Chapter 1 – Intro to C Coding

By the end of this chapter you should be able to:

- Use input and output functions in C
- Use the basic conventions of programming to show good practice

How the chapters work….

Each chapter introduces a new structure or idea and has:

- Key terms
  Important for understanding exam Qs & learning definitions
- Examples of C code
  For you to copy and run so you can follow the explanations (which are colour coded) and see how they work
- Tasks for you to write on your own
  To ensure that you can apply what you have learned

Key terms to ensure you know:

- **Variable**: a memory space that you give a name that you can use to store data
- **Pointer:** in C this is the & sign – it's used in an input statement (a scanf) which tells the compiler where to send data once it's been entered
- **Library:** ready-made and pre-compiled bits of code/ functions that you can use
- **Buffer:** a temporary storage area
- **Assignment statement:** the line of code that stores data to a variable
- **Initialisation / initialising:** giving your variables a start value/ clearing them
- **Format code:** the bits of code that set/ change the way a variable is input and output/ displayed

Example 1

```
#include<stdio.h>
#include<stdlib.h>
//this program reads in a number and outputs it

main()
{
        //declares and initialises a variable
        int value=0;

        //outputs the message to screen
        printf("Enter a number :");

        //gets the user input
        scanf("%d", &value);
        printf("value is %d \n",value);
}
```

- Include loads basic libraries so you can use built-in functions to read in and out data
- Comments are lines to explain your program and are ignored by the compiler – you start them with two slashes e.g. //
- We declare an integer **variable** (a named memory location) called *value* and **initialise** it as 0; this ensures it has no data in it
- **printf** calls the ready made print library function. It outputs & displays text on screen
- **scanf** is the input library function that gets what you type on the keyboard.
- When the IDE gets to a **scanf**, it pauses and waits for user input & enter to be pressed.
- **%d** format code tells the scanf function that the input is going to be stored as an integer
- The **&** sign is a variable pointer, it makes the compiler understand that 'value' is the memory space to 'point' and store the data when the scanf reads it in
- To output a variable, we use the **%d** format code in the text as a placeholder and then list the variable **at the end** of the printf statement

Extra info about scanf

You can also read in more than one value at a time using the same scanf e.g.

scanf ("%d%d",&val1, &val2);

scanf ("%d , %d",&val1, &val2);

- The top version will read in 2 values separated by a space - the first will stored in the variable **val1** and the second will into **val2**.
- The bottom version will read in 2 values separated by a comma i.e. the user types in **4,4**

Data Types & Format Codes

There are a whole range of different types of data and format codes that you can use in C. Each data type reserves a different amount of memory.

The main ones you will use are in bold:

int	:	**%d – a whole number.** Uses 2 bytes of storage.
float	:	**%f – a real number e.g. has a decimal point.** Uses 4/ 8 bytes.
char	:	**%c – one individual ASCII character.** Uses one byte.
string	:	**%s – a list of ASCII characters e.g. a word or series of words**
long long	:	%lld – a very large whole number up to 2^{63}
hex	:	%x – integer in hexadecimal form
exponential	:	%e – exponential notation

Inputs: Memory Buffers

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

main()
{
    int val=0;
    printf("Enter a number :");
    //clear buffer before next input
    fflush(stdin);
    scanf("%d", &val);
    printf("%d squared is %d \n",val, val*val);
}
```

When we use **scanf** for input it uses **memory buffers**, which are temporary memory storage areas, until the data can be moved to wherever the variable pointer sends it in memory.

These buffers are reused again and again and this means you can't always be sure what is contained in them e.g. sometimes data is left in them.

To prevent this, a C function called **fflush** is used which flushes the keyboard buffer and gets rid of any unwanted data stored there.

It's a **good habit** to get into to insert it **before** EVERY *scanf* statement line.

Exercises

1. Using the pseudocode example below, write a program that asks the user for 2 float numbers, separated by commas and then multiplies them, and outputs the answer.

