Practical 1B

Introduction to C - Decimal to radix-n converter

Before the practical ensure that you complete the following:

Read and revise:

Chapter 6 - C for microcontrollers in the **EEE2046F/EEE2050F course notes**. You may also want to refer to the **Modern C** textbook for more detailed information.
Please note that the practicals for this course contain a large self-study component and therefore be prepared before the start of the practical.

1B.1 Introduction

The aim of this practical is to introduce you to basic C programming step by step by designing a decimal to radix-n converter program where $n \in \{2, 3, 4, \ldots, 16\}$.

This practical contains both coding and a report submission and may take longer to complete than the 2 hour practical session. Please ensure you copy and paste your code from each question into a document and that it is well formatted so that it can be easily marked after the practical. Submit both your document and **.c** file on VULA by the due date. You are also welcome to complete the coding during free period sessions in the Red Laboratory.

Please name your practical report as follows:

Prac1B-STUDENTNUMBER1

Call a teaching assistant or tutor if you need assistance at any stage during the practical session.

1B.2 Background: The compilation process

C was developed in the late 1960s and early 1970s in Bell Labs by Dennis Ritchie as a cross-platform general purpose programming language that allowed low level memory access and control. This meant that it could be used to program many types of processor architecture from microcontrollers to personal computers. It has subsequently been standardised by the *American National Standards Institute* (ANSI) and later the *International Standards Organisation* (ISO) which enables portability between devices. These standards define the language syntax as well as the standard libraries such as **stdint.h** and **stdio.h**. The latest standard is C11 or more formally ISO/IEC 9899:2011.

C is a *procedural* programming language. This means that it is made up of a set of *procedures* called functions/sub-routines. Each *function* contains a set of instructions that are run sequentially. A function can make calls to other functions from within it. The instructions in these functions will then run first, before the program returns to where it left off. A logical list of instructions and functions are saved as a *source file* [1]. A source file is a plain text (**.c**) file. This file contains a series of digital 8-bit bytes each representing a single ASCII[1] character (see Appendix B.3). However a processor can only process machine code (opcodes and memory/register addresses). There therefore needs to be a process which can convert the (**.c**) file into an executable *binary file* containing a sequential set of machine instructions. This is performed in stages by four programmes:

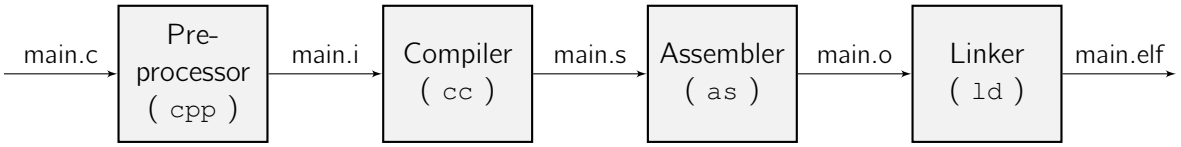


Figure 1B.1: A block diagram showing the compilation process. Adapted from [1].

- **Pre-processor** runs through the original text C program and implements all pre-processor commands (prefaced by # character). It produces a text-based C file with the **.i** file extension.
- **Compiler** converts the C instructions into assembly instructions. Each assembly instruction represents one machine-code instruction but in text form, making it easier for a programmer to understand, however it still needs to be further processed before it can be read by the target processor. This produces a text-based assembly file with the **.s** file extension.
- **Assembler** converts the the text-based assembly program into a binary *relocatable object file*. Each byte in this file encodes machine-code instructions, not ASCII characters as in the previous files. This file is *relocatable* as the final addresses for each section have not been set at this stage. This produces a binary object file with the **.o** file extension.
- **Linker** links all object files together and combines them into a single executable binary file, where the final absolute address for each section is assigned. This file can be run on the target. The function of the linker is to tie all of the object code files together to create one linked set of code. All cross references within the linker are checked and tied together. All of the symbolic addresses (labels, variable names) within your code are converted to absolute memory addresses and all C library references are tied into your code where they have been called. This requires that the linker knows several basic facts about your system. The linker needs to know where to place code in memory. The linker also needs to know where to place variables in memory. Finally the linker needs to know which files to link together and which libraries are to be used. It then produces a binary executable with the **.elf** or **.exe** file extension.

