# Laboratory 1

Title of the Laboratory Exercise: Introduction to C programming

1. Introduction and Purpose of Experiment

Students get familiar with the data types and local variables and random number  
generation. Basic concepts such as data types and local variables are part and parcel of  
almost all the c programs. Hence sound knowledge is most essential in this regard. Also, the  
random number generation essential for many applications, for ex. rolling a dice for many in  
gaming applications such as backgammon which requires a random number generation from  
1 to 6.

1. Aim and Objectives

Aim

* To design and develop a C programs using Data types, local variables and Random  
  number generation to demonstrate the use and significate of the same in programming.

Objectives

At the end of this lab, the student will be able to

* Use variables of the basic data types with proper declarations
* Read and validate the input data
* Generate random numbers for any application

1. Experimental Procedure
   * 1. Analyse the problem statement
     2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     3. Implement the algorithm in C language
     4. Compile the C program
     5. Test the implemented program
     6. Document the Results
     7. Analyse and discuss the outcomes of your experiment
2. Questions

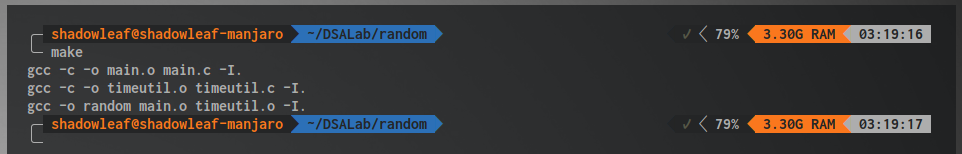
Demonstrate the use of data types, local variables and Random numbers by designing appropriate algorithms for the below problems. Tabulate the output for various inputs and verify against expected values. Analyse the efficiency of the algorithm. Describe your learning along with the limitations of overall approach if any. Suggest how these can be overcome.

* + - 1. Write a C program to illustrate random number generation. Modify the program to generate a random number between 75 to 85
      2. Write a C program to find sum of n elements, allocate memory dynamically using malloc() and calloc() function. Modify the program include both the allocation strategies in a single program.
      3. Combine both random number generation and memory allocation in a single program to demonstrate the allocation of random number of memory blocks.

1. Calculations/Computations/Algorithms
2. Presentation of Results



*Figure 1.1: Source Code*



*Figure 1.2 : Build Successful*



*Figure 1.3 : Run Succesful*

1. Analysis and Discussions

rand() is an library function that is defined in rand.c as

/\* Return a random integer between 0 and RAND\_MAX. \*/

int

rand (void)

{

return (int) \_\_random ();

}

If we take a look at how \_\_random works

/\* If we are using the trivial TYPE\_0 R.N.G., just do the old linear congruential bit. Otherwise, we do our fancy trinomial stuff, which is the same in all the other cases due to all the global variables that have been set up. The basic operation is to add the number at the rear pointer into the one at the front pointer. Then both pointers are advanced to the next location cyclically in the table. The value returned is the sum generated, reduced to 31 bits by throwing away the "least random" low bit. Note: The code takes advantage of the fact that both the front and rear pointers can't wrap on the same call by not testing the rear pointer if the front one has wrapped. Returns a 31-bit random number. \*/

As the source-code of \_\_random from glibc states, it uses Linear Congruential Generator to generate the pseudo-random numbers, to generate a different set of numbers at every run event, the algorithm is seeded with a seeding value, and here we take the system time to do so.

The generator is defined by:

Where

1. Conclusions
2. Comments

1. Limitations of Experiments

2. Limitations of Results

3. Learning happened

4. Recommendations