Experiment No. 03

Name : Soham U. Patil

PRN NO : 22510019

Batch : B6

Title – Implementation of Euclidean and Extended Euclidean Algorithm

Objectives:

Develop a program to implement the Euclidean Algorithm and the Extended Euclidean Algorithm for computing the Greatest Common Divisor (GCD) of two integers and, in the extended version, also compute the modular inverse and the coefficients of Bézout's identity.

1. Euclidean Algorithm

- The Euclidean Algorithm is an efficient method to compute the Greatest Common Divisor (GCD) of two integers a and b.
- Idea:
  - gcd(a, b) = gcd(b, a mod b)
  - Repeat until remainder = 0.
  - The non-zero number left is the GCD.

Example:
Find gcd(56, 15)

56 = 15 * 3 + 11

15 = 11 * 1 + 4

11 = 4 * 2 + 3

4 = 3 * 1 + 1

3 = 1 * 3 + 0 → GCD = 1

2. Extended Euclidean Algorithm

- In addition to finding GCD, it finds integers x and y such that:

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

- These integers x and y are called Bezout coefficients.
- If gcd(a, m) = 1, then the Extended Euclidean Algorithm can also compute the modular inverse of a modulo m:

$a-1\equiv x(modm)a^{-1} \equiv x \pmod{m}a-1≡x(modm)$

3. Modular Inverse

- For integer a and modulus m, the modular inverse is an integer x such that:

$a \cdot x \equiv 1 \pmod{m}$a · x≡1(modm)a \cdot x \equiv 1 \pmod{m}a · x≡1(modm)

- This only exists if gcd(a, m) = 1.

Procedure

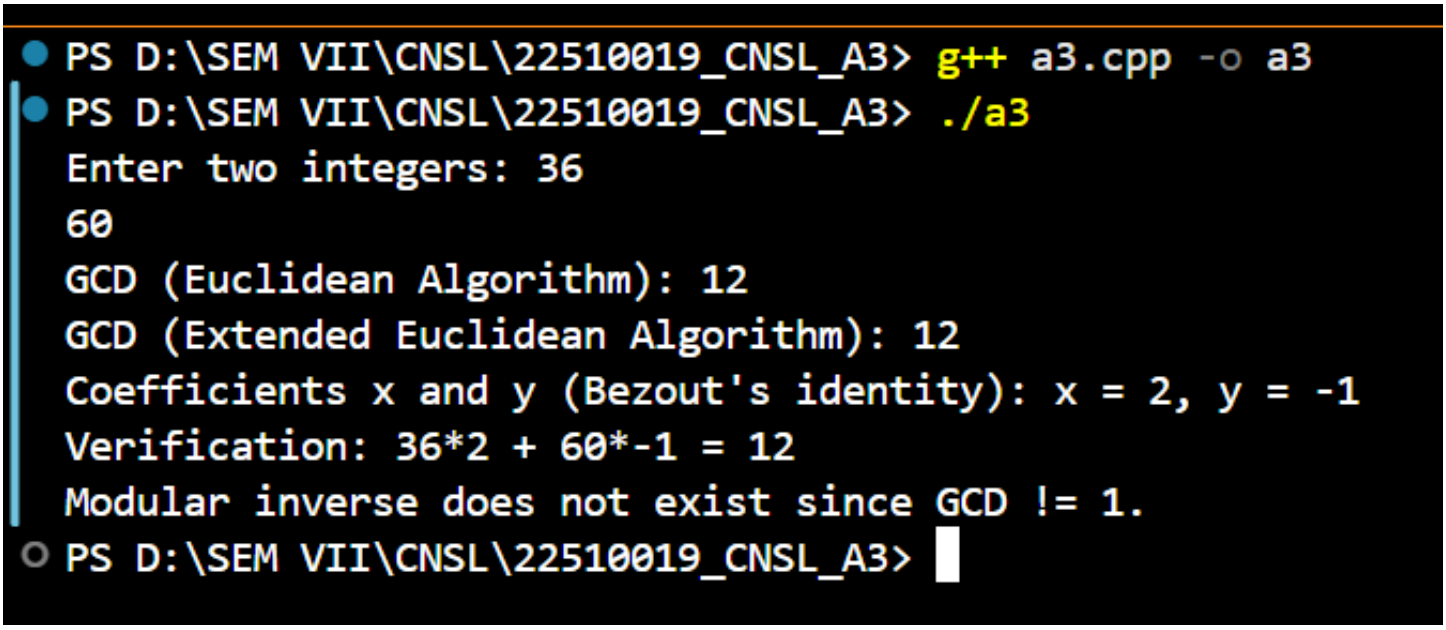
1. **Input two integers** a and b.
2. **Euclidean Algorithm**:
   - Repeat division a = bq + r until r = 0.
   - GCD = last non-zero remainder.
3. **Extended Euclidean Algorithm**:
   - Start with base case:
     - If b = 0, then gcd(a, b) = a, coefficients are (x=1, y=0).
   - Otherwise, recursively apply extended algorithm:
     - gcd(b, a mod b) gives coefficients (x1, y1).
     - Update:
     - x = y1
     - y = x1 - (a/b) * y1
4. **Modular Inverse (if required)**:

- If gcd(a, m) = 1 → inverse = (x mod m).
- Else → inverse does not exist.

Steps :

1. **Start Program**
   - Take input values a and b.
2. **Apply Euclidean Algorithm**
   - Use a loop or recursion to compute GCD.
   - Print steps (optional for understanding).
3. **Apply Extended Euclidean Algorithm**
   - Implement recursive/iterative function to compute (gcd, x, y).
   - Print coefficients x, y such that ax + by = gcd(a, b).
4. **Check Modular Inverse**
   - If gcd(a, b) = 1, compute inverse = (x mod b).
   - Print modular inverse.
   - Otherwise, print "Inverse does not exist".
5. **Output Results**
   - Display GCD.
   - Display Bézout coefficients.
   - Display modular inverse (if exists).

}



Code :

```
#include<bits/stdc++.h>
using namespace std;

int euclideanGCD(int a, int b)
{

    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

// Returns gcd(a, b), and finds x, y such that ax + by = gcd(a, b)
int extendedEuclidean(int a, int b, int &x, int &y) {
```

```
    if (b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }

    int x1, y1;
    int gcd = extendedEuclidean(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;

    return gcd;
}

int modInverse(int a, int m)
{
    int x, y;
    int g = extendedEuclidean(a, m, x, y);
    if (g != 1)
        return -1;
    else
        return (x % m + m) % m;
}

int main()
{
    int a, b;
    cout << "Enter two integers: ";
    cin >> a >> b;

    int gcd = euclideanGCD(a, b);
    cout << "GCD (Euclidean Algorithm): " << gcd << endl;
```

```
    int x, y;
    int gcd_ext = extendedEuclidean(a, b, x, y);

    cout << "GCD (Extended Euclidean Algorithm): " << gcd_ext << endl;
    cout << "Coefficients x and y (Bezout's identity): x = " << x << ", y = " << y << endl;


    cout << "Verification: " << a << "*" << x << " + " << b << "*" << y << " = " << (a*x + b*y) << endl;

    if (gcd_ext == 1)
    {
        int inv = modInverse(a, b);
        cout << "Modular inverse of " << a << " mod " << b << " is: " << inv << endl;
    } else {
        cout << "Modular inverse does not exist since GCD != 1." << endl;
    }

    return 0;
}
```

Observations:

a	b	GCD (Euclidean)	x (Bézout Coefficient)	y (Bézout Coefficient)	Modular Inverse of a mod b
56	15	1	-4	15	11
120	23	1	-9	47	14
81	153	9	2	-1	Not applicable

Conclusions

- The Euclidean Algorithm correctly computed the GCD by iterative remainder calculation.
- The Extended Euclidean Algorithm successfully computed Bézout coefficients x and y, verifying that $ax + by = gcd(a, b)$
- When $gcd(a , b) = 1$, the algorithm computed the modular inverse of a modulo b, which is essential in public-key cryptography (e.g., RSA).
- This experiment demonstrated the fundamental number-theoretic concepts used in modern cryptographic systems.