BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJ.)
CS F111 Computer Programming

LAB SESSION #4

(More C Programming and usage of `<math.h>` library)

Create a directory "**lab4**" inside "**myprogs**" directory that you have created in the last week's lab session. This week, we will write a bit more advanced C programs. We will also learn how to use `<math.h>` library for evaluating various mathematical expressions. You shall need to use the basics that were covered in the previous lab session.

Let us write a few more C programs using the fundamentals covered in the previous lab session.

1. In this question, you will learn more about data types and format specifiers. Copy the following program in "**lab4_sizes.c**", and then compile and execute it.

```
#include<stdio.h>
int main()
{
float f;
printf("Sizeof (char) = %lu bytes\n", sizeof(char)); // datatype
printf("Sizeof (short)= %lubytes\n", sizeof(short));
printf("Sizeof (int)= %lu bytes\n", sizeof(int));
printf("Sizeof (long)= %lu bytes\n", sizeof(long));
printf("Sizeof (float)= %lu bytes\n", sizeof(f)); // variable
printf("Sizeof (double)= %lu bytes\n", sizeof(double));
printf("Sizeof (1.55)= %lu bytes\n", sizeof(1.55)); // constant
printf("Sizeof (1.55L)= %lu bytes\n", sizeof(1.55L));
printf("Sizeof (str)= %lu bytes\n", sizeof("Hello")); // string
return 0;
}
```

Please note the use of **`sizeof()`** operator. ***It is used to find the size of any datatype or variable or constant in C***. It essentially returns the number of bytes occupied in the memory. For example, **`sizeof(int)`** would return 4 on a gcc compiler, as any integer variable would occupy 4 bytes. Similarly, **`sizeof(char)`** would return 1. Also note that the values returned by the **`sizeof()`** are of unsigned long data type. That's why we use **`%lu`** as the format specifier while printing the values returned by the **`sizeof()`** operator.

The **Qualifiers** are the keywords that when applied to the data types change their meaning. The ***short*** and ***long*** are size qualifiers; the ***signed*** and ***unsigned*** are sign qualifiers. Size Qualifiers are prefixed to the primary data types to increase or decrease the space allocated to the variable. The point to be noted is that the size allocated to a data type depends on the compiler and the machine on which the compiler is installed. ***short*** is used to allocate less memory than the usual size, and similarly ***long*** is used for allocating more memory than the usual size for the datatype.

For example, let us say that our compiler supports 4 –byte integer variables. If we declare a variable ***int a* = *14***, the compiler will **allocate 4 bytes** to the variable ***a***. If we declare a variable with ***short int a* = *30***, the compiler will allocate only **2 bytes**. Similarly, if we de

C also allows you to use the following syntax to declare a short integer variable: ***short a* = *30***. In this case, short is treated as a datatype. Using short as a datatype vs a qualifier makes no difference except for the syntax **in the case of int variables only**. Both can be interchangeably used. The same thing applies to long as well. For all other variable types, short and long can be used as qualifiers only.

Sign Qualifier is used to specify the signed nature of the integer. Declaring an integer variable means it can store a positive number or a negative number. By default, integer variable declaration int a = 14 means the variable can store *positive* or *negative* numbers. On the other hand, *we can use the unsigned qualifier to allow only positive numbers to be stored*.

Now let us do the following exercises:

a. Find out the sizes of data types when prefixed with the keywords signed and unsigned.
b. Now, try various combinations of qualifiers: (**short and long**) with the keywords: **unsigned** and **signed**. You should try checking which of these combinations the compiler accepts, and which the compiler doesn't accept. For example, **long unsigned int** is valid, whereas **long unsigned double** is not.

Let us now move to the second program for today's lab: Swapping the values of two integer numbers entered by the user.

```
2. #include<stdio.h>
   int main(){
      int a, b, c;
      printf("Enter the first number: ");
      scanf("%d", &a);
      printf("Enter the second number: ");
      scanf("%d", &b);
      c = a;
      a = b;
      b = c;
      printf("Value of a = %d, b =%d\n", a ,b);
      return 0;
   }
```

Try running the above program and understand how it is swapping the values of two integers. Can we interchange the lines 9 & 10 in the code? Try and understand what would happen if we do so.

3. It is also possible to swap two numbers without using a third variable and only with arithmetic operations. Check out this link to know more.
https://www.geeksforgeeks.org/swap-two-numbers-without-using-temporary-variable/
Implement it on your computer.

Type casting
Converting one data type into another is known as type casting or type-conversion. If you want to store a 'long' value into a simple integer, then you can type cast 'long' to 'int'. You can convert the values from one type to the other explicitly using the cast operator as follows.

<type_name> expression

Refer to the slides on Module 5 to know more about typecasting in C.

