Experiment No. 1
Title: Random Number Generator

Batch: A4 Roll No.: 16010420117 Experiment No.: 01

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

i) Period of the numbers generated is >=100

ii) 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.

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.

$X_{i+1} = (a X_i + c) \bmod m$, i = 0, 1, 2…

The initial value X_0 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, X_0 satisfying one of the following conditions

1. For m, a power of 2 ($m = 2^b$) and $c \neq 0$ period $p = 2^b$ is achieved provided c is relatively prime to m and a = 1+4k , k = 0,1,2,…
2. For $m = 2^b$ and c = 0 , period $p = 2^{b-2}$ is achieved provided X_0 is odd and multiplier a = 3+8k or a = 5+8k , k = 0,1,2,…
3. 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)

Case 1: When $c \neq 0$

Code:

```
#include <iostream>

class LinearCongruentialRandomGenerator {
    private:
        int seed, multiplier, increment, modulus, period;
        int lastDigit;

        int powerOfTwoHigherThan(int n) {
            int result = 2;
            while (result < n) result *= 2;
            return result;
        }

        int gcd(int a, int b) {
            if (b == 0) return a;
            return gcd(b, a%b);
        }

        bool areCoprime(int a, int b) {
            return gcd(a, b) == 1;
        }

        int coprimeTo(int a) {
            for (int i = 2; i < a; i++)
                if (areCoprime(a, i)) return i;
        }
```

```
    public:
        LinearCongruentialRandomGenerator(int x, int p) :
            seed(x),
            period(p),
            lastDigit(x)
        {
            modulus = powerOfTwoHigherThan(p);
            increment = coprimeTo(modulus);
            multiplier = 1 + 4*2;
        }

        void printInfo() {
            std::cout << "Seed: " << seed << std::endl;
            std::cout << "Multiplier: " << multiplier << std::endl;
            std::cout << "Increment: " << increment << std::endl;
            std::cout << "Modulus: " << modulus << std::endl;
        }

        float nextNumber() {
            lastDigit = (multiplier*lastDigit + increment) % modulus;
            return (float) lastDigit / modulus;
        }
};

int main() {
    int seed, increment;
    std::cout << "Enter a value for seed: " << std::endl;
    std::cin >> seed;

    LinearCongruentialRandomGenerator generator(seed, 100);
    std::cout << "A linear congruential random number generator has been
initialized" << std::endl;
    generator.printInfo();

    std::cout << "The generator will now generate the first 100 random numbers
now" << std::endl;
    for (int i = 0; i < 100; i++)
        std::cout << generator.nextNumber() << std::endl;
}
```

```
Enter a value for seed:
3
A linear congruential random number generator has been initialized
Seed: 3
Multiplier: 9
Increment: 3
Modulus: 128
The generator will now generate the first 100 random numbers now
0.234375
0.132812
0.21875
0.992188
0.953125
0.601562
0.4375
0.960938
0.671875
0.0703125
0.65625
0.929688
0.390625
0.539062
0.875
0.898438
0.109375
0.0078125
0.09375
0.867188
0.828125
0.476562
0.3125
0.835938
0.546875
0.945312
0.53125
0.804688
0.265625
0.414062
0.75
0.773438
0.984375
0.882812
0.96875
0.742188
0.703125
0.351562
```

```
0.0625
0.585938
0.296875
0.695312
0.28125
0.554688
0.015625
0.164062
0.5
0.523438
0.734375
0.632812
0.71875
0.492188
0.453125
0.101562
0.9375
0.460938
0.171875
0.570312
0.15625
0.429688
0.890625
0.0390625
0.375
0.398438
0.609375
0.507812
0.59375
0.367188
0.328125
0.976562
0.8125
0.335938
0.046875
0.445312
0.03125
0.304688
0.765625
0.914062
0.25
0.273438
0.484375
0.382812
0.46875
0.242188


...Program finished with exit code 0
Press ENTER to exit console.
```

Case 2: When c = 0

Code:

```
#include <iostream>

class LinearCongruentialRandomGenerator {
    private:
        int seed, multiplier, increment, modulus, period;
        int lastDigit;

        int powerOfTwoHigherThan(int n) {
            int result = 2;
            while (result < n) result *= 2;
            return result;
        }

    public:
        LinearCongruentialRandomGenerator(int x, int p) :
            seed(x),
            period(p),
            lastDigit(x)
        {
            modulus = powerOfTwoHigherThan(p) * 2 * 2;
            increment = 0
            multiplier = 3 + 8*7;
        }

        void printInfo() {
            std::cout << "Seed: " << seed << std::endl;
            std::cout << "Multiplier: " << multiplier << std::endl;
            std::cout << "Increment: " << increment << std::endl;
            std::cout << "Modulus: " << modulus << std::endl;
        }

        float nextNumber() {
            lastDigit = (multiplier*lastDigit + increment) % modulus;
            return (float) lastDigit / modulus;
        }
};

int main() {
    int seed, increment;
    std::cout << "Enter a value for seed (must be odd): " << std::endl;
    std::cin >> seed;

    LinearCongruentialRandomGenerator generator(seed, 100);
    std::cout << "A linear congruential random number generator has been
initialized" << std::endl;
    generator.printInfo();
```

```
    std::cout << "The generator will now generate the first 100 random numbers
now" << std::endl;
    for (int i = 0; i < 100; i++)
        std::cout << generator.nextNumber() << std::endl;
}
```

```
Enter a value for seed (must be odd):
7
A linear congruential random number generator has been initialized
Seed: 7
Multiplier: 59
Increment: 0
Modulus: 512
The generator will now generate the first 100 random numbers now
0.806641
0.591797
0.916016
0.0449219
0.650391
0.373047
0.00976562
0.576172
0.994141
0.654297
0.603516
0.607422
0.837891
0.435547
0.697266
0.138672
0.181641
0.716797
0.291016
0.169922
0.0253906
0.498047
0.384766
0.701172
0.369141
0.779297
0.978516
0.732422
0.212891
0.560547
0.0722656
0.263672
0.556641
0.841797
0.666016
0.294922
0.400391
0.623047
0.759766
0.826172
0.744141
0.904297
0.353516
```

```
0.748047
0.134766
0.951172
0.119141
0.0292969
0.728516
0.982422
0.962891
0.810547
0.822266
0.513672
0.306641
0.0917969
0.416016
0.544922
0.150391
0.873047
0.509766
0.0761719
0.494141
0.154297
0.103516
0.107422
0.337891
0.935547
0.197266
0.638672
0.681641
0.216797
0.791016
0.669922
0.525391
0.998047
0.884766
0.201172
0.869141
0.279297
0.478516
0.232422
0.712891
0.0605469
0.572266
0.763672
0.0566406
0.341797
0.166016
0.794922


...Program finished with exit code 0
Press ENTER to exit console.
```

Questions:

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

Ans: Many real life scenarios or systems have some unpredictable factor at play. Simulations require random numbers to model the unpredictable behavior of such system. For example, the behavior of customers in a queue, their arrival time, classical games of chance like gambling, chances of natural calamities, etc.

2. List the methods for generating random numbers.

Ans:

- Linear Congruence Method
- Middle square method
- Mersenne Twister Algorithm
- Cryptography based methods

Outcomes:

CO2: Generate pseudorandom numbers and perform empirical tests to measure the quality of a pseudo random number generator.

Conclusion: (Conclusion to be based on outcomes)

Understood the Linear Congruential Method of generating random numbers

Grade: AA / AB / BB / BC / CC / CD /D

Signature of faculty in-charge with date

References:

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

.