LEARNING HUB

SHAHABAD MARKANDA

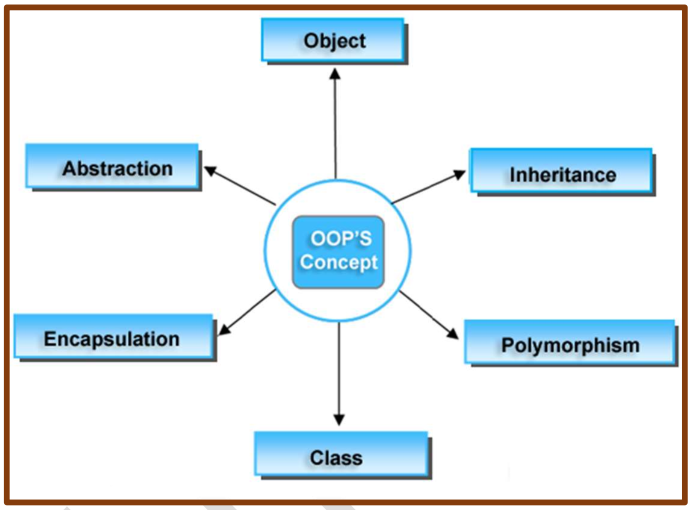
CALL- 77000 90800

- ALL COMPUTER COURSES
- ENGLISH LANGUAGE COURSES
- Brand Marketing & Custom Software Solution
- TUITION NURSERY TO 10TH (ALL SUBJECTS)

WEBSITE: https://learninghubshahabad.in

C++ OOPS

OOPS ek programming approach hai jisme hum program ko objects ke through design karte hain, na ki sirf functions ke through. C++ ek object-oriented language hai jo real-world cheezon ko code ke form mein represent karti hai.



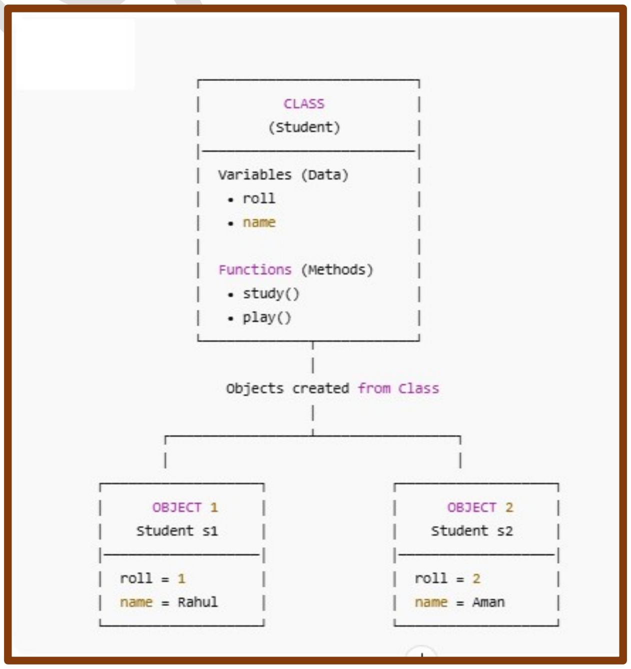
Advantages of OOPS (Object-Oriented Programming System) –

- Code reuse hota hai (same code baar-baar likhne ki zarurat nahi)
- Program well-organized aur structured hota hai
- Maintenance easy ho jaata hai
- Data secure rehta hai (data hiding)
- Real-world problems ko easily represent kar sakte hain
- Code duplication kam hota hai
- Large projects manage karna easy hota hai
- Polymorphism se flexibility milti hai
- Debugging aur testing easy hoti hai
- Development time kam lagta hai

Class and Object

Class

- Class ek blueprint / template hoti hai
- Isme data members (variables) aur member functions (functions) hote hain
- Class sirf design hoti hai, memory tab tak allocate nahi hoti
- 👉 Example:
  Class = Car ka design



Object

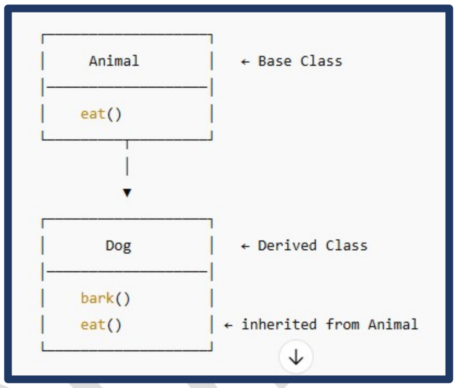
- Object class ka real instance hota hai
- Object banate hi memory allocate hoti hai
- Object class ke data aur functions ko use karta hai
- 👉 Example:
  Object = Actual Car (BMW, Audi)

