LAB 1

Experiment No. 1

Date: 2081/02/18

OBJECTIVE:

To generate random number using linear congruential method in C programming.

THEORY:

Linear Congruential Method is a class of Pseudo Random Number Generator (PRNG) algorithms used for generating sequences of random-like numbers in a specific range. This method can be defined as:

$$\mathbf{X_{i+1} = aX_i + c \bmod m}$$

Where,

X, is the sequence of pseudo-random numbers
m, (>0) the modulus
a, (0,m) the multiplier
c, (0,m) the increment
X_0, [0,m) - Initial value of sequence known as seed

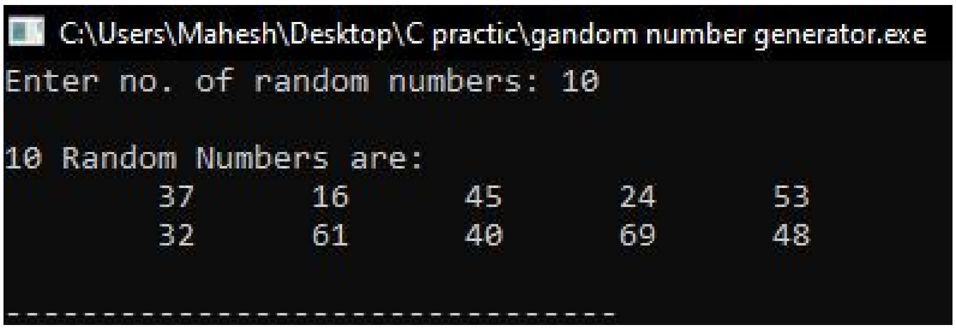
m, a, c, and X_0 should be chosen appropriately to get a period almost equal to m.

PROGRAM:

Source Code

```
#include<stdio.h>
int main(){
	int x0 = 37, a = 51, c = 29, m = 100;
	int randomNumber[50], n ,i;

	printf("Enter no. of random numbers: ");
	scanf("%d",&n);

	randomNumber[0] = x0;
	for(i = 1;i < n; i++)
		randomNumber[i] = ((randomNumber[i-1] * a) + c) % m;

	printf("\n%d Random Numbers are:\n",n);
	for(i = 1;i <= n; i++){
		printf("	%d",randomNumber[i-1]);
		if(i%5==0) printf("\n");
	}

	return 0;
}
```

Output:


C:\Users\Mahesh\Desktop\C practic\gandom number generator.exe

```
Enter no. of random numbers: 10

10 Random Numbers are:
        37      16      45      24      53
        32      61      40      69      48


--------------------------------
```

RESULTS AND DISCUSSION:

The experiment was successful to generate random numbers using linear congruential method in C programming. This program helps to analysis algorithms in C programming language.

CONCLUSION:

This laboratory exercise provided a hands-on experience in C program. Students gained practical knowledge of implementing algorithms in C programming and are now better equipped to undertake more complex programming tasks in the future.

Experiment No. 2

Date:- 2081/02/18

OBJECTIVE:

To implement Euclidean Algorithm to find GCD of two integers.

THEORY:

It is used to compute Greatest Common Divisor (GCD). The GCD represented as gcd(a,b) and defined as:

gcd(a,b) = d, where d is largest number that divide both a and b.

(if gcd(a,b)=1 then we say that a and b are relatively prime, which means both a and b do not divide each other).

Pseudo code for Euclidean Algorithm:

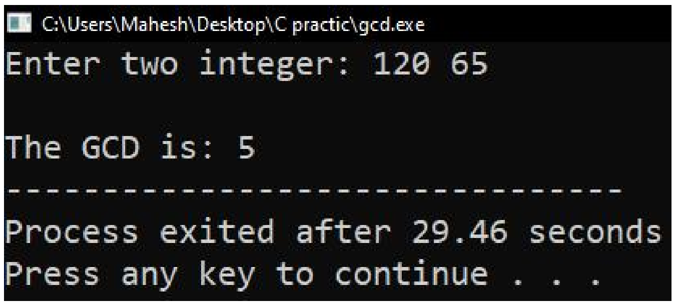
```
INPUT: Two non-negative integers a and b with a >= b
OUTPUT: gcd(a, b).
    1. While b > 0 do
        Set r = a mod b,
        a = b
        b = r
    2. Return a
```

