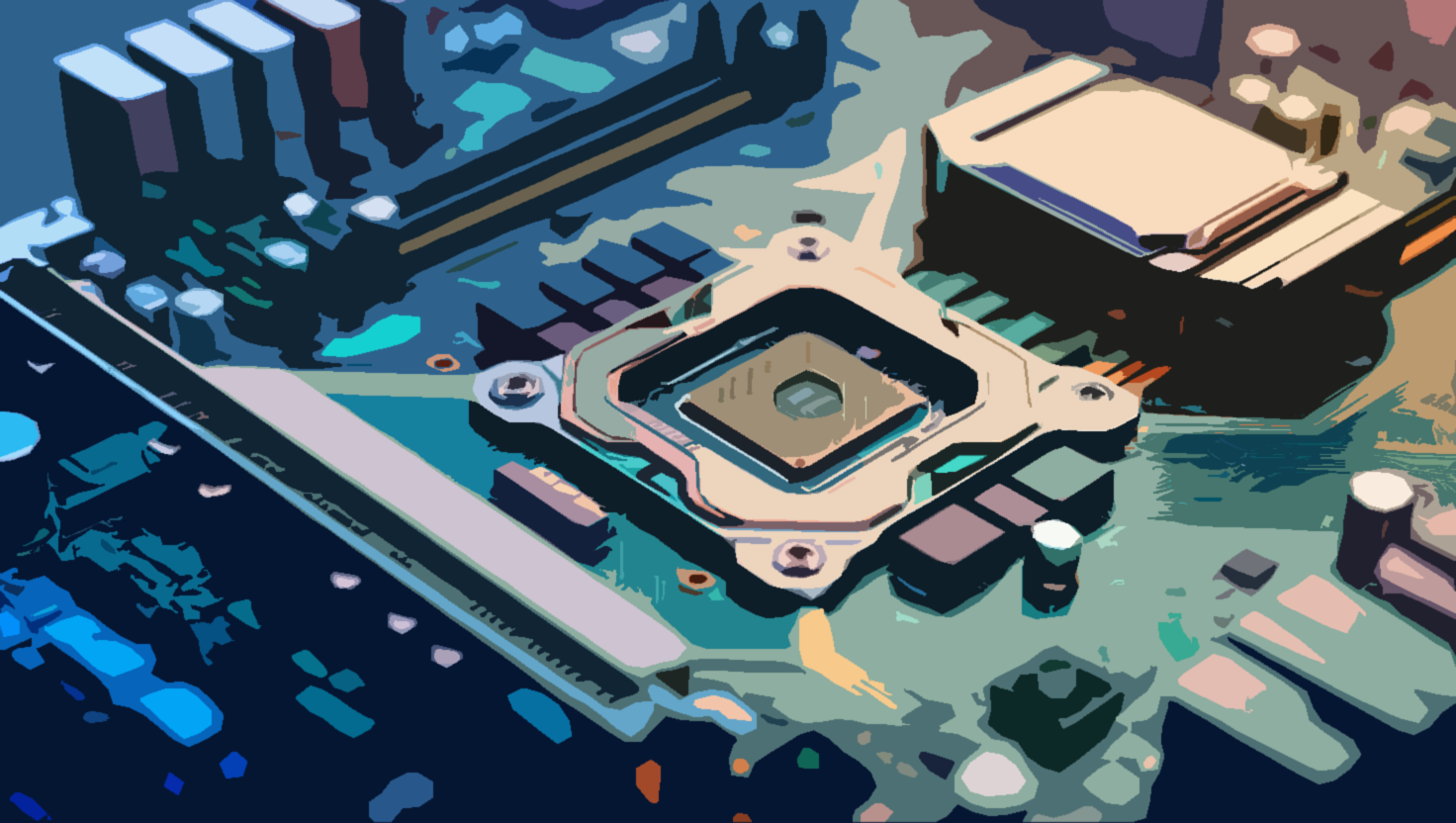
****

**C++ Notes:**

**🎯** #include <iostream>   
using namespace std;   
int main () {   
  cout << "Hello World!";   
  return 0;   
} // The main difference between C and C++ is that C++ supports classes and objectss

What is C++ ?

* **C++** is a powerful, cross-platform programming language used to build high-performance applications.
* It was developed by **Bjarne Stroustrup** as an extension of the C programming language.
* Widely used in **game development**, **system software**, and **performance-critical applications**, C++ remains one of the most popular programming languages.
* It offers programmers **fine-grained control** over system resources and memory, making it ideal for low-level programming tasks.

Why Use C++?

* C++ plays a crucial role in the development of **operating systems**, **graphical user interfaces (GUIs)**, and **embedded systems**.
* As an **object-oriented programming (OOP)** language, it promotes clean code structure, **modularity**, and **code reuse**, leading to lower development costs.
* Its **portability** ensures that applications can be easily adapted across various platforms.
* C++ shares similarities with **C, C#, and Java**, making it easier for developers familiar with those languages to transition smoothly.

1. #include <iostream> is a **header file library** that lets us work with input and output objects. It add functionality to C++ programs.

2. using namespace std means that we can use **names** for objects and variables from the standard library.

--**C++ considers blank lines ignores in compiler.**

3. cout is an **object** used together with the *insertion operator* (<<) to output/print text.

4. **Multi-line comments start with /\* and ends with \*/.**

5. **Variables** are containers for storing data values:

* Int - stores integers (whole numbers), without decimals, such as 123.
* double –It stores floating point numbers, with decimals, such as 19.99
* Char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
* String - stores text, such as "Hello World". String values are surrounded by double quotes
* bool - stores values with two states: true or false

## **Syntax: type variableName = value;**

6. The general rules for naming variables are:

* Names can contain letters, digits and underscores
* Names must begin with a letter or an underscore (\_)
* Names are case-sensitive (myVar and myvar are different variables)
* Names cannot contain whitespaces or special characters like! #, %.
* Reserved words (like C++ keywords, such as int) can’t be used.

7. **Constants**: When you do not want others (or yourself) to change existing variable values, use the const keyword (which means **unchangeable and read-only**)

When you declare a constant variable, it must be assigned with a value:

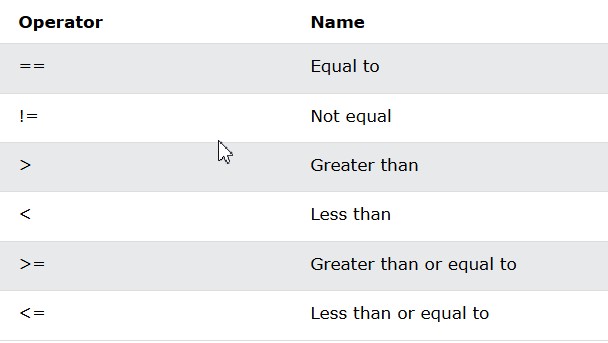
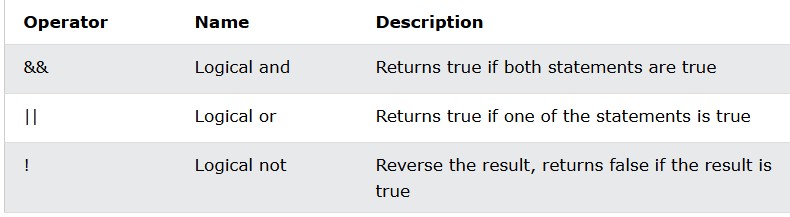
Like this:

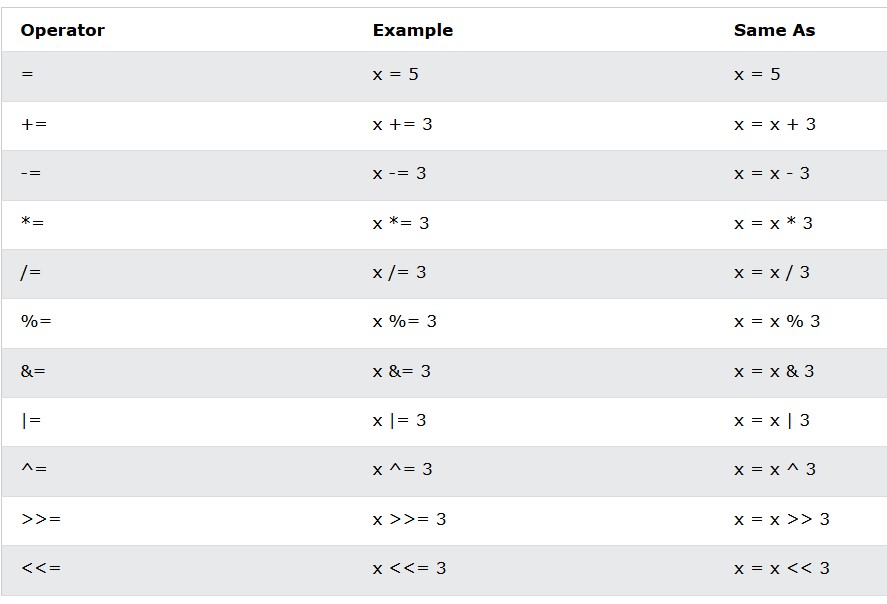
Const int minutesPerHour = 60;

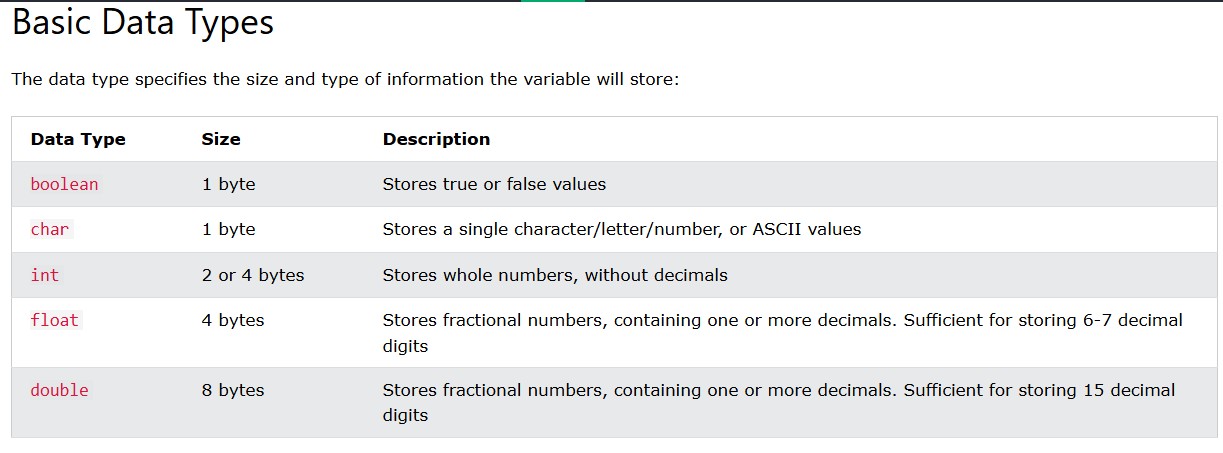
This however, **will not work**:

Const int minutesPerHour;   
minutesPerHour = 60; // error

8. Float vs. double

The **precision** of a floating point value indicates how many digits the value can have after the decimal point. The precision of float is only six or seven decimal digits, while double variables have a precision of about 15 digits. Therefore it is safer to use double for most calculations.





9.[Strings](file:///C:\Users\Gateway\Desktop\Programs\C++\14.cpp) : include <string>;Strings are used for storing text/characters.

10. Array is a collection of items of similar type stored in contiguous method.

**🎯** Sometime, a simple variable is not enough to hold all data.eg: let say we want to store the marks of 2500 stud, having 2500 variable for this task is not feasible.

**🎯** To solve this we can use an array with size 2500 that can hold marks.

EG: string cars [4] = {"Volvo", "BMW", "Ford", "Mazda"};   
cout << cars [0]; // Outputs Volvo

**🎯If and else Syntax:**

if (*condition1*) {  
  *// block of code to be executed if condition1 is true*  
} else if (*condition2*) {  
  *// block of code to be executed if the condition1 is false and condition2 is true*  
} else {  
  *// block of code to be executed if the condition1 is false and condition2 is false*  
}

**🎯Short Hand If else(Ternary Operator):**

**Syntax:** *variable* = (*condition*) ? *expressionTrue* : *expressionFalse*;

**🎯Switch:**

**Syntax:** switch(*expression*) {  
  case x:  
    *// code block*  
    break;  
  case y:  
    *// code block*  
    break;  
  default:  
    *// code block*  
}

**🎯**while (*condition*) {  
*// code block to be executed*  
}

**🎯**do {  
*// code block to be executed*}  
while (*condition*);

**🎯**for (*Initial code1*;*Condition block*; *Iteration block*) {  
 *// code block to be executed*  
}

**🎯**[**Structures**](https://www.w3schools.com/cpp/cpp_structs.asp)**,** [**Enums**](https://www.w3schools.com/cpp/cpp_enum.asp)**.**

**🎯** A reference variable is a "reference" to an existing variable, and it is created with the & operator

Eg: string food = "Pizza";   
string &meal = food;   
  
cout << food << "\n"; // Outputs Pizza  
cout << meal << "\n"; // Outputs Pizza

11. [**Object-oriented programming**](https://www.w3schools.com/cpp/cpp_classes.asp) is about creating objects that contain both data and functions.

* OOP is faster and easier to execute
* 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
* Principal of data hiding helps build secure systems.

Tip**:** The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

**🎯** Class is a template for objects and an object is an instance of a class.  C++ is associated with classes and objects, along with attributes and methods. Eg: in real life, a car is an **object**. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake. Procedural programming language (Ppl)

Eg: class (Fruit) & objects apple-banana-mango.

* C++ language was designed with the main features of adding OO features to c Lang.As the size of program increases readability, main ability and bug free nature of program decreases.
* Major problem of c was relied upon func or procedures (hence the name ppl came into standby). Data was almost neglected data security was easily compromised so possibilities of not addressing problem was very high.
* Ppl : Consist of writing a set of instr for comp to follow. Main focus is on func and not on the flow of data.
* Func can be either use local or global data. Data moves openly from func1 to func2.

Terminology:

* **Objects**: Basic run time entities.
* **Data abstraction and encapsulation**: wrapping data and function into single unit.
* **Inheritance**: Properties off one class can be inherited into others.
* **Polymorphism**: ability to take more than one forms.
* **Message Passing-Object.** Function (Info) call format.
* **Dynamic Binding**: code which will execute is not known until the program
* [**Encapsulation**](https://www.w3schools.com/cpp/cpp_encapsulation.asp): It is to make sure that "sensitive" data is hidden from us

-ers. To achieve this, you must declare class variables/attributes as private (cannot be accessed from outside the class). If you want others to read or modify the value of a private member, you can provide public **get** and **set** methods.

**🎯** [Inheritance](https://www.w3schools.com/cpp/cpp_inheritance.asp)

What reason C++ is used for ?

• In C++ we can reuse a class and add additional features to it and saves time &cash.

• Reusing already tested and debugged classes will save a lot of effort of developing and debugging the same thing again

• The concept of reusability in C++ is supported using inheritance

• We can reuse the properties of an existing class by inheriting it

## Single Inheritance in C++

Single inheritance is a type of inheritance in which a derived class is inherited with only one base class. For example, we have two classes “employee” and “programmer”. If the “programmer” class is inherited from the “employee” class which means that the “programmer” class can now implement the functionalities of the “employee class”

## Multiple Inheritances in C++

Multiple inheritances are a type of inheritance in which one derived class is inherited with more than one base class. For example, we have three classes “employee”, “assistant” and “programmer”. If the “programmer” class is inherited from the “employee” and “assistant” class which means that the “programmer” class can now implement the functionalities of the “employee” and “assistant” class.

[**Syntax for inheriting in mutliple inheritance**](file:///C:\Users\Gateway\Desktop\Programs\C++\40.cpp):

**class DerivedC: visibility-mode base1, visibility-mode base2**

**{**

**Class body of class "DerivedC"**

**};**

## Hierarchical Inheritance

A hierarchical inheritance is a type of inheritance in which several derived classes are inherited from a single base class. For example, we have three classes “employee”, “manager” and “programmer”. If the “programmer” and “manager” classes are inherited from the “employee” class which means that the “programmer” and “manager” class can now implement the functionalities of the “employee” class.

## Hybrid Inheritance in C++

Hybrid inheritance is a combination of multiple inheritance and multilevel inheritance. In hybrid inheritance, a class is derived from two classes as in multiple inheritances. However, one of the parent classes is not a base class. For example, we have four classes “animal”, “mammal”, “bird”, and “bat”. If “mammal” and “bird” classes are inherited from the “animal” class and “bat” class is inherited from “mammal” and “bird” classes which means that “mammal” and “bird” classes can now implement the functionalities of “animal” class and “bat” class can now implement the functionalities of “mammal” and “bird” classes.

In C++, it is possible to inherit attributes and methods from one class to another:

* **derived class** (child) - the class that inherits from another class
* **base class** (parent) - the class being inherited from

// Base class  
class Vehicle {  
 public:  
    string brand = "Ford";  
    void honk () {  
      cout << "Tutu, tutu! \n" ;  
    }  
};  
  
// Derived class  
**class Car: public Vehicle** {  
  public:  
    string model = "Mustang";  
};  
  
int main() {  
  Car myCar;  
  myCar.honk();  
  cout << myCar.brand + " " + myCar.model;  
  return 0;

**Friend function can access private members of a class to add two complex**

class Complex {

private:

int a, b; // Private data members

public:

void setNumber(int x, int y); // Set real and imaginary parts

void printNumber(); // Print the complex number

friend Complex addComplex(Complex c1, Complex c2); // Friend function

};

Complex:: addComplex(Complex c1, Complex c2) {

Complex c3;

c3.setNumber(c1.a + c2.a, c1.b + c2.b);

return c3;

}

**🎯 Key Concepts Demonstrated:**

| **Feature** | **Description** |
| --- | --- |
| **Friend Function** | addComplex() has access to private members of Complex. |
| **Encapsulation** | Private data members, accessed only through interface. |
| [**Object Usage**](https://www.w3schools.com/cpp/cpp_class_methods.asp) | Objects passed as arguments to non-member function. |

**🚀**  Friend Function Properties **(Condensed):**

1. **Not in class scope**, but declared with friend.
2. Can access **private/protected** members and Can be declared in **public or private**.
3. **Invoked like a regular function** (not with object).
4. Requires **object references** to access members.
5. **Not inherited** by derived classes.

**🚀** Interview-Ready Enhancements**:**

**1. Use const references for efficiency:**

friend Complex addComplex(const Complex &c1, const Complex &c2);

**2. Add a Constructor for initialization:**

Complex(int x = 0, int y = 0) { a = x; b = y; }

**3. Overload the + Operator (alternate to friend function):**

Complex operator+(const Complex &obj) {

return Complex(a + obj.a, b + obj.b);

}

**4. Overload << Operator for better output:**

friend ostream& operator<<(ostream &out, const Complex &c) {

out << c.a << " + " << c.b << "i";

return out;

}

**🎯** Constructor is a **special method** that is automatically called when an object of a class is created.

To create a constructor, use the same name as the class, followed by parenthesis ():

class MyClass {     // The class  
  public:           // Access specifier  
    MyClass() {     // Constructor  
      cout << "Hello World!";  
    }  
};  
  
int main() {  
  MyClass myObj;    // Create an object of MyClass (this will call the constructor)  
  return 0;  
}

**Note:** The constructor has same name as the class, it is always public, and doesn’t return value.

**🎯** [Constructor Parameters](https://www.w3schools.com/cpp/cpp_constructors.asp) :

Constructors can also take parameters (just like regular functions).

When we call the constructor (by creating an object of the class), we pass parameters to the constructor, which will set the value of the corresponding attributes to the same:

Just like functions, constructors can also be defined outside the class. First, declare the constructor inside the class, and then define it outside of the class by specifying the name of the class, followed by the scope resolution :: operator, followed by the name of the constructor (which is the same as the class):

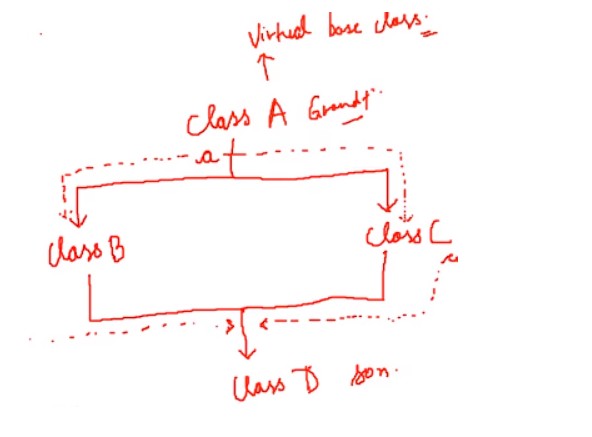
class Car {        // The class  
  public:          // Access specifier  
    string brand;  // Attribute  
    string model;  // Attribute  
    int year;      // Attribute  
    Car(string x, string y, int z); // Constructor declaration  
};  
  
// Constructor definition outside the class  
Car::Car(string x, string y, int z) {  
  brand = x;  
  model = y;  
  year = z;  
}  
  
int main() {  
  // Create Car objects and call the constructor with different values  
  Car carObj1("BMW", "X5", 1999);  
  Car carObj2("Ford", "Mustang", 1969);  
  
  // Print values  
  cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";  
  cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";  
  return 0;  
}

**🎯** [AMBIGUITY RESOLUTION IN INHERITANCE](file:///C:\Users\Gateway\Desktop\Programs\C++\42.cpp)

Ambiguity in inheritance can be defined as when one class is derived for two or more base classes then there are chances that the base classes have functions with the same name. So it will confuse derived class to choose from similar name functions. To solve this ambiguity scope resolution operator is used “::”

**🎯** [Virtual Classroom](file:///C:\Users\Gateway\Desktop\Programs\C++\43.cpp) :

The virtual base class is a concept used in multiple inheritances to prevent ambiguity between multiple instances. For example: suppose we created a class “A” and two classes “B” and “C”, are being derived from class “A”. But once we create a class “D” which is being derived from class “B” and “C” as shown in figure 1.

class A

{

public:

void say()

{

cout << "Hello world" << endl;

}

};

class B : public virtual A

{

};

class C : public virtual A

{

};

class D : public B, public C

{

};

**🎯** [Destructor](https://archive.codewithharry.com/videos/cpp-tutorials-in-hindi-35/) is a type of function which is called when the object is destroyed. Destructor never takes an argument nor does it return any value.

**🎯** Virtual Function: A function that is in the parent class but redefined in the child class is called a virtual function. “virtual” keyword is used to declare a virtual function

[CONSTRUCTORS IN DERIVED CLASS IN C++](file:///C:\Users\Gateway\Desktop\Programs\C++\44.cpp)

* If the base class constructor does not have any arguments, there is no need for any constructor in the derived class
* But if there are one or more arguments in the base class constructor, derived class need to pass argument to the base class constructor
* If both base and derived classes have constructors, base class constructor is executed first

Constructors in Multiple Inheritances

* In multiple inheritances, base classes are constructed in the order in which they appear in the class deceleration. For example if there are three classes “A”, “B”, and “C”, and the class “C” is inheriting classes “A” and “B”. If the class “A” is written before class “B” then the constructor of class “A” will be executed first. But if the class “B” is written before class “A” then the constructor of class “B” will be executed first.
* In multilevel inheritance, the constructors are executed in the order of inheritance. For example if there are three classes “A”, “B”, and “C”, and the class “B” is inheriting classes “A” and the class “C” is inheriting classes “B”. Then the constructor will run according to the order of inheritance such as the constructor of class “A” will be called first then the constructor of class “B” will be called and at the end constructor of class “C” will be called.

Special Syntax

* C++ supports a special syntax for passing arguments to multiple base classes
* The constructor of the derived class receives all the arguments at once and then will pass the call to the respective base classes
* The body is called after the constructors is finished executing

Derived-Constructor (arg1, arg2, arg3….): Base 1-Constructor (arg1,arg2), Base 2-Constructor(arg3,arg4)

{

} Base 1-Constructor (arg1,arg2)

[this](file:///C:\Users\Gateway\Desktop\Programs\C++\49.cpp) : “this” is a keyword that is an implicit pointer. “this” pointer points to the object which calls the member function.

**🎯** [Polymorphism](https://www.w3schools.com/cpp/cpp_polymorphism.asp):

Polymorphism means "many forms". It allows the same interface to behave differently based on the object.[**Inheritance**](https://www.w3schools.com/cpp/cpp_inheritance.asp) lets us inherit attributes and methods from another class.

### 🔹 **Types of Polymorphism in C++**

1. **Compile-Time Polymorphism (Static Binding):**

Compile-time polymorphism, it is already known which function will run. Compile-time polymorphism is also called early binding, which means that you are already bound to the function call and you know that this function is going to run.

* + Achieved using:
    - **Function Overloading**
    - **Operator Overloading:** . Operator loading is also an example of compile-time polymorphism because the compiler already knows at the compile time which operator has to perform the task.
  + Resolved at **compile time**.

1. **Run-Time Polymorphism (Dynamic Binding):**

In the run-time polymorphism, the compiler doesn’t know already what will happen at run time. Run time polymorphism is also called late binding. The run time polymorphism is considered slow because function calls are decided at run time. Run time polymorphism can be achieved from the virtual function.

* + Achieved using:
    - **Virtual Functions**
    - **Base class pointers/references pointing to derived class objects**
  + Resolved at **runtime** using **vtable (virtual table)**.

### 🔹 **Function Overloading (Compile-Time)**

* Same function name, different parameter list.

void show(int);

void show(double);

### 🔹 **Operator Overloading (Compile-Time)**

* Redefine operators for user-defined types.

Complex operator+(Complex c);

### 🔹 **Virtual Functions (Run-Time)**

* Declared using virtual keyword in base class and enables late binding.

class Base {

public:

virtual void show() { cout << "Base"; }

};

class Derived : public Base {

public:

void show() override { cout << "Derived"; }

};

### 🔹 **Key Points**

* Virtual function must be **non-static**.
* Destructors should be declared **virtual** in base classes for proper cleanup.
* **Polymorphism promotes code reusability** and **extensibility**.

## 📁 File I/O in C++

#include <fstream>

There are some useful classes for woring with files in c++:

* fstreambase – Base class
* ifstream – For reading files 🡪derived from fstreambase
* ofstream – For writing files 🡪derived from fstreambase

### Writing to a File using Constructor

ofstream out("filename.txt"); // Automatically opens file for writing

out << "Your text here";

out.close(); // Always close after use

### 📖 Reading from a File using Constructor

ifstream in("filename.txt"); // Automatically opens file for reading

in >> variable; // Reads word by word

in.close();

### Example Program: Read & Write in Same File

Steps:

1. Create sample60.txt in the same directory.
2. Use ofstream to write input to the file.
3. Use ifstream to read from the file.
4. Maintain separate stream objects and use .close() to disconnect.

### Code:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

// Writing to file

ofstream hout("sample60.txt");

string name;

cout << "Enter your name: ";

cin >> name;

hout << name + " is my name";

hout.close();

// Reading from file

ifstream hin("sample60.txt");

string content;

hin >> content; // Reads only up to the first space

cout << "The content of the file is: " << content;

hin.close();

return 0;

}

### 📝 Output Example:

Enter your name: Harry

The content of the file is: Harry

Only "Harry" is printed because hin >> content reads till the first space. Use getline(hin, content) to read the full line.

* open() function to open files dynamically
* eof() to detect end-of-file during reading

### 📚 Useful Classes:

* fstreambase – Base class
* ifstream – For reading files (derived from fstreambase)
* ofstream – For writing files (derived from fstreambase)

### 📂 Ways to Open a File:

1. **Constructor method** (e.g. ofstream out("file.txt");)
2. **Member function** open() (e.g. out.open("file.txt");)

### ✍️ Writing to a File

#include <iostream>

#include <fstream>

using namespace std;

int main() {

string st = "Harry bhai";

ofstream out("sample60.txt"); // Open file for writing

out << st; // Write string to file

return 0;

}

Output: Writes "Harry bhai" to sample60.txt

### Reading from a File

#include <iostream>

#include <fstream>

using namespace std;

int main() {

string st2;

ifstream in("sample60b.txt"); // Open file for reading

in >> st2; // Reads only first word

getline(in, st2); // Reads the entire line

cout << st2; // Display the line

return 0;

}// Output: Displays full line from sample60b.txt