BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJ.)

Second Semester 2017-18 CS F111 Computer Programming

LABORATORY SESSION #4

(Writing simple C programs)

In this lab, you will learn how to write, compile and execute a C program. You will also learn about some syntactical elements of C by trying out the exercises.

1. Type the following C program in a file named **lab4_sum.c** using the vi editor:

```
#include <stdio.h> /*Preprocessor directive to include a header file*/
int main()         /* main() is the entry point of the program */
{
   int a,b,sum; /* a, b and sum are declared as integer variables */
   a = 10;      /* Assigning values to the variables */
   b = 20;
   sum = a + b; /* storing the sum of the two numbers in sum */
   printf("The addition result is %d.\n",sum);
   return 0;      /* main() function returns with a value 0 */
}
```

   Use the **gcc** command to get an executable of the above code in Linux using **gcc**.

   $ gcc lab4_sum.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 as follows: **$./a.out**

   There are four main stages through which a source code passes in order to finally become an 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 the entire pipeline by typing various options for **gcc**.

   For instance, try **gcc -S lab4_sum.c** for just invoking the assembler. The assembly code for the C program is now generated and stored in **lab4_sum.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.

2. Take the above C program, and copy it without the .c extension (Hint: use the **cp** command on Linux). Now try compiling this copy. Some C compilers will complain; others will not. On Unix systems, the complaint may be quite cryptic, with words such as bad magic number or unable to process using elf libraries. What happens on your system?

3. 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

4. Every language has *keywords* and *identifiers*, which are only understood by its compiler. Keywords are predefined reserved words, which possess special meaning, e.g., `int`, `return`. An identifier is a unique name given to a particular variable, function or label in the program. Which of the following are not valid C identifiers and why?

```
3id     yes    o_no_o_no     00_go     int      star*it
_i_am_gr8      one_i_aren”t        me_to-2      xYshouldI
```

You will declare variables of each of these names and compile your program to check which of these are valid. Also find out the list of 32 keywords that are used in C.

5. Type the following C program in a file named **`lab4_product.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  float a, b, prod;
 5  printf(“Enter value of a”);
 6  scanf(“%f”,&a);        // Reading user input for the variable a
 7  printf(“Enter the value of b”);
 8  scanf(“%f”,&b);        // Reading user input for the variable b
 9  prod = a * b;
10  printf(“Product of %f and %f is: %f\n”, a, b, prod);
11  return 0;
12 }
```

In the above code, `scanf()` is used to accept the input from the user through keyboard and `printf()` is 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. You will learn more about this in later classes – don't worry for now. But note that the `printf()` does not use & to print the value of the variable.

Format Specifier: Specifies the `scanf()`, which type of data – integer, character, floating point number or double-precision floating point number it should accept. The following characters, after the % character, in a `scanf()` argument, have the following effect.
`%d` is for `int`; `%f` is for `float`; `%c` is for `char`; and `%lf` for `double`.

Now, modify the `printf()` statement by typing the following (line #10):

```
printf(“Product of %.2f and %.2f is: %.4f\n”, a, b, prod);
```

You now see that the same numbers are printed differently to give better-looking output.

6. Let us see how good you are in “reverse engineering”! You can inspect the output of a program that has been written by us by copying `/home/share/hailstones` to your current directory and then executing it by typing `./hailstones`. Based on the output, can you infer what the program does? At this stage, you are not expected to write the program, but you should be able to draw the flowchart that corresponds to it. Try it!

7. Write a C program which, takes two floating point numbers from the user as input, calculates and displays their sum, product, quotient and difference. Name your program as **`lab4_basicMathOperations.c`**.

8. Based on what you know about for loop, write a C program to print the math tables for the given number the user inputs between 2 and 20. For example, if the user inputs 4, four tables must be printed as follows:

```
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
....
....
....
4 x 10 = 40
```

9. Write a C program to calculate the total distance travelled by a vehicle in "t" seconds, given by: $d = ut + 1/2\ (at^2)$. Get user input for u, a and t. Output the value of d.

10. This exercise is meant for you to explore and learn more about printf() format specifiers. Try each of these and infer what is meaning:

int x = 12; printf("%5d", x);	int x=12; printf("%-5d", x);	float x = 234.5678; printf("%-8.2f", x);
float x = 234.5678; printf("%+8.2f",x);	float x = 234.5678; printf("%+-8.2f",x);	char ch = 'Y'; printf("%c", ch);