Deference b/w class and method

## C++ Functions

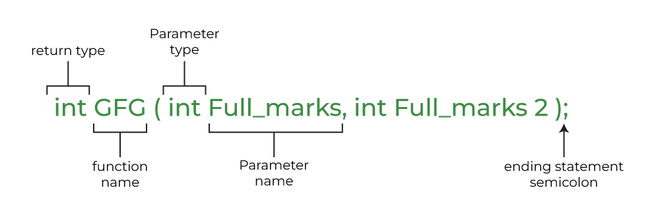
A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

Functions are used to perform certain actions, and they are important for reusing code: Define the code once and use it many times.

**Create a Function**

C++ provides some **pre-defined functions**, such as main(), which is used to execute code. But you can also create your own functions to perform certain actions.



// Create a function  
void myFunction() { //function Declaration part  
  cout << "I just got executed!";  
}  
int main() {  
  **myFunction();** // call the function  
  return 0;  
}  
// Outputs "I just got executed!"

**Note:** If a user-defined function, such as myFunction() is declared after the main() function, **an error will occur**:

## Method/Functions:

1. C++ Algorithm **sort()** function is used to sort the elements in the range [first, last) into ascending order.

int array[] = {10, 35, 85, 93, 62, 77, 345, 43, 2, 10};

int n = sizeof(array)/sizeof(array[0]);

// 'sizeof' gives the size of total array i.e. size of each character \* no. of characters

// so to get no. of characters

// we divide the sizeof(array) with the size of any one character of the array

// here it is array[0]

sort(array, array+n);

**O/p** 2 10 10 35 43 62 77 85 93 345

1. for swap two elements

**swap**(a[0], b[0])

## **OOPs (Object Oriented Programming System)**



**Object:**

* Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.
* An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.
* Object take up space in memory and have an associated address like a record in pascal or structure or union in C.
* When a program is executed, the objects interact by sending messages to one another.
* Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other’s data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.
* Object take up space in memory and have an associated address like a record in pascal or structure or union in C.
* When a program is executed the objects interact by sending messages to one another.
* Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other’s data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

**Class:**

* Collection of objects is called class. It is a logical entity.
* A Class is a user-defined data-type which has data members and member functions.
* Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behavior of the objects in a Class.

class MyClass {       // The class  
  public:             // Access specifier  
    int myNum;        // Attribute (int variable)  
    string myString;  // Attribute (string variable)  
};  
int main() {  
  MyClass **myObj**;  // Create an object of MyClass  
  
  // Access attributes and set values  
  **myObj.myNum** = 15;   
  **myObj.myString** = "Some text";  
  
  // Print attribute values  
  cout << myObj.myNum << "\n";  
  cout << myObj.myString;  
  return 0;  
}

| **S. No.** | **Class** | **Object** |
| --- | --- | --- |
| 1 | Class is used as a template for declaring and  creating the objects. | An object is an instance of a class. |
| 2 | When a class is created, no memory is allocated. | Objects are allocated memory space whenever they are created. |
| 3 | The class has to be declared only once. | An object is created many times as per requirement. |
| 4 | A class cannot be manipulated as they are not available in the memory. | Objects can be manipulated. |
| 5 | A class is a logical entity. | An object is a physical entity. |
| 6 | It is declared with the class keyword | It is created with a class name in C++ and  with the new keywords in Java. |
| 7 | Class does not contain any values which  can be associated with the field. | Each object has its own values, which are associated with it. |
| 8 | A class is used to bind data as well as methods together as a single unit. | Objects are like a variable of the class. |
| 9 | Syntax for Declaring Class in C++:  class <classname> {  }; | **Syntax for Instantiating an object for a Class in C++:**  class Student {     public:        void put(){            cout<<“Function Called”<<endl;        }  };   // The class is declared here  int main(){           Student s1;   // Object created           s1.put();  } |
| 10 | Example: Bike | Example: Ducati, Suzuki, Kawasaki |

**Constructors** are special class members which are called by the compiler every time an object of that class is instantiated. Constructors have the same name as the class and may be defined inside or outside the class definition.  
There are four types of constructors in C++:

1. Default Constructors
2. Parameterized Constructors
3. Copy Constructors
4. Dynamic Constructors

// C++ program to demonstrate constructors

#include <bits/stdc++.h>

using namespace std;

class Geeks

{

public:

int id;

//Default Constructor

Geeks()

{

cout << "Default Constructor called" << endl;

id=-1;

}

//Parameterized Constructor

Geeks(int x)

{

cout << "Parameterized Constructor called" << endl;

id=x;

}

};

int main() {

// obj1 will call Default Constructor

Geeks obj1;

cout << "Geek id is: " <<obj1.id << endl;

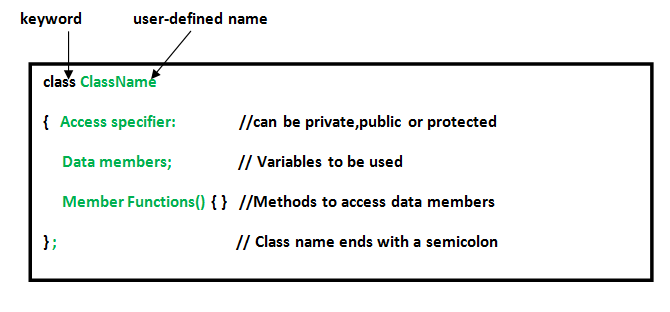
// obj2 will call Parameterized Constructor

Geeks obj2(21);

cout << "Geek id is: " <<obj2.id << endl;

return 0;

}



……………………………………………………………………………………………

**Inheritance**

inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically.

Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”.

**Derived Classes:** A Derived class is defined as the class derived from the base class.  
**Syntax**:

class <derived\_class\_name> : <access-specifier> <base\_class\_name>

{

//body

}

**Where**  
**class**    — keyword to create a new class  
derived\_class\_name   — name of the new class, which will inherit the base class  
**access-specifier** — either of private, public, or protected. If neither is specified, PRIVATE is taken as default  
**base-**class-name — name of the base class

**Note**: A derived class doesn’t inherit ***access*** to private data members However, it does inherit a full parent object, which contains any private members which that class declares.

**Modes of Inheritance:**There are 3 modes of inheritance.

1. **Public Mode**: If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.
2. **Protected Mode**: If we derive a subclass from a Protected base class. Then both public members and protected members of the base class will become protected in the derived class.
3. **Private Mode**: If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.

**Note:**The private members in the base class cannot be directly accessed in the derived class, while protected members can be directly accessed. For example, Classes B, C, and D all contain the variables x, y, and z in the below example. It is just a question of access.

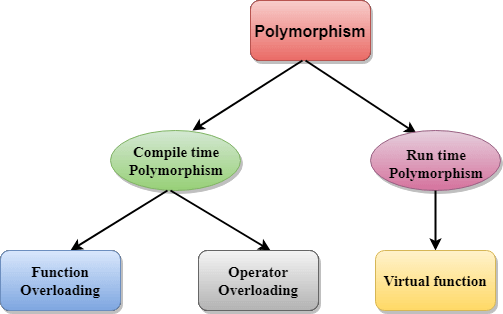
**Types Of Inheritance: -**

1. ***Single inheritance:*** a class is allowed to inherit from only one class.
2. ***Multilevel inheritance:*** Multiple Inheritance is a feature of C++ where a class can inherit from more than one class.
3. ***Multiple inheritance***: a derived class is created from another derived class.
4. ***Hierarchical inheritance:*** more than one subclass is inherited from a single base class.
5. ***Hybrid inheritance:*** Hybrid Inheritance is implemented by combining more than one type of inheritance.

Polymorphism

**Polymorphism** is an important concept of object-oriented programming. It simply means more than one form. That is, the same entity (function or operator) behaves differently in different scenarios.

For Example: A man at the same time is a father, a husband, and an employee. So, the same person exhibits different behavior in different situations. This is called polymorphism.



**1. Compile-Time Polymorphism**

This type of polymorphism is achieved by function overloading or operator overloading.

**A. Function Overloading**

When there are multiple functions with **the same name but different parameters**, then the functions are said to be **overloaded,**hence this is known as Function Overloading. Functions can be overloaded by **changing the number of arguments** or/and **changing the type of arguments**. In simple terms, it is a feature of object-oriented programming providing many functions to have the same name but distinct parameters when numerous tasks are listed under one function name. There are certain [Rules of Function Overloading](https://www.geeksforgeeks.org/cpp-polymorphism/) that should be followed while overloading a function.

**B. Operator Overloading**

C++ has the ability to provide the operators with **a special meaning for a data type**, this ability is known as operator overloading. For example, we can make use of the addition operator (+) for string class to concatenate two strings. We know that the task of this operator is to add two operands. So a single operator ‘+’, when placed between integer operands, adds them and when placed between string operands, concatenates them.

**2. Runtime Polymorphism**

This type of polymorphism is achieved by **Function Overriding**. Late binding and dynamic polymorphism are other names for runtime polymorphism.The function call is resolved at runtime in [runtime polymorphism](https://www.geeksforgeeks.org/virtual-functions-and-runtime-polymorphism-in-cpp/). In contrast, with compile time polymorphism, the compiler determines which function call to bind to the object after deducing it at runtime.

### **A. Function Overriding**

Function Overriding occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

### **B. Virtual Function**

A[**virtual function**](https://www.geeksforgeeks.org/virtual-function-cpp/) is a member function that is declared in the base class using the keyword virtual and is re-defined (Overridden) in the derived class.

Some Key Points About Virtual Functions:

Virtual functions are Dynamic in nature.

They are defined by inserting the keyword “**virtual**” inside a base class and are always declared with a base class and overridden in a child class

A virtual function is called during Runtime.

**Encapsulation**

**Encapsulation**is defined as wrapping up of data and information under a single unit.

* We cannot access any function from class directly. We need an object to access that function which is using the member the variable of that class.
* The function which we are making inside the class must use the all-member variable then only it is called encapsulation.

Abstraction is hiding irrelevant information and providing only the essential information of the data to the outside world(user).

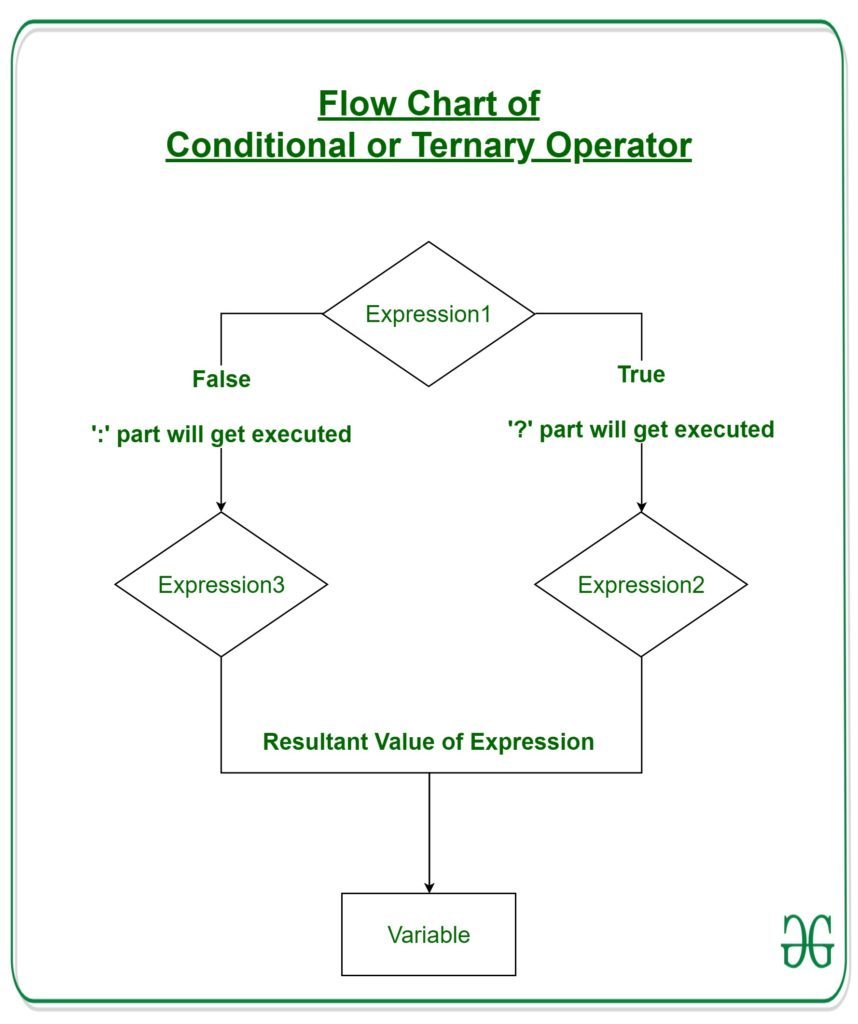
**public:** These data members are accessible both inside and outside the class.  
**private:** Private data members are not accessible outside the class and are not inheritable. They are available only inside the body of the class.  
**protected:** Protected members are not accessible outside the class but are inheritable in nature i.e. they are accessible to the derived/base class.

## **Getter (Accessor Function)**

## **Setter (Mutator Function)**

**\*\*Conditional or Ternary Operator (?:) in C/C++**

The conditional operator is kind of similar to the if-else statement as it does follow the same algorithm as of if-else statement, but the conditional operator takes less space and helps to write the if-else statements in the shortest way possible.



Text

Description automatically generated

# **Time Complexity**

**Time Complexity:** The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input. Note that the time to run is a function of the length of the input and not the actual execution time of the machine on which the algorithm is running on.

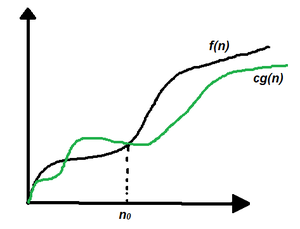
In detail: [Time Complexity Examples - Simplified 10 Min Guide (crio.do)](https://www.crio.do/blog/time-complexity-explained/)

# Big – Ω (Big- Omega) Notation

**Mathematical Representation:**

*Ω(g) = {f(n): there exist positive constants c and n0 such that 0 ≤ c\*g(n) ≤ f(n) for all n ≥ n0}   
Note: Ω (g) is a set*

**Graphical Representation:**



*Graphical Representation*

Follow the steps below to calculate Big – Omega(Ω) for any program:

1. Break the program into smaller segments.
2. Find the number of operations performed for each segment (in terms of the input size) assuming the given input is such that the program takes the least amount of time.
3. Add up all the operations and simplify it, let’s say it is f(n).
4. Remove all the constants and choose the term having the least order or any other function which is always less than f(n) when n tends to infinity, let say it is g(n) then, Big – Omega (Ω) of f(n) is Ω(g(n)).

Some examples:

Liner time = O(n)

Logarithm time > O(Log n )

Quadratic time >> O(n2)

Cubic time >> O(n3)

**Increasing complexity**

Min complexity O(1) >O(log n) >O(n)>> O(nlog n) >>O(n2) >>O(n3) >> O(2n) > O(n!) Max complexity

# **Array**

1. It is a group of variables **of similar data types** referred to by a single element.
2. Its elements are stored in a **contiguous memory location**.
3. The size of the array should be mentioned while declaring it.
4. Array elements are always counted from zero (0) onward.
5. Array elements can be accessed using the position of the element in the array.
6. The array can have one or more dimensions.

**Complexity of array**

* Time complexity - O(n)
* Space complexity - O(n)

**// array declaration by specifying size**

* **int** arr1[10];
* **int** arr[] = { 10, 20, 30, 40};
* **int** arr[2] = { };

**Types of Arrays:**

1. **Single Dimensional Arrays** Array\_Name [ ].
2. **Two Dimensional Arrays** a form of rows(i) \* columns(j), for instance an A[2][3] will have 2 rows and 3 columns allocating 6 elements.
3. **Three Dimensional Arrays:** as A[k][i][j] where k, i, j represents depth, rows and columns respectively.

**Size of Multidimensional Arrays:**

The total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.

**Basic Operations Performed with an Array**

1. **Traverse:** Traversing literally means travelling through an area, and in terms of a programming language it means running a loop in an array and performing the required operation.
2. **Insertion:** Adding an element inside an array on the given position(index) in the array.
3. **Deletion:** Removing any element from an array at a given position(index).
4. **Search:** Searches an element of an array by the element index or the value.
5. **Update:** Updates an element on a given position(index).

 sizeof() operator returns the size of a type in **bytes**.

Example: int myNumbers[5] = {10, 20, 30, 40, 50};  
 cout << **sizeof(myNumbers)**;

OUTPUT / 20 //20 is Bytes

int myNumbers[5] = {10, 20, 30, 40, 50};  
int getArrayLength = **sizeof(myNumbers) / sizeof(int)**;  
cout << getArrayLength;

Result: **5**

**Array problem list:** <https://www.geeksforgeeks.org/c-programs-gq/array-programs-gq/>

# Vector in C++ STL

Vectors are the same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. In vectors, data is inserted at the end. Inserting at the end takes differential time, as sometimes the array may need to be extended. Removing the last element takes only constant time because no resizing happens. Inserting and erasing at the beginning or in the middle is linear in time.

Vector : https://www.geeksforgeeks.org/vector-in-cpp-stl/

String Data Structure

Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character ‘\0’.

1. **Using string::size:** The method string::size returns the length of the string, in terms of bytes.
2. **Using string::length:**The methodstring::length returns the length of the string, in terms of bytes. Both string::size and string::length are synonyms and return the exact same value.

URL: <https://www.geeksforgeeks.org/string-data-structure/?ref=shm>

The **ispunct()** function checks whether a character is a punctuation character or not.  
The term “**punctuation**” as defined by this function includes all printable characters that are neither alphanumeric nor a space. For example, ‘@’, ‘$’, etc.  
This function is defined in **ctype.h** header file.

**strcoll()** is a built-in library function and is declared in **<string.h>** header file. This function compares the string pointed to by *str1* with the one pointed by *str2*.The **strcoll()** function performs the comparison based on the rules of the current locale’s**LC\_COLLATE** category. **Syntax:**

int strcoll(const char \*str1, const char \*str2)

Problem :

1. Function to copy string (Iterative and Recursive)
2. [**https://www.geeksforgeeks.org/rearrange-characters-string-no-two-adjacent/**](https://www.geeksforgeeks.org/rearrange-characters-string-no-two-adjacent/)

# **Linked List**

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers

A Linked List is a linear data structure that is a collection of objects, called nodes. Each node in a linked list consists of two parts, the first part contains the Data and the second part contains the Address of the next node in the Linked List. A Linked List is a dynamic data structure, i.e., memory is allocated at run time, and memory size can be changed at run time according to our requirements.

If the linked list is empty, then the value of the head points to NULL.

Linked List in C++ works using pointers, each node is connected to the next node using C++ pointers.

There are mainly three **types of Linked Lists**:

* Singly Linked Lists
* Doubly Linked Lists
* Circular Linked Lists

**Simple Linked List/ Singly Linked Lists** – In this type of linked list, one can move or traverse the linked list in only one direction

**Doubly Linked List** – In this type of linked list, one can move or traverse the linked list in both directions (Forward and Backward)

**Circular Linked List** – In this type of linked list, the last node of the linked list contains the link of the first/head node of the linked list in its next pointer and the first/head node contains the link of the last node of the linked list in its prev pointer

**Advantages of Linked Lists over arrays**:

* Dynamic Array.
* Ease of Insertion/Deletion.

**Drawbacks of Linked Lists:**

Random access is not allowed. We have to access elements sequentially starting from the first node(head node). So we cannot do a binary search with linked lists efficiently with its default implementation.

Extra memory space for a pointer is required with each element of the list.

Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

Types of Linked Lists:

Basic operations on Linked Lists:

* Deletion
* Insertion
* Search
* Display