Name: SATYAJIT GHANA **Roll Number**: 17ETCS002159

Laboratory 1

Title of the Laboratory Exercise: Random number generation

1. Introduction and Purpose of Experiment

   Random Number generation is a pretty commonly used algorithm in computing, it’s used in large scales such as the monte-carlo simulation, and it’s also used in games to bring in the element of unpredictability. Random draws are often used to make a decision where no rational or fair basis exists for making a deterministic decision, or to make unpredictable moves.

2. Aim and Objectives

   Aim

   To design and develop a C program for 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

   - Generate random numbers for any application

1. Pseudo Code

```
Get min, max, n
Seed random with current system time
Set i = 0
While i < n
    Print rand()%(max - min + 1) + min
    Set i = i + 1
Stop
```

The generator is defined by:

$$X_{n+1} = (aX_n + c) \bmod m$$

Where

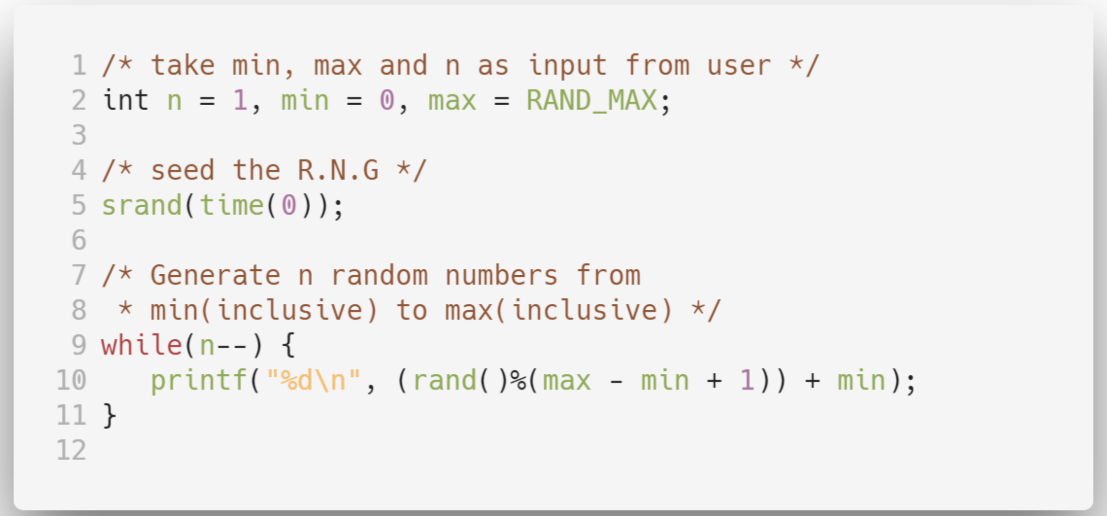
$m, 0 < m - modulus$

$a, 0 < a < m - multiplier$

$c, 0 \leq c < m - increment$

$X_0, 0 \leq X_0 < m - \boldsymbol{the\ seed}$

2. Implement C Code



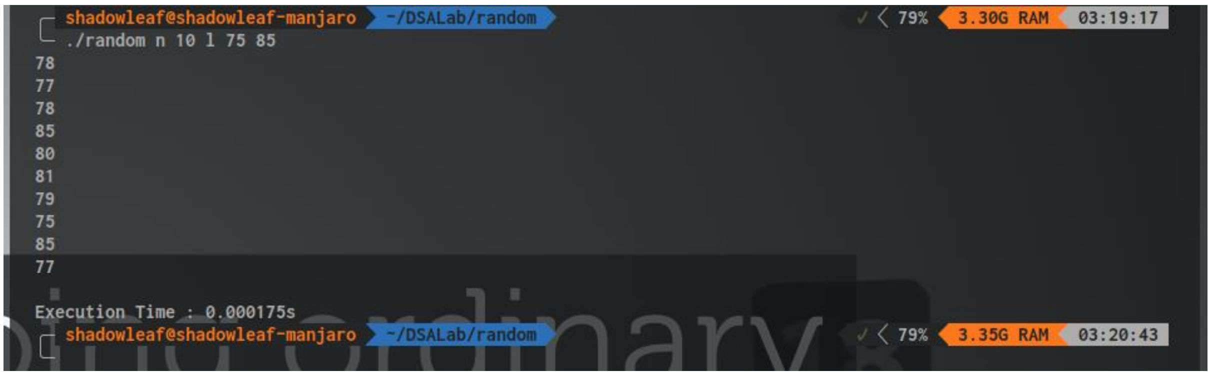
```c
/* take min, max and n as input from user */
int n = 1, min = 0, max = RAND_MAX;

/* seed the R.N.G */
srand(time(0));

/* Generate n random numbers from
 * min(inclusive) to max(inclusive) */
while(n--) {
    printf("%d\n", (rand()%(max - min + 1)) + min);
}

```

3. Presentation of Results



```
┌ shadowleaf@shadowleaf-manjaro  ~/DSALab/random          ✓ < 79%  3.30G RAM  03:19:17
└ ./random n 10 1 75 85
78
77
78
85
80
81
79
75
85
77

Execution Time : 0.000175s
┌ shadowleaf@shadowleaf-manjaro  ~/DSALab/random          ✓ < 79%  3.35G RAM  03:20:43
└
```

This generates 10 random numbers from 75 to 85

4. Conclusions

A benefit of LCGs is that with appropriate choice of parameters, the period is known and long. Although not the only criterion, too short a period is a fatal flaw in a pseudorandom number generator. While LCGs are capable of producing pseudorandom numbers which can pass formal tests for randomness, this is extremely sensitive to the choice of the parameters m and a. For example, `a = 1` and `c = 1` produces a simple modulo-m counter, which has a long period, but is obviously non–random.

C uses LCGs internally inside rand for generating these pseudorandom numbers, this can be verified by looking at the definition written in `glibc`. This cannot be used for cryptographic purposes at all, due to the fact that two systems producing generating random numbers using this algorithm may generate the same numbers arising conflicts and encryption will fail.

Nevertheless this algorithm still holds good for small applications such as in an embedded systems where memory is severely limited.