Introduction to Computer Programming Using the C Programming Language
LABORATORY EXERCISE 1

OBJECTIVES
At the end of this lesson, you should be able to:
1. identify the different commands used in LINUX environment,
2. use the appropriate command/s for a given problem
3. implement a simple program and,
4. distinguish common errors in C programming.

LINUX PROGRAMMING ENVIRONMENT
Under *Linux* there are two interfaces:
- GUI – Graphical User Interface
  - point, click and drag
- CLI – Command Line Interface
  - type commands
  - command prompt(in Windows OS)/Terminal
  - **CASE SENSITIVE**

CLI File and Directory Commands
1. `cd`
   - The **cd** command changes directories.
   - To navigate into the root directory, type:
     - `cd /`
   - To navigate to your home directory, type:
     - `cd` or `cd ~`
   - To navigate up one directory level, type:
     - `cd ..`
   - To navigate to the previous directory (or back), type:
     - `cd -`
   - To navigate through multiple levels of directory at once, specify the full directory path. For example, type:
     - `cd /var/www`
2. `pwd`
   - The **pwd** command will show which directory you're located in.
3. `ls`
   - The **ls** command shows you the files in your current directory.
4. `cp`
   - The **cp** command makes a copy of a file for you.
5. `mv`
   - The **mv** command moves a file to a different location or will rename a file.
6. `rm`
   - The **rm** command to remove or delete a file in your directory
7. `mkdir`
   - The **mkdir** command will allow you to create directories.

Other Useful Things: *Save on Typing*
Here are some useful ways on pasting commands.

Up Arrow	Scrolls through the command you've entered previously.
Down Arrow	Takes you back to a more recent command.
Tab	It *autocompletes* any commands or filenames, if there's only one option, or else gives you a list of options.

EXERCISE:

1. In the home directory create a folder named CMSC11_<Section> (e.g. CMSC11_A1L)
2. Create a folder named <SURNAME>_EXER (e.g. BENNET_EXER) inside the CMSC11<Section> folder.
3. Create a folder named EXTRA inside the CMSC11_<Section> folder.
4. Create a text file named <initials>.txt inside the CMSC11_<Section> folder.
5. Moved <initials>.txt in the folder <SURNAME>EXER.
6. Delete the EXTRA folder.

A FIRST PROGRAM

```
/* Author: Maverick C. Crisostomo
   Date created: November 17, 2008
*/
#include <stdio.h>

main( )
{
       printf("\nHello World!\n");
}
```

- Use any text editor to type your first program. You may use kate, kwrite or gedit text editors.
- After typing the code, save the file in your own folder using the filename:

```
hello.c
```

- In the *Terminal*, make sure that you are in the directory where you have saved the file. Then, compile it. To compile, type:

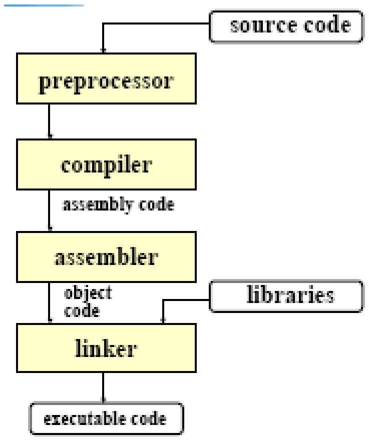
```
gcc -o hello hello.c
```

- Finally, execute the program simply by typing

```
./hello
```

- You should see the words "Hello World" printed out on the *Terminal.*

THE C COMPILATION MODEL
- The Preprocessor accepts source code as input and
  - removes comments
  - extends the code according to the preprocessor directives included in the source code (lines starting with #)
- The Compiler takes the output of the preprocessor and produces assembly code
- The Assembler takes the assembly code and produces machine code (or object code)
- The Linker takes the object code, joins it with other pieces of object code and libraries and produces code that can be executed



STRUCTURE of a C PROGRAM

- C program contains the following elements:
  - Preprocessor Commands
  - Type Definitions
  - Function Prototypes
  - Variables
  - Functions
- All programs must contain a single *main( )* function. All functions, including *main,* have the following format:

```
type function_name ( parameters ) {
       local variables
       statements
}
```

SECOND PROGRAM

```
#include <stdio.h>
#define pi 3.14159
main()
{
    char charName;
    char charMI='A';
    int intX, intY=10, intZ=10.5;
    float floatNum1=100.25;
    double doubleNum1=10.432432;

    printf("Enter 2 numbers(separated by space): ");
    scanf("%d %d", &intX, &intY);
    printf("intX=%d intY=%d intZ=%d", intX, intY, intZ);

    getchar();
    printf("\n\nEnter your firstname: ");
    scanf("%c", &charName);
    printf("charName=%c charMI=%c", charName, charMI);

    printf("\n\ndoubleNum1=%lf floatNum1=%f
",doubleNum1, floatNum1);
    printf("\npi = %f", pi);
}
/*end of program*/
```

DATA TYPES

- ***VARIABLES***
  - *C has the following data types*
  - *Every variable name must start with a letter; the rest of the name can consist of letters, numbers and underscore characters.*
  - *C recognizes upper and lower case characters as being different.*
  - *You cannot use any of C's keywords like main, while, switch, etc as variable names.*
  - *[Assign: Look for the range of values of each of the data type given below.]*

Type	Use
char	characters
int	integers
float	real numbers
double	large real numbers

- ***CONSTANTS***
  - one can introduce symbolic constants using #define, for example:

```
#define pi 3.14159
```

COMMON C PROGRAMMING ERRORS

- Forgetting to put an ampersand (&) on arguments
  - causes SEGMENTATION FAULT
  - scanf() must have the address of the variable to store input into. This means that often the ampersand address operator is required to compute the addresses. Here's an example:

```
int x;

scanf("%d", x); /*it should be &x*/
```

- Missing operand and using the wrong format for operand
  - C compilers do *not* check that the correct format is used for arguments of a scanf() and printf() call. The most common errors are incompatibilities in the file format used and the variable associated to it.

```
int x;

scanf("%c %d", &x);
```

- Missing closing and terminating characters
  - causes SYNTAX error
  - Omitting a semicolon or a closing brace.
  - Omitting quotation character.

```
#include <stdio.h>

main()
{
   int x;
   float y

   printf("enter an integer: ");
   scanf("%d", &x);
   printf("enter a real number: ";
   scanf("%f, &y);

/*end of program*/
```

- Undeclared variables
  - A variable should be declared before it can be used. The compiler should know the type of the data that can be stored in the variable.

```
main()
{
       int x;

       scanf("%d", &y);
}
```

- Using a forward slash when a backslash is required (for example, substituting "/n" for "\n.")

PRACTICE EXERCISES

- Write a program to display your full name on the monitor.
- Modify the program to display your address and cellphone number on separate lines by adding two additional **printf()** statements.
- Declare an integer variable ***age*** and use this to store your age. Use **scanf()** to get the input from the user. Then, output the age.