| | |
|---|---|
| **Government College of Engineering, Jalgaon** **(An Autonomous Institute of Govt. of Maharashtra)** | |
| **Name:** | **PRN:** |
| **Class:** L.Y **Semester:** VII | **Batch:** |
| **Date of Performance:** ______________ | **Date of Completion:** _______________ |
| **Subject:** CO407U CNSL | **Subject Teacher:** Ms. Shrutika Mahajan |

# Practical - 2

**Aim:** Implementation of Euclidean algorithm and Extended Euclidean algorithm.

## Theory:

### 1. Euclidean Algorithm

The Euclidean Algorithm is one of the oldest and most efficient methods for finding the Greatest Common Divisor (GCD) of two integers.
The GCD of two numbers is the largest positive integer that divides both numbers without leaving a remainder.

**Working Principle:**

- Based on the property:
- GCD(a,b)=GCD(b,amodb),for a>b
- The process continues until the remainder becomes **0**.
- The non-zero divisor at this stage is the GCD.

**Example:**

Find GCD(56, 98)

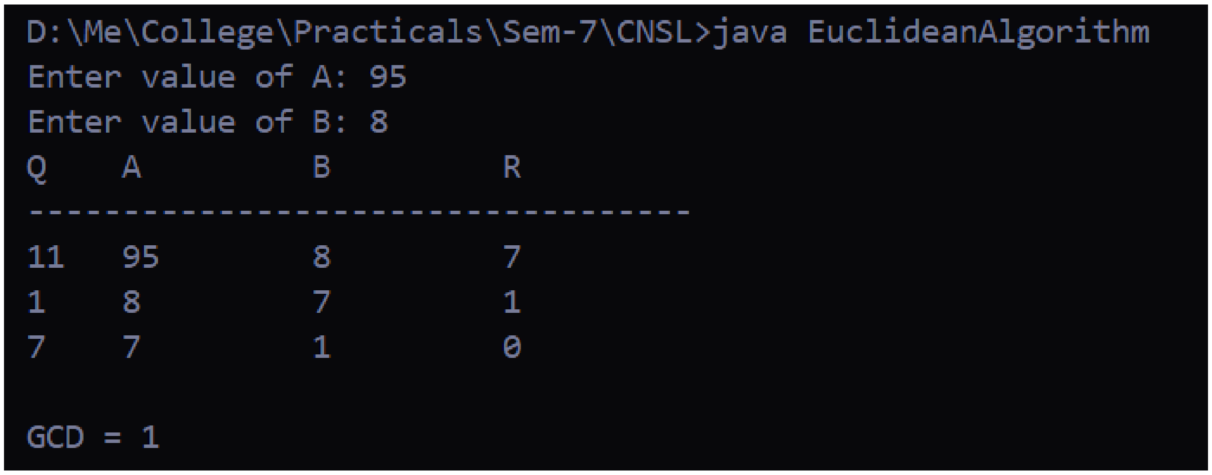
98 ÷ 56 = 1 remainder 42
56 ÷ 42 = 1 remainder 14
42 ÷ 14 = 3 remainder 0
So, GCD(56, 98) = 14

## Code:

```
import java.util.Scanner;

public class EuclideanAlgorithm {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Enter value of A: ");
        int A = sc.nextInt();
        System.out.print("Enter value of B: ");
        int B = sc.nextInt();

        System.out.printf("%-5s%-10s%-10s%-10s%n", "Q", "A", "B", "R");
        System.out.println("----------------------------------");

        while (B != 0) {
            int Q = A / B;
            int R = A % B;
            System.out.printf("%-5d%-10d%-10d%-10d%n", Q, A, B, R);

            // shift values: B → A, R → B
            A = B;
            B = R;
        }

        System.out.println("\nGCD = " + A);
    }
}
```

**Output:**



```
D:\Me\College\Practicals\Sem-7\CNSL>java EuclideanAlgorithm
Enter value of A: 95
Enter value of B: 8
Q    A         B         R
----------------------------------
11   95        8         7
1    8         7         1
7    7         1         0

GCD = 1
```

**2. Extended Euclidean Algorithm**

The Extended Euclidean Algorithm is an extension of the Euclidean Algorithm.
In addition to computing the GCD, it also finds integers **x** and **y** such that:

$ax+by=GCD(a,b)$

This is known as Bézout’s identity.

**Working Principle:**

- Uses recursion or iteration along with the Euclidean steps.
- Back-substitutes the remainders to express GCD as a linear combination of a and b.
- Very useful in cryptography (e.g., RSA) for finding modular inverses.

