CPP

What is C++

C++ is a general purpose, case-sensitive, free-form programming language that supports object-oriented, procedural and generic programming.

C++ is a middle-level language, as it encapsulates both high and low level language features.

C++ Standard Libraries

Standard C++ programming is divided into three important parts:

- The core library includes the data types, variables and literals, etc.
- The standard library includes the set of functions manipulating strings, files, etc.
- The Standard Template Library (STL) includes the set of methods manipulating a data structure.

Usage of C++

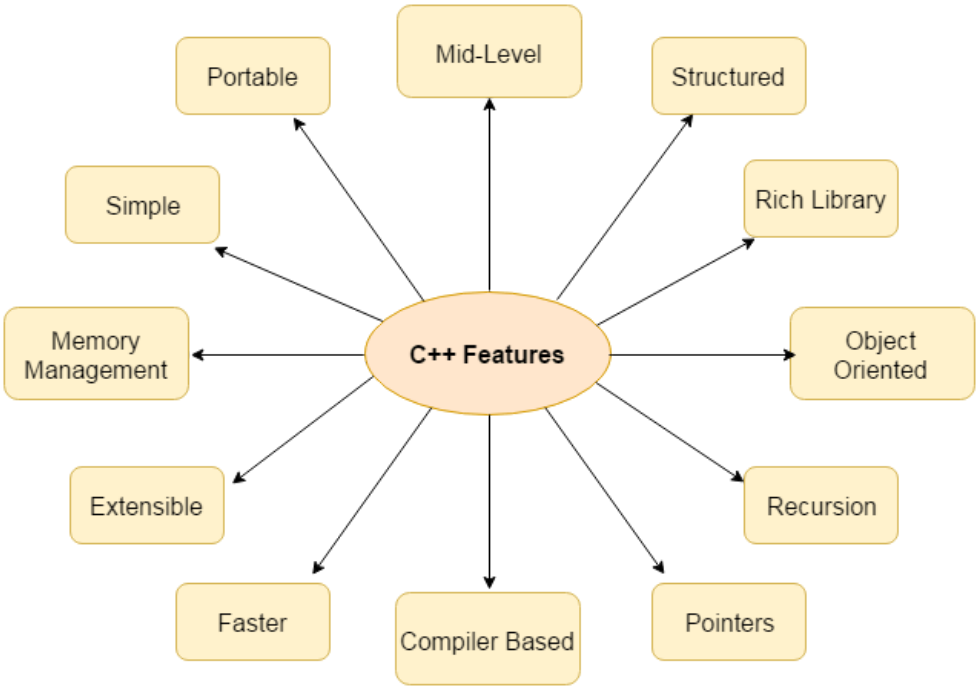
With the help of C++ programming language, we can develop different types of secured and robust applications:

- Window application
- Client-Server application
- Device drivers
- Embedded firmware etc

No.	C	C++
1)	C follows the **procedural style programming.**	C++ is multi-paradigm. It supports both **procedural and object oriented.**
2)	Data is less secured in C.	In C++, you can use modifiers for class members to make it inaccessible for outside users.
3)	C follows the **top-down approach.**	C++ follows the **bottom-up approach.**
4)	C does not support function overloading.	C++ supports function overloading.
5)	In C, you can't use functions in structure.	In C++, you can use functions in structure.
6)	C does not support reference variables.	C++ supports reference variables.
7)	In C, **scanf() and printf()** are mainly used for input/output.	C++ mainly uses stream **cin and cout** to perform input and output operations.
8)	Operator overloading is not possible in C.	Operator overloading is possible in C++.
9)	C programs are divided into **procedures and modules**	C++ programs are divided into **functions and classes.**
10)	C does not provide the feature of namespace.	C++ supports the feature of namespace.
11)	Exception handling is not easy in C. It has to perform using other functions.	C++ provides exception handling using Try and Catch block.
12)	C does not support the inheritance.	C++ supports inheritance.

https://www.javatpoint.com/c-vs-cpp

C++ Features :



C++ Basic Input/Output

← Prev Next →

C++ I/O operation is using the stream concept. Stream is the sequence of bytes or flow of data. It makes the performance fast.

