EE101 C programming and SW engineering 1
Lab Practice 2 – Basic C Exercises

Objectives

The main objective of this Practical is to let you review the basic C language elements introduced during the lecture by solving simple programming exercises.

General Guidelines

Write the code for each exercise using the Microsoft Visual Studio 2013 Compiler (remember to follow the instructions on opening a new C project in Lab Practice 1) or your preferred compiler can be used. Add comments to your code as necessary so that someone else can read it and understand what your program is supposed to do. Save the code for each exercise as exercise1.c, exercise2.c, etc. and keep them for later reference. Compile the code for every exercise making sure you sort out all syntax problems. Run the executable file and make sure your program works correctly. Try inputting incorrect data to see what happens and if there is any way to solve the problem.

Exercise 1: (Using Escape Characters)

```
#include <stdio.h>

main(){
printf("I am writing this text on one line");
}
```

Compile and run the program. Now modify the program as follows

```
#include <stdio.h>

main(){
printf("I am still writing this text"); printf(" on one line");
}
```

Did you notice any changes? You shouldn't! The reason is that you need to request explicitly if you want something to be printed on a new line. Here's how you do it

```
#include <stdio.h>

main(){
printf("Now I am writing this text\n"); printf(" on two lines!");
}
```

The secret is to insert two characters, \n (read "backslash n") at the point where you wish the text to appear on a new line. The "backslash" character \ is known as the escape character because it lets the character that follows to 'escape' being printed on screen. The complete sequence of characters \n is known as an escape sequence.

Exercise 2: (Writing a simple program)

Write a program that displays your name vertically. Use printf() **only once**. Compile and run your program to check if you were right.

Exercise 3: (Writing a simple program)

Write a program that displays your name on two lines inside a rectangle of asterisks as follows:

```
**********
*  First name  *
*  Surname     *
**********
```

Make sure the asterisks are properly aligned. To do this you will need to insert spaces (blanks) in the right place. Use printf() exactly **four times**. Test your program.

Here are some more Escape Sequences:

\a	bell	\0	null character
\r	carriage return	\b	backspace
\f	form feed	\n	newline
\v	vertical tab	\t	horizontal tab
\’	single quote	\\	backslash
\?	question mark	\"	double quote

Write a program that displays your name on a single line and makes the computer beep at the end! Find which escape sequence from the previous list you need to use. Test your program to see and hear if it works!

Exercise 4 (Use of #define directive)

The following example shows how you can use the #define preprocessor directive to introduce constants in your program

```
#include <stdio.h>
#define UK_SPEED_LIMIT 70

main(){
printf(“UK Speed Limit also known as UK_SPEED_LIMIT %d is”,
UK_SPEED_LIMIT);
}
```

Edit, compile and run this program. What happened? The preprocessor - which prepares the code to be compiled- has replaced all occurrences of your constant

UK_SPEED_LIMIT in the file you have just edited with 70 before passing the text to the compiler. Note however that the identifier name in between the quoted string (“”) has not been altered. Note also that because UK_SPEED_LIMIT has been assigned an integer value we had to use the conversion symbol % followed by the conversion character d to indicate that we want the result to be a decimal integer.

So if you write a program that involves the speed limit in UK, you will only need to modify your program once, in the preprocessor section, when the speed limit will finally be in line with the current models of Ferrari.

Exercise 5 (Variable declarations and initialisation)

Have you thought about what happened once you have declared your own set of variables in a program? What is their value once they have been declared?
Let's try a simple example

```
#include <stdio.h>

main(){
int my_int;
printf("Variable my_int is: %d", my_int);
}
```

What did you see? Was it what you expected? And why do you think you saw this?

Exercise 6: (variable initialisation)

Write a similar program to test the value of a float variable you have declared in your program. Use the conversion specifier %f to print out the result.
Let's declare and initialise our variables in one go:

```
#include <stdio.h>

main(){
int my_int = 120;
printf(“Variable my_int is: %d”, my_int) ;
}
```

Exercise 7: (variable initialisation)

Write a similar program that declares and initialises the value of a char value and then prints the result on the screen. Use the conversion specifier %c to format the number on screen.

Exercise 8: (Some simple maths)

Lets try a simple example that involves some simple arithmetic operations

```
#include <stdio.h>

main(){

int miles, yards;
float kilometres;

miles = 26;
yards = 385 ;

kilometres = l.609 * ( miles + (yards/1760.0) );

printf("\n To win a marathon you must run for %f kilometres!", kilometres);
}
```

What is the distance in kilometres you have to run to complete the marathon?

Exercise 9: (Some simple maths)

Write a similar program that converts 3 hours and 25 minutes into seconds and displays the result. Use the #define directive to define the constants SECONDS_PER_HOUR and MINUTES_PER_HOUR. Use these constants in your program.