A *toolchain* is a collection of software "tools" and programs which are used to develop source code,

[1] The American Standard Code for Information Interchange is a standardised encoding procedure for text characters.

convert the code into the appropriate form and then debug, program and run the code on the target processor.

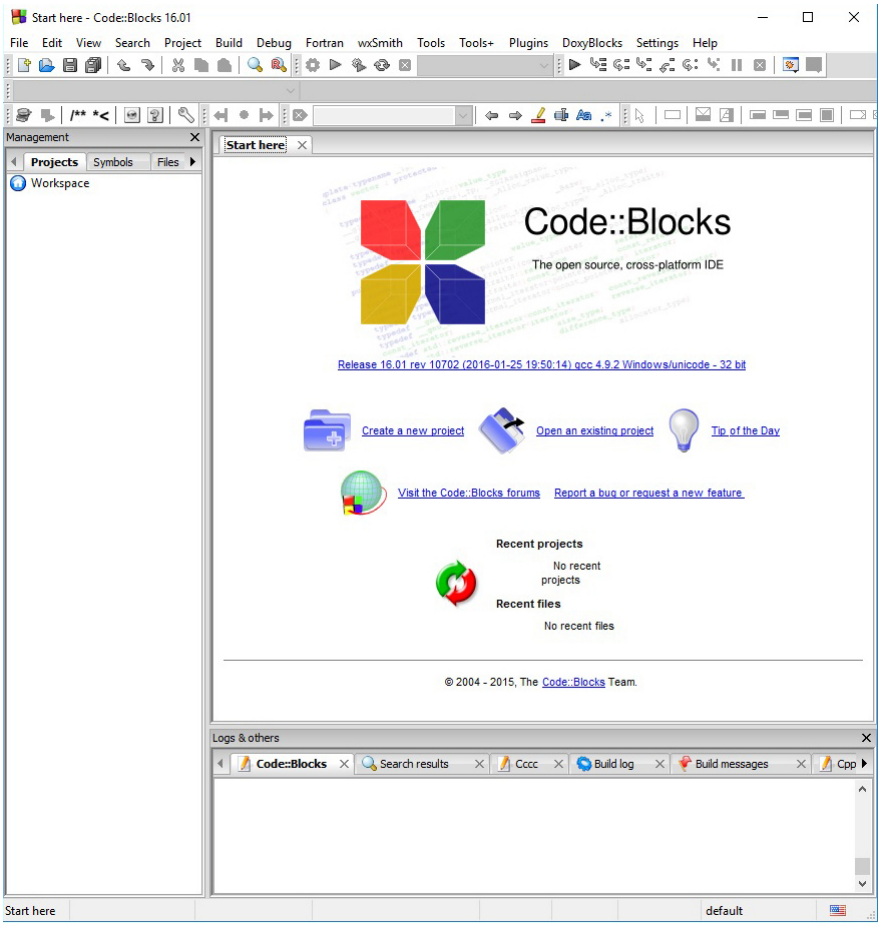
1B.3 Background: Integrated Development Environment (IDE)

An *Integrated Development Environment* (IDE) is a program used for editing, compiling, debugging and programming. We will use Code::Blocks IDE (cross platform IDE; **(www.codeblocks.org)**) to write C code and compile it for a PC in this practical. The toolchain is included in the IDE and we can run the complete compilation, linking and assembling process from one program[2]. We will start by creating a new console project in Codeblocks using the following process.

- Download the **ctemplate.c** file from VULA.

  VULA → **EEE2046F** → **Resources** → **Part B: C and microcontrollers** → **Code**.

