

**Lab no.:- 3 Date:** 2079-09-22

**Title**: **Write a program to find GCD and Extended Euclidean Algorithm**

Greatest Common Divisor

The greatest common divisor (GCD), also called the greatest common factor, of two numbers is the largest number that divides them both. For instance, the greatest common factor of 20 and 15 is 5, since 5 divides both 20 and 15 and no larger number has this property. The concept is easily extended to sets of more than two numbers: the GCD of a set of numbers is the largest number dividing each of them.

The GCD is used for a variety of applications in number theory, particularly in modular arithmetic and thus encryption algorithms such as RSA. It is also used for simpler applications, such as simplifying fractions. This makes the GCD a rather fundamental concept to number theory, and as such a number of algorithms have been discovered to efficiently compute it.

The GCD is traditionally notated as gcd(a,b), or when the context is clear, simply (a, b).

The concept of the greatest common divisor or the highest common factor is used in many real-life incidents as below.

A shopkeeper has 420 balls and 130 bats to pack in a day. She wants to pack them in such a way that each set has the same number in a box, and they take up the least area of the box. What is the number that can be placed in each set for this packing purpose?

In the above problem, the greatest common divisor of 420 and 130 will be the required number.

Other application like arranging students in rows and columns in equal number, diving a group of people into smaller sections,etc.,

**IDE** : Dev-C++

**Language** : C++

**Source Code**

*// Program to demonstrate GCD*

#include <iostream>

using namespace std;

*// Function to return gcd of a and b*

int gcd(int a, int b) {

if (a == 0)

return b;

else

return gcd(b%a, a);

}

int main() {

int a,b;

cout<<"Enter the Value of a and b to find gcd(): ";

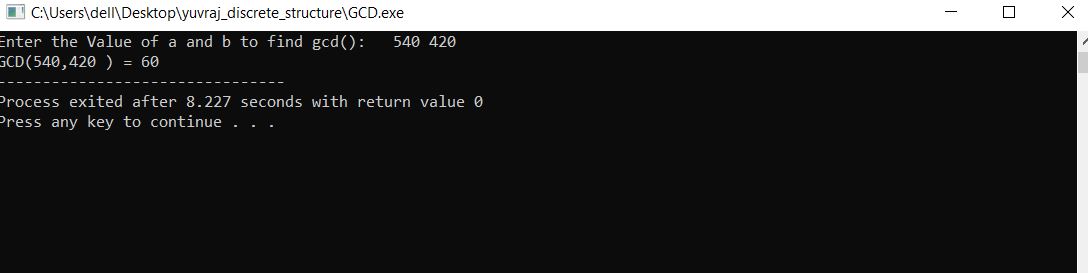
cin>>a>>b;

cout<<"GCD("<<a<<","<<b<<" ) = "<<gcd(a, b);

return 0;

}

**Output**



Extended Euclidean Algorithm

The extended Euclidean algorithm is an algorithm to compute integers *x* and *y* such that ax + by = gcd(a,b) given *a* and *b*.

The existence of such integers is guaranteed by Bézout's lemma.

The extended Euclidean algorithm can be viewed as the reciprocal of modular exponentiation.

By reversing the steps in the Euclidean algorithm, it is possible to find these integers *x* and y. The whole idea is to start with the GCD and recursively work our way backwards. This can be done by treating the numbers as variables until we end up with an expression that is a linear combination of our initial numbers. We shall do this with the example we used above.

We start with our GCD. We rewrite it in terms of the previous two terms: =26−2×12.

We replace for 12 by taking our previous line (38=1×26+12) and writing it in terms of 12: 2=26−2× (38−1×26).

Collect like terms, the 26's, and we have 2=3×26−2×38.

Repeat the process: 2=3×(102−2×38)−2×38.

The final result is our answer:.2=3×102−8×38.

Thus *x* and *y* are 3 and -8.

**IDE** : Dev-C++

**Language** : C++

**Source Code**

*// Program to demonstrate working of Extended Euclidean Algorithm*

#include <iostream>

using namespace std;

int gcdExtended(int a, int b, int\* x, int\* y)

{

*// Base Case*

if (a == 0) {

\*x = 0;

\*y = 1;

return b;

}

int x1, y1; // To store results of recursive call

int gcd = gcdExtended(b % a, a, &x1, &y1);

*// Update x and y using results of recursive call*

\*x = y1 - (b / a) \* x1;

\*y = x1;

return gcd;

}

int main()

{

int a,b,x=1, y=1;

cout<<"Enter the Value of a and b : ";

cin>>a>>b;

int g = gcdExtended(a, b, &x, &y);

cout<<endl<<"GCD("<<a<<","<<b<<") = "<< g;

cout<<endl<<" The Extended Euclidean Algorithm is : "<<g<<" = "<< x <<" \* " <<a <<" + "<< y<< " \* " <<b ;

return 0;

}

**Output**

