# A Beginner's Guide To C Programming V1.2

## Lab Exercises

James Bonnyman
Ed. Louise Brown
Louise.Brown@Nottingham.ac.uk

*The University of Nottingham*
*UK*

2022
The University of Nottingham

# Some common mistakes in C

When starting to learn C, there are a few common mistakes that people make... hopefully having them here will help you as you program (please use the remaining space to add any other ones you think of - and let me know to add them to the list!).

- C is case sensitive - as such, X is not the same as x
- Variables must be defined before they can be used
- New variables do not contain zero when defined
- Each line of code in C ends with a semicolon
- Arrays in C start at zero

# Contents

# Chapter 1

# Introduction

Remember when developing code for the module (and indeed any time you develop code) you must apply, from the outset, best practice.

The following, if not followed, will cost marks in the formally assessed laboratory exercises - the same criteria will also be applied when marking any code you develop as part of project and/or exam work associated with this or any other course!

- Code must be designed before you start writing it (this will save you an enormous amount of time!).
- Code MUST be commented as you develop it - NOT AT THE END
- Variable names MUST be meaningful for the values they are storing
- Function names MUST indicate the task they perform
- Global variables are not to be used in these exercises
- Input should always be checked to see if values provided are valid
- When a task can fail (e.g. opening a file, allocating memory) this MUST be checked for (with any appropriate action taken).

# Chapter 2

# Designing Code

## 2.1 Introduction - and free software

The laboratory exercises for chapter 2 will introduce you to draw.io, a drawing package that is particularly useful for drawing flow charts. This can be accessed at https://www.drawio.com/.

## 2.2 Exercise 1

Use draw.io to draw out a flowchart for a quadratic equation & linear equation solver.

In addition to the features outlines in the case study in chapter 2 of the course it should be extended to solve linear equations.

In addition to your flowchart

- Provide a table that lists the inputs and outputs indicating the type of number each will be (real, integer etc.).
- Generate a table of test data that could be used to validate the correct operation of the code.

## 2.3 Exercise 2

Develop a flow chart for a program that is able to calculate the area of a triangle given the lengths of the three sides. Your flowchart should include logic in your code to reject **ALL** invalid input cases,

In addition to your flowchart

- Provide a table that lists the inputs and outputs indicating the type of number each will be (real, integer etc.).
- Generate a table of test data that could be used to validate the correct operation of the code.

# Chapter 3

# Hello World

This exercise is designed to get you familiar with Code::Blocks to create new projects, edit, compile and execute code

## 3.1 Exercise 1

Following the guide in Appendix B of the course book create a new folder and '.c' file. After following the instructions you will have created a program which,when run will display 'Hello World' in a new command window - once you have achieved this, you are on your way to being a programmer!

**NOTE:** It is quicker to compile your code if you work on the local drive of the PC and, when the session is complete you then copy your work to your OneDrive (or other media).

## 3.2 Exercise 2

Modify the code to change the text displayed on the screen.

You might like to read ahead to see what the \n does to the text as written to the screen (or add a few extra words and see if you can work this out).

# Chapter 4

# The very basics of C

There are no specific lab exercises associated with this chapter

# Chapter 5

# Output

The exercises in this chapter will allow you to gain skills in writing out information to the screen, both as fixed text and displaying the values stored in variables (which themselves may contain the result of a calculation).

Remember to comment your code and to use meaningful names for variables.

## 5.1 Exercise 1

Please type in, compile and run the chapter 5 example program 'c5\displaying_variables.c'.

## 5.2 Exercise 2

Define a number of different numerical variable types (int, char, float) and assign them values (either when declared or as a second step).

Using the printf statement and the appropriate formatting characters display these on the screen, each value to be separated by a single space.

## 5.3 Exercise 3

Modify the code you have written for exercise 2 such that each value is displayed on a new line.

## 5.4 Exercise 4

You are required to write code to calculate the volume and surface area of a sphere or radius r. The equations for these are presented below

$$\text{Volume} = \frac{4\pi r^3}{3}$$

$$\text{Surface Area} = 4\pi r^2$$

As we have not yet covered reading in values, the value for r will need to be set in the code.

In your calculations,

- Use M_PI to obtain a value for $\pi$ (note: you need add #include <math.h> at the top of your program to be able to use this definition)
- For $r^2$ use r*r (r^2 means something else in C!), for $r^3$ use r*r*r

## 5.5 Exercise 5

Modify the code from exercise 4 such that the answer is displayed to 2 decimal places.