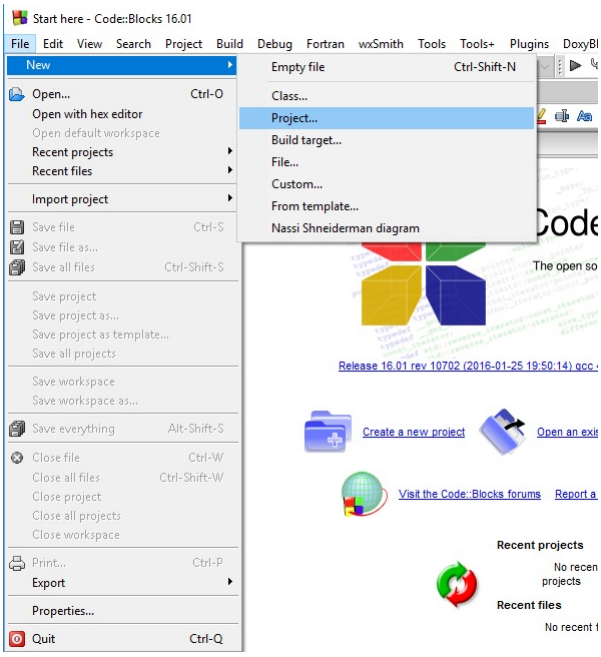
- Open Code::Blocks on your computer



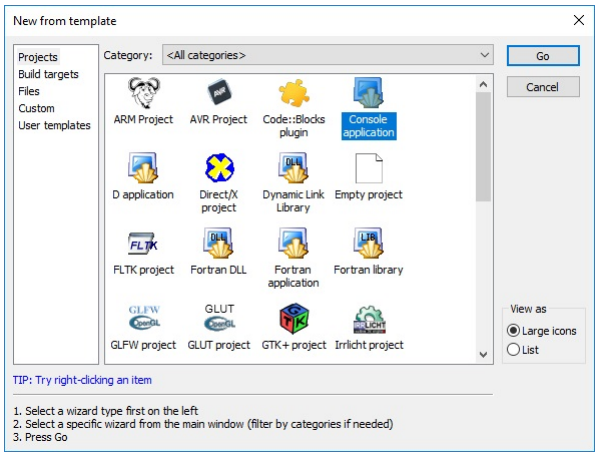
- Create a new project.

  File → **New** → **Project...**.

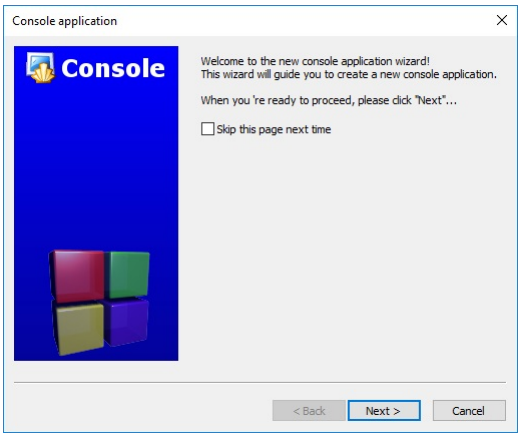
[2]Note if you prefer to write your code in a text editor and compile it from the command line or use an alternative IDE, you are welcome to do so and using Code::Blocks is not a requirement for the course.



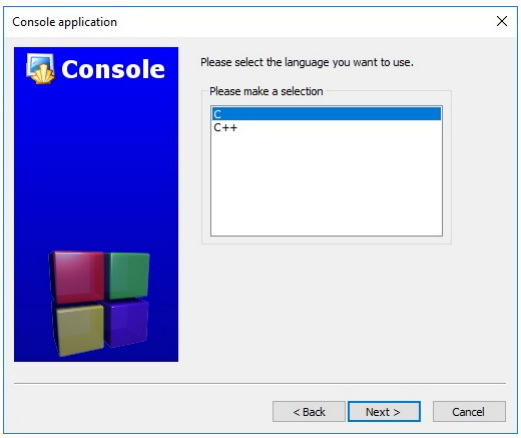
- Select **Console application** and click **Go**.



- Click **Next**.



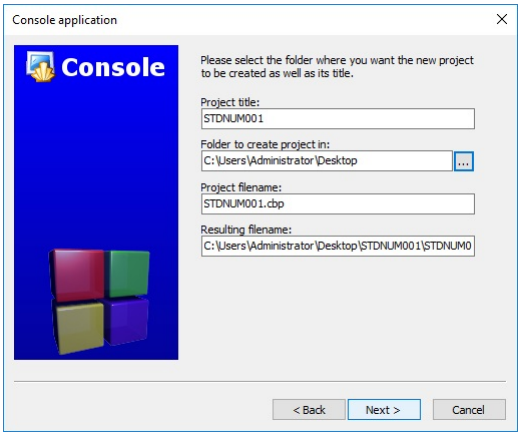
- Select **C** and click **Next**.