```
num1 = INPUT ("Enter number 1")
num2 = INPUT ("Enter number 2")
result = num1 * num2
OUTPUT result
```

Try it: enter invalid data e.g. enter integers and letters – does it crash? Display 0's? Why…..?

2. We can use format codes to output/ display a stored data type as a different one e.g.: I can read a number into an integer variable and ask C to output it as a char. This will display the int value as a character - whatever its equivalent letter is. (values between 65-90 (upper case) and 97-122 (lower case) but you can try other values and see what comes out!)

   Write a program that asks the user to input **one** letter, and then outputs that letter's ASCII code value. Think about how you can use the different format codes as described above in a printf statement.

Challenge Exercise – Error Checking

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

int main()
{
   int value=0, check =0;
   printf("Enter a number  :");
   fflush(stdin);
   check = scanf("%d", &value);
   printf("Scanf returned: %d \n", check);
}
```

- Many of the inbuilt functions in C return a number which can be used to check if it has worked correctly
- In the code example, I've added in a new variable called *check* (but it could be anything, it's just a number variable).
- This scanf is supposed to read in a single integer variable. So the function will return the number of values of that type it's read in, which we can use (later on in IF statements) to error check.
- The scanf will return 0 if it hasn't read any value it should have, a 1 if it's read in one number successfully, a 2 if it has read in two values etc

Exercise 3) Add to your program so it uses a new variable to catch the return value from scanf. Make sure it displays the value of **error, like my program above**.
Add another couple of lines of code to then read in two numbers, again using your **error** variable, and display the value of **error**.
Now try it with three values.
What is the value of **error** when an invalid character is typed in?
Comment out the **fflush(stdin)** between reading in the data. What happens?

Computational thinking – decomposition…

- Bob is building a house. But Bob is very bad at maths (poor Bob) and keeps ordering either too many or too few pallets of bricks.
- He wants you to build him a program to work out how many bricks he needs & how many to order:
  - What do you need to know before you can build Bobs program?
  - What do you already know/have assumed?
  - How can we break the problem into sub-tasks?

Decomposition task

- Inputs- number of walls, length of walls in metres (assume they are all the same length at this point), height of walls (number of rows).
- Processing – it takes 1.5 bricks to build 1m of 1 row of wall. There are 100 bricks on a pallet
- Outputs- how many bricks are needed, how many pallets are needed, how many bricks will be left over.

TASK - Work in pairs to plan an algorithm that may solve Bob's problem.

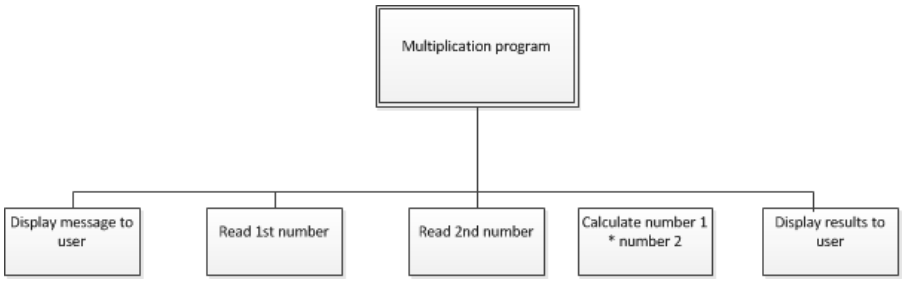
When your algorithm's been checked you are to individually code & test it in CLion

Decomp – always ID the components of a problem

- With any problem it is important to first identify the components of the problem.
  - What do you know?
  - What don't you know?
  - Can you break the problem down?
  - Can you break the sub-tasks down even more (refine)
  - Are some elements of the problem more important?
  - Do you need to do some elements in order?

Structure Diagrams

- We can use diagrams to break down a problem and show how it's been decomposed. This is a **top down** approach.
- Being able to produce these is a vital exam skill and we'll be getting you to do this quite a bit!



Pathway 1 Practice

1. Write a program that takes the following inputs:
   - A whole number
   - A decimal number
   - A letter

   The program should output user friendly messages to show:

   a) The result of the sum: the decimal number subtracted from the whole number
   b) The result of the sum calculated in **a** above multiplied by the ascii value of the letter

2. Write a program that asks for a letter to be input in upper case. The program should then output that letter in lower case.

   To do this, think about each letter's binary ASCII values and how you could use these to convert them. Useful info: capital 'A' = 65 and lower case 'a' = 97.

3. Draw a structure diagram using VISIO/ Draw.io to represent one of the programs above.

Pathway 2 Practice

1. You should **plan** an algorithm as a structure diagram that gives the user two options.

   Choice 1 allows a user to:
   - input a whole number
   - outputs whether it is a prime number or a composite number (P or C)

   Choice 2 means that:
   - The program randomly displays a ‘P’ or a ‘C’ and waits for input from the user to input a number value
   - Checks if they have entered a correct example
   - Outputs whether they were correct or not