University of Nottingham
UK | CHINA | MALAYSIA

EEEE1042/1032 - **Practical 3**
Strings, variables, input/output.

This practical session gives you hands-on practice of the theories, concepts and codes taught in the lectures. It is best to work through the practical session while having the class notes beside you. You should submit your `.c` code to Moodle at the end of the session. Answer the questions in this practical **directly** in your `.c` file using comments and/or `printf()` to print to the screen.

For example, in your C code, include comments and printf()'s so I can tell which question your code is answering like this:

```
1 // Q1.1: What is the result when you add together a signed and an unsigned int?
  // Answer to 1.1: type in your answer here.
  printf("**************\nAnswer to 1.1\n***************\n");
```

Name your `.c` file as: `yourName_practical3_EEEE1042.c`. You can put all your answers into 1 huge `.c` file, or alternatively you can put each answer into separate `.c` files and name your files `yourName_practical3_EEEE1042.1.c` instead.

Now you know how to answer the questions and how to submit your work in properly named `.c` files to Moodle, let's move on to the main part of the practical session.

1 Chars, Strings and Pointers

In this section we will practice the concepts of strings and pointers learnt in class. Write a C-program following these instructions.

1. As we will use functions in the string library `#include<string.h>` as a second header file along with the usual `stdio.h` at the start of your program.

2. Declare and initialize `char alphabet[]` as a string (i.e. pointer to a char) consisting of the 26 letters of the alphabet a to z. Examples on how to do this are in the lecture notes.

3. Declare `s` as a pointer to a char as:

   ```
   2 char *s;
   ```

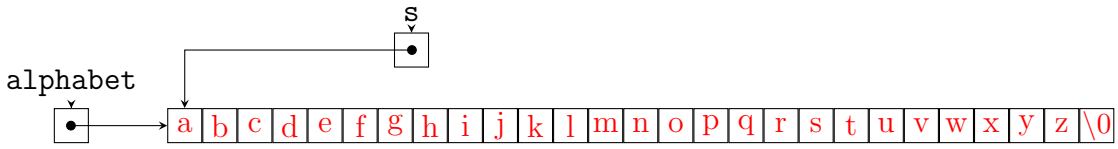
   Q1: What does `s` contain at this point (without any assignment?) Should you print out the string pointed to by `s`? Why or why not?

4. Print out `strlen` of `alphabet`. Note `strlen` has the following declaration in the string header file:

```
3 unsigned int strlen(const char *s);
```

   Meaning it takes a pointer to a char (i.e. a string) as input and it returns an unsigned int as output. `const` means the input string is not to be modified by the function, the function counts from the character pointed to by `*s` until it finds an end-of-string marker '\0' and returns the number of characters it counted.

5. Point `s` to the start of your alphabet (`s=alphabet`)and print out `strlen(s)`.



   Q2: What do you get and why?

6. Add one to s and printout `strlen(s)` again. print out `s` as a string with the `%s` conversion character.
   Q3: Explain why you get the observed result.

7. Point `s` to the the element `k` in the array `alphabet`. You use `alphabet[i]` to access the `i`'th element in the array `alphabet`. Use `&alphabet[i]` to get the **address** of the `i`'th element in the array `alphabet`. Print `strlen(s)` and `s` again.
   Q4: Explain again why you are getting what you observe.

2 Truth tables

In this section we will use the C Logical operators `&&`, `||` and ^ to print out AND, OR and XOR truth tables.

1. Declare two integers `x` and `y`.

2. Set `x=0` and `y=0`. Print out the value of `x&&y` using the `%d` conversion char.

3. Repeat when `x=1` and `y=0`; `x=0` and `y=1`; `x=1` and `y=1`, putting the results into a truth table that looks as follows:

```
     AND   | x=0 | x=1 |
=========================
    y=0    |     |     |
    y=1    |     |     |
```

   Use `printf` to print out the truth table in a nice visually pleasing format filling the appropriate values into the table.

4. Repeat the truth table, using OR: `||` in place of AND:`&&`.

5. Repeat again for XOR which uses $\wedge$.

3 Counting in Binary

In this section we will write a function that will convert decimal to binary for us and print out the binary number.

1. First setup your main program that will call a subfunction that you will declare and define. Declare an integer `x` and assign it a value in your `main` and call a function `printBinary(x);` The function `printBinary()` does not exist yet. We are going to create it now.

2. **Define** a new function definition **above** your `main` function (after your `#include` lines). This function is called `printBinary` and it takes an integer as input, and prints it to the screen. It returns nothing to calling environment (return type is void). The function **declaration** would look like this:

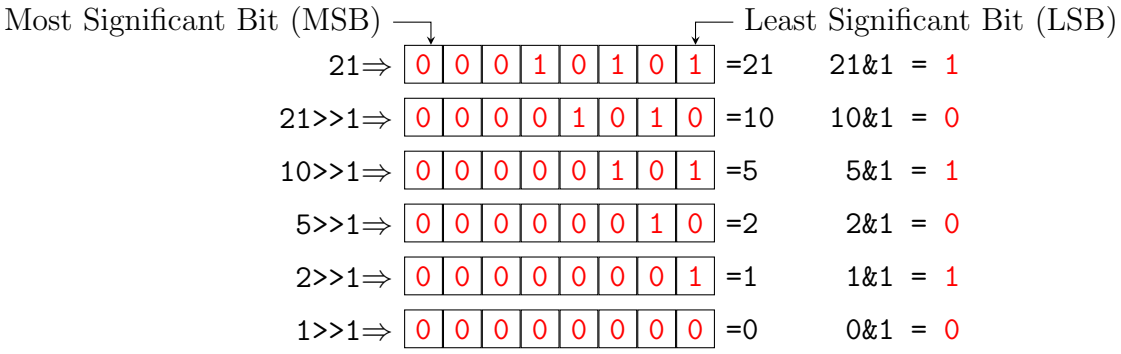
```
4 void printBinary(int x);
```

   but in this case we're going to be a little bit lazy and just **define** the function above, which simultaneously **declares** the function to the compiler. The function **definition** looks like this:

```
5 void printBinary(int x){ /*Function definition goes inside here*/ }
```

   the return type `void` just means that this function returns nothing to the calling environment. You should not put `return(0);` inside, as this is trying to return something.