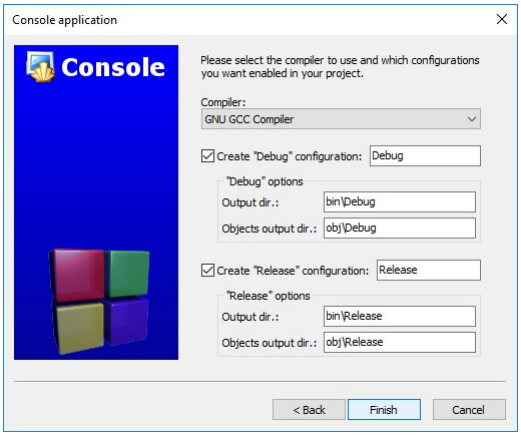
- Title the project:

  Prac1B-STUDENTNUMBER1

  and create the project folder on the desktop (it must be a local drive, not a network drive).



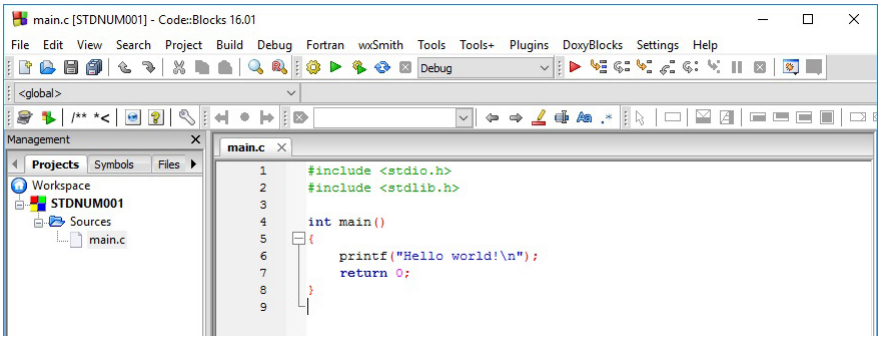
- Leave the compiler settings at their default values and click **Finish**.



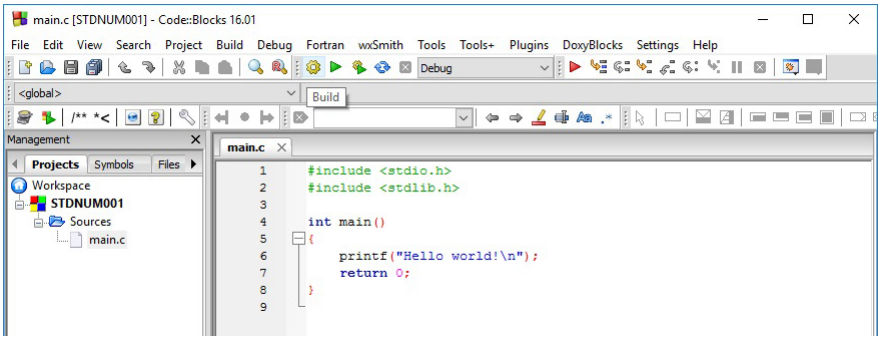
- Open **ctemplate.c** file and copy its contents to your **main.c** file.
- Modify the comments at the start of the file, including your name, the date etc.
- Type the following code in the main function (lines 6 and 7):

```
1  //=========================================================================
2  // MAIN FUNCTION
3  //-------------------------------------------------------------------------
4  int main(void)
5  {
6          printf("Hello world!");
7          return 0;
8  }// End of main
9  //*************************************************************************
10 //*                              END OF CODE                              *
11 //*************************************************************************
```

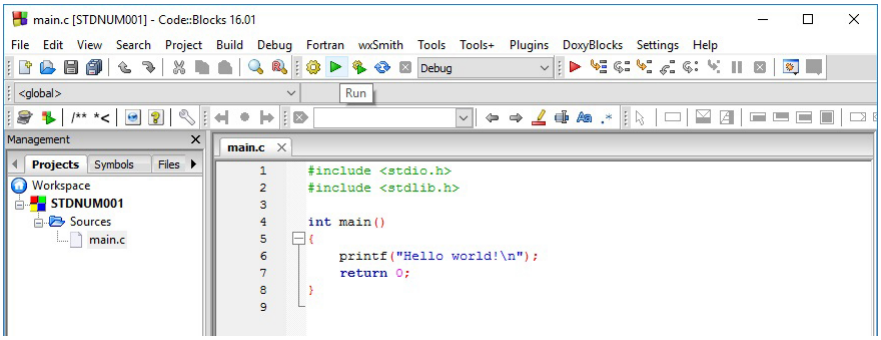
- Save your file (**Ctrl-S**).