If bytes flow from main memory to device like printer, display screen, or a network connection, etc, this is called as **output operation.**

If bytes flow from device like printer, display screen, or a network connection, etc to main memory, this is called as **input operation.**

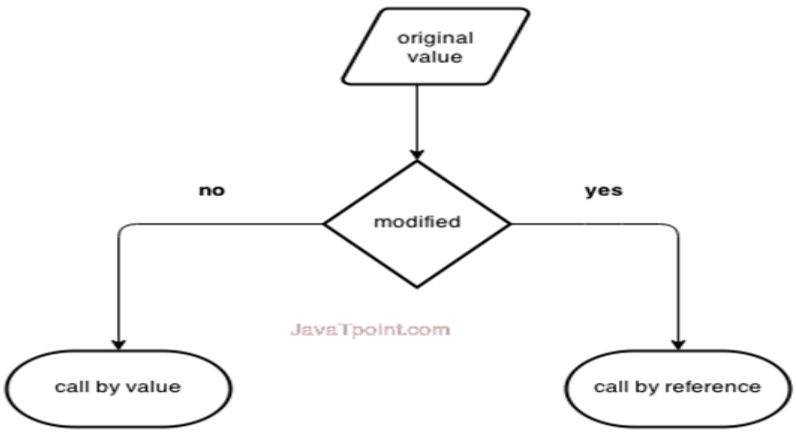
I/O Library Header Files

Let us see the common header files used in C++ programming are:

Header File	Function and Description
<iostream>	It is used to define the **cout, cin and cerr** objects, which correspond to standard output stream, standard input stream and standard error stream, respectively.
<iomanip>	It is used to declare services useful for performing formatted I/O, such as **setprecision and setw.**
<fstream>	It is used to declare services for user-controlled file processing.

Data Types in C++

Identifiers	Keywords
Identifiers are the names defined by the programmer to the basic elements of a program.	Keywords are the reserved words whose meaning is known by the compiler.
It is used to identify the name of the variable.	It is used to specify the type of entity.
It can consist of letters, digits, and underscore.	It contains only letters.
It can use both lowercase and uppercase letters.	It uses only lowercase letters.
No special character can be used except the underscore.	It cannot contain any special character.
The starting letter of identifiers can be lowercase, uppercase or underscore.	It can be started only with the lowercase letter.
It can be classified as internal and external identifiers.	It cannot be further classified.
Examples are test, result, sum, power, etc.	Examples are 'for', 'if', 'else', 'break', etc.



Storage Class	Keyword	Lifetime	Visibility	Initial Value
Automatic	auto	Function Block	Local	Garbage
Register	register	Function Block	Local	Garbage
Mutable	mutable	Class	Local	Garbage
External	extern	Whole Program	Global	Zero
Static	static	Whole Program	Local	Zero

Concepts of C

Static Memory Allocation:

- In case of **static memory allocation**, memory is **allocated at compile time**, and **memory can't be increased while executing the program**. It is used in the array.
- The lifetime of a variable in static memory is the lifetime of a program.
- The static memory is **allocated using the static keyword**.
- The static memory is **implemented using stacks or heaps**.
- The pointer is required to access the variable present in the static memory.
- **Static memory is faster than dynamic memory**.
- In static memory, **more memory space is required to store the variable**.

1. **For example:**
2. **int a[10];**

Dynamic Memory Allocation:

- In case of **dynamic memory allocation,** memory is **allocated at runtime** and **memory can be increased while executing the program**. It is used in the linked list.
- The **malloc() or calloc() function** is required to **allocate the memory at the runtime.**
- An **allocation or deallocation of memory** is done at the execution time of a program.
- No dynamic pointers are required to access the memory.
- The dynamic memory is **implemented using data segments.**
- **Less memory space** is required to store the variable.

1. **For example**
2. **int *p= malloc(sizeof(int)*10);**

Functions used in Dynamic Allocation?

1) Malloc:

- The malloc() function is used to **allocate the memory during the execution of the program.**
- It does **not initialize the memory** but **carries the garbage value**.
- It returns **a null pointer** if it is **not able to allocate the requested space.**

Syntax

- **ptr = (cast-type*) malloc(byte-size) // allocating the memory using malloc() function.**

2)calloc(): The **calloc() is same as malloc()** function, but the **only difference is that it initializes the memory with zero value**.

Syntax: ptr = (cast-type*)calloc(n, element-size);// allocating the memory using calloc() function.