Types Of Class in C++

Base Class (Parent Class)

- Ye original class hoti hai
- Iske properties aur methods child class inherit kar sakti hai

```
class Animal {      // Base Class
public:
    void eat() {
        cout << "Animal is eating";
    }
};
```



Derived Class (Child Class)

- Ye class Base class ke features inherit karti hai
- Apni extra properties aur methods bhi add kar sakti hai

```
class Dog : public Animal {  // Derived Class
public:
    void bark() {
        cout << "Dog is barking";
    }
};
```

Explanation:

- Animal = Base class (parent)
- Dog = Derived class (child)
- Dog inherits eat() from Animal and has its own bark() method

Employee Management System

Scenario: Company mein employees Regular ya Manager ho sakte hain. Managers ke paas extra properties hoti hain.

```
#include <iostream>
using namespace std;
// Base Class
class Employee {
public:
    string name;
    int id;
    void showEmployee() {
        cout << "Name: " << name << ", ID: " << id << endl;
    }
};
// Derived Class
```

```
Output
Name: Rohan, ID: 101
Department: Sales
```

```
class Manager : public Employee {
public:
    string department;
    void showManager() {
        cout << "Department: " << department << endl;
    }
};
int main() {
    Manager m1;
    m1.name = "Rohan";
    m1.id = 101;
    m1.department = "Sales";

    m1.showEmployee(); // Base class ka method
    m1.showManager();  // Derived class ka method
    return 0;
}
```

Explanation

- Base Class (Employee)
- Common properties rakhti hai: name aur id
- Method showEmployee() employee details display karta hai
- Derived Class (Manager)
- Employee class se inherit karti hai (public inheritance)
- Apni extra property rakhti hai: department
- Method showManager() department display karta hai
- main() mein
- Manager class ka object m1 create kiya
- Base class ke members (name, id) aur derived class ka member (department) assign kiya
- m1.showEmployee() → base class method call
- m1.showManager() → derived class method call

Bank Account System

Scenario: Bank ke paas general accounts hain, aur SavingsAccount ka interest calculate hota hai.

```
#include <iostream>
using namespace std;
// Base Class
class BankAccount {
public:
    string accountHolder;
    double balance;
    void showAccount() {
        cout << "Account Holder: " << accountHolder << ", Balance: $" << balance << endl;
    }
```

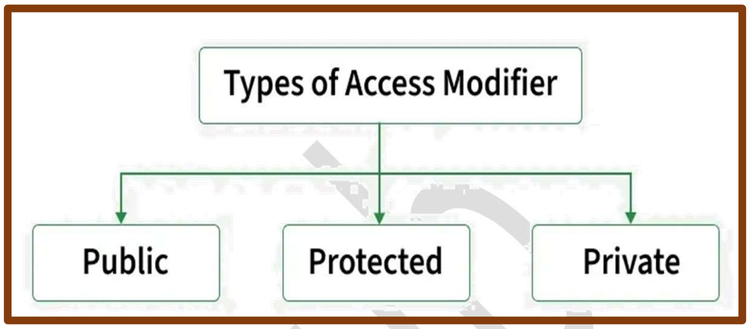
```
Output
Account Holder: Neha,
Balance: $5000
```

```
};
// Derived Class
class SavingsAccount : public BankAccount {
public:
    double interestRate;
    void calculateInterest() {
        cout << "Interest: $" << balance * interestRate / 100 << endl;
    }
};
int main() {
    SavingsAccount sa1;
    sa1.accountHolder = "Neha";
    sa1.balance = 5000;
    sa1.interestRate = 5;
    sa1.showAccount();      // Base class ka method
    sa1.calculateInterest();// Derived class ka method
    return 0;
}
```

Explanation

- Base Class (BankAccount)
- Properties: accountHolder aur balance
- Method showAccount() account holder aur balance display karta hai
- Derived Class (SavingsAccount)
- BankAccount se inherit karti hai (public inheritance)
- Apni extra property: interestRate
- Method calculateInterest() interest calculate karke display karta hai
- main() function mein
- SavingsAccount ka object sa1 create kiya
- Base class members (accountHolder, balance) aur derived class member (interestRate) assign kiye
- sa1.showAccount() → Base class ka method call
- sa1.calculateInterest() → Derived class ka method call

Access Modifiers in C++



Access modifiers decide karte hain ki class ke members (variables aur functions) ko kaun access kar sakta hai. Ye data security aur encapsulation ke liye use hote hain.