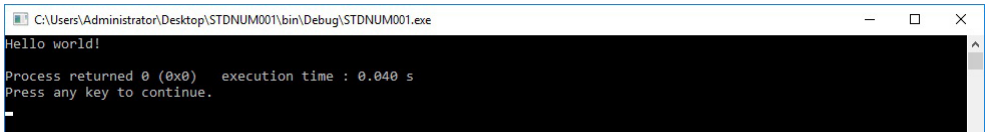
- Compile your code **(Build)**.



- If there are no errors, run your code **(Run)**.



- You should see a similar output on the terminal.



NOTE: Clear, concise and well commented coding is encouraged. Please ensure that you explain all steps in your code and follow good coding practice.

You are now ready to begin your task.

1B.4 Task

In this practical you will learn how to display values to the screen, read in values from the keyboard, create preprocessor constants, loops and functions using C. You will then use these skills to write a decimal to radix-n conversion program.

1B.4.1 The `printf` function

The **printf** function writes a 'string'[3] to the standard output stream (the screen in this case). This C function is part of the **stdio.h** library. A brief description of the function and how to use this it is given below.

```
int printf (const char *format, ... );
```

It can be used as follows (Modified from [4]):

```
/*===========================================================================
// printf format specifiers
//---------------------------------------------------------------------------
printf("Hello world!\n");
printf("Characters: %c %c \n",'a', 65);
printf("Decimals: %d %ld \n",1977, 650000L);
printf("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
printf("Floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, -3.1416);
printf("%s \n","Welcome to EEE2046F");
```

Outputs the following to the screen

```
Hello world!
Characters: a A
Decimals: 1977 650000
Some different radices: 100 64 144 0x64 0144
Floats: 3.14 +3e+000 3.141600E+000
Welcome to EEE2046F
```

For more detailed information on the function and all the format specifiers see Cplusplus reference site.

[3] Note C does not have a 'string' datatype. Stings are represented by an array of characters. If we want this to be displayed correctly, each character in the array must be the ASCII value for the character. The **printf** function converts all numerical values to their correct ASCII characters before incorporating them into the array, based on the value of the format specifier.

1B.4.2 The `scanf` function

The **scanf** function stores input from the standard input stream, in a specified format, to a location pointed to by a reference to a variable. This C function is part of the **stdio.h** library.

```
int scanf (const char *format, ... );
```

It can be used as follows[4]:

```
//=====================================================================
// scanf
//---------------------------------------------------------------------
int age;
char name[80];

printf("Enter your name: ");
scanf("%79s",name);
printf("Enter your age: ");
scanf("%d",&age);
```

Outputs the following to the screen

```
Enter your name: Robyn
Enter your age: 20
```

For more detailed information on the function and all the format specifiers see **Cplusplus** reference site.

We will now start our program design, which we are going to complete step by step.

1B.5 Questions and coding

In your **.c** file start to write the following program step by step. Compile and run your code after each step and make sure it behaves as expected, troubleshooting as you progress.

(a) Write a basic program, using the template, which displays the following on the screen. Copy this code and paste it in your report. **[3]**

```
*****************************
 DECIMAL TO RADIX-n converter
 Written by: Name
 Date: 2018
*****************************
```

(b) Create three constants called `TITLE`, `AUTHOR` and `YEAR` using the preprocessor command `#define`. Update your `printf` code from step one to use the information from these constant values. Copy this code and paste it in your report. **[3]**

[4] Note the `scanf` function, expects an address (or reference) of where the data are to be written in memory, as an argument after the format specifier. To access the address of a variable we use the unary `&` operator in front of the variable name. The name of an array is already a reference or address to the first value in the array and therefore does not need the `&` operator to extract the address.

