PPS

Programming for Problem Solving

Mini Project

Paridhi Agarwal

RA2111019010048

Problem Statement

Write a program in C to generate Pseudorandom Numbers by John Von Newmann’s Middle Square Algorithm and the Mersenne Twister (The Linear Congruenital Generator)

Analysis:

* Accept the User Choice
* Accept input according to user choice
* Generate Pseudorandom Numbers with the respective algorithm
* Display the generated numbers

FDT-Function Description Table

|  |  |  |  |
| --- | --- | --- | --- |
| Function Name | Return Type | Purpose | Parameter List |
| main | int | To take user’s inputs | - |
| count\_digit | int | To count the number of digits | long long seed,int digit |

Algorithm

## The Middle Square Method:-

Von Neumann proposed the middle square method of generating pseudo-random numbers in 1949, in a paper published a bit later. The method is simple: you take a seed, say 4 digits long, you square it, and extract the middle 4 digits, which become the next seed. For example:

1231

19123129

4373

While it seems random enough, is it?

A good pseudo-random generator should not depend much on the seed. It should also have a maximally long period, that is, visit every number in its range before it starts repeating itself. The transition graph would be just one big cycle.

Well, the middle square generator does both very badly:

* Depending on the seed, you may get very quickly into a cycle. For example, using a 2 digit seed, you get

2500

50

50

* The squaring operator isn’t very good at covering its range. We may think it’s a good operation because if you feed it n different numbers you’ll get n2 results, but you don’t. As we extract only the middle digits, we find that abound only 60% of the different values are actually reached (that seems to be true regardless of how many digits we use).
* The path from a seed to a cycle can be long, but cycles, when they occur, are very short.

## [Linear Congruential Generator for Pseudo-random Number Generation with R](https://aaronschlegel.me/linear-congruential-generator-r.html):-

Linear congruential generators (LCGs) are a class of pseudorandom number generator (PRNG) algorithms used for generating sequences of random-like numbers. The generation of random numbers plays a large role in many applications ranging from cryptography to Monte Carlo methods. Linear congruential generators are one of the oldest and most well-known methods for generating random numbers primarily due to their comparative ease of implementation and speed and their need for little memory. Other methods such as the Mersenne Twister are much more common in practical use today.

Linear congruential generators are defined by a recurrence relation:

Xi+1= (aXi+c) mod mXi+1= (aXi+c) mod m

There are many choices for the parameters m, the modulus, a, the multiplier, and c the increment.

Source Code

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Generating Pseudorandom Numbers

John Von Newmann's Middle Square Algorithm for Generation of Pseudorandom Numbers

Linear Congruential Generator for Generation of Pseudorandom Numbers

Aditya Jain RA2111019010064

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<conio.h>

#include<math.h>

#include<stdlib.h>

int count\_digit (long long, int);

int main ()

{

printf ("Enter a Choice: \n1) Linear Congruential Generator\n2) Middle Square

Algorithm\n");

int ch;

int xo, x1;

int a, c, m;

int i, j, n;

int array[20];

int a[20];

long long seed;

long long p;

int count;

int divisor;

int x[20];

scanf ("%d", &ch);

switch (ch)

{

case 1:

printf ("Enter the seed value: ");

scanf ("%d", &xo);

printf ("Enter the constant multiplier: ");

scanf ("%d", &a);

printf ("Enter the increment: ");

scanf ("%d", &c);

printf ("Enter the modulus: ");

scanf ("%d", &m);

printf ("How many random numbers you want to generate: ");

scanf ("%d", &n);

printf ("\n");

for (i=0; i<n; i++)

{

x1= (a\*xo+c) %m;

array[i] =x1;

xo=x1;

}

printf ("The generated random numbers are: ");

for (i=0; i<n; i++)

{

printf ("%d", array[i]);

printf ("\t");

}

getch ();

break;

case 2:

printf ("Enter a number to generate the random number: ");

scanf ("%lld", &seed);

p=seed;

int digit=0;

digit=count\_digit (p, digit);

count=digit;

printf ("How many random number you want to generate: ");

scanf ("%d", &n);

printf ("\nThe random numbers are: \n");

for (i=0; i<n; i++)

{

seed=seed\*seed;

p=seed;

digit=0;

digit=count\_digit (p, digit);

digit=ceil (digit/4.0);

for (j=0; j<digit; j++)

{

seed=seed/10;

}

divisor=pow (10, count);

seed=seed%divisor;

x[i] =seed;

printf ("%lld\t", seed);

}

getch ();

break;

default:

printf ("Wrong Choice!!!");

break;

}

return 0;

}

int count\_digit (long long seed, int digit)

{

long long p;

p=seed;

while (p! =0) // Digit count

{

p=p/10;

digit++;

}

return digit;

}

VDT-Variable Description Table

Sample Input Output

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Data Type | Purpose | Scope |
| ch | int | To accept the user choice | int main () |
| xo | int | To store the seed value | int main () |
| x1 | int | To store the next seed | int main () |
| a | int | To store the multiplier | int main () |
| c | int | To store the increment | int main () |
| m | int | To store the modulus | int main () |
| n | int | To store the number of random numbers to be generated | int main () |
| j | int | LCV-Loop Control Variable | int main () |
| i | int | LCV-Loop Control Variable | int main () |
| array[20] | int | To store the random numbers | int main () |
| count | int | To store the digit count of a number | int main () |
| divisor | int | To calculate the next seed | int main () |
| x[20] | int | To store the random numbers | int main () |
| seed | long long | To store the seed value | int main (), int count\_digit () |
| p | long long | To copy the seed value | int main (), int count\_digit () |
| digit | int | To store the digit count | int main (), int count\_digit () |

Sample Input Output