3)realloc():

- The realloc() function is used to **reallocate the memory to the new size.**
- If **sufficient space is not available in the memory,** then the **new block is allocated to accommodate the existing data.**

4)free(): The free() function releases the **memory allocated by either calloc() or malloc() function.**

Syntax : free(ptr); // memory is released using free() function.

	calloc()	malloc()
Description	The malloc() function allocates a single block of requested memory.	The calloc() function allocates multiple blocks of requested memory.
Initialization	It initializes the content of the memory to zero.	It does not initialize the content of memory, so it carries the garbage value.
Number of arguments	It consists of two arguments.	It consists of only one argument.
Return value	It returns a pointer pointing to the allocated memory.	It returns a pointer pointing to the allocated memory.

Pointer in C:

A pointer is a variable that **refers to the address of a value**. It makes the **code optimized and makes the performance fast**. Whenever a **variable is declared inside a program**, then the **system allocates some memory** to a variable. The **memory contains some address numbers**. The v**ariable that holds this address number** is known as the pointer variable. **Data_type *p;**

Usage of Pointer:

- **Accessing array elements:** Pointers are used in traversing through an array of integers and strings. The string is an array of characters which is terminated by a null character '\0'.
- **Dynamic memory allocation:** Pointers are used in allocation and deallocation of memory during the execution of a program.
- **Call by Reference:** The pointers are used to pass a reference of a variable to another function.
- **Data Structures like a tree, graph, linked list, etc**.: The pointers are used to construct different data structures like tree, graph, linked list, etc.

NULL Pointer: A pointer that **doesn't refer to any address of value but NULL** is known as a NULL pointer. When we assign a **'0' value to a pointer of any type**, then it becomes a Null pointer**.**

Dangling Pointer:

- If a **pointer is pointing to any memory location**, but **meanwhile another pointer deletes the memory occupied by the first pointer** while the first pointer still points to that memory location, **the first pointer will be known as a dangling pointer.** This problem is known as a dangling pointer problem.
- Dangling pointer arises when an object is **deleted without modifying the value of the pointer.** The pointer points to the deallocated memory.
- The problem of a **dangling pointer can be overcome by assigning a NULL** value to the dangling pointer

Basis for comparison	Local variable	Global variable
Declaration	A variable which is declared inside function or block is known as a local variable.	A variable which is declared outside function or block is known as a global variable.
Scope	The scope of a variable is available within a function in which they are declared.	The scope of a variable is available throughout the program.
Access	Variables can be accessed only by those statements inside a function in which they are declared.	Any statement in the entire program can access variables.
Life	Life of a variable is created when the function block is entered and destroyed on its exit.	Life of a variable exists until the program is executing.
Storage	Variables are stored in a stack unless specified.	The compiler decides the storage location of a variable.

OOPS

Q)What is oops?

Ans: Object Oriented programming (OOP) is a **programming paradigm** that relies on the concept of **classes** and **objects**. It is used to structure a **software program into simple, reusable pieces of code blueprints (usually called classes),** which are used to create individual instances of objects.

Procedural programming is about writing **procedures or functions** that **perform operations on the data**, while object-oriented programming is about **creating objects** that contain **both data and functions.**

Object-oriented programming has several **advantages** over procedural programming:

- OOP is **faster** and **easier** to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the **C++ code DRY "Don't Repeat Yourself",** and makes the code easier to maintain, modify and debug
- OOP makes it possible to **create full reusable applications with less code and shorter development time**

Q)What is an Object?

Ans: Object is a **real world entity,** for example, a chair, car, pen, mobile, laptop etc.

In other words, an object is an entity that **has state and behavior**. Here, **state means data** and **behavior means functionality**.

Object is a **runtime entity**, it is created at runtime.

Object is an **instance of a class**. All the members of the class can be accessed through objects.

An object consists of -

1. State: It is represented by **attributes of an object**. It also reflects the **properties of an object.**

2. Behavior: It is represented by **methods of an object**. It also reflects the response of an **object with other objects**.

3. Identity: It gives a unique name to an object and enables one object to interact with other objects.

Q)What is Class?

Ans: The class is a **user-defined data type** which **defines its properties** and **its functions.**

Class is a **group of similar objects**.A class is a user **defined blueprint or prototype** from **which objects are created**.

Classes can also **contain functions**, called **methods** available **only to objects of that type.** These functions are defined within the class and perform some action helpful to that specific type of object.