PROGRAM:

Source Code

```
#include<stdio.h>

int GCD(int a, int b){
    int r;
    while(b>0){
        r = a % b;
        a = b;
        b = r;
    }
    return a;
}

void main(){
    int n1,n2;
    int gcd,lcm;
    printf("Enter two integer: ");
    scanf("%d%d",&n1,&n2);

    gcd = GCD(n1,n2);
    printf("\nThe GCD is: %d",gcd);
}
```

Output:



```
C:\Users\Mahesh\Desktop\C practic\gcd.exe
Enter two integer: 120 65

The GCD is: 5
--------------------------------
Process exited after 29.46 seconds
Press any key to continue . . .
```

RESULTS AND DISCUSSION:

The experiment successfully demonstrated the Euclidean Algorithm to find gcd of two integers in C programming. Students were able to implement Euclidean Algorithm n C programming.

CONCLUSION:

This program provides a basic implementation of the Euclidean algorithm to compute the GCD of two integers in C. It serves as a useful example for understanding how to implement fundamental arithmetic operations and control flow in C programming.

Experiment No. 3

Date: 2081/02/18

OBJECTIVE:

To implement extended Euclidean Algorithm.

THEORY:

It is just another way of finding greatest common division as we did it using Euclidean algorithm.

The pseudo code is illustrated below:

```
INPUT: Two non-negative integers a and b with a >= b.
OUTPUT: d = gcd(a, b) and integers x and y satisfying ax + by = d.

 1. If b = 0 then set d = a, x = 1, y = 0, and return(d, x, y).
 2. Set x2 = 1, X1 = 0, y2 = 0, y1 = 1
 3. While b > 0, do
     a. q = floor(a/b), r = a - qb, x = x2 - qx1, y = 2-qy1.
     b. a = b, b = r, x2 = X1, X1 = X, Y2 = Y1, Y1 = y.
 4. Set d = a, x = x2, y = y2, and return(d, x, y).
```

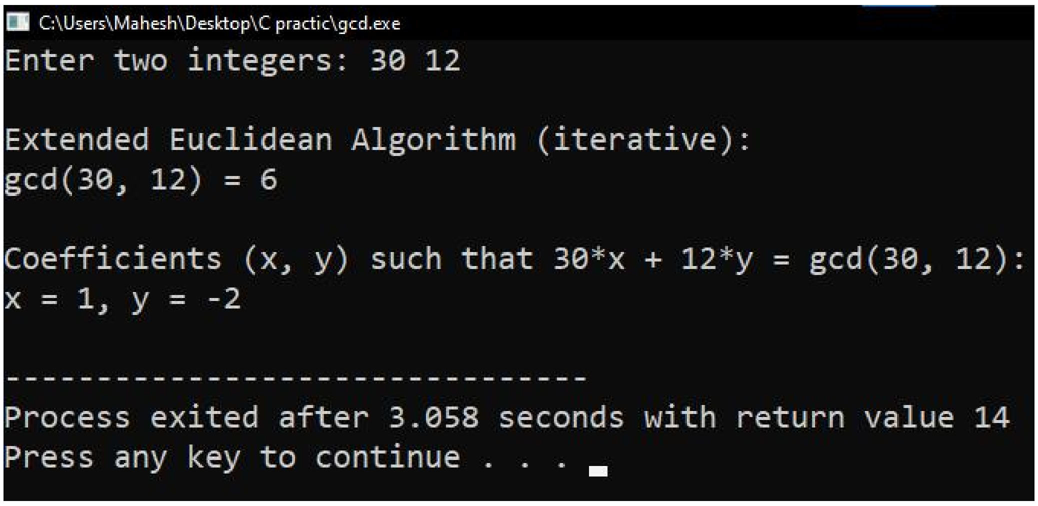
PROGRAM:

Source Code

```c
#include <stdio.h>

int extendedEuclid(int a, int b, int *x, int *y) {
   // Initial values
   int x1 = 1, y1 = 0, x2 = 0, y2 = 1;
   int q, r, xn, yn;

   while (b != 0) {
      q = a / b;
      r = a % b;

      xn = x1 - q * x2;
      yn = y1 - q * y2;

      // Update values for next iteration
      a = b;
      b = r;
      x1 = x2;
      y1 = y2;
      x2 = xn;
      y2 = yn;
   }
```

```
    // Setting coefficients x and y
    *x = x1;
    *y = y1;

    return a; // Returns gcd(a, b)
}

// Main function
void main() {
    int a, b, x, y;

    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    int gcd = extendedEuclid(a, b, &x, &y);

    // Output results
    printf("\nExtended Euclidean Algorithm (iterative):\n");
    printf("gcd(%d, %d) = %d\n\n", a, b, gcd);
    printf("Coefficients (x, y) such that %d*x + %d*y = gcd(%d, %d):\n", a, b, a,
b);
    printf("x = %d, y = %d\n", x, y);
}
```

Output:



```
C:\Users\Mahesh\Desktop\C practic\gcd.exe
Enter two integers: 30 12

Extended Euclidean Algorithm (iterative):
gcd(30, 12) = 6

Coefficients (x, y) such that 30*x + 12*y = gcd(30, 12):
x = 1, y = -2

--------------------------------
Process exited after 3.058 seconds with return value 14
Press any key to continue . . .
```

RESULTS AND DISCUSSION:

This program effectively implements the extended Euclidean algorithm in C, showcasing its ability to compute the GCD and find coefficients that satisfy a linear combination equation. By using an iterative approach, it ensures simplicity and efficiency, making it suitable for various computational tasks, including cryptography and modular arithmetic. The output illustrates its practical utility in solving mathematical problems involving number theory and demonstrates the fundamental principles of algorithmic design and implementation.

CONCLUSION:

This program successfully demonstrates the implementation of the extended Euclidean algorithm in C without using recursion. By computing the GCD and coefficients x and y, it showcases how mathematical algorithms can be practically implemented to solve number theory problems. The iterative approach ensures efficiency and clarity, making it a reliable method for various applications in computational mathematics.

Experiment No. 4

Date: 2081/02/18

OBJECTIVE:

To find the sum of two Binary numbers using Addition Algorithm.

THEORY:

Binary addition follows rules similar to decimal addition, where each digit (bit) can be either 0 or 1. The addition algorithm works as follows:

- Single Bit Addition:
  0 + 0 = 0
  0 + 1 = 1
  1 + 0 = 1
  1 + 1 = 10 (1 carry over to the next higher bit)

- Carry Handling:
  - If the sum of two bits is 1 or 0, no carry is generated.
  - If the sum is 10 (binary for 2), a carry of 1 is generated, and 0 is written as the result in that bit position.

- Iterative Process:
  - Start from the least significant bit (rightmost) and move towards the most significant bit (leftmost).
  - Carry generated from one bit addition is added to the next higher bit addition.

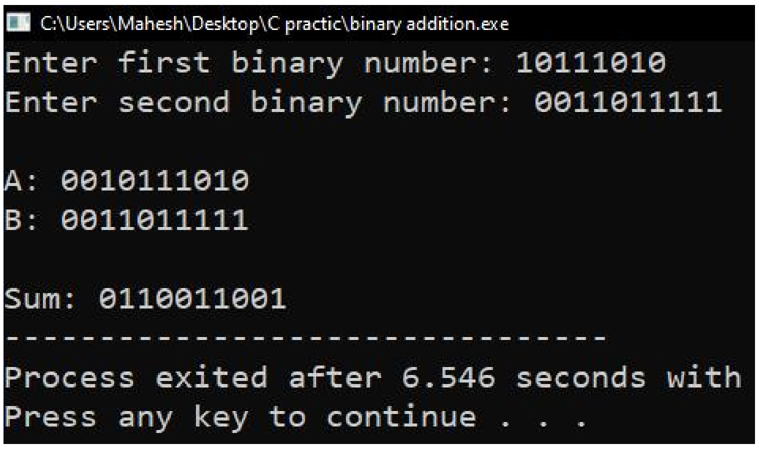
PROGRAM:

```
#include<stdio.h>

// To check if entered data is binary or not..
int isBinary(char arr[],int size){
        int i, t=1;
        for(i=0;i<size;i++)
                if(arr[i]!='0' && arr[i]!='1') return 0;
        return 1;
}

// To find the length of bits...
int length(char arr[]){
        int i,count=0;
        for(i=0; arr[i]!='\0'; i++)
                count++;
        return count;
}
```