# Chapter 6

# Operators in C

These exercises show you how the order of the operator ++ and −− operator work.

You will also gain further use of printf to display the value held in a variable.

Following this, you will test the bitwise operator functions.

## 6.1 Exercise 1

Please type in, compile and run the chapter 6 example program 'c6\inc_dec_example.c'.

Modify the code so that the current values of the variable are displayed (using printf and the appropriate place holder) following each assignment and increment or decrement operation.

## 6.2 Exercise 2

For this exercise please define all variables as **unsigned int**

Declare variables A and B of this type and assign them the values 60 and 13 respectively (as per the values used in Section 6.5).

Declare a third variable R which will be used hold the result of the various bitwise operations.

Develop code that performs each of the bitwise operations from section 6.5, displaying the result of the calculation on the screen (using printf).

## 6.3 Exercise 3

Modify the code from Exercise 2 to declare the values of A and B in hexadecimal and to print out the results with the hex number in the form 0xnn.

# Chapter 7

# Input: Reading in information

The exercises in this chapter will give you experience in reading input from the user (both numerical and text values) and displaying this information on screen.

## 7.1 Exercise 1

Please type in, compile and run the chapter 7 example program 'c7\scanf_example_1.c'.

## 7.2 Exercise 2

Modify the code you developed in section 5.4 so that the user is prompted to enter the value for r - this value is then read in through the use of scanf.

Note: For this exercise there is no need to validate user input (this will be done in later exercises).

## 7.3 Exercise 3

The surface area of a cylinder can calculated as

$$\text{SA} = 2\pi r^2 + 2\pi rh$$

Develop code where the user is prompted to enter values for r and h. The values can be read separately or through a single scanf statement.

The result of the calculation is to be displayed to 3 decimal places.

In your calculations,

- Use M_PI to obtain a value for $\pi$ (note: you need add #include <math.h> at the top of your program to be able to use this definition)
- For $r^2$ use r*r (or, as an alternative, you can use pow(r,2) )

Note: For this exercise there is no need to validate user input (this will be done in later exercises).

## 7.4 Exercise 4

Develop a short program that prompts you to enter your name which is then displayed on the screen prefixed with 'Hello' (e.g. if you entered *James* it would display *Hello James*.

Use both scanf, gets to read in your name.

For each case, note what happens if you enter both your first name and surname.

Note: Remember to define a string that is long enough to hold all the characters to be entered plus one extra for the 'end of string'

## 7.5 Exercise 5

Develop a program that first prompts the user to enter their name after which they are requested to enter their age.

The values entered should be stored in suitably declared variables that have names appropriate to the information they will be storing.

Using a single printf statement, display the information entered in the format presented below

> Hello {*name*}
>
> You are {*age*} old

Where {*name*} and {*age*} are replaced with the values entered by the user

# Chapter 8

# Program Flow in Code

The laboratory exercises up to this point have all consisted of programs where every line of code is executed. As discussed in lectures, there are cases where portions of code are only to be executed if a test condition is met.

The exercises in this chapter will skill you in using the different approaches to implementing program flow.

You will complete the lab be enhancing code previously developed to include error checking to reject invalid input.

## 8.1 Exercise 1

Please type in, compile and run the chapter 8 example program 'c8\if_example.c'.

Modify the code so that the user is prompted to enter values for a & b - test that the condition statements work as you would expect for different values of a & b.

## 8.2 Exercise 2

Develop a program that prompts the user to enter an integer value.

If the value is in the range 0 to 10 (inclusive) the program should display on the screen the text 'The number is in range', for all other values no message should be displayed.

## 8.3 Exercise 3

Modify the code from section 8.2 so that if the number **is not** in the range specified the text 'The number is not in range' is displayed.

## 8.4 Exercise 4

Develop a program that, based on the input of an integer value from the user, displays a message as defined in table 8.1

| Age Range | Message to Display |
|---|---|
| <= 5 | Still a baby |
| > 5 and <= 12 | The junior years |
| > 12 and < 20 | Teenage Years |
| >= 20 | Downhill all the way now! |

Table 8.1: Messages to display for given age ranges

## 8.5 Exercise 5

Using the if/else if/else approach develop code that, based on user input of an integer displays a message as defined in table 8.2.