(c) Now add code to your program that will prompt an external user to enter an integer decimal number and display it on the screen, as follows:

```
*****************************
 DECIMAL TO RADIX-n converter
 Written by: Name
 Date: 2018
*****************************
Enter a decimal number: 23
The number you have entered is 23
```

Copy this code and paste it in your report. **[3]**

(d) Add code to your program that will prompt an external user to enter an integer between 2 and 16, which will be the radix of the number system that we will convert our decimal value to, and display it on the screen as follows:

```
*****************************
 DECIMAL TO RADIX-n converter
 Written by: Name
 Date: 2018
*****************************
Enter a decimal number: 23
The decimal number you have entered is 23
Enter a radix for the converter between 2 and 16: 3
The radix you have entered is 3
```

Copy this code and paste it in your report. **[1]**

(e) Modify your program so that it, continuously prompts the user to enter a decimal number and radix and display it on the screen, until the user types in a number less than 0. After which the program should terminate and display the message `EXIT` on the screen.

```
*****************************
 DECIMAL TO RADIX-n converter
 Written by: Name
 Date: 2018
*****************************
Enter a decimal number: 23
The number you have entered is 23
Enter a decimal number: 1
Enter a radix for the converter between 2 and 16: 3
The radix you have entered is 3
Enter a decimal number: -1
EXIT
```

Copy this code and paste it in your report. **[5]**

(f) Now find the $\log_2 n$ of the entered decimal number and display the result to the screen[5].Copy this code and paste it in your report. **[2]**

(g) Now display the entered decimal number divided by the radix and display the integer result to the screen. Copy this code and paste it in your report. **[2]**

(h) Now display the remainder of the decimal number divided by radix and display the result to the screen. Copy this code and paste it in your report. **[1]**

The output from your code should now look like this:

[5]HINT: Include the **math.h** library and look up the function `log2()`

```
*****************************
 DECIMAL TO RADIX-n converter
 Written by: Name
 Date: 2018
*****************************
Enter a decimal number: 16
The number you have entered is 16
Enter a radix for the converter between 2 and 16: 2
The radix you have entered is 2
The log2 of the number is 4.00
The integer result of the number divided by 3 is 8
The remainder is 0
Enter a decimal number: -1
EXIT
```

We are now going to write a C program to convert an entered decimal number and display it as a radix-n value on the screen.

(i) Write out the step by step algorithm to convert an integer decimal number into any radix-n value (where $n \in \{2, 3, 4, ..., 16\}$) in your practical report. Pseudocode and flow diagrams are important and useful tools when design or presenting your code. **[5]**

(j) Create a function called `char* Dec2RadixN(int decValue, int radValue)` which takes in the decimal number and the radix value entered by the user and displays the equivalent radix-n number[6]. The output from your code should now look like this:

```
*****************************
 DECIMAL TO RADIX-n converter
 Written by: Name
 Date: 2018
*****************************
Enter a decimal number: 16
The number you have entered is 16
Enter a radix for the converter between 2 and 16: 2
The radix you have entered is 2
The log2 of the number is 4.00
The integer result of the number divided by 3 is 8
The remainder is 0
The radix-2 value is 10000
Enter a decimal number: -1
EXIT
```

Copy this code and paste it in your report. **[5]**

(k) Ensure that your **main.c** file is well commented and **copy the entire contents of the main.c file and paste it onto a new page in your report**. **[10]**

[6] Note that we follow the same single symbol convention as Hexadecimal for values above 9_{10}.

1B.6 Practical Submission

Submit your completed practical report on VULA as a **.pdf**[7] under the correct assignment. Show all your calculations.

Your document must be named as follows so that it is easily identifiable:

Prac1B-STUDENTNUMBER1.pdf

Additionally you must submit your **main.c** as a separate file on VULA. It must be renamed as follows so that it is easily identifiable:

Prac1B-STUDENTNUMBER1.c

Do **NOT** zip the files together but upload them individually to VULA. Ensure that your name and student number is clearly written in your practical report. Practical reports must be completed and submitted on VULA by 23h55 by the due date on the VULA calendar.

1B.7 Marks Breakdown

Marks	
Questions	
(a)	3
(b)	3
(c)	3
(d)	1
(e)	5
(f)	2
(g)	1
(h)	2
(i)	5
(j)	5
Code	10
Total	**40 Marks**

Up to 5 marks will be deducted for untidy reports or uncommented code. **5% will be deducted per day for late hand in's for up to one week, after which you will receive 0 and the practical report will no longer be accepted.** Please ensure that you submit the files in the correct place and read the instructions given on VULA with regards to file naming. **NO EMAILED** pracs will be accepted.

[7] **Note** if it is submitted as a **.doc/.docx** or equivalent format file, it will **NOT** be marked.