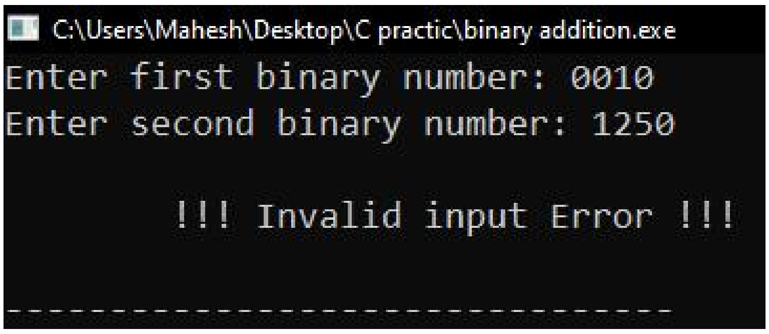
```
// To match bits with another...
void bitMatch(char arr[], int n){
        int i,j;
        int var;
        int     arrlen = length(arr);

        for(i=0; i < (n-arrlen); i++){
                var = length(arr);
                for(j=var; j > 0; j--)
                        arr[j] = arr[j-1];
                arr[0] = '0';
                arr[var+1] = '\0';
        }
}

// To Make both bits equal...
void makeEqualBits(char a[], char b[]){
        int al = length(a), bl = length(b);

        if(al > bl)             bitMatch(b,al);
        else if(bl > al)        bitMatch(a,bl);
}

// Adding two binary numbers...
void addBinary(char aArr[], char bArr[],char result[],int size){
        int d = 0,c = 0,i,j;
        int a,b;

        for(i=size-1;i>=0;i--){
                a = (aArr[i]=='1')?1:0;
                b = (bArr[i]=='1')?1:0;

                // Addition algorithm implementation...
                d = (a + b + c)%2;
                c = (a + b + c)/2;
                result[i] = d?'1':'0';
        }

        if(c){ // carry added if exist...
                for(j=size; j > 0; j--)
                        result[j] = result[j-1];
                result[0] = c?'1':'0';
                size++;
        }
        result[size] = '\0';
}
```

```
int main(){
        char a[50],b[50],sum[50]; //array to read binary number
        int n;  //define size of binary bits..

        printf("Enter first binary number: "); scanf("%s",a);
        printf("Enter second binary number: "); scanf("%s",b);

        //Error handling....
        if(!isBinary(a,length(a)) || !isBinary(b,length(b))){
                printf("\n\t!!! Invalid input Error !!!\n");
                return 0;
        }

        makeEqualBits(a,b);
        addBinary(a,b,sum,length(a));

        printf("\nA: %s\nB: %s",a,b);
        printf("\n\nSum: %s",sum);

        return 0;
}
```

<u>Output:</u>



```
C:\Users\Mahesh\Desktop\C practic\binary addition.exe
Enter first binary number: 10111010
Enter second binary number: 0011011111

A: 0010111010
B: 0011011111

Sum: 0110011001
--------------------------------
Process exited after 6.546 seconds with
Press any key to continue . . .
```



```
C:\Users\Mahesh\Desktop\C practic\binary addition.exe
Enter first binary number: 0010
Enter second binary number: 1250

        !!! Invalid input Error !!!

--------------------------------
```

RESULTS AND DISCUSSION

- **Algorithm Efficiency:** The binary addition algorithm implemented in the program iteratively computes the sum without converting binary numbers to decimal, ensuring efficiency.
- **Edge Cases:** The program handles cases where binary numbers have different lengths or where carry propagation is required effectively.

CONCLUSION

This lab exercise successfully demonstrates the implementation of the binary addition algorithm in C. It illustrates the fundamental principles of binary arithmetic and provides practical experience in handling binary operations programmatically. Understanding binary addition is crucial for various applications in computer science, including digital logic design and low-level programming.