If the value entered is not listed then the default message 'Invalid input' should be displayed.

| Value | Message to Display |
|---|---|
| 0 | Black |
| 1 | Brown |
| 2 | Red |
| 3 | Orange |
| 4 | Yellow |
| 5 | Green |
| 6 | Blue |
| 7 | Violet |
| 8 | Grey |
| 9 | White |

Table 8.2: Messages to display for given inputs

## 8.6 Exercise 6

Repeat task 8.5 using the switch-case approach

## 8.7 Exercise 7

This final task has you improve existing code to make it reject invalid user input (that may cause the code to crash or which would result in a 'meaningless' answer).

Modify the code you previously developed to calculate the surface area of a cylinder (section 7.3) such that the calculation is **only** performed if both inputs are valid.

You are free to adopt any approach to this (there a number of options) - you may be asked to justify your choice by a demonstrator!

# Chapter 9

# Loops - repeating things

Loops play a crucial role in programming, allowing code to be repeatedly executed while a test condition is met or for a range of numbers. The exercises in this chapter are designed to give you experience in using all types of loops.

## 9.1 Exercise 1

Please type in, compile and run the chapter 7 example program 'c9\while_loop.c'.

## 9.2 Exercise 2

Modify the loop code (i.e. that controlled by the *while* condition) to include a condition such that if the age entered is 18 or 21 it additionally displays the message 'You have come of age'

## 9.3 Exercise 3

Develop a program that first asks the user to enter their name, it should then continually display the name entered on a new line (with no method of the loop breaking)

Hint: To stop the code press the Control Key and the letter C at the same time

## 9.4 Exercise 4

Implement the code from 9.2 using a do-while loop, note how in this case when the age entered is zero it is displayed on the screen as part of the 'You are...' output.

## 9.5 Exercise 5

Develop a program that counts from 1 to 15 inclusive and displays the value of the loop counter on the screen - each value to be displayed on the same line, separated from the previous by a single space (i.e. 1 2 3 4 .... 15 )

## 9.6 Exercise 6

Add additional code to that developed in section 9.5 so that the lower and upper values of the loop are entered by the user (hint: you will need to declare two additional variables for this task and use these in place of the 1 and 15).

What test should you put in place to ensure the loop will execute as expected (consider the values a user might enter, perhaps if they misread the order to enter them).

Keep a copy of this code - you will be modifying it in the next two exercises

## 9.7 Exercise 7

With for loops, we often make use of the loop variable within a calculation (perhaps displaying the output within the loop).

Expand the code from section 9.6 to display both the value of the loop variable and this value squared (remember not to use ^2 to square a number - for reasons you now know!).

This time put the pairs of values on a new line.

## 9.8 Exercise 8

With for loops, as you have seen, we can make use of the loop variable within a calculation, displaying the output within the loop.

This task is a little different! This time you are required to expand the code from section 9.6 to keep a running sum of the loop variable, displaying the final result **only** when the loop terminates.

E.g. if the loop counted from 1 to 5 inclusive the final result would be 1+2+3+4+5 = 15, if the loop was from 1 to 10 inclusive it would be 1+2+3+4+5+6+7+8+9+10 = 55

# Chapter 10

# Functions (part 1)

The lab exercises associated with this chapter are designed to give you experience in developing functions that take a number of parameters and return a value.

You may find it helpful when undertaking these tasks to consult the 'Function Flowchart' from Appendix C of the course handbook.

For these exercises you will need to develop both in main() and a function (or functions) external to main().

## 10.1 Exercise 1

Please type in, compile and run the chapter 10 example program 'c10\function_example.c'.

## 10.2 Exercise 2

When working with the trigonometric functions in C we need to use radians.

This exercise will requires you to develop a function (that will be external to main) that when passed a float will return this value converted to radians.

The conversion you need to use is

$$radians = \frac{\pi * degrees}{180.0}$$ (remember to use M_PI for $\pi$)

Remember, when developing your function to use meaningful names both for the function, the variables passed and any variables you declare locally (both in main() and in your function).

To test your function, develop code in main() that prompts the user to enter a value in degrees, this should then be passed to the function you have developed.

You can either store the result in a variable and display this (using printf) or, if you prefer, use the function itself in the printf statements.

## 10.3 Exercise 3

This exercise is designed to show you how well written code can be reused!

Create a new program and paste in the code you developed for section 9.7

Now paste, above main(), the function you developed in section 10.2

Change the code in the loop to call the function (rather than calculating the square of the loop variable). Remember to change the formatting character as the second value you will be displaying will now be a float.

## 10.4 exercise 4

This (and the next exercise) will bring together a few aspects of the module to develop a rather 'clever' piece of code.

It is possible to determine the day of the week based for any given date using the following approach

Assume that "year", "month" and "day" have been given as three integers, for example 1965, 8, 23 respectively for August 23rd 1965. The calculation continues.

```
if ( month < 3 )
{
      month=month+12;
      year=year-1;
}
```

Then

$$nd = (\frac{13*month+3}{5} + day + year + \frac{year}{4} - \frac{year}{100} + \frac{year}{400})\ mod\ 7$$

You are required to develop a program that, in main(), prompts the user to enter values for day, month and year (store these in integer variables). You can assume that the values entered will be valid (validating them would be too complex a task).

Develop a function designed to receive day, month and year as parameters (these **MUST** be defined as integers for this to work) and then use the function provided to calculate *nd*. Note: To calculate modulus in C use % (e.g. a % b means a mod b ).

Pass the values entered to your function and store the value returned in (another) integer variable.

Display the calculated value on the screen; this value gives the day of the week (based on 0=Monday, 1=Tuesday etc.).

## 10.5 exercise 5

To further improve our 'day of the week' program we are now going add a second function to which we pass the calculated value. This function should display the relevant day of the week.

The function will not be returning a value (only receiving a single integer) so the return type will be **void**.

You can use any of the different methods to display the day of the week based on the value passed (e.g. if, if/else if/else if/...., switch-case).

# Chapter 11

# Arrays

## 11.1 Exercise 1

Please type in, compile and run the chapter 11 example program 'c11\array_loop_example_1.c'.

## 11.2 Exercise 2

You are required to write a program that creates a float array of size 90.

Using a for loop, populate the array element whose index is the value of the loop variable with the value of the loop variable (e.g. array[0] = 0, array[1] = 1 etc.)

Using a second loop display the loop index and the value in the array - each pair of numbers is to be displayed on a new line.

## 11.3 Exercise 3

Again we see how well written functions can be reused!

Copy the code you developed for 11.2 into a new project.

Copy the function you developed for converting from degrees to radians (section 10.2 ) into your code (above or below the main() code - your choice).

Modify the code so that the number stored in the array is not the array index but the value of the array index converted to radians (i.e. pass the loop variable to your function and store the result in the array).

# Chapter 12

# Variables - Part 2

There are no specific lab exercises associated with this chapter

# Chapter 13

# Pointers (part 1)

These exercises are designed to show you how the value in an existing variable can be modified if we know the address at which it is stored.

This you should now appreciate is how scanf works. We pass the address of an existing variable, scanf interprets what is entered at the keyboard and then stores the result at the address of the variable - we can then make use this variable as required in our code.

## 13.1 Exercise 1

Please type in, compile and run the chapter 13 example program 'c13\accessing_via_pointers_1.c'.

## 13.2 Exercise 2

Modify the example code to change the value stored in c to 20 **via the pointer before** it is used to set the value of d (i.e. the value of d, when displayed, should be 20 ).

## 13.3 Exercise 3

Copy the code developed for section 13.2 to a new project and modify it to work with float variables.

# Chapter 14

# Functions (part 2)

In this chapter, the exercises are designed to show you how to develop functions that can return multiple values.

We will (again) be making use of the 'degree to radians' functions previously developed (showing how re can easily reuse well written code).

## 14.1 Exercise 1

In this exercise you will be developing a function that is passed x & y coordinates and return the spherical polar coordinates of this point.

The equations you need to perform the calculation are

$$r = \sqrt{x^2 + y^2} \qquad theta = \frac{y}{x}$$

Develop a function that is passed four parameters. The first two should be for x & y (which can be passed as values), the second two for r & theta (these need to be declared such as they receive the addresses of existing variables).

Consider carefully the type of variables to use to make this a practical, general function.

Develop main() code that prompts the user to enter values for x & y. These should then be passed to the function (along with the addresses of suitably declared variables to receive the result).

Display on the screen the two values returned from the the function.

## 14.2 Exercise 2

This exercise will see you developing a function that is passed an angle in degrees and returns four values, the value converted to radians and the sine, cosine and tangent of the angle passed.

To be efficient the code you develop will perform the degrees to radian calculation using your previously developed function. This value is then passed to the sine, cosine and tangent functions (as these work on angles in radians).

To demonstrate your function, develop code in main() that prompts the user to enter a value in degrees. This value is then to be passed to your function (along with the addresses of suitably declared values to store the four returned values).

All values are then to be displayed on the screen from within main().

## 14.3 Exercise 3

Copy the code you have developed for section 14.2

Rather than have the user enter a single value, have them enter two integer values.