In general, class declarations can include these components, in order:

1. Modifiers: A class can be **public or has default access.**

2. Class name: The name should begin with an initial letter (capitalized by convention).

3. Superclass(if any): The name of the **class's parent (superclass),** if any, preceded by the keyword. A class can only extend (subclass) one parent.

4. Interfaces(if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.

5. Body: The class body surrounded by braces, { }.

```
class Student {

  public:
```

```
int id;//data member (also instance variable)
  string name;//data member(also instance variable)
 };
```

Class consists of:

i)**Data members**

ii)**Members function**

Above two can be accessed within the class,outside the class,and/or inherited by the class.

we can access class using a term known as **Modifiers:**

Modifiers are of three types:

1)Public : Objects can access the **data members and function outside** the class.

2)Private : Objects cannot access the data members and function outside the class.These members can **only be accessed inside the class**.

3)Protected: Objects cannot access the data members and function outside the class.These members can be **accessed inside the class and inherited class**.

https://www.javatpoint.com/access-modifiers

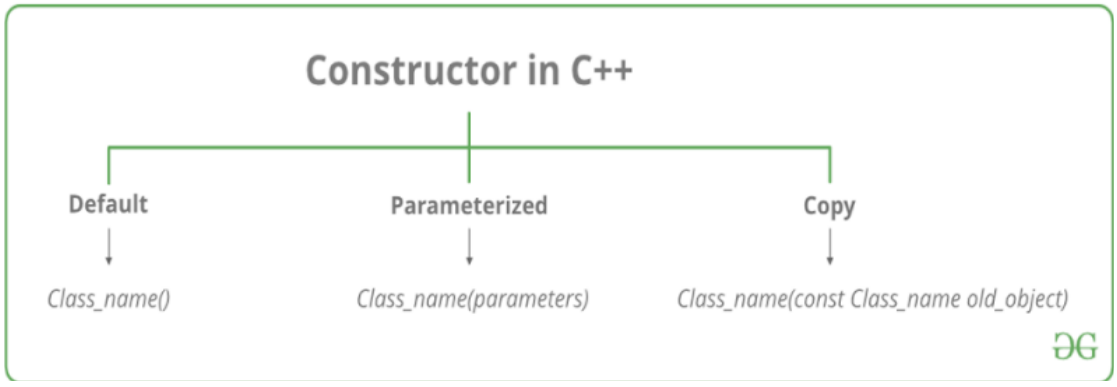
Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Q) What is Constructor?

Ans: A constructor is a **special type of member function of a class** which **initializes objects of a class.**
In C++, **Constructor is automatically** called **when an object(instance of class) is created.**
It is a special member function of the class because **it does not have any return type**.

A constructor is different from normal functions in the following ways:

- Constructor has **the same name as the class itself**
- Constructors **don't have return type**
- A constructor is **automatically called when an object is created.**
- It must be placed in the public section of the class.
- If we do not specify a constructor, C++ compiler **generates a default constructor for the objec**t (expects **no parameters and has an empty body).**



1. Default Constructors: Default constructor is the **constructor which doesn't take any arguments. It has no parameters**.

2. Parameterized Constructors: A **constructor which has parameters is called a parameterized constructor.** It is used **to provide different values to distinct objects.**

3. Copy Constructor: A copy constructor is a member function which initializes **an object using another object of the same class**.

https://www.javatpoint.com/cpp-copy-constructor

Q) What is a Destructor?

Ans: A destructor works **opposite to a constructor**; it **destructs the objects of classes.** It can be defined only once in a class. Like constructors, **it is invoked automatically.**

A destructor is defined like a constructor. It must have the **same name as the class.** But it is **prefixed with a tilde sign (~).**

```
class student{
    string name; // by default all are private
    public:

    int age;
    bool gender;

    void setName(string s) // setter function
    {
        name = s;
    }

    // Default Constructor
    // it does shallow copy
    //in shallow copy only pointer gets copied and not there address they
are pointing
```

```
    student()
    {
        cout << "Default constructor" << endl;
    }

    // parameterised constructor
    student(string s,int a,int g)
    {
        cout << "parameterised constructor" << endl;
        name = s;
        age = a;
        gender = g;
    }
      // shallow copy - only data
    // deep copy - both data and address


    // destructor
    ~student(){
         cout << "Destructor called" << endl;
    }

    void getName(){   // getter function
        cout <<"Name = ";
        cout << name << endl;
    }

    void printInfo(){
        cout <<"Name = ";
         cout << name << endl;
        cout <<"Age = ";
        cout << age << endl;
        cout <<"Gender = ";
        cout << gender << endl;
    }

    // operator overloading
   bool operator == (student &a){
        if(name==a.name && age==a.age && gender==a.gender)
        {
```