3. Now we are going to work on creating (defining) our `printBinary` function. The input to this function is `x` which is an integer. The figure below suggests a method to convert a decimal number (`x=21`) into binary:



   To implement this algorithm, define an array:

```
6 char c[9];
```

   that will hold the output of `char`'s, an array of `'1'`s and `'0'`s. We will only be converting up to 8 bits, so there are 8 chars for our 8 bits and one char for the end of string. We will populate it from the right most bit as suggested by the figure above. But first let's initialize the last bit to be the end of string:

```
7 c[8]='\0';
```

   Remember that C counts arrays from `0` and `c[8]` is the 9th (and last) char in the array.

4. We will also want to make a copy of our input `x` because the algorithm is going to destroy `x` in the process and we will want to print out its value later together with its binary representation. So let's make a copy of it now:

```
8 int y=x;
```

5. Now we work on `c[7]` (the least significant bit or LSB) which according to the above figure should be `x&1`, which is the LSB of `x`. However remember that `x` is an `int` and `c` is a string that we are going to want to print later. We want our `char`'s to take on the values `'1'` or `'0'`, (not integers 1 or 0) which are actually ascii values of 48 for `'0'` and 49 for `'1'`. We don't have to remember that 48 corresponds to `'0'` and 49 corresponds to `'1'`. We can just use `'0'` and `'1'` as values for a `char`, C will store it internally in binary which can be output as the decimal values with `%d` if we wanted the numbers 48 and 49. But we do need to store `'0'` in `c[7]` or `'1'` in `c[7]` if `(x&1)==0` or `(x&1)==1` respectively. We can do this using the **ternary operator**:

```
9 c[7]=((x&1)==1 ? '1' : '0');
```

This is our ternary operator and it says: assign `'1'` in `c[7]` if the LSB of `x` is 1. Otherwise assign `'0'` to `c[7]`. Note we are assigning one of two `char`'s into `c[7]`, either `'0'` or `'1'`, which take on values of 48 or 49. This explains why we couldn't have done this: `c[7]=x&1` because these values would be 0 or 1, not `'0'` or `'1'`.

6. Now we need to change our value of `x` so that the entire set of bits shifts down. This is done with the right-shift operator:

```
10 x=x>>1;
```

7. And now we are setup ready to compute `c[6]` in the same way:

```
11 c[6]=((x&1)==1 ? '1' : '0');
   x=x>>1;
```

Copy, paste and edit these lines to fill the other elements of the array `c`.

8. Once we have filled our `c` array all the way down to `c[0]`, it contains an array of 8 binary digits `'1'` or `'0'`, terminated with the NULL character `'\0'`. It is ready to be printed as a string together with its decimal equivalent:

```
12 printf("%3d: %s\n",y,c);
```

after which we can successfuly return to the from the function to the calling environment.

9. Set up the `printBinary()` function as described above and try calling it with a few different decimal numbers as inputs and use your calculator to check that you are getting the correct 8-bit binary representation each time.

4 Input from command line, output to a file

In this section we will write a short program that takes input from the command line and redirects them into a file using `fopen()`, `fclose()` and `fprintf()`

1. Create your usual main function in a new C-program. Be sure to take in `int argc` and `char **argv` as inputs to main, as you will need `argv`.

2. Create a file pointer `FILE *f;` in your main.

3. Open a file `"myOutput.txt"` for writing using the file pointer `f`. The code for this is given in the Lecture3 class material in the Section 5 Input/Output. Be sure to open the file **for writing** using the character `"w"` as the second parameter to `fopen`. Note that the `fopen()` command with the `"w"` flag will overwrite the file if it already exists. Include the `fclose()` function and close the outer `if` statement bracket } as it is good practice to close brackets that you open and close files that you open immediately so you don't forget them later. You should be able to compile and execute the program after including the `fopen() fclose()` pair as shown in the Lecture3 notes.

4. Your subsequent code should go inside the `if` statement that `fopen`'d the file and **before** the `fclose()` statement so that you can write to your opened file. Write the first 3 inputs from the command line to the file. This can be done with code like this:

```
13 fprintf(f,"%s\n",argv[1]);
```

   and similarly for the second and third input arguments from the command line. Remember that `argv[0]` is the name of the calling function, so you should start saving from `argv[1]`.

5. Compile your program, but **do not** run it using the usual `make` because your program now needs 3 input arguments to work properly. Instead, compile it manually from the command line terminal using `gcc -o myfilename myfilename.c` (replace `myfilename` with your filename) and run it passing three inputs for example, if your program is called `practical3.4` type (note the $ is the command prompt, not to be typed):

```
14 $ ./practical3.4 my three inputs
   $ la
   -rwxrwxr-x 1 kheong kheong 17K Okt 18 11:47 practical3.4
   -rw-rw-r-- 1 kheong kheong 16 Okt 18 11:47 myOutput.txt
   $ cat myOutput.txt
```

   The command `cat` tells the shell to list the contents of the file to the screen.
   Q5: What output do you get from the `cat` command, explain what has happened.