Using a **for** loop, count over this range defined by these two values. Pass the loop value to the function (rather than then value entered by the user as before) and, as before, display all the values on the screen with the float values displayed to a precision of 3 decimal points).

If you have time, perhaps add headers to the output (using a printf statement before the loop starts).

When run, if the limits for the loop were 10 and 20 the output should resemble that below

```
Degs    Rad     sin     cos     tan
10      0.175   0.174   0.985   0.176
11      0.192   0.191   0.982   0.194
12      0.209   0.208   0.978   0.213
13      0.227   0.225   0.974   0.231
14      0.244   0.242   0.970   0.249
15      0.262   0.259   0.966   0.268
16      0.279   0.276   0.961   0.287
17      0.297   0.292   0.956   0.306
18      0.314   0.309   0.951   0.325
19      0.332   0.326   0.946   0.344
20      0.349   0.342   0.940   0.364
```

# Chapter 15
# Pointers (part 2)

There are no specific lab exercises associated with this chapter

# Chapter 16

# Dynamic Memory Allocation

The exercises in this chapter will familiarise you with the process for dynamically allocating memory in C using calloc or malloc.

## 16.1 Exercise 1

Please type in, compile and run the chapter 16 example program 'c16\alloc_example_4.c'.

## 16.2 Exercise 2

Modify the code to prompt the user how big the array is to be and use this user entered value to allocate memory (i.e. replace the '10000' with the variable in wish the user input was stored).

## 16.3 Exercise 2

As you have done previously (section 11.2)), add a loop to the code that populate the array element whose index is the value of the loop variable the value of the loop variable (e.g. array[0] = 0, array[1] = 1 etc.) - the difference here is that the upper limit is defined by the value entered by the user.

Display the loop variable and the value stored at that index in the array - this can be within the same loop or, if you prefer in a second loop.

# Chapter 17

# Functions (Part 3)

## 17.1 Exercise 1

Please type in, compile and run the chapter 17 example program 'c17\arrays_to_functions_example_2.c'.

## 17.2 Exercise 2

Using the code from 17.1 as a template, modify the code (or better yet, write it from scratch!) to meet the following requirements

- The array should be of type float
- The populate function should by modified to populate the array with the value of the loop variable converted to radians [*]
- The display function should be updated to display the float values passed to 3 decimal places of precision.

[*] You may have a function written that you can use to do this :-)

# Chapter 18

# Using Files

This chapter will give you practise in reading & writing files - both text and binary.

## 18.1 Exercise 1

Please type in, compile and run the chapter 18 example program 'c18\text_file_example.c'.

## 18.2 Exercise 2

Modify the code from 18.1 so that the value of the loop counter and this value squared are written to the text file.

Make the necessary changes to the code to enable it to read the data back from the file, displaying the pairs of values on the screen (hint: you will need to declare a second variable into which you read the squared value).

## 18.3 Exercise 3

Add additional code to that developed for 18.2 to prompt the user to enter the name for the file.

Modify the fopen statements to use the file name entered by the user (rather than the fixed file name 'numbers.txt').

## 18.4 Exercise 4

This exercise is going to take some existing code and provide additional functionality.

It is quite a challenging exercise so take a little time to think about it before you start coding.

As a starting point, copy the code you developed for Section 17.2.

You are now going to create an additional function (based on the 'display on screen' function) that writes the output to a specified file name, as such, in addition to the array it needs to be passed an additional parameter which is the file name (that has been declared and entered in main() ).

All the checking that the file can be opened, writing to the file and closing of the file is to be done in this new function.

## 18.5 Exercise 5

We are now going to make a final addition to the code from 18.4 - to also write the data to a binary file (the name for which needs to be prompted for in main() ).

In some ways this is easier to do as we can write the entire array with a single **fwrite** instruction - the downside is that we cannot 'view' the data written to the file (without using other code).

In the next exercise we will read this binary file.

## 18.6 Exercise 6

Develop a program that within main() opens the binary file created in 18.5 and reads the values into a suitably defined array (you can use a fixed array or, if you wish to practice your skill, use malloc/calloc to define the array).

# Chapter 19

# Advanced Data Types in C

This chapter will focus on the use of structures in C as they can make programming (especially where a large number of parameters need to be passed to a function) considerably easier.

You will also see how a structure can be written and read from a file.

## 19.1 Exercise 1

Define a struct (outside of main) that contains the following members

- age - an integer
- forename - a string that can hold up to 30 characters
- surname - a string that can hold up to 50 characters

Within main() declare a variable of this type.

Using scanf read into each member information that is appropriate for you - remember that to read into a member you need to use the dot operator to specify the member (if you are not sure, ask a demonstrator for help).

Using printf, display the stored information on the the screen - remembering again to use the dot operator to select a member of the structure (and to use the appropriate formatting character - e.g. for 'age' it will be %d).

## 19.2 Exercise 2

We can pass a structure to a function the same as any other variable in C - the difference is that, by doing this, we are passing all the members of the structure (so this approach is great if you need to pass a very large number of parameters to a function).

Develop a void function that received the populate structure from exercise 19.1 and use this to displays the information in the structure (rather than doing this in main).

## 19.3 Exercise 3

We are now going to see how we can add an additional item to our structure and how it is then immediately available to any function to which the structure is passed (using the approach so far adopted, we would need to update the function definition and add the new parameter in all function calls - which could take a very long time!).

Copy the code from section 19.2 into a new project.

Add a new integer member 'year_of_birth' to the structure and add additional code to main to prompt for and store a value in this new structure member.

Update the function to display this additional value.

## 19.4 Exercise 4

Another time when a structure can be of particular use is as a file header (information at the start of a file that describes the contents). We can read this information in 'one go' and, based on this, know how to process the information.

Note: When reading/writing we use fread/fwrite as - we can still use **sizeof()** to obtain the total size of the structure (it 'adds' together the sizes of the individual members for us).

This exercise requires you to extend the code from 19.3 to ask if the user wishes to enter data and write this to a file or to read an existing file and to display the information, as such you will need to add program flow logic to the code.

Copy the code developed for section 19.3 into a new project.

Prompt the user to choose to ether

- (1) enter data values and store them to a file
- (2) Read data from a file and display it on the screen

In the case of (1) you can use a fixed file name when developing the code (though you may later like to prompt for this as you did in the exercise in section 18.3). You should then open the file (with checking!) in **binary** mode for writing and use fwrite to save the data to file - remember then to close the file.

For (2) you need to open a file created from running your code and having selected (1), as before the file name can be fixed or you can ask the user to enter this. Open the file (with checking) and read the data into the structure (using fread). Pass the now populated structure to the function previously developed (so displaying the information read on the screen).

# Chapter 20

# Compiler Preprocessor Directives

This exercise will have you update one of your existing programs to provide a 'debug' mode that can be turned on/off as required (this is something that you may find particularly useful to be able to do in the project work).

## 20.1 Exercise 1

Select any of the programs you have written (ideally one that makes use of functions) and copy the code to a new project.

Add in some 'debug' messages to the code that are only compiled into the code when the compiler variable DEBUG is defined (as per example 20.2 in the course book).

Such messages might include 'Program Started', 'Function {name of function} started', 'Function ending' etc.

You can even have different compiler variables to turn on different debugging messages/levels e.g.

- define a compiler variable DEBUG_MAIN and use this to turn on/off messages in main()
- define a compiler variable DEBUG_FNS and use this to turn on/off messages in functions you write outside of main()

NOTE: Although this exercise uses DEBUG as compiler variable, you can use (within the usual constraints) any name you wish (e.g. DEMO_MODE, APPLE_BUILD etc.).

# Chapter 21

# Command Line Arguments

Command line arguments are useful to us as they allow us to pass information directly into our code (without the need to any additional user input).

We often make use of this to 'batch process' where we create a file containing the commands we would type at the keyboard and have the operating system 'read' from this file (leaving us free to do other things as we do not have to wait for the program to finish so we can start the next 'run').

Note: To pass arguments into code being executed by Code::Blocks you need to set the arguments; to do this select 'Project' from the menu, then 'Set programs' arguments...'. Make sure you have the same target set as you are compiling for (Debug/Release) and then enter the arguments in the lower box (leaving a space between items).

## 21.1 Exercise 1

Create a new project based on the code example from the lecture c21\get_them.c

Set arguments for the program and run the code to confirm you are able to set values to be read into your programs.

You will notice that there is one additional item displayed that you did not enter, this is the item in argv[0] which is **ALWAYS** the name of the program being executed.

## 21.2 Exercise 2

You are now going to modify the code you developed to display the day of the week based on the user entering three integer values for day, month & year.

If there are exactly three items passed to the program (argc will be 4 in this case - the file name & three parameters) your code should take the values from the command line (you may assume they are entered in the correct order).

If the value of argc is any value other than 4 the code should execute as it previously did (prompting the user to enter values for day, month & year).

# Chapter 22

# A few more examples to try