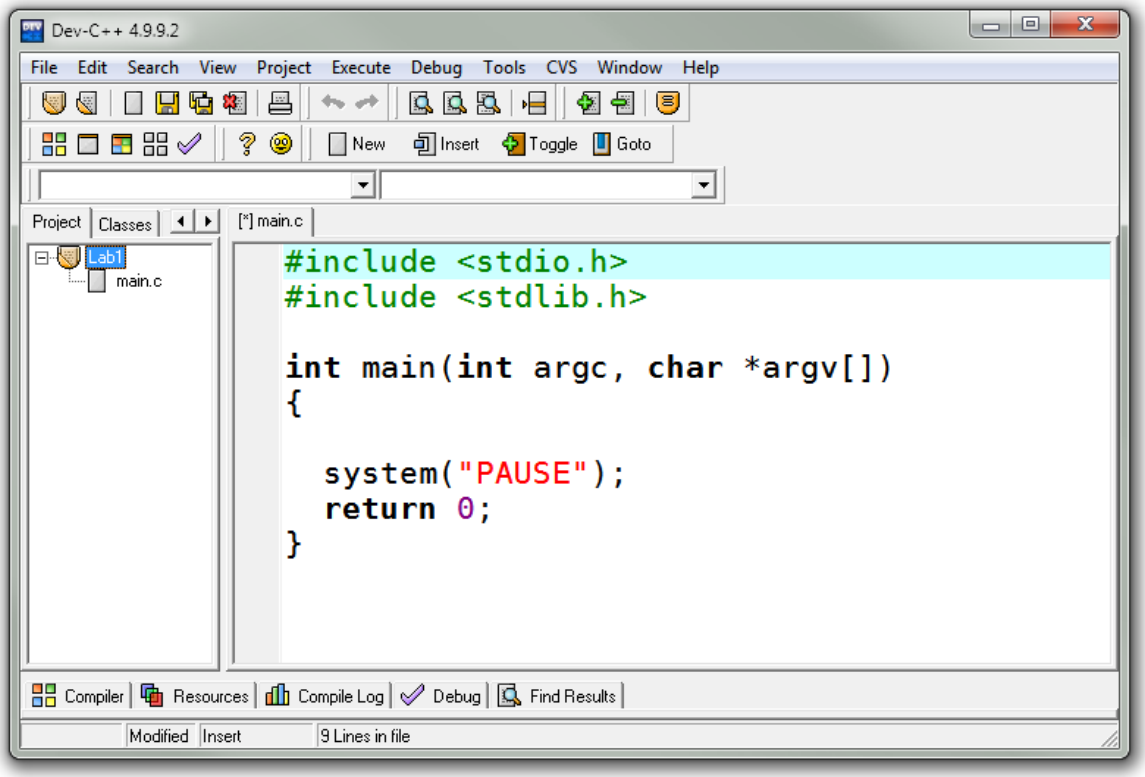

For this Lab (and through most of this course), we want to create a C Project | Console Application. To do this select the **Console Application** icon and click on the **C Project** radio button in the bottom right. You can go ahead and make it the default language too.

The last thing we want to do on this window is to give your project a name (bottom left). It is recommended for these labs that you follow a logical naming convention, not just to keep your work in order but also so that you can go back and refer to the material later.

**A good rule to follow is to name your projects by Lab, and the source files by Exercise. So for example, for this lab call your project Lab1.**

Once you have clicked Ok you will be asked where you'd like to save your project. Save it to the Lab 1 folder you created earlier.

You should now be back on the main Dev-C++ window, similar to the screenshot below.




In creating a new Console project, Dev-C++ has provided a basic skeleton program for us.

Go to the menu and click on File->Save. This will bring up a window asking where you'd like to save your source code. The default file name given is main.c, which is fine. Make sure the file is saved in the Lab1 folder.

The Project bar on the left shows you the files associated with this project. If you've followed all the steps so far you should see Lab1. If you click on the [+] icon next to Lab1 you'll find that it contains one file - main.c.

## The main function

A .c file is basically the equivalent to a .java file. It contains the source code of our program. In Dev-C++, the main window on the right should show you the code contained in the file main.c.

The first two lines (**#include**) tell the compiler to provide access to two **st**andar**d** libraries which contain a lot of functionality we will be using over the course of this course, including ways of getting input and output (**stdin.h**), and methods for converting, allocating and generating data (**stdlib.h**).

