BIRLA INSTTUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJ.)
Second Semester 2019-20 CS F111 Computer Programming

LAB SESSION #3
(Getting started with C programming)

Working on the *Prithvi* server

Starting this lab, you shall now work in your own personal Linux account in the "Prithvi" server. Your login name is your complete BITS ID number. For instance, if your ID number is 2019A2PS0076P, then your login name would be: **2019A2PS0076P**. The default password is the same.

To login to the Prithvi server, type the following command on your Linux terminal:

ssh 2019A2PS0076P@172.24.16.31

Note that you should enter your own login name. When prompted for the password, enter your password (note that this will not be visible as you type it).

Next, type the following command on the shell prompt: **passwd**

Follow the instructions to change your password to *something that you can remember*. Also note it down for your future reference.

(Note: Those who are **not able to** log in, please send an **email to the Instructor-in-Charge by Saturday 01:00PM (01-02-2020). After the deadline** any request regarding login and password **would not be entertained**. Till then, work on the local machine in IPC.)

Writing C programs

C programs is a collection of instructions written in C language that performs a specific task when executed by a computer. Whenever you write a c program in a file, save the file in .c format.

1. Type the following C program in a file named **lab3_1.c** using the vi editor:

```
#include <stdio.h> /*Preprocessor directive to include header file*/
    int main()      /* main() is the entry point of the program */
    {
      printf("This is my first C programming lab");
      printf("This lab is based on understanding the concepts of printf and scanf");
    }
```

Use the following **gcc** command to get an executable of the above code in Linux.
$ gcc lab3_1.c

If there were no errors in the entire process, a file called **a.out** would be created in the same directory. You can now run the program by typing**./a.out** at the shell prompt.

There are four main stages through which a source code is passed in order to finally get a runnable executable. They are: *pre-processing*, *compilation*, *assembly* and *linking*. By invoking **gcc** command, all these steps are accomplished, and you get the resultant executable in **a.out**. You can stop this pipeline at a specific stage by giving specific options for **gcc**.

For instance, try **gcc –S lab3_1.c** for just invoking the assembler. The assembly code for the C program is now generated and stored in **lab3_1.s** that you can inspect using vi. You may not understand the statements, but you should learn to identify how an assembly program looks like.

Instead of having the executable stored in a.out, you can also mention the destination file of the executable by using the **–o** option with **gcc**. That is, typing **gcc –o exe_1 lab3_1.c** will not produce the default file a.out, but will store the executable in the file **exe_1** which can be run by typing **./exe_1** at the shell prompt.

2. Type the following C program in a file named **lab3_2.c** using the vi editor:
(The line numbers given here are not to be typed by you! You can use the **:set nu** option in *vi* editor to see the line numbers automatically on the screen.)

```
1 #include<stdio.h>
2 int main()
3 {
4 int x, y;
5  float a,b;
6  printf("Enter value of a: ");
7  scanf("%f",&a);  // Reading user input for the variable a
8  printf("Enter the value of b: ");
9  scanf("%f",&b);  // Reading user input for the variable b
10 prod= a * b;
11 printf("Enter value of x and y");
12 scanf("%d%d",&x, &y);  // Reading values of two variables using single
 scanf
13 printf("Product of %f and %f is: %f\n", a, b, prod);
14.printf("Product of %d and %d is: %d\n", x, y, x*y); //printing the product
 of two numbers without storing the result in any variable
15  return 0;
16 }
```

In the above code, scanf() is a *function* being used to accept the input from the user through keyboard and printf() is another function being used to print out the output. scanf() has two parts; format specifier "%f", and a variable prefixed with & sign. The & (ampersand) prefixed to the variable fetches the address of the location where the variable is stored in memory.

Format Specifier: Specifies the scanf(), which type of data – integer, character, floating pointnumber or double-precision floating point number it should accept. The following characters listed, after the % character, are valid to be used in scanf():
%d is for int; %f is for float; %c is for char; and %lf for double.

3. Type the following code in your system and observe the format of the output printed by various printf statement in the program

```
#include <stdio.h>
int main()
{
   char ch = 'A';    char str[20] = "Computer";
   float flt = 10.234;    int no = 150;   double dbl = 20.123456;
   printf("Character is %c \n", ch);
   printf("String is %s \n" , str);
   printf("Float value is %f \n", flt);
   printf("Integer value is %d\n" , no);
   printf("Double value is %lf \n", dbl);
   printf("Octal value is %o \n", no);
   printf("Hexadecimal value is %x \n", no);
   return 0;
}
```

4. Make the following changes in the code written in the file lab3_2.c and observe the effect of each change on the compilation process:

a) Delete the x variable present in the fourth line of the code and see what the compiler does when you forget to declare a variable
b) Delete a semicolon and see what happens.
c) Leave out one of the braces
d) Remove one of the parentheses next to the main function

By simulating errors like these, you can learn about different compiler errors, and that will make your typos easier to find when you make them for real.

5. List all the errors you might get on compilation of the following code. Also, correct the errors.

```
int main()
{
   char ch = 'A';
   float flt 10.234;
   int no = 150
   printf("Character is %c \n", ch);
   scanf(%c, &ch);
   printf("Character is %c \n", ch);
   printf("Float value is %f \n", flt);
   scanf("%f", flt);
   printf("Float value is %f \n", flt);
   printf("Integer value is %d\n" , no);
   scanf("%d", &n);
   printf("Integer value is %d\n" , no);
   return 0;
}
```

6. Now that you know how variables are declared and which type of data requires what format specifier, fill the blank spaces in the following code with appropriate format specifier. Save the complete code in a file named lab3_6.c and execute it. Observe the behaviour of the presence of the newline character (\n) in your code.

```
#include <stdio.h>
int main()
{
   int c;
   float f;
   printf("Enter any integer");   \\note the absence of \n. Observe the effect of
the presence of  \n inside a printf statement
   scanf("_____", &c);
   printf("Entered integer is _________ \n", );
   printf("Enter any decimal value\n");
   scanf("___________", &f);
   printf("Entered value is ________ \n", f);
}
```

7. Now that you have learned input and output functions (printf and scanf) and a little bit about data types (int, float, char), you are ready to write your own C program that performs basic mathematical operations. Write a C code that takes two integrs from the user as input, calculates their sum and difference, and displays the answers on the terminal.

8. Generally, C does not provide nesting of comments, although many compilers provide an option for this. Try the following line in your program, use gcc, and see what happens:

```
This is an attempt /* to nest */ a comment. */
```

Also check to see the following alternative style of commenting in your program.

```
//This is a single line comment
```