Software and Embedded System Lab 2 (ELEE08022)

Bit Operation & Dynamic Memory Allocation

1. Write a program that reads an **`unsigned`** decimal number (between 0 and 255) and prints out its logic codeword in *binary format*, with the Most Significant Bit (MSB) on the left and the Least Significant Bit (LSB) on the right. (Note: there is no **`printf`** format specifier for this - you will have to design a *simple* algorithm, perhaps including shift and mask operations, to achieve this).

2. Write a C program to read a 16-bit *hexadecimal* codeword and store it in a **`short`** integer named **`original`**. The program should then store the bits from variable **`original`** into a second variable **`reversed`**, in **`reverse order`**. Finally, the program should print the codeword stored in **`reversed`** in hexadecimal format. Thus, given an input of **`1F01`**, your program should produce an output of **`80F8`**.

Note: reading in and writing out an **`unsigned short`** in hexadecimal can be accomplished with the format specifier **`%hx`** in **`scanf()`** and **`printf()`**.

3. Write down dynamic memory allocation statements, which will return a pointer to the following:

an array of 25 characters
an array of 10 integers
an array of 10 doubles
a float
a student record structure
an array of 5 such structures

Note: use **`%p`** in **`printf( )`** to print address.
Finally release those memories using **`free()`**.

4. Modify the last example program in this session, keep using structure pointer and functions, but not use dynamic memory allocation. The structure template of student record should be defined as:

```
struct studrec        /* structure template for student record */
  {
    char name[30];                /* Name as character string */
    long matric;               /* Matriculation number as long */
    char addr[50];                  /* Term Address as string */
  };
```

After this exercise, you can appreciate the difference of using and not using dynamic memory allocation.

5. Design, implement and test a program, which can read a string of seven characters from the keyboard and print it to the screen.

There are many ways of approaching this problem. What I suggest on this occasion is to request memories using **`malloc()`** for two strings as input go along, then keep re-copying your growing string between two strings, at last print out one string out. Don't forget to **`free()`** any memory you no longer need, as you go along.