**Description:**

**Implementation of random number generation techniques:**

1. **linear congruential**
2. **blum-blum shub generator**

The linear congruential method produces a sequence of integers X1, X2, X3, ... between **zero** and **m-1** according to the following recursive relationship:



* The initial value X0 is called the seed;
* a is called the constant multiplier;
* c is the increment
* m is the modulus

The selection of **a, c, m** and **X0** drastically affects the statistical properties such as mean and variance, and the cycle length.

**Example:** m=123, a=5, c=2, seed X0=73

X1=(5x73+2) mod 123

=367 mod 123

=121

X2=(5x121+2) mod 123

=607 mod 123

=115

X3=(5x115+2) mod 123

=577 mod 123

=85

……….

If we want to generate random bit

X0=73 ≡ 1 mode 2=>1

X1=121≡ 1 mode 2=>1

X2=115≡ 1 mode 2=>1

X3=85≡ 1 mode 2=>1

X4=58≡ 0 mode 2=>0

….

We get random 0 and 1….

**Instructions:**

Take the input from the file i.e. m, a, c, X0 and the range in which a random number is to be generated, and print the random numbers

**The Blum Blum Shub (BBS)** method is as pseudorandom number generator and was created by Lenore Blum, Manuel Blum and Michael Shub in 1968. It uses the form of:

Xi=(Xi-1)2 mod n

where X0 is a random seed. The value of **n** is equal to pq and where p and q are prime numbers.

These values of p and q are both congruent to 3 mod 4 (p=q=3 (mod4) ). What does that mean? Well when I take the values of p or q and divide them by 4, I will get a remainder of 3.

So, p=7 is possible (as 7 divided by 4 is 1 remainder 3), and p=11 is also possible (as 11 divided by 4 is 2 remainder 3). A value of 13 is not possible is at will be 3 remainder 1

n=p.q=77

X1=(5)2 mod 77

X2=(25)2 mod 77

X3=(9)2 mod 77

X4=(4)2 mod 77

In this case we are using p=7 and q=11, and then a seed of X0=5 , and the random sequence is **25, 9, 4 and 16**

**Instruction:** Read the input from the file i.e*. p, q, X0* and the range in which a random number is to be generated, and print the random numbers

**Implementation of random number generation techniques:**

1. **linear congruential**

* **methodology followed**

#include <iostream>

#include <bits/stdtr1c++.h>

#include <fstream>

using namespace std;

int main()

{

    ifstream fin;

    fin.open("input.txt");

    string s1;

    string s2;

    string s3;

    string s4;

    string s5;

    getline(fin, s1);

    getline(fin, s2);

    getline(fin, s3);

    getline(fin, s4);

    getline(fin, s5);

    long long int m = stoi(s1);

    long long int a = stoi(s2);

    long long int c = stoi(s3);

    long long int x0 = stoi(s4);

    long long int n = stoi(s5);

    vector<int> x;

    x.push\_back(x0);

    for (int i = 1; i <= n; i++)

    {

        x.push\_back((a \* x[i - 1] + c) % m);

    }

    for (int i = 0; i <= n; i++)

    {

        cout << "X" << i << " : " << x[i] << "\n";

    }

    string binary;

    for (int i = 0; i <= n; i++)

    {

        binary.push\_back(x[i] % 2 + '0');

    }

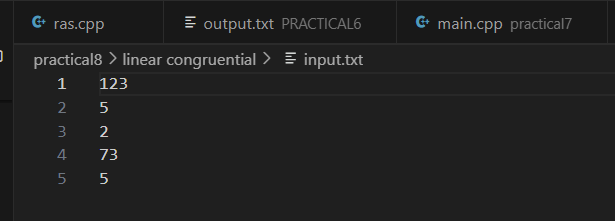
    cout << binary << "\n";

    fin.close();

    return 0;

}

INPUT:



1. **blum-blum shub generator**

* **methodology followed**

#include <iostream>

#include <bits/stdtr1c++.h>

#include <fstream>

using namespace std;

int main()

{

    ifstream fin;

    fin.open("input2.txt");

    string s1;

    string s2;

    string s3;

    string s4;

    getline(fin, s1);

    getline(fin, s2);

    getline(fin, s3);

    getline(fin, s4);

    long long int p = stoi(s1);

    long long int q = stoi(s2);

    long long int x0 = stoi(s3);

    long long int nx = stoi(s4);

    long long int n = p \* q;

    vector<long long int> x;

    x.push\_back(x0);

    for (int i = 1; i <= nx; i++)

    {

        x.push\_back((x[i - 1] \* x[i - 1]) % n);

    }

    for (int i = 0; i <= nx; i++)

    {

        cout << "X" << i << " : " << x[i] << "\n";

    }

    string binary;

    for (int i = 0; i <= nx; i++)

    {

        binary.push\_back(x[i] % 2 + '0');

    }

    cout << binary << "\n";

    fin.close();

    return 0;

}

// range  =  x%(max-min)+ min +1;

INPUT:

