

**PROGRAMMING IN C++**

**CST-153**

SEMESTER-2

**Submitted To: Submitted By:** Er.Shallini Anchal Malik

16BCS1828

**TITLE**

C++ PROGRAM TO FIND GREATEST COMMON DIVISOR OF TWO NUMBERS USING RECURSIVE EUCLID ALGORITHM

**ABOUT THE PROJECT**

In [mathematics](https://en.wikipedia.org/wiki/Mathematics), the **Euclidean algorithm**, or **Euclid's algorithm**, is an efficient method for computing the [greatest common divisor](https://en.wikipedia.org/wiki/Greatest_common_divisor) (GCD) of two numbers, the largest number that divides both of them without leaving a [remainder](https://en.wikipedia.org/wiki/Remainder). It is named after the ancient Greek [mathematician](https://en.wikipedia.org/wiki/Mathematician) [Euclid](https://en.wikipedia.org/wiki/Euclid), who first described it in [Euclid's *Elements*](https://en.wikipedia.org/wiki/Euclid%27s_Elements) . It is an example of an [*algorithm*](https://en.wikipedia.org/wiki/Algorithm), a step-by-step procedure for performing a calculation according to well-defined rules, and is one of the oldest algorithms in common use. It can be used to reduce [fractions](https://en.wikipedia.org/wiki/Fraction_(mathematics)) to their [simplest form](https://en.wikipedia.org/wiki/Irreducible_fraction), and is a part of many other number-theoretic and cryptographic calculations.

The Euclidean algorithm is based on the principle that the greatest common divisor of two numbers does not change if the larger number is replaced by its difference with the smaller number. For example, 21 is the GCD of 252 and 105 (252 = 21 × 12 and 105 = 21 × 5), and the same number 21 is also the GCD of 105 and 147 = 252 − 105. Since this replacement reduces the larger of the two numbers, repeating this process gives successively smaller pairs of numbers until the two numbers become equal. When that occurs, they are the GCD of the original two numbers. By [reversing the steps](https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm), the GCD can be expressed as a [sum](https://en.wikipedia.org/wiki/Linear_combination) of the two original numbers each multiplied by a positive or negative [integer](https://en.wikipedia.org/wiki/Integer), e.g., 21 = 5 × 105 + (−2) × 252. The fact that the GCD can always be expressed in this way is known as [Bézout's identity](https://en.wikipedia.org/wiki/B%C3%A9zout%27s_identity" \o "Bézout's identity).

The version of the Euclidean algorithm described above (and by Euclid) can take many subtraction steps to find the GCD when one of the given numbers is much bigger than the other. A more efficient version of the algorithm shortcuts these steps, instead replacing the larger of the two numbers by its remainder when divided by the smaller of the two (with this version, the algorithm stops when reaching a zero remainder). With this improvement, the algorithm never requires more steps than five times the number of digits (base 10) of the smaller integer. This was proven by [Gabriel Lamé](https://en.wikipedia.org/wiki/Gabriel_Lam%C3%A9) in 1844, and marks the beginning of [computational complexity theory](https://en.wikipedia.org/wiki/Computational_complexity_theory). Additional methods for improving the algorithm's efficiency were developed in the 20th century.

The Euclidean algorithm has many theoretical and practical applications. It is used for reducing [fractions](https://en.wikipedia.org/wiki/Fraction_(mathematics)) to their [simplest form](https://en.wikipedia.org/wiki/Irreducible_fraction) and for performing [division](https://en.wikipedia.org/wiki/Division_(mathematics)) in [modular arithmetic](https://en.wikipedia.org/wiki/Modular_arithmetic). Computations using this algorithm form part of the [cryptographic protocols](https://en.wikipedia.org/wiki/Cryptographic_protocol) that are used to secure [internet](https://en.wikipedia.org/wiki/Internet) communications, and in methods for breaking these cryptosystems by [factoring large composite numbers](https://en.wikipedia.org/wiki/Integer_factorization). The Euclidean algorithm may be used to solve [Diophantine equations](https://en.wikipedia.org/wiki/Diophantine_equation), such as finding numbers that satisfy multiple congruences according to the [Chinese remainder theorem](https://en.wikipedia.org/wiki/Chinese_remainder_theorem), to construct [continued fractions](https://en.wikipedia.org/wiki/Continued_fraction), and to find accurate [rational approximations](https://en.wikipedia.org/wiki/Diophantine_approximation) to real numbers. Finally, it can be used as a basic tool for proving theorems in [number theory](https://en.wikipedia.org/wiki/Number_theory) such as [Lagrange's four-square theorem](https://en.wikipedia.org/wiki/Lagrange%27s_four-square_theorem) and the [uniqueness of prime factorizations](https://en.wikipedia.org/wiki/Fundamental_theorem_of_arithmetic). The original algorithm was described only for natural numbers and geometric lengths (real numbers), but the algorithm was generalized in the 19th century to other types of numbers, such as [Gaussian integers](https://en.wikipedia.org/wiki/Gaussian_integer) and [polynomials](https://en.wikipedia.org/wiki/Polynomial) of one variable. This led to modern [abstract algebraic](https://en.wikipedia.org/wiki/Abstract_algebra) notions such as [Euclidean domains](https://en.wikipedia.org/wiki/Euclidean_domain).

**PROCEDURE TO FIND GCD USING EUCLID ALGORITHM**

The Euclidean algorithm proceeds in a series of steps such that the output of each step is used as an input for the next one. Let *k* be an integer that counts the steps of the algorithm, starting with zero. Thus, the initial step corresponds to *k* = 0, the next step corresponds to *k* = 1, and so on.

Each step begins with two nonnegative remainders *rk*−1 and *rk*−2. Since the algorithm ensures that the remainders decrease steadily with every step, *rk*−1 is less than its predecessor *rk*−2. The goal of the *k*th step is to find a [quotient](https://en.wikipedia.org/wiki/Quotient) *qk* and [remainder](https://en.wikipedia.org/wiki/Remainder) *rk* that satisfy the equation

***rk-2 = qkrk + r0***

{\displaystyle r\_{k-2}=q\_{k}r\_{k-1}+r\_{k}}

and that have *rk* < *rk*−1. In other words, multiples of the smaller number *rk*−1 are subtracted from the larger number *rk*−2 until the remainder *rk* is smaller than *rk*−1.

In the initial step (*k* = 0), the remainders *r*−2 and *r*−1 equal *a* and *b*, the numbers for which the GCD is sought. In the next step (*k* = 1), the remainders equal *b* and the remainder *r*0 of the initial step, and so on. Thus, the algorithm can be written as a sequence of equations

***a = q0b + r0***

***b = q1r0 + r1***

***r0 = q2r1 + r2***

***.....***

{\displaystyle {\begin{aligned}a&=q\_{0}b+r\_{0}\\b&=q\_{1}r\_{0}+r\_{1}\\r\_{0}&=q\_{2}r\_{1}+r\_{2}\\r\_{1}&=q\_{3}r\_{2}+r\_{3}\\&\dotsb \end{aligned}}}

If *a* is smaller than *b*, the first step of the algorithm swaps the numbers. For example, if *a* < *b*, the initial quotient *q*0 equals zero, and the remainder *r*0 is *a*. Thus, *rk* is smaller than its predecessor *rk*−1 for all *k* ≥ 0.

Since the remainders decrease with every step but can never be negative, a remainder *rN* must eventually equal zero, at which point the algorithm stops.[[13]](https://en.wikipedia.org/wiki/Euclidean_algorithm#cite_note-13) The final nonzero remainder *rN*−1 is the greatest common divisor of *a* and *b*. The number *N* cannot be infinite because there are only a finite number of nonnegative integers between the initial remainder *r*0 and zero.

**ABOUT THE LANGUAGE –C++**

C++ is a [general-purpose programming language](https://en.wikipedia.org/wiki/General-purpose_programming_language). It has [imperative](https://en.wikipedia.org/wiki/Imperative_programming), [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) and [generic](https://en.wikipedia.org/wiki/Generic_programming) programming features, while also providing facilities for [low-level](https://en.wikipedia.org/wiki/Low-level_programming) [memory](https://en.wikipedia.org/wiki/Memory_(computing)) manipulation.

It was designed with a bias toward [system programming](https://en.wikipedia.org/wiki/System_programming) and [embedded](https://en.wikipedia.org/wiki/Embedded_software), resource-constrained and large systems, with [performance](https://en.wikipedia.org/wiki/Performance_(software)), efficiency and flexibility of use as its design highlights.[[5]](https://en.wikipedia.org/wiki/C%2B%2B#cite_note-Stroustrup1-5) C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications,[[5]](https://en.wikipedia.org/wiki/C%2B%2B#cite_note-Stroustrup1-5) including [desktop applications](https://en.wikipedia.org/wiki/Application_software), servers (e.g. [e-commerce](https://en.wikipedia.org/wiki/E-commerce), [web search](https://en.wikipedia.org/wiki/Web_search_engine) or [SQL](https://en.wikipedia.org/wiki/SQL) servers), and performance-critical applications (e.g. [telephone switches](https://en.wikipedia.org/wiki/Telephone_switches) or [space probes](https://en.wikipedia.org/wiki/Space_probes)).[[6]](https://en.wikipedia.org/wiki/C%2B%2B#cite_note-applications-6) C++ is a [compiled language](https://en.wikipedia.org/wiki/Compiled_language), with implementations of it available on many platforms. Many vendors provide [C++ compilers](https://en.wikipedia.org/wiki/List_of_compilers#C.2B.2B_compilers), including the [Free Software Foundation](https://en.wikipedia.org/wiki/Free_Software_Foundation), [Microsoft](https://en.wikipedia.org/wiki/Microsoft), [Intel](https://en.wikipedia.org/wiki/Intel), and [IBM](https://en.wikipedia.org/wiki/IBM).

C++ is standardized by the [International Organization for Standardization](https://en.wikipedia.org/wiki/International_Organization_for_Standardization) (ISO), with the latest standard version ratified and published by ISO in December 2014 as [*ISO/IEC 14882*](https://en.wikipedia.org/wiki/C%2B%2B#Standardization)*:2014* (informally known as [C++14](https://en.wikipedia.org/wiki/C%2B%2B14)).[[7]](https://en.wikipedia.org/wiki/C%2B%2B#cite_note-isocpp2011-7) The C++ programming language was initially standardized in 1998 as *ISO/IEC 14882:1998*, which was then amended by the [C++03](https://en.wikipedia.org/wiki/C%2B%2B03), *ISO/IEC 14882:2003*, standard. The current C++14 standard supersedes these and [C++11](https://en.wikipedia.org/wiki/C%2B%2B11), with [new features](https://en.wikipedia.org/wiki/C%2B%2B14) and an enlarged [standard library](https://en.wikipedia.org/wiki/C%2B%2B#Standard_library). Before the initial standardization in 1998, C++ was developed by [Bjarne Stroustrup](https://en.wikipedia.org/wiki/Bjarne_Stroustrup) at [Bell Labs](https://en.wikipedia.org/wiki/Bell_Labs) since 1979, as an extension of the [C language](https://en.wikipedia.org/wiki/C_(programming_language)) as he wanted an efficient and flexible language similar to C, which also provided high-level features for program organization. The [C++17](https://en.wikipedia.org/wiki/C%2B%2B17) standard is due in July 2017, with the draft largely implemented by some compilers already, and C++20 is the next planned standard thereafter.

**FEATURES OF C++**

* **Data Abstraction**

**Data abstraction** refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

* **Data encapsulation**

**Encapsulation** is an Object Oriented Programming concept that binds together the**data** and functions that manipulate the **data**, and that keeps both safe from outside interference and misuse. ... **C++** supports the properties of **encapsulation** and **data**hiding through the creation of user-defined types, called classes.

* **Data Hiding**

**Data hiding** is a software development technique specifically used in object-oriented programming (OOP) to **hide** internal object details (**data** members). **Data hiding**ensures exclusive **data** access to class members and protects object integrity by preventing unintended or intended changes.

* **Message Passing**

**Message Passing**. An object-oriented program consists of a set of objects that communicate with each other.

* **Dynamic Binding**

**Dynamic binding** also called **dynamic** dispatch is the process of linking procedure call to a specific sequence of code (method) at run-time. ... **Dynamic binding** is also known as late **binding** or run-time **binding**. **Dynamic binding** is an object oriented programming concept and it is related with polymorphism and inheritance.

* **Polymorphism**

The word **polymorphism** means having many forms. Typically, **polymorphism**occurs when there is a hierarchy of classes and they are related by inheritance. **C++ polymorphism** means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

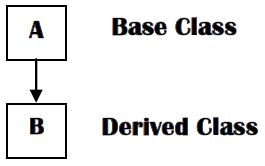
* **Inheritance**

**Inheritance** in Object Oriented Programming can be described as a process of creating new classes from existing classes. New classes **inherit** some of the properties and behavior of the existing classes. An existing class that is "parent" of a new class is called a base class.

**TYPES OF INHERITANCE**

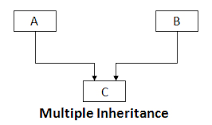
* **SINGLE LEVEL**

**When one class is derived from another class, it is referred to as single level inheritance.**



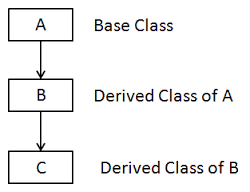
* **MULTIPLE**

**When a single class is derived from more than one bases classes, it is called multiple inheritance.**

****

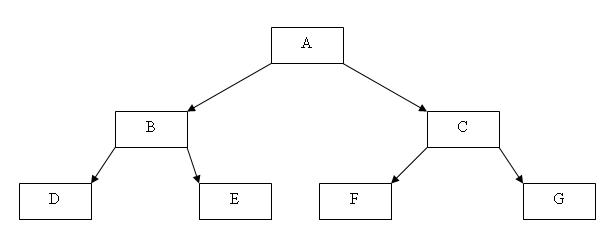
* **MULTILEVEL**

**When one class is derived from another class and one more class is derived from the previously derived class, it is then called multilevel inheritance.**

****

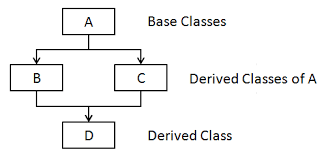
* **Hierarchical**

**When more than one class is derived from a single base class, it is called hierarchical inheritance.**

****

* **Hybrid**

**When all types of inheritance are used to derive a class, it is called Hybrid inheritance.**

****

**ADVANTAGES OF INHERITANCE**

- It allows the code to be reused as many times as needed. 

- The base class once defined and once it is compiled, it need not be reworked.

- Saves time and effort as the main code need not be written again.

- Code reusability

- Saves time in program development.

**TEMPLATES IN C++**

**Templates** are a feature of the C++ programming language that allows functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one.

Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.

A template is a blueprint or formula for creating a generic class or a function. The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept.

There is a single definition of each container, such as **vector**, but we can define many different kinds of vectors for example, **vector <int>** or **vector <string>**.

**ADVANTAGES OF TEMPLATES**

* reducing the repetition of code (generic containers, algorithms)
* reducing the repetition of code advanced (MPL and Fusion)
* static polymorphism (=performance) and other compile time calculations
* policy based design (flexibility, reusability, easier changes, etc)
* increasing safety at no cost (i.e. dimension analysis via Boost Units, static assertions, concept checks)

**FILE HANDLING IN C++**

**File Handling** concept in C++ language is used for store a data permanently in

computer. Using file handling we can store our data in Secondary memory (Hard disk).

**Why use File Handling?**

* For permanent storage.
* The transfer of input - data or output - data from one computer to another can be easily done by using files.

**How to achieve File Handling?**

For achieving file handling in C++ we need follow following steps

* Naming a file
* Opening a file
* Reading data from file
* Writing data into file
* Closing a file

**SOURCE CODE**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

int gcd(int u, int v)

{

return (v != 0) ? gcd(v, u % v) : u;

}

int main(void)

{

int num1, num2, result;

cout << "Enter two numbers to find GCD using Euclidean algorithm: ";

cin >> num1 >> num2;

result = gcd(num1, num2);

if (gcd)

cout << "\nThe GCD of " << num1 << " and " << num2 << " is: " << result

<< endl;

else

cout << "\nInvalid input!!!\n";

return 0;

}

**OUTPUT**

Enter two numbers to find GCD using Eculidean algorithm: 2 24

The GCD of 2 and 24 is :2

**ENHANCEMENTS THAT CAN BE DONE IN THE PROJECT**

* **The code can be done using file handling in order to store the data and information permanently.**
* **The code can also be done using classes and objects for efficient programming.**
* **There can also be the use of pointers in order to reduce the memory space occupied by the variables.**

**REFERENCES**

* **Programming in c++ by Sumita Arora.**
* [**www.wikipedia.com**](http://www.wikipedia.com)
* [**www.sanfoundry.com**](http://www.sanfoundry.com)
* [**www.tutorialspoint.com**](http://www.tutorialspoint.com)
* [**www.geeksforgeeks.com**](http://www.geeksforgeeks.com)

**THANK YOU!**

****

**SEMESTER 2**

**PROJECT ON C++ TO FIND THE GREATEST COMMON DIVISOR USING RECURSIVE EUCLIDIAN ALGORITHM**

**PROGRAMMING IN C++**

**CST 153**

**SUBMITTED TO: SUBMITTED BY:**

**Ms.DEEPANSHU NAME:R.UPASNA**

**UID:16BCS1351**