**Example:**
Find x and y such that:

gcd(30,20)=30x+20ygcd(30, 20) = 30x + 20ygcd(30,20)=30x+20y

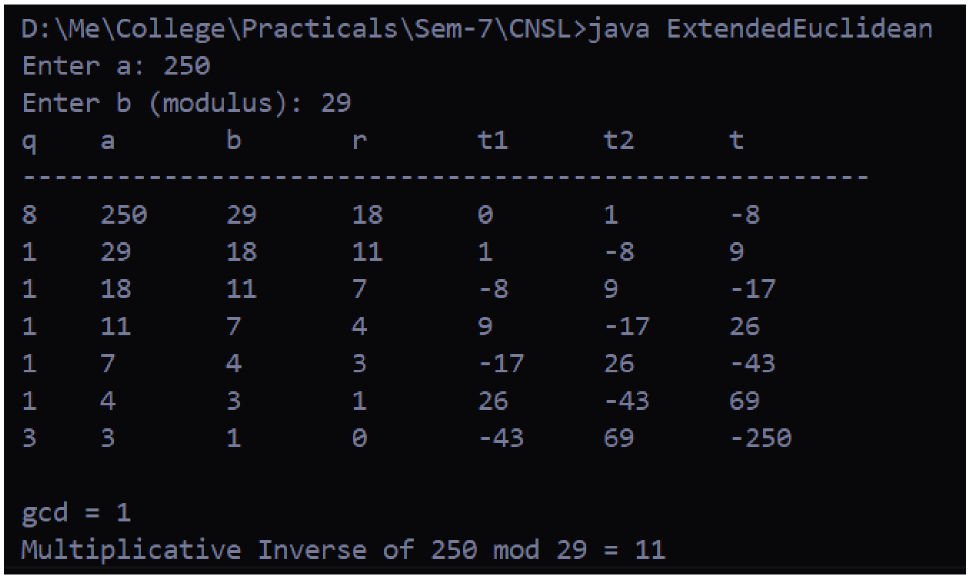
Steps:
30 = 20 × 1 + 10
20 = 10 × 2 + 0   So,
gcd(30,20) = 10
Back substitution:
10 = 30 - 20 × 1
  = 30 × (1) + 20 × (–1)
Hence, x = 1, y = –1

**Code:**

```
import java.util.Scanner;

public class ExtendedEuclidean {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter a: ");
      int a = sc.nextInt();
      System.out.print("Enter b (modulus): ");
      int b = sc.nextInt();

      int A = a, B = b;
      int t1 = 0, t2 = 1, t = 0;

      System.out.printf("%-5s%-8s%-8s%-8s%-8s%-8s%-8s%n", "q", "a", "b", "r", "t1", "t2", "t");
      System.out.println("---------------------------------------------------");

      while (B != 0) {
         int q = A / B;
         int r = A % B;
         t = t1 - q * t2;

         System.out.printf("%-5d%-8d%-8d%-8d%-8d%-8d%-8d%n", q, A, B, r, t1, t2, t);

         A = B;
         B = r;
         t1 = t2;
         t2 = t;
      }

      System.out.println("\ngcd = " + A);
```

```
    if (A == 1) {
            int inverse = (t1 % b + b) % b;
            System.out.println("Multiplicative Inverse of " + a + " mod " + b + " = " + inverse);
        } else {
            System.out.println("No Multiplicative Inverse exists since gcd != 1.");
        }
    }
}
```

**Output:**



```
D:\Me\College\Practicals\Sem-7\CNSL>java ExtendedEuclidean
Enter a: 250
Enter b (modulus): 29
q    a       b       r       t1      t2      t
------------------------------------------------------
8    250     29      18      0       1       -8
1    29      18      11      1       -8      9
1    18      11      7       -8      9       -17
1    11      7       4       9       -17     26
1    7       4       3       -17     26      -43
1    4       3       1       26      -43     69
3    3       1       0       -43     69      -250

gcd = 1
Multiplicative Inverse of 250 mod 29 = 11
```

**Applications:**

- Finding GCD of large numbers (useful in number theory).
- Used in cryptography (RSA algorithm for modular inverse).
- Solving Diophantine equations.
- Error correction codes in digital communication.

**Conclusion:**

- The Euclidean Algorithm provides an efficient way to compute the GCD of two integers using successive division.
- The Extended Euclidean Algorithm not only finds the GCD but also gives integers x and y satisfying Bézout’s identity.
- These algorithms form the foundation for many applications in network security, cryptography, and computational mathematics.

**Course Teacher**
**Ms. Shrutika Mahajan**