Also provided is the main function. This function is the starting point of every C Program. You'll notice that the main function contains two arguments (**argc** and **argv[]** ). These allow us to pass command line arguments into our function. We will take a closer look at these in a later lab.

The line **system("PAUSE")** is simply a way to pause the program after it's finished running - so that the program doesn't run and then automatically close. We can also use **getchar();** for this which will produce similar behaviour without printing the message "Press any key to continue".

The last line of the main function simply returns the integer 0 as is standard. Returning 0 basically means “everything ran ok, no errors to report”.

## Your first C program

It's time to create your first C program! Go to File->New->Source File and create another .c file called Lab1Exercise1.c. Don’t forget the .c file extension – without it you won’t be able to compile.

Paste the following code into the empty document, and then go to File->Save.

```
exercise1()
{

}
```

As programming tradition dictates, let's make the computer say "Hello World!" by adding the following line between the curly brackets:

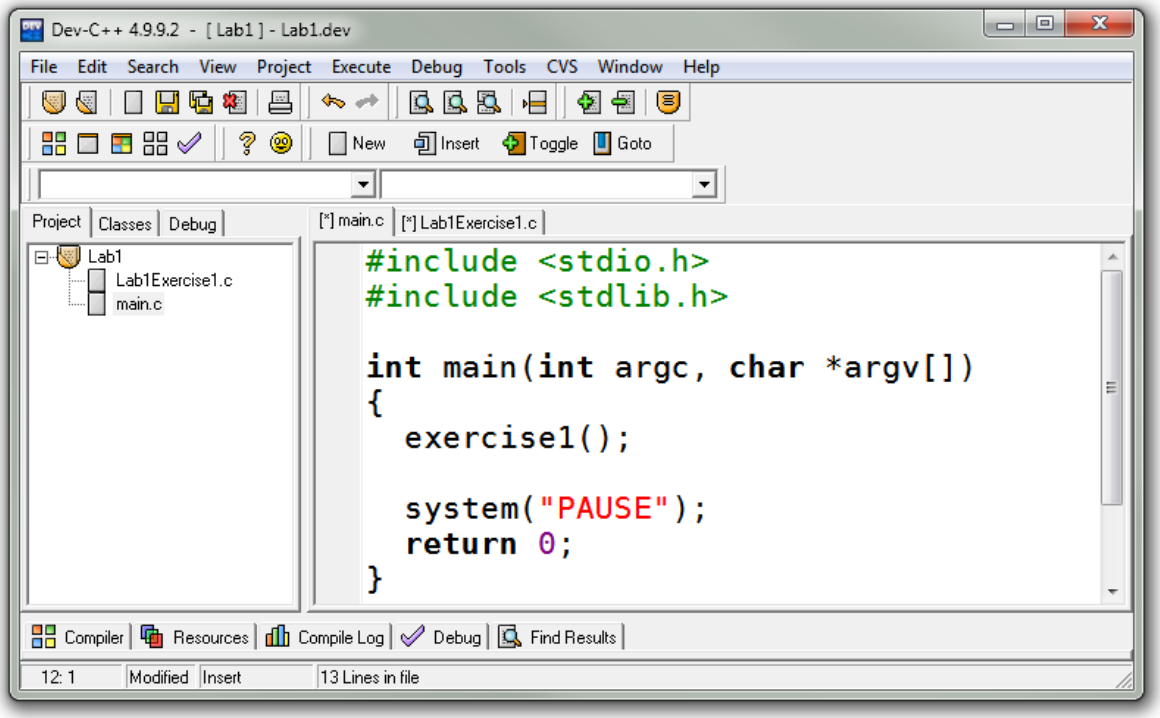
```
printf( "Hello World!\n" );
```

The last thing we need to do is **call** the function exercise1 from the main function. To do this, simply add the following line to main.c:

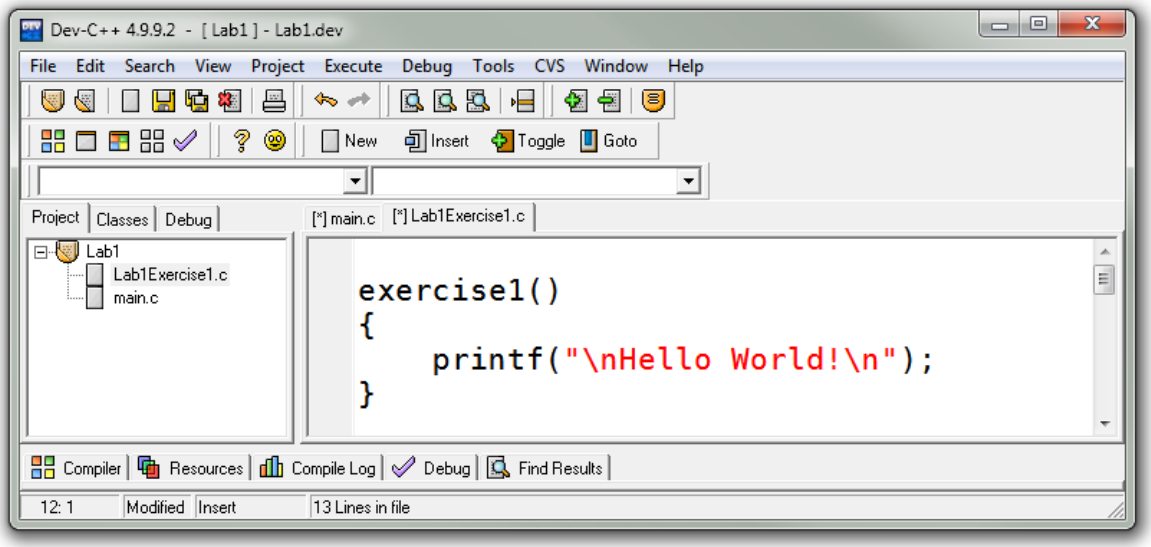
# exercise1();

By this stage your project should look something like this:

## main.c




## Lab1Exercise1.c



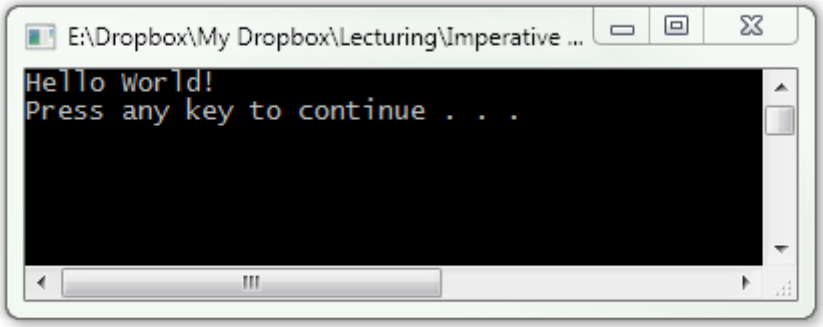
Basically what we have done here is created a main() function, an exercise1() function, and then told the program to run the contents of the exercise1() function by calling it from within main().

## Compiling and Executing

Dev-C++ provides buttons to compile, run, compile + run, build and debug our program.

Click on the left most icon. This should compile the program. Once the compile-link process has finished, Dev-C++ will give you a summary - including whether the compilation was successful or not and how many errors or warnings it came across.

Assuming everything has gone according to plan we should now be able to run our program - click on the second icon to run. This will run the program, starting with the code contained in the main() function. You should see the following output:




Congratulations - You have created, edited, compiled and executed your first C program!

# Exercise 2 – Let's try that again

Now that you have learned how to create and run a program, it's time to do it all over again! You want to get used to creating new source files for each exercise, so let's start by creating Lab1Exercise2.c

Go to File->New->Source File and add a new file to the project. This will create a new entry in the Project pane on the left called something like Untitled1. Go to File->Save and save the

new file under the name Lab1Exercise2.c. Alternatively you can right click on the name of the file in the Project pane and select Rename file.

For this exercise, we are going to create a new function called exercise2(). To do this, simply copy and paste the code below into the newly created Lab1Exercise2.c.

```
exercise2()
{
    printf("\nHello I'm Exercise 2\n\n");

    system("PAUSE");
}
```

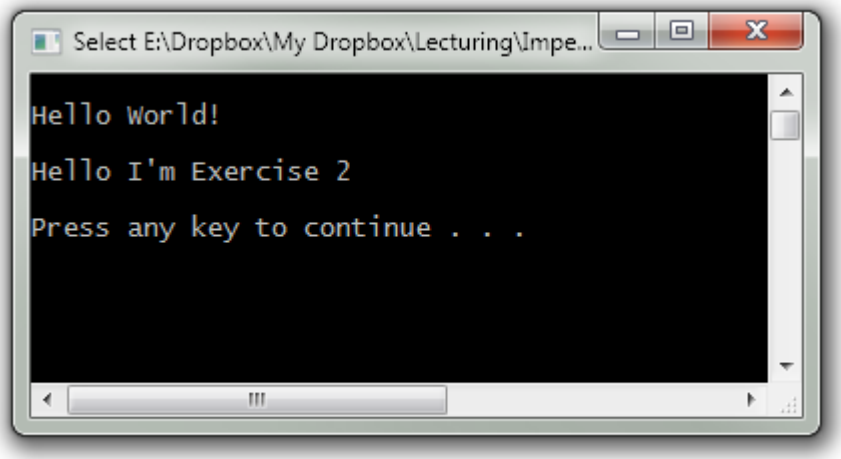
As with the previous exercise, to run the code contained in the function exercise2, we need to call the function from the main method. Modify main() so it looks like this:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    exercise1();
    exercise2();

    system("PAUSE");
    return 0;
}
```

If you compile and run your code you should produce the following output:



By now it should be clear why we are doing this. By having a main method that calls a different function (or sub-program) for each exercise, you are able to store all of the lab exercises under a single project. . **Remember to create a new source file for each exercise and call the function from main() as appropriate.**

# Comments

Comments in C are very similar to other languages. For single line comments, we use // like this:

```
int x = 0;    /* comment c style */

int y = 0;    // comment c++ style
```

For multi-line comments, we wrap the text in /* and */. For example:

```
/*
This is a
multi line
comment
*/
```

While there is no absolute need to include a comment at the start of a source file, its good practice and what most C Programmers (or any programmer) will expect to find. **For any exercise in this course, you should always include a multiline comment detailing the author and date, and a description of what the program does**:

```
/*
 * Author: Llyr ap Cenydd
 * Purpose: Main method for Lab 1. All exercises for the Lab
 *          are called from this function.
 * Date : 27/1/14
 */

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
      exercise1();
      exercise2();

      system("PAUSE");
      return 0;
}
```

We can also take advantage of more advanced methods of commenting code, allowing us to quickly comment and uncomment while testing our program.

```
/*
    printf("\nHello World!");
    printf("\nHello World!");
    printf("\nHello World!");
    printf("\nHello World!");
// */
```

Notice the extra double forward slash // on the last line. By commenting like this, we can switch the code block on and off by adding a single / character on the first line. Learning to use this shortcut now will save you many hours over the course of your career!

```
//*
    printf( "\nHello World!" );
    printf( "\nHello World!" );
    printf( "\nHello World!" );
    printf( "\nHello World!" );
// */
```

# Coding Style Conventions

You should take the time to read the Coding Style Conventions for C/C++ programming. Taken from the document:

*"The use of a guide or set of convention gives programmers a set of rules for code normalization or coding style that establishes how to format code, name variables, place comments or any other non-language dependent structural decision that is used on the code.* ***This is very important, as you share a project with others****. Agreeing to a common set of coding standards and recommendations saves time and effort, by enabling a greater understandings and transparency of the code base, providing a common ground for undocumented structures, making for easy debugging, and increasing code maintainability. These rules may also be referred to as Source Code Style, Code Conventions, Coding Standards or a variation of those."*

While the compiler won't enforce any formatting restrictions on you, it is expected (and you should always) conform to the coding style conventions. You should aim to do this right from the start - this is where bad habits start forming.

# Tips and Tricks

- Take a look at Tools->Editor Options in Dev-C++. It allows you to specify how many spaces pressing tab indents, change the font, switch line numbering on and off etc. There are some very useful options here, but it is up to you as a coder what preferences you like.
- Shift-Right click on any folder in windows allows you to select *Open a Command Prompt here*.
- You can drag-select multiple lines of code in Dev-C++ and press TAB to indent the code. Shift-Tab will unindent.

# Homework

That's it for Lab 1! Hopefully you should now be in a position to start coding in C. If you plan on working on non-University machines you should take the time over the coming week to install an IDE on your laptop/desktop and make sure you can compile and run a "Hello World" application. If there are any problems please let us know.

# Summary

Here is a list of things we have covered in Lab 1:

- General introduction to ICP-1029 : Imperative Programming in C
- Created a folder system on your M: drive for the course
- Created a new project in Dev-C++
- Edit, Compiled and Run your first C Program
- Created a few source files and example programs (Lab1Exercise1.c, Lab1Exercise2.c)
- Single and Multiline commenting
- Coding Style Conventions


PRIFYSGOL
BANGOR
UNIVERSITY