Types of Access Modifiers

1. **Public**
   - Members class ke andar, derived class mein aur class ke bahar sab jagah accessible hote hain.
   - Real-Life Example: Car ka Start/Stop button – koi bhi use kar sakta hai.
2. **Private**
   - Members sirf class ke andar accessible hote hain. Derived class aur class ke bahar direct access nahi hota.
   - Real-Life Example: Car ka engine/fuel level – sirf car ke methods ke through access hota hai.
3. **Protected**
   - Members class ke andar aur derived class ke liye accessible hote hain, lekin class ke bahar direct access nahi hota.
   - Real-Life Example: Car ka speed limit – sirf SportsCar (derived class) access kar sakti hai.

Class Methods

Class Methods wo functions hote hain jo class ke andar define kiye jaate hain aur class ke objects ke saath use hote hain. Ye methods class ke data members ko access aur manipulate karte hain.

Define a Method Inside the Class
Definition:
Jab hum class ke andar hi function (method) ko define karte hain, usse "method defined inside the class" kehte hain.

- Iska matlab hai ki function ka code directly class ke andar likha gaya hai, na ki class ke bahar.
- Ye usually chhote functions ke liye use hota hai aur compiler ise inline treat kar sakta hai.

```
#include <iostream>
using namespace std;
class Student {
public:
    string name;
    int roll;

    // Method defined inside the class
```

```
Output
Name: Rahul, Roll: 1
```

```
    void display() {
        cout << "Name: " << name << ", Roll: " << roll << endl;
    }
};
int main() {
    Student s1;
    s1.name = "Rahul";
    s1.roll = 1;
    s1.display();  // Object se method call
    return 0;
}
```

Explanation

- Class Student ke andar hi display() method define kiya gaya hai.
- Object s1 banaya aur uske data members (name aur roll) set kiye.
- s1.display() call karne par method directly object ke data ko access karke output print karta hai.

Define a Method Outside the Class

Jab hum class ke andar sirf method ka declaration karte hain aur method ka code/class ke bahar likhte hain, usse “method defined outside the class” kehte hain.

- Isme scope resolution operator (::) ka use hota hai, jo batata hai ki method ka belonging kis class se hai.
- Ye bade programs aur complex functions ke liye useful hai.

```
#include <iostream>
using namespace std;
class Student {
public:
    string name;
    int roll;
    // Method declaration (inside class)
    void display();
};
// Method definition (outside class)
void Student::display() {
    cout << "Name: " << name << ", Roll: " << roll << endl;
}
int main() {
    Student s1;
    s1.name = "Aman";
    s1.roll = 2;
    s1.display();  // Object se method call
    return 0;
}
```

```
Output
Name: Aman,
Roll: 2
```

Explanation

- Class Student ke andar display() ka sirf declaration hai.
- Method ka code class ke bahar Student::display() me define kiya gaya hai.
- s1.display() call karne par method object ke data members ko access karke output print karta hai.

Difference Between Method Defined Inside the Class and Outside the Class

Point	Inside the Class	Outside the Class
Definition	Method ka code class ke andar likha jata hai	Method ka code class ke bahar likha jata hai, declaration class me
Syntax	No scope resolution operator (::) needed	Scope resolution operator (::) required
Compiler Behavior	Usually inline function ke tarah treat hota hai	Inline nahi hota by default (unless explicitly declared)
Use Case	Chhote aur simple methods ke liye	Bade aur complex functions ke liye, organized code ke liye
Readability	Simple programs ke liye easy	Large programs ke liye better readability
Example	void display() { cout << name; }	void Student::display() { cout << name; }

Parameters

Parameters wo values ya variables hote hain jo hum function/method ko input ke roop mein pass karte hain.

- Ye function ke andar use hote hain data ko process karne ke liye.
- Parameters function ke behavior ko customize karte hain.

Types of Parameters

1. Formal Parameters (Method ke andar define kiye gaye)
   - Ye function ke declaration/definition me likhe jaate hain.
   - Example: void add(int a, int b) → yahan a aur b formal parameters hain.
2. Actual Parameters (Call ke time pass kiye gaye)
   - Ye function call ke time provide kiye jaate hain.
   - Example: add(5, 3) → yahan 5 aur 3 actual parameters hain.