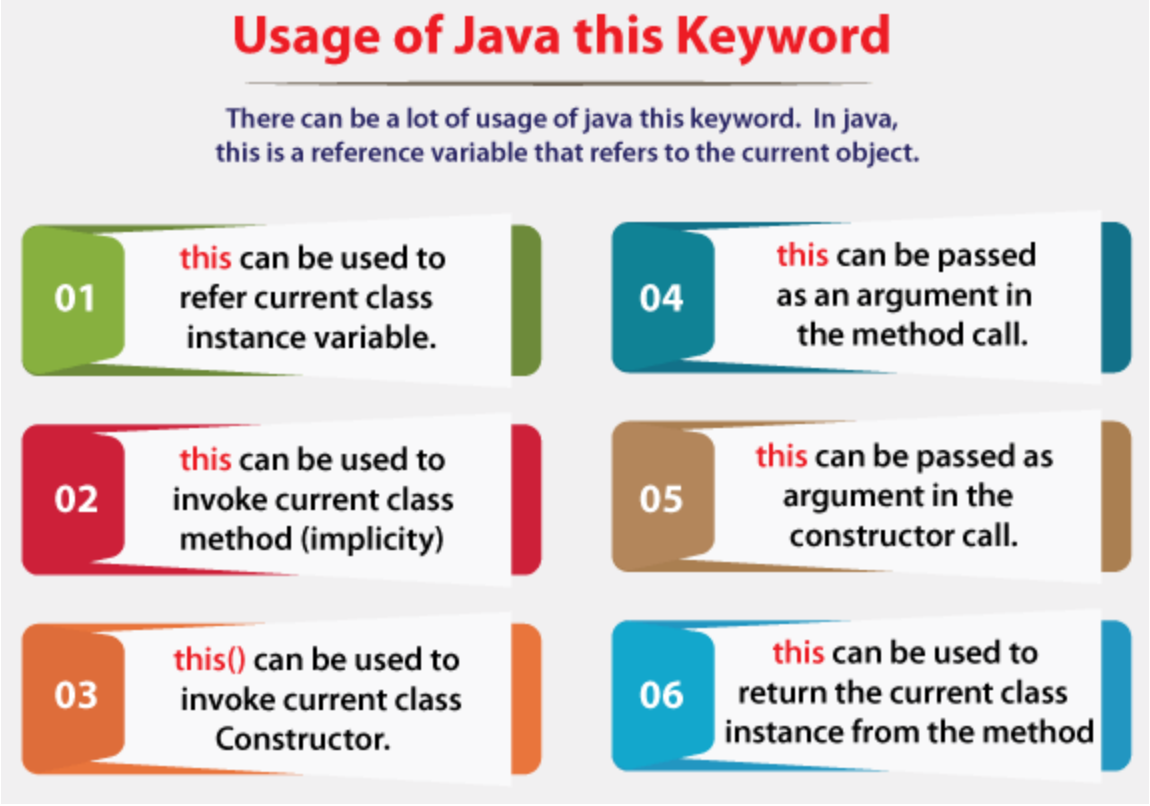
```
            return true;
        }
        else
        {
            return false;
        }
    }
};


int main()
{
  /* student arr[3];
   for(int i=0;i<3;i++)
   {
       string s;
       cin>>s;
       arr[i].setName(s);
       cin>>arr[i].age;
       cin>>arr[i].gender;
   }
   for(int i=0;i<3;i++)
   {
       arr[i].getName();
       arr[i].printInfo();
   } */

   student a("Sumit",21,1);
 //  a.printInfo();
   student b;
   student c=a; // copy constructor
//   c.printInfo();

   if(c==a) cout << "same" << endl;
   else cout << "not same" << endl;
   return 0;
}
```

Q) this Keyword?

Ans: this is a keyword that refers to the **current instance of the class**.

There can be 3 main usages of this keyword in C++.

- **It can be used to pass the current object as a parameter to another method.**
- **It can be used to refer to the current class instance variable.**
- **It can be used to declare indexers.**



Q) Static Keyword?

Ans: **Static** is a **keyword or modifier that belongs to the type**, not instance.So instance is not required to access the static members.In C++, static can be field, method, constructor, class, properties, operator and event.

Advantage of C++ static keyword

Memory efficient: Now we don't need to create an instance for accessing the static members, so it saves memory. Moreover, it belongs to the type, so it will not get memory each time an instance is created.

C++ Static Field

A field which is declared as static is called a static field. Unlike an instance field which gets memory each time whenever you create an object, there is only one copy of the static field created in the memory. It is shared with all the objects.

https://www.javatpoint.com/cpp-static

Q) What is structure ? and how is it different from classes?

Ans: Classes and structs are blueprints that are used to create the instance of a class.

C++ Structure is a **collection of different data types.** It is similar to the class that holds different types of data.

```
struct Student  {
   char name[20];
    int id;
    int age;  } ;
```

https://www.javatpoint.com/cpp-structs

Structure vs class in C++

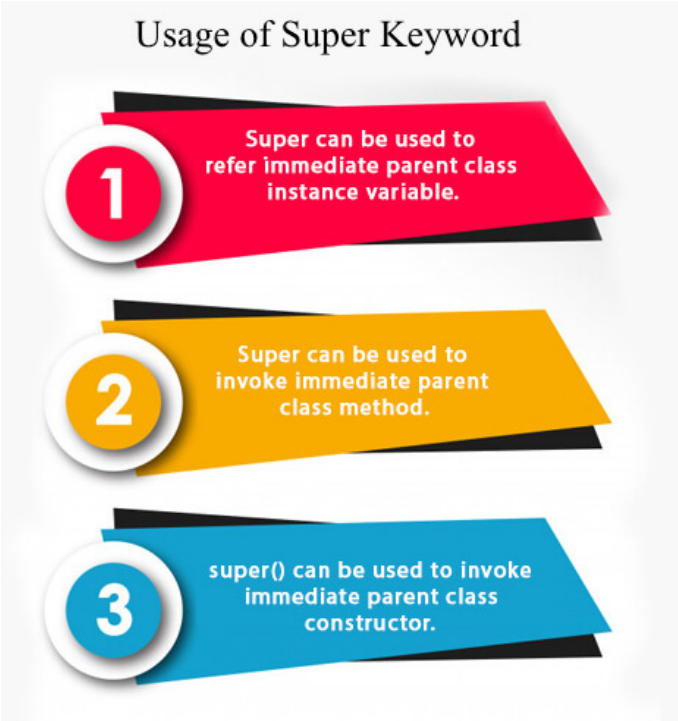
Enumeration : Enum in C++ is a data type that contains a fixed set of constants.

Friend Function : If a function is defined as a friend function in C++, then the **protected and private data of a class can be accessed using the function**.
For accessing the data, the declaration of a friend function **should be done inside the body of a class starting with the keyword friend**.

New Keyword : **new** keyword is used to create an instance of the class. In other words, it instantiates a class by allocating memory for a new object and returning a reference to that memory.

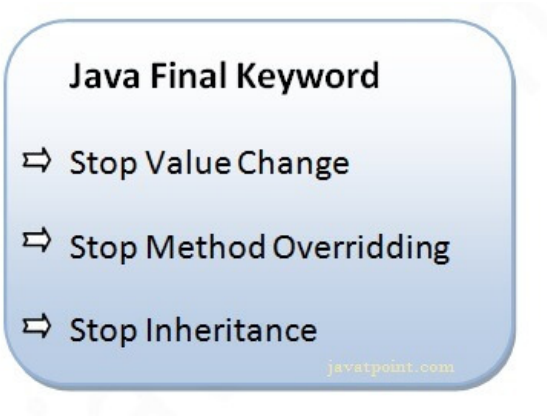
Super Keyword : The **super keyword** in Java is a **reference variable** which is used to **refer to the immediate parent class object**.

Whenever you **create the instance of a subclass**, an instance of the **parent class is created implicitly which is referred to by a super reference variable**.



Final Keyword: The final keyword in java is used to restrict the user. The java final keyword can be used in many contexts. Final can be:

1. variable
2. method
3. class



Encapsulation: Hiding sensitive data from user

Encapsulation is a mechanism **which binds the data and associated operations together** and **thus hides the data from the outside world**. Encapsulation is also known as data hiding. In C++, It is achieved using the **access specifiers, i.e., public, private and protected.**

Advantages of Encapsulation :

i)Good coding practice

ii)Increase security.

```
class A{
    public:
    int a;
    void funcA()
    {
        cout << "Func A \n" ;
    }
    private:
    int b;
    void funcB()
    {
         cout << "Func B \n" ;
    }
    protected:
    int c;
    void funcC()
    {
         cout << "Func C \n" ;
    }

};
```

```
int main()
{
   A obj;
   obj.funcA();
   obj.funcB(); // will give an error of being private
   obj.funcC(); // will give an error of being protected

   return 0;}
```

Types of relation between classes:

1)Inheritance(IS-A relationship)
2)Association(Has-A):- Aggregation and composition

Inheritance (IS-A relation)

Inheritance is a process in which **one object acquires all the properties and behaviors of its parent object automatically**. In such a way, you can **reuse, extend or modify the attributes and behaviors which are defined in other classes**.
It is possible to inherit attributes and methods from one class to another.
Only the **public and protected** are inherited.
i) **Derived class(child class)**:The class that inherits from another class.
ii) **Base Class(Parent class)**:The class being inherited from.

Types of Inheritance:

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hybrid Inheritance
- Hierarchical Inheritance

Link: https://www.javatpoint.com/cpp-inheritance

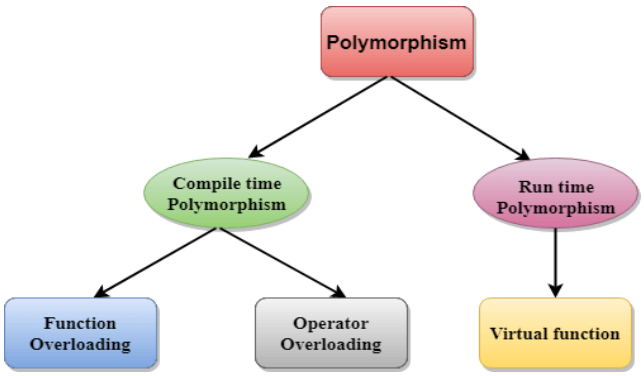
```cpp
#include <iostream>
using namespace std;
 class Account {
   public:
   float salary = 60000;
 };
   class Programmer: public Account {
   public:
   float bonus = 5000;
   };
int main(void) {
     Programmer p1;
     cout<<"Salary: "<<p1.salary<<endl;
     cout<<"Bonus: "<<p1.bonus<<endl;
    return 0;
}
```

Association: Has **a non-blood relation unlike inheritance**.in this **class is not tightly coupled**. By using a **ref variable or new keyword**.

Aggregation: Aggregation is a process in **which one class defines another class as any entity reference**. It is another way to reuse the class. It is a form of association that represents the HAS-A relationship.
weak bonding b/w class.
Composition: Strong bonding b/w classes.

Polymorphism :

The term "Polymorphism" is the combination of "poly" + "morphs" which **means many forms**. **Polymorphism** refers to the ability of OOPs programming languages to **differentiate between entities with the same name efficiently.**



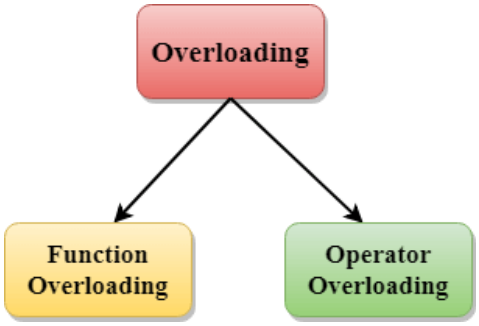
Differences b/w compile time and run time polymorphism.

Compile time polymorphism	Run time polymorphism
The function to be invoked is known at the compile time.	The function to be invoked is known at the run time.
It is also known as overloading, early binding and static binding.	It is also known as overriding, Dynamic binding and late binding.
Overloading is a compile time polymorphism where more than one method is having the same name but with the different number of parameters or the type of the parameters.	Overriding is a run time polymorphism where more than one method is having the same name, number of parameters and the type of the parameters.
It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and pointers.
It provides fast execution as it is known at the compile time.	It provides slow execution as it is known at the run time.
It is less flexible as mainly all the things execute at the compile time.	It is more flexible as all the things execute at the run time.

Overloading:

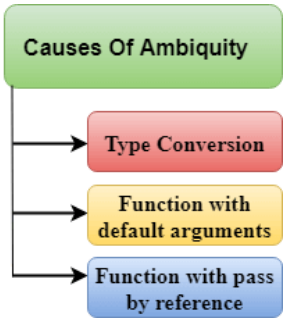
If we create two or more members having the **same name but different in number or type of parameter,** it is known as C++ overloading. In C++, we can overload:

- methods,
- constructors, and
- indexed properties



Function Overloading: Function Overloading is defined as the process of having **two or more functions with the same name, but different in parameters** is known as function overloading in C++.
In function overloading, the function is redefined by using either different types of arguments or a different number of arguments. It is only through these differences that the compiler can differentiate between the functions.

The **advantage** of Function overloading is that **it increases the readability of the program** because you don't need to use different names for the same action.



Operator Overloading: Operator overloading is a **compile-time polymorphism** in which the **operator is overloaded to provide** the **special meaning to the user-defined data type**. Operator overloading is used to overload or redefine most of the operators available in C++.
It is used to perform the operation on **the user-defined data type**. For example, C++ provides the ability to add the variables of the user-defined data type that is applied to the built-in data types.

The advantage of operator overloading is to perform different operations on the same operand.

Overriding : If a **derived class defines the same function as defined in its base class,** it is known as function overriding in C++. **It is used to achieve runtime polymorphism**. It enables you to provide **specific implementations of the function which is already provided by its base class.**

Virtual Function:

- A virtual function is used to **replace the implementation provided by the base class.** The replacement is always called whenever the object in question is actually of the derived class, even if the object is accessed by a base pointer rather than a derived pointer.
- A virtual function is a member function which is present in the base class and redefined by the derived class.
- When we use the **same function name in both base and derived classes, the function in the base class is declared with a virtual keyword.**
- When the function is made virtual, then C++ determines at run-time which function is to be called based on the type of the object pointed by the base class pointer. Thus, by making the base class pointer to point to different objects, we can execute different versions of the virtual functions.

Rules of a virtual function:

- The virtual functions should be a member of some class.
- The virtual function cannot be a static member.
- Virtual functions are called by using the object pointer.
- It can be a friend of another class.
- C++ does not contain virtual constructors but can have a virtual destructor.

Abstraction:

Abstraction is used to **hide the internal implementations** and **show only the necessary details to the outer world.**
Abstraction in C++ is the process to **hide the internal details** and show **functionality only.**
If the members are defined with a **public keyword, then the members are accessible outside. If the members are defined with a private keyword, then the members are not accessible by outside methods.**

- C++ provides a great level of abstraction. For example, **the pow() function is used to calculate the power of a number** without knowing the algorithm the function follows.

Data abstraction is implemented using **interfaces and abstract classes in C++**.

In C++ class is made **abstract by declaring** at least one of its functions as a **strong pure virtual function**.
A pure virtual function is specified by **placing "= 0" in its declaration**. Its implementation must be provided by derived classes.

Abstraction using classes: An abstraction can be achieved using classes. A class is **used to group all the data members** and **member functions into a single unit by using the access specifiers**. A class has the responsibility to determine which data member is to be visible outside and which is not.

Abstraction in header files: Another type of a**bstraction is header file**. For example, the pow() function is used to calculate the power of a number without actually knowing which algorithm function is used to calculate the power. Thus, we can say that header files hide all the implementation details from the user.

NameSpaces

- The namespace is a **logical division of the code which is designed to stop the naming conflict**.
- The namespace defines the scope where the identifiers such as variables, class, and functions are declared.
- The main purpose of using namespace in C++ is to remove the ambiguity. Ambiguity occurs when a different task occurs with the same name.
- For example: if two functions exist with the same name such as add(). In order to prevent this ambiguity, the namespace is used. Functions are declared in different namespaces.
- C++ consists of a standard namespace, i.e., std which contains inbuilt classes and functions. So, by using the statement "using namespace std;" includes the namespace "std" in our program.

Exception Handling:

Exception Handling in C++ is a process to handle runtime errors. We perform exception handling so the normal flow of the application can be maintained even after runtime errors.

In C++, we use 3 keywords to perform exception handling:

- try
- catch, and
- throw