

# Batch: A3 Roll No.:16010421119 Experiment No.: 1

**Aim:** To study and implement a PseudoRandom Number Generator (PRNG) using Linear Congruential Method

**Resources needed:** Turbo C / Java / python

# Theory

**Problem Definition:**

Write a Program for generating random numbers using Linear Congruential method such that

1. Period of the numbers generated is >=100
2. Density of the numbers generated is maximum (average gap between random numbers is < 0.1).

# Concepts:

**Random Numbers:** Random numbers are a necessary basic ingredient in simulation of almost all discrete systems. Most computer languages have a subroutine, object or function that will generate a random number. A simulation language generates random numbers that are used to generate event times and other random variables.

# Properties of random Numbers:

A sequence of random number R1, R2 … must have two important statistical properties, uniformity and independence.

# Uniformity :

If the interval (0, 1) is divided into „n‟ classes or subintervals of equal length , the expected number of observations in each interval is N/n, where N is total number of observations.

# Independence:

The probability of observing a value in a particular interval is independent of the previous drawn value.

# Problems faced in generating random numbers:

1. The generated number may not be uniformly distributed.
2. The number may be discrete valued instead of continuous values.
3. The mean of the numbers may be too high or low
4. The variance of the number may be too high or low.
5. The numbers may not be independent

e.g. a. Autocorrelation between numbers

b. Numbers successively higher or lower than adjacent numbers.

# Criteria for random no. generator:

1. The routine should be fast.
2. The routine should be portable.
3. The routine should have a sufficient long cycle. The cycle length or period represents the length of random number sequence before previous numbers begin to repeat themselves in an earlier order. A special case of cycling is degenerating. A routine degenerates when some number appears repeatedly which is unacceptable.
4. The random number should be replicable.
5. Most important, the generated random numbers should closely approximate to the ideal statistical properties of uniformity and independence.

6.

# Procedure / Approach /Algorithm / Activity Diagram:

**Linear Congruential Method:**

The Linear Congruential method produces a sequence of integers X1, X2,… between 0 and m-1 according to the following recursive relationship.

X i+1= (a X i + c) mod m , i = 0, 1, 2…

The initial value X0 is called the seed, a is constant multiplier, c is the increment and m is the modulus. Maximal period can be achieved by a, c, m, X0 satisfying one of the following conditions

1. For m, a power of 2 (m = 2b) and c≠0 period p = 2b is achieved provided c is relatively prime to m and a = 1+4k , k = 0,1,2,…
2. For m = 2b and c = 0 , period p = 2b-2 is achieved provided X0 is odd and multiplier a

= 3+8k or a = 5+8k , k = 0,1,2,…

1. For m a prime number and c = 0, period p = m-1 is achieved provided a has the property that the smallest integer is such that a k-1 is divisible by m is k = m-1.

# Results: (Program printout with output / Document printout as per the format)

#include<iostream>

#include<algorithm>

#include<vector>

#include<unordered\_map>

using namespace std;

#define noOfRandom 200

vector<int> random\_number(int seed,int m,int a,int c){

    vector<int> randoms(noOfRandom,seed);

    for(int i  = 1;i<noOfRandom;i++)

    {

        randoms[i] = ((randoms[i-1]\*a)+c)%m;

    }

    return randoms;

}

int period\_finder(vector<int> r)

{

    unordered\_map<int,int> mp;

    int count = 0;

    for(int num:r)

    {

        mp[num]++;

        if(mp[num] == 2)

        {

            break;

        }

        count++;

    }

    return count;

}

void print(vector<int> &r)

{

    for(int num:r)

    {

        cout<<num<<" ";

    }

    cout<<endl;

}

int main(){

*// For c != 0*

    int seed1 = 13;

    int a1 = 9;

    int c1 = 9;

    int m1 = 128;

    vector<int> r1;

    r1 = random\_number(seed1,m1,a1,c1);

    int period1 = period\_finder(r1);

    cout<<"Seed: "<<seed1<<endl;

    cout<<"a (multiplier): "<<a1<<endl;

    cout<<"c (incrementer): "<<c1<<endl;

    cout<<"m (modulus) "<<m1<<endl;

    cout<<"Period for C != 0 is "<<period1<<endl;

    print(r1);

*// For c = 0*

    int seed2 = 13;

    int a2 = 13;

    int c2 = 0;

    int m2 = 128; *// period 2^b-2 is achievable for m = 2^b*

    vector<int> r2;

    r2 = random\_number(seed2,m2,a2,c2);

    int period2 = period\_finder(r2);

    cout<<"Seed: "<<seed2<<endl;

    cout<<"a (multiplier): "<<a2<<endl;

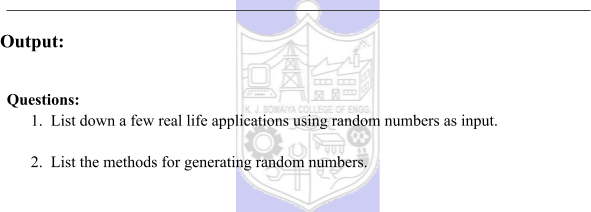
    cout<<"c (incrementer): "<<c2<<endl;

    cout<<"m (modulus) "<<m2<<endl;

    cout<<"Period for C = 0 is: "<<period2<<endl;

    print(r2);

}

****

# Questions:

1. **List down a few real life applications using random numbers as input.**

Used to model complex systems like traffic flow, financial markets, and biological processes.

1. **Randomized Algorithms:**

Employed in algorithms like randomized quicksort and Monte Carlo methods.

1. **Cryptographic Applications:**

Crucial for generating secure keys and nonces in cryptographic protocols.

1. **Game Development:**

Utilized for introducing variability in video games, determining enemy behavior, and loot drops.

1. **Random Sampling in Surveys:**

Used to select representative samples in statistical surveys.

1. **List the methods for generating random numbers.**
2. **Pseudorandom Number Generators (PRNG):**

Algorithms that use deterministic processes to generate a sequence of numbers that appears random. They start from an initial value called the seed. Popular algorithms include Linear Congruential Generators (LCG) and Mersenne Twister.

1. **Hardware Random Number Generators (HRNG):**

Utilize physical processes, such as electronic noise or radioactive decay, to generate truly random numbers. These generators are often used in cryptography for higher unpredictability.

1. **Middle-Square Method:**

A simple PRNG where the square of the current value is taken, and the middle digits are used as the next random number. This method has limitations but is easy to implement.

1. **Lagged Fibonacci Generator:**

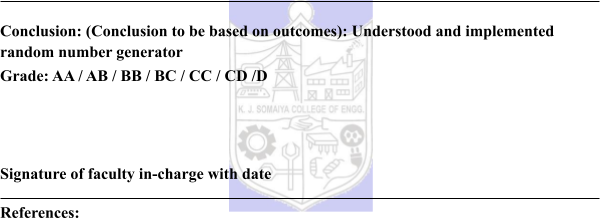
An extension of the Fibonacci sequence where the sum of the last two values is used to generate the next value. Lagged Fibonacci Generators are more complex than basic LCGs and can have longer periods.

1. **Cryptographic Hash Functions**:

Hash functions, like SHA-256, can be used to generate pseudorandom numbers by taking a seed value and hashing it to produce seemingly random output.

# Outcomes:

**CO1: Apply the experiment with process of simulation for model building using simulation languages and tools**



# Books/ Journals/ Websites:

**Text Book:**

Banks J., Carson J. S., Nelson B. L., and Nicol D. M., “Discrete Event System Simulation”, 3rd edition, Pearson Education, 2001.

# Websites:

1. <http://en.wikipedia.org/wiki/Pseudorandom_number_generator>
2. <http://en.wikipedia.org/wiki/Linear_congruential_generator>

.