Example 1: we are converting a double value into an int.

```
double d = 75.25;
int i;
i = (int)d;

// Note that 'i' will have the value 75 as it can't store the
fraction part
```

Example 2: Consider the following example where the cast operator causes the division of one integer variable by another, to be performed as a floating-point operation.

```
#include <stdio.h>

int main() {

   int sum = 17, count = 5;
   double mean;

   mean = (double) sum / count;
   printf("Value of mean : %f\n", mean );
   return 0;
}
```

4. Write a C program, "**ascii_test.c**", which takes two characters as input and returns the sum of their ASCII values as output. For instance, input is A and B, the output should be 131 (sum of the ASCII of A and B). [**Hint:** Use explicit typecast to convert character to integer values. **End of Hint**]

Using <math.h> library

C supports extensive mathematics operations using various functions implemented in a library called <math.h>. We will learn how to use this library for evaluating various mathematical expressions. Your program should start with importing this library using #include <math.h>. And then we can call the functions that are available in it. A sample program is shown below:

myFirstMathProg.c

```
#include <stdio.h>
#include <math.h>
```

```
int main()
{
      int a = 4;
      double b = sqrt((double)a);
      /* Computes square root of "a" after converting it into a double
      variable */
      printf("Printing the value of square root of a: %lf \n", b);
      double angle = 2.3; // angle in radians
      double c = sin(angle);
      printf("Sin %lf is: %lf \n", angle, c);
      return 0;
}
```

This program computes square root of a number using `sqrt()` function and sin value of an angle in radians using `sin()` function. Both these functions are available in `math.h` library. Both of them take double values as input and return double values as output which is to be captured in a separate variable. We will study more about *functions* in subsequent classes and lab sessions.

One thing to be noted here is that while compiling any C program that uses `math.h` library, you must add `-lm` flag as shown below:

```
gcc myFirstMathProg.c -o math_exe -lm
./math_exe // runs the program
```

Without this flag, the compiler will throw an error. Now it is an exercise for you to figure out what is `-lm` and why do you need it. Please take help of GOOGLE!

The complete list of functions available in `math.h` library is as follows:

Function	**Description**
floor( )	This function returns the nearest integer which is less than or equal to the argument passed to this function.
round( )	This function returns the nearest integer value of the float/double/long double argument passed to this function. If decimal value is from ".1 to .5", it returns integer value less than the argument. If decimal value is from ".6 to .9", it returns the integer value greater than the argument.
ceil( )	This function returns nearest integer value which is greater than or equal to the argument passed to this function.
sin( )	This function is used to calculate sine value.
cos( )	This function is used to calculate cosine.
cosh( )	This function is used to calculate hyperbolic cosine.
exp( )	This function is used to calculate the exponential "e" to the x^{th} power.
tan( )	This function is used to calculate tangent.

tanh( )	This function is used to calculate hyperbolic tangent.
sinh( )	This function is used to calculate hyperbolic sine.
log( )	This function is used to calculates natural logarithm.
log10( )	This function is used to calculates base 10 logarithm.
sqrt( )	This function is used to find square root of the argument passed to this function.
pow( )	This is used to find the power of the given number.
trunc( )	This function truncates the decimal value from floating point value and returns integer value.

Now, let us try to write some programs using `math.h` library.

5. Write a C program to calculate x raised to the power n (x^n).

6. As you know, the roots x1 and x2 of a quadratic equation $ax^2 + bx + c = 0$ are calculated by:

$$x1 = (-b + \sqrt{(b^2 - 4ac)})/2a,$$

$$x2 = (-b - \sqrt{(b^2 - 4ac)})/2a$$

Write a C program named “**quadroots.c**”, which should take a, b and c as inputs, and output the values of x1 and x2.

Additional Practice Exercises

1. Write a C program to enter P, T, R and calculate Compound Interest and show it to the user.

   Hint:

   $A = P(1+(R/100))^t$

A	=	final amount
P	=	initial principal balance
R	=	interest rate
t	=	number of years

2. Write a C program in “**math_ops.c**” that evaluates and prints the values of the following arithmetic expressions. Here *x* and *y* are floating point numbers to be taken as input from the user. The value of *pi* is 3.142. Use *exp, sin, cos* and *tan* functions from math.h library. It is now your task to figure out how to use the above functions, what is their syntax. Take help of GOOGLE!

$$\text{expr1} = \frac{e^x \sin 60^\circ + 5.6\times10^{-5}}{3\cos 30^\circ}$$

$$\text{expr2} = \sin\left(\frac{\tan^{-1} 0.33 + \pi}{2y}\right)$$

3. Explain why the following code prints the largest integral value on your system:

   ```
   unsigned long long val = -1;
   printf("The biggest integer value: %llu\n", val);
   ```

   Note the conversion specifier `u` (for unsigned integers) preceded by the length modifier `ll` (ell-ell is the pronunciation) used to print the value stored in val. Explore other length modifiers and conversion specifiers by reading the online manual for `printf()`, by typing the following command at the Linux shell prompt: `man 3 printf`.