**PRN No: 2020BTECS00023**

**Full name: Sumit Narake**

**Batch: B2**

**Assignment: 7**

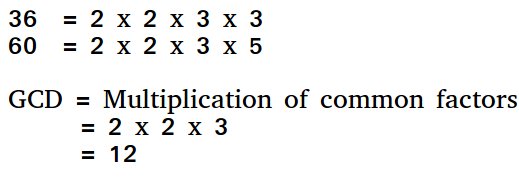
**Title of assignment:** Implementation of Euclidean and Extended Euclidean Algorithm.

1. **Aim:**

Implementation of Euclidean and Extended Euclidean Algorithm.

1. **Theory:**

The Euclidean algorithm is a way to find the greatest common divisor of two positive integers. GCD of two numbers is the largest number that divides both of them. A simple way to find GCD is to factorize both numbers and multiply common prime factors.



**Basic Euclidean Algorithm for GCD:**

The algorithm is based on the below facts.

* If we subtract a smaller number from a larger one (we reduce a larger number), GCD doesn’t change. So if we keep subtracting repeatedly the larger of two, we end up with GCD.
* Now instead of subtraction, if we divide the smaller number, the algorithm stops when we find the remainder 0.

Extended Euclidean algorithm also finds integer coefficients x and y such that: ax + by = gcd(a, b)

**Examples:**

***Input:****a = 30, b = 20****Output:****gcd = 10, x = 1, y = -1  
(Note that 30\*1 + 20\*(-1) = 10)*

***Input:****a = 35, b = 15****Output:****gcd = 5, x = 1, y = -2  
(Note that 35\*1 + 15\*(-2) = 5)*

## **How does Extended Algorithm Work?**

As seen above, x and y are results for inputs a and b,

*a.x + b.y = gcd                      —-(1)*

And x1 and y1 are results for inputs b%a and a*(b%a).x1 + a.y1 = gcd*

When we put b%a = (b – (⌊b/a⌋).a) in above, we get following. Note that ⌊b/a⌋ is floor(b/a) *(b – (⌊b/a⌋).a).x1 + a.y1  = gcd*

Above equation can also be written as below

*b.x1 + a.(y1 – (⌊b/a⌋).x1) = gcd      —(2)*

After comparing coefficients of ‘a’ and ‘b’ in (1) and (2), we get following,   
*x = y1 – ⌊b/a⌋ \* x1  
y = x1*

## **How is Extended Algorithm Useful?**

The extended Euclidean algorithm is particularly useful when a and b are coprime (or gcd is 1). Since x is the modular multiplicative inverse of “a modulo b”, and y is the modular multiplicative inverse of “b modulo a”. In particular, the computation of the modular multiplicative inverse is an essential step in RSA public-key encryption method.

**Code:**

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

class menu

{

    public :

   long long find\_multiplicative\_inverse(long long a, long long b) {

    long long q, r, t1 = 0, t2 = 1, t, main\_a = a;

cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    cout << "|\tQ\t|\tA\t|\tB\t|\tR\t|\tT1\t|\tT2\t|\tT\t|\n";

  cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    while (b > 0) {

        q = a / b;

        r = a % b;

        t = t1 -  (t2 \* q );

        cout << "|\t" << q << "\t|\t" << a << "\t|\t" << b << "\t|\t" << r << "\t|\t" << t1 << "\t|\t" << t2 << "\t|\t" << t << "\t|\n";

      cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

        a = b;

        b = r;

        t1 = t2;

        t2 = t;

    }

    cout << "|\t" << q << "\t|\t" << a << "\t|\t" << b << "\t|\t" << r << "\t|\t" << t1 << "\t|\t" << t2 << "\t|\t" << t << "\t|\n";

    cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    if (t1 < 0) {

        t1 += main\_a;

    }

    return t1;

}

    long long find\_large\_number\_gcd(long long a,long long b)

    {

        long long q,r;

            cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

            cout<<"|\t\tQ\t\t|\t\tA\t\t|\t\tB\t\t|\t\tR\t\t|\n";

          cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

            while(b>0)

            {

                    q=a/b;

                    r=a%b;

                    cout<<"|\t\t"<<q<<"\t\t|\t\t"<<a<<"\t\t|\t\t"<<b<<"\t\t|\t\t"<<r<<"\t\t|\n";

                 cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

                    a=b;

                    b=r;

            }

             cout<<"|\t\t"<<q<<"\t\t|\t\t"<<a<<"\t\t|\t\t"<<b<<"\t\t|\t\t"<<r<<"\t\t|\n";

    cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

            cout<<endl;

            return a;

    }

};

int main()

{

    main\_menu:

    cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    cout<<"\n1.Find Multiplicative Inverse (Extended Euclidien Algo ) \n2.Find GCD Of large numbers(Euclideian Algo ) \n";

    cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    cout<<"Enter Choice Code :\t";

    menu object;

    int ch;

    cin>>ch;

    cout<<"\n";

    long long a,b,ans;

    switch(ch)

    {

        case 1 :

            cout<<"\nEnter  A and B ( must be A>B)  :\t";

            cin>>a>>b;

            ans=object.find\_multiplicative\_inverse(a,b);

            cout<<"Multiplicative Inverse Of  "<<a<<"\tAnd "<<b<<"\t :\t"<<ans<<endl;

            goto main\_menu;

        case 2:

            cout<<"\nEnter  A and B  :\t";

            cin>>a>>b;

             ans=object.find\_large\_number\_gcd(a,b);

            cout<<"\nGCD Of   Of  "<<a<<"\tAnd "<<b<<"\t :\t"<<ans<<endl;

            goto main\_menu;

        default:

            cout<<"Invalid Input !";

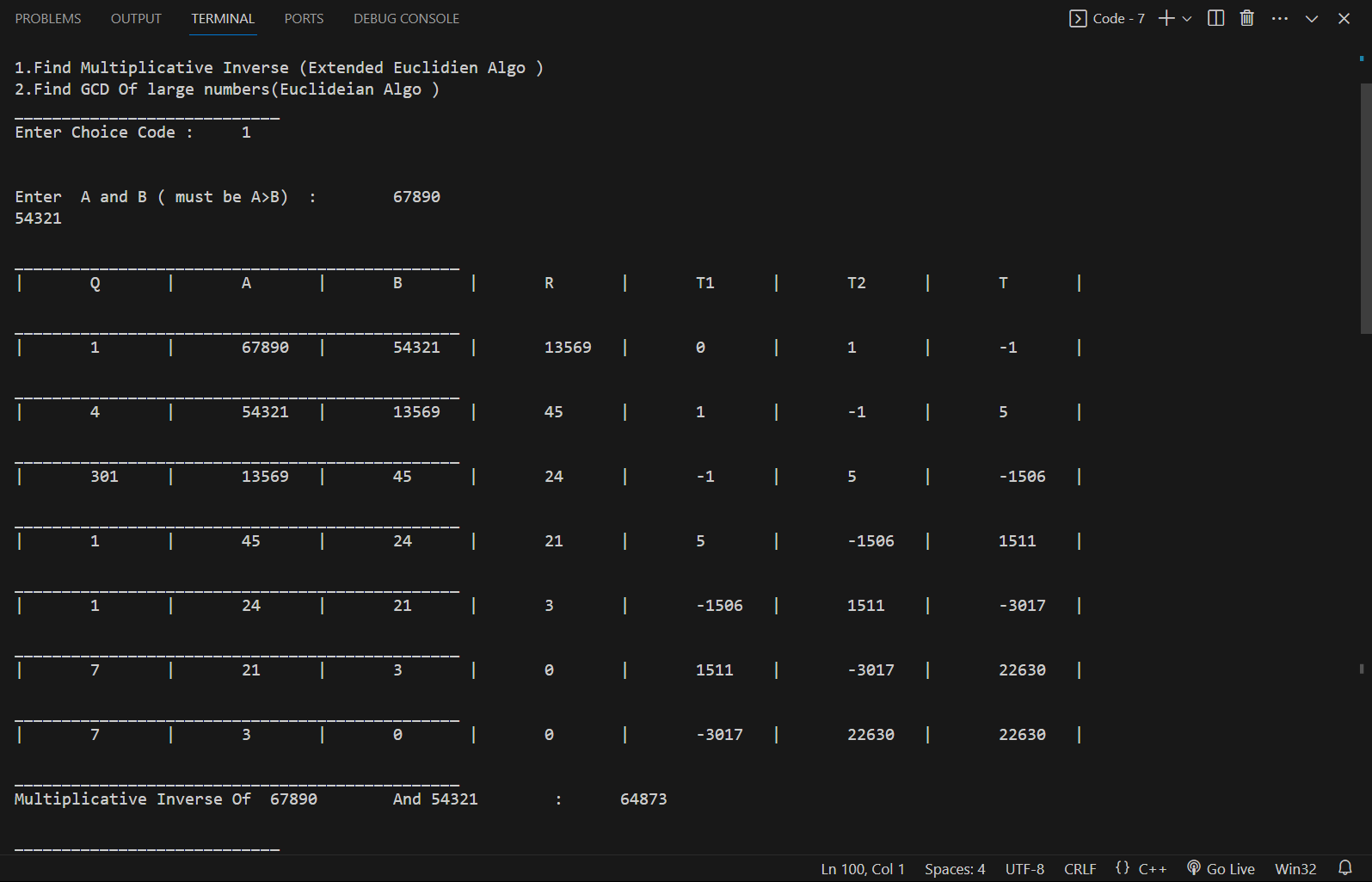
            break;

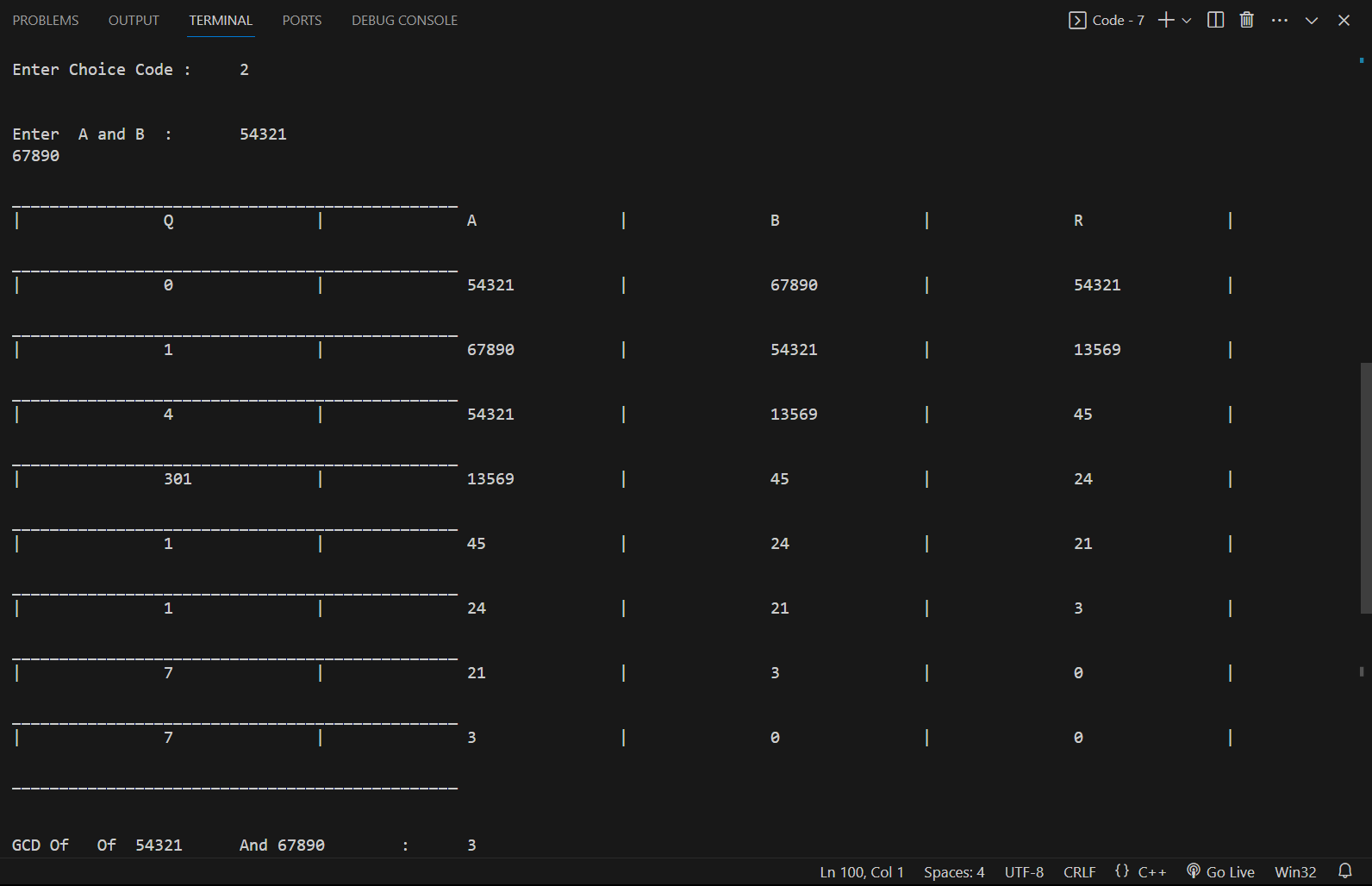
    }

    return 0;

}

**Output:**

****

****