Example: Function with Parameters

```
#include <iostream>
using namespace std;
class Calculator {
public:
    void add(int a, int b) {   // a, b = formal parameters
        cout << "Sum: " << a + b << endl;
```

```
Output
Sum: 30
```

```
    }
};
int main() {
    Calculator calc;
    calc.add(10, 20);          // 10, 20 = actual parameters
    return 0;
}
```

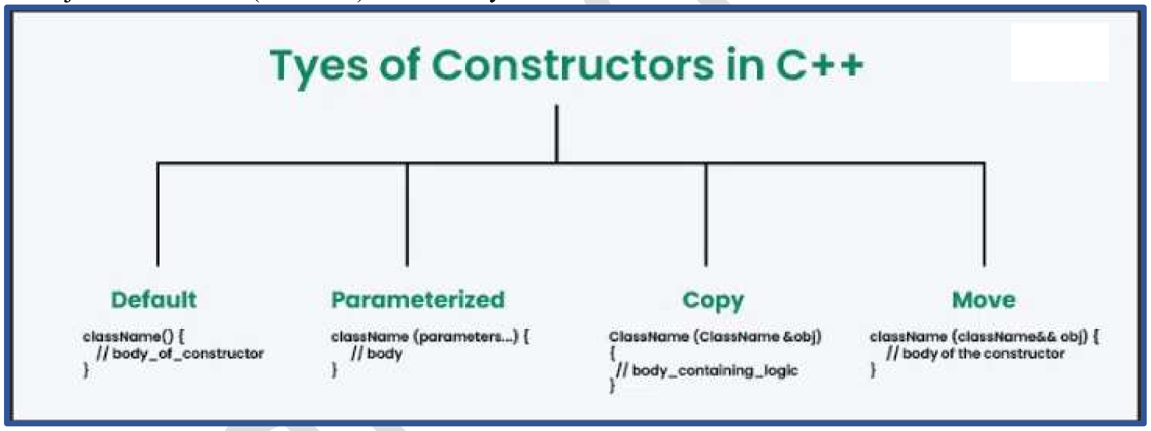
Explanation

- add(int a, int b) → formal parameters
- add(10, 20) → actual parameters
- Method formal parameters ke through actual parameters ke values ko process karta hai

C++ Constructors

Definition:

Constructor ek special type ka function hota hai jo automatically call hota hai jab bhi class ka object create kiya jata hai.
Ye object ko initialize (value set) karne ke liye use hota hai



```
className() {
  // body_of_constructor
}
```

```
className (parameters...) {
    // body
}
```

```
ClassName (ClassName &obj)
{
 // body_containing_logic
}
```

```
className (className&& obj) {
  // body of the constructor
}
```

Types of Constructors

- **Default Constructor** – No parameters (compiler bhi bana deta hai agar user define na kare)
- **Parameterized Constructor** – Arguments accept karke object ko specific values se initialize karta hai
- **Copy Constructor** – Ek object ko dusre object ki values se copy karta hai
- **Move Constructor** – Temporary object ke resources transfer karta hai (C++11)

Default Constructor

Question

Ek C++ program likho jisme Product class ho. Use a default constructor (constructor jo koi parameter nahi leta) to initialize product ka naam "Unknown" aur price ko 0 rakho. Fir ek object create karke product details display karo.

- Yeh waise case ko dikhata hai jab customer product ki details nahi batata, to default values automatically set ho jati hain.

```
#include <iostream>
using namespace std;
class Product {
public:
    string name;
    int price;
    // Default constructor
    Product() {
        name = "Unknown";  // default product name
        price = 0;         // default price
    }
    void display() {
        cout << "Product Name: " << name << endl;
        cout << "Price: " << price << endl;
    }
};
int main() {
    Product p1;  // Calls default constructor
    p1.display();  // Display product details
    return 0;
}
```

```
Output
Product Name:
Unknown
Price: 0
```

Explanation

- Product() ek default constructor hai (koi argument nahi leta) jo object banate hi name = "Unknown" aur price = 0 set karta hai.

- p1 object banate hi constructor call hota hai aur fir display() se details print hoti hain.
- Ye wahi default behaviour hai jo customer jab product detail nahi batata tab values automatically set ho jati hain

Parameterized Constructor

Question

Ek C++ program likho jisme Student class ho.
Is class me name, roll, aur marks ho. Ek parameterized constructor likho jo object banate waqt in values ko initialize kare.
Fir ek object bana ke student details display karo.

- Yeh waise situation ko dikhata hai jab tum student ke details ko directly object create karte waqt set karte ho.

```
#include <iostream>
using namespace std;
class Student {
public:
    string name;
    int roll;
    int marks;
    // Parameterized constructor
    Student(string n, int r, int m) {
        name = n;
        roll = r;
        marks = m;
    }
    void display() {
        cout << "Student Name: " << name << endl;
        cout << "Roll No: " << roll << endl;
        cout << "Marks: " << marks << endl;
    }
};
int main() {
    // Object banate waqt values pass kiye
    Student s1("Rahul", 5, 89);
    s1.display();  // Display student details
    return 0;
}
```

```
Output
Student Name: Rahul
Roll No: 5
Marks: 89
```

Explanation

- Constructor Student(string n, int r, int m) object banate waqt values receive karta hai aur class ke data members ko set karta hai.
- Jab Student s1("Rahul", 5, 89); likha, to constructor "Rahul", 5 aur 89 se data initialize hua.
- s1.display() se student details screen pe print ho gaye.

Copy Constructor
Question

Ek C++ program likho jisme BankAccount class ho.
Is class me holderName aur balance members ho.
Ek copy constructor likho jo ek existing account object ki details ko copy karke naya account object banaye.
Fir original aur copied account dono ki details display karo.
(Jaise bank me jab naye account me purane account ki details copy karai jati hain.)

```
#include <iostream>
using namespace std;
class BankAccount {
public:
    string holderName;
    double balance;
    // Parameterized constructor
    BankAccount(string name, double bal) {
        holderName = name;
        balance = bal;
    }
    // Copy constructor
    BankAccount(const BankAccount &oldAccount) {
        holderName = oldAccount.holderName;
        balance = oldAccount.balance;
        cout << "Copy constructor called!" << endl;
    }
    void showDetails() {
        cout << "Account Holder: " << holderName << endl;
        cout << "Balance: $" << balance << endl;
    }
};
int main() {
    // Original account object
    BankAccount acc1("Neha", 5000);

    // New account object created using copy constructor
    BankAccount acc2 = acc1;  // Copying values from acc1 to acc2
```

Output
```
Copy constructor called!
Original Account:
Account Holder: Neha
Balance: $5000
```

```
    cout << "Original Account:" << endl;
    acc1.showDetails();
    cout << "\nCopied Account:" << endl;
    acc2.showDetails();
    return 0;
}
```

Explanation

- BankAccount acc1("Neha", 5000); → pehle account banaya gaya.
- BankAccount acc2 = acc1; → yahan copy constructor call hota hai, jo acc1 ki details ko acc2 me copy karta hai.
- Dono accounts ke details print kiye jaate hain.

Move Constructor
Question
Ek C++ program likho jisme Mobile class ho.
Is class me brand, model, aur ram ho.
Ek move constructor likho jo temporary object se resources naye object me transfer kare bina copying ke.
Fir original aur moved object dono ka data display karo.
(Yeh waise situation ko dikhata hai jab temporary object ka data directly ek real object me shift karna chahte ho for better performance.)

```
#include <iostream>
using namespace std;
class Mobile {
public:
    string brand;
    string model;
    int* ram;  // dynamically allocate RAM to demonstrate move
    // Parameterized constructor
    Mobile(string b, string m, int r) {
        brand = b;
        model = m;
        ram = new int(r);
        cout << "Parameterized constructor called\n";
    }
   // Move constructor
    Mobile(Mobile&& temp) noexcept {
        cout << "Move constructor called\n";
        brand = temp.brand;
        model = temp.model;
```

```
Output
Parameterized constructor
called
Move constructor called
Original Mobile (m1): Object
has no data (moved)
Moved Mobile (m2): Brand:
Samsung, Model: Galaxy,
RAM: 8GB
```

```
      ram = temp.ram;     // transfer resource
        temp.ram = nullptr; // original object loses ownership
    }
    void showData() {
        if (ram != nullptr) {
            cout << "Brand: " << brand
                 << ", Model: " << model
                 << ", RAM: " << *ram << "GB\n";
        } else {
            cout << "Object has no data (moved)\n";
        }
    }
    ~Mobile() {
        delete ram;
    }
};
int main() {
    Mobile m1("Samsung", "Galaxy", 8);
    // Move constructor is called
    Mobile m2 = std::move(m1);
    cout << "Original Mobile (m1): ";
    m1.showData();  // m1 no longer owns memory
    cout << "Moved Mobile (m2): ";
    m2.showData();
    return 0;
}
```

Explanation

- Mobile m1("Samsung", "Galaxy", 8);
  - Yeh parameterized constructor call hota hai aur m1 ko initialize karta hai with brand, model aur dynamically allocated ram.
- Mobile m2 = std::move(m1);
  - Move constructor call hota hai because we used std::move(m1).
  - Move constructor original object m1 se resources (pointer) ko transfer karta hai m2 me bina memory copy kiye.
  - Fir original (m1) ka pointer nullptr set hota hai — matlab m1 ab data nahi rakhta.
- m1.showData();
  - Ab m1 ke paas koi data nahi hai (moved state), isliye output me “Object has no data (moved)” dikhata hai.

- m2.showData();
  - m2 ne directly m1 se data move kiya, isliye brand, model aur RAM value print hota hai.

Why We Used Constructor

- Objects ko automatically initialize karte hain jab object banta hai. =
- Initialization se undefined/garbage values se bachate hain.
- Constructors se objects valid aur consistent state me start hote hain.
- Multiple constructors se flexible initialization milti hai (default/parameterized/copy/move).
- Encapsulation strong hoti hai kyunki initialization logic class ke andar hi rehta hai.
- Resource management (memory/file handles etc.) safely handle hota hai through constructors.
- Code readable aur organized hota hai jab initialization class ke andar hi hoti hai.
- Default constructors se common values automatically set ho jati hain bina extra code likhe.

C++ Constructor Overloading

C++ me agar ek class ke ek se zyada constructors likhe ja sakte hain jinke parameters alag-alag hote hain (number ya type different), to is concept ko Constructor Overloading kehte hain. Ye function overloading jaisa hi hota hai, lekin sab constructors ka naam class ke naam jaisa hi hota hai.

Key Points (Hinglish)

- Ek class me multiple constructors ho sakte hain.
- Har constructor ka parameter list different hota hai.
- Object banate waqt compiler decide karta hai ki kaunsa constructor call hona chahiye based on arguments.
- Isse flexibility milti hai object ko alag-alag tarike se initialize karne ka.

Question Constructor Overloading ko samjhane ke liye Vehicle Registration par ek C++ program likhiye.
Ek class Vehicle banaiye jisme:

- vehicle number
- vehicle type

ho.
Class me 3 constructors banaiye:

1. Ek constructor jo koi value na le (default).
2. Ek constructor jo sirf vehicle number le.
3. Ek constructor jo vehicle number aur vehicle type dono le.

Ek function display() banaiye jo vehicle ki details print kare.
main() function me:

- Vehicle class ke 3 objects banaiye.
- Har object ke liye details print kijiye.

```
#include <iostream>
using namespace std;
class Vehicle {
```

Output
```
Vehicle Number: 0,
Vehicle Type:
Unknown
Vehicle Number: 101,
Vehicle Type:
    int vehicleNumber;
    string vehicleType;
public:
    // 1. Default constructor
    Vehicle() {
        vehicleNumber = 0;
        vehicleType = "Unknown";
    }
    // 2. Constructor with vehicle number
    Vehicle(int vNum) {
        vehicleNumber = vNum;
        vehicleType = "Unknown";
    }
    // 3. Constructor with vehicle number and type
    Vehicle(int vNum, string vType) {
        vehicleNumber = vNum;
        vehicleType = vType;
    }
   void display() {
        cout << "Vehicle Number: " << vehicleNumber
             << ", Vehicle Type: " << vehicleType << endl;
    }
};
int main() {
    Vehicle v1;                 // Default constructor
    Vehicle v2(101);            // Constructor with number
    Vehicle v3(102, "Car");     // Constructor with number + type
    v1.display();
    v2.display();
    v3.display();
    return 0;
}
```

Explanation
Class Definition:

- class Vehicle banaya.
- Data members:
  - vehicleNumber → Vehicle ka number
  - vehicleType → Vehicle ka type (Car, Bike, etc.)

Constructor Overloading:

- **Default constructor** → Koi parameter nahi, vehicleNumber = 0, vehicleType = "Unknown"
- **Constructor 2** → Sirf vehicleNumber parameter, vehicleType = "Unknown"
- **Constructor 3** → vehicleNumber aur vehicleType dono parameters

Display Function:

- void display() → Vehicle details print karta hai
- Format: "Vehicle Number: <number>, Vehicle Type: <type>"

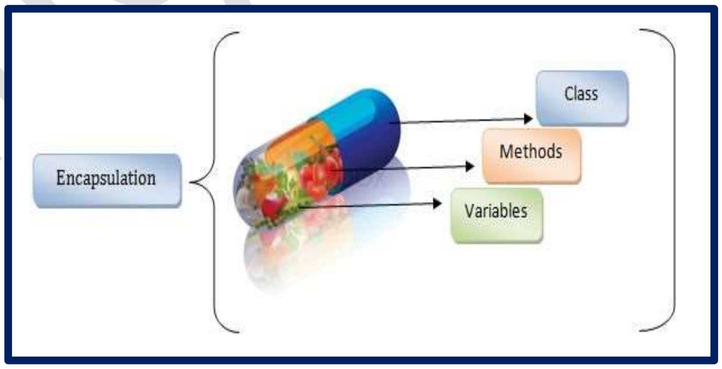
Main Function:

- Vehicle v1; → Default constructor call → 0, Unknown
- Vehicle v2(101); → Constructor 2 call → 101, Unknown
- Vehicle v3(102, "Car"); → Constructor 3 call → 102, Car
- display() function har object ke liye call kiya → details print hui

Key Point:

- Constructor Overloading = Ek hi class me multiple constructors, different parameters ke saath
- Real-life example → Vehicle registration me alag-alag details ke saath objects create karna

Encapsulation



Encapsulation ka matlab hai data ko class ke andar hide karna aur usko control karna.

- Data members ko private banate hain
- Unko access karne ke liye public functions (getters/setters) use karte hain

Why we use it?

1. Data safe rahe → directly change na ho
2. Code easy to maintain ho
3. Object-oriented programming ka basic principle

Question Student Marks System (Encapsulation)
Student marks ko secure karne ka program likhiye using Encapsulation.

- Class Student banaiye jisme private data member ho: marks (int type)
- Public functions:
  1. setMarks(int m) → marks set kare (0-100 ke beech)
  2. getMarks() → marks return kare

```
#include <iostream>
using namespace std;
class Student {
private:
    int marks;  // private data → encapsulated
public:
    // Setter function
    void setMarks(int m) {
        if(m >= 0 && m <= 100)
            marks = m;
        else
            cout << "Invalid marks! Enter 0-100." << endl;
    }
    // Getter function
    int getMarks() {
        return marks;
    }
};
int main() {
    Student s1;
    s1.setMarks(85);  // marks set kare
    cout << "Student Marks: " << s1.getMarks() << endl;
    s1.setMarks(120); // invalid marks
    cout << "Student Marks: " << s1.getMarks() << endl;
    return 0;
}
```

```
Output
Student Marks: 85
Invalid marks! Enter 0-100.
Student Marks: 85
```

Explanation

- marks → private → direct access nahi
- setMarks() → marks ko control ke saath set karta hai (0–100)
- getMarks() → marks read karne ke liye use hota hai
- Yehi hai Encapsulation ka real-life example

Question Mobile Phone Details (Constructor Overloading + Encapsulation)

- Mobile phone details ka program likhiye using Constructor Overloading aur Encapsulation.
- Class Mobile banaiye jisme private data members:
  - brand → mobile brand
  - price → mobile price
- Constructors:
  - Default → brand="Unknown", price=0
  - Brand only
  - Brand and price

```
#include <iostream>
using namespace std;
```

```
Output
Brand: Unknown, Price: 0
Brand: Samsung, Price: 0
Brand: Apple, Price: 80000
```

```
class Mobile {
private:
    string brand;  // encapsulated
    int price;     // encapsulated
public:
    // 1. Default constructor
    Mobile() {
        brand = "Unknown";
        price = 0;
    }
    // 2. Constructor with brand only
    Mobile(string b) {
        brand = b;
        price = 0;
    }
    // 3. Constructor with brand and price
    Mobile(string b, int p) {
        brand = b;
        price = p;
    }
    // Function to display details
    void show() {
        cout << "Brand: " << brand << ", Price: " << price << endl;
    }
};
int main() {
    Mobile m1;                // Default constructor
    Mobile m2("Samsung");     // Brand only
    Mobile m3("Apple", 80000); // Brand and price
    m1.show();
    m2.show();
    m3.show();
    return 0;
}
```

Explanation

- Encapsulation: brand aur price ko private banaya → direct access nahi
- Constructor Overloading: 3 constructors → different ways to create objects
- m1 → default constructor → Unknown, 0
- m2 → brand only → Samsung, 0
- m3 → brand + price → Apple, 80000

Abstraction

- Abstraction ka matlab hai sirf essential features ko dikhana aur unnecessary details ko hide karna.
- User ko kya karna hai dikhai deta hai, kaise karna hai hidden rehta hai.
- Example: Car chalana → driver ko sirf steering, brake, accelerator dikhte hain, engine ka kaam hidden hota hai.

Why We Used Abstraction

1. Complex system ko simple banaye
2. Code maintain karna easy ho
3. Data ko safe rakhe
4. Real-world objects ka essential feature show kare

Simple Real-Life Analogy

- Car chalate waqt driver engine ke internal mechanism ko nahi dekhta — sirf steering, brake aur accelerator ka use karta hai. Isi tarah programming me essential actions dikhte hain, complexity hide hoti ha

Question: Device Power Control (Abstraction)
Ek abstract class Device banaiye jisme pure virtual functions:

- turnOn()
- turnOff()

Phir two derived classes banaiye:

- Laptop → functions implement kare
- Smartphone → functions implement kare

main() me base class pointer se dono devices ko on/off kijiye.

```
#include <iostream>
using namespace std;
// Abstract base class
class Device {
public:
    virtual void turnOn() = 0;   // Pure virtual function
    virtual void turnOff() = 0;  // Pure virtual function
};
// Derived class 1
class Laptop : public Device {
public:
    void turnOn() {
        cout << "Laptop turned ON" << endl;
    }
    void turnOff() {
        cout << "Laptop turned OFF" << endl;
    }
};
// Derived class 2
```

```
Laptop turned ON
Laptop turned OFF
Smartphone turned ON
Smartphone turned OFF
```

```
class Smartphone : public Device {
public:
    void turnOn() {
        cout << "Smartphone turned ON" << endl;
    }
    void turnOff() {
        cout << "Smartphone turned OFF" << endl;
    }
};
int main() {
    Device *d;              // Base class pointer
    Laptop lap;
    Smartphone phone;
    d = &lap;              // Point to Laptop
    d->turnOn();
    d->turnOff();
    d = &phone;            // Point to Smartphone
    d->turnOn();
    d->turnOff();
    return 0;
}
```

Explanation

Virtual Function virtual keyword base class me function ko special banata hai.

- Virtual function derived class me override hota hai.
- Runtime polymorphism achieve hota hai — yani function call runtime pe decide hota hai.
- Base class pointer/reference se call karne par derived class ka function run hota hai.
- Without virtual, base class ka function hi call hota (early binding).
- Base class me pure virtual function (= 0) → abstract class banata hai.
- Abstract class ka object directly create nahi kiya ja sakta.

Virtual Function Kyun Use Karte Hain?

Normally C++ mein compile-time binding hoti hai.
Lekin virtual function use karne se dynamic (runtime) binding hoti hai, jisse derived class ka function call hota ha

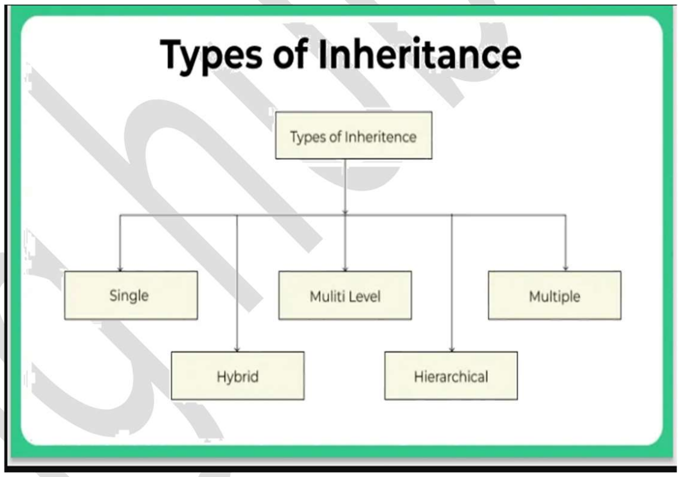
Question 1:- Socho aap TV remote use karte ho. Aapko ye nahi pata hota ki button dabane par andar circuit kaise kaam karta hai. Aapko sirf button ka use pata hota hai. using abstraction in c++.

```
#include <iostream>
using namespace std;
// TV class with abstraction
class TV {
private:
    int volume;      // Internal state (hidden)
    int channel;     // Internal state (hidden)
public:
    TV() {
        volume = 10;
        channel = 1;
    }
    // Public functions act like "buttons" on remote
    void increaseVolume() {
        volume++;
        cout << "Volume: " << volume << endl;
    }
    void decreaseVolume() {
        volume--;
        cout << "Volume: " << volume << endl;
    }
    void changeChannel(int newChannel) {
        channel = newChannel;
        cout << "Channel: " << channel << endl;
    }
};
int main() {
    TV myTV;
    // User interacts with TV via "abstraction" (public methods)
    myTV.increaseVolume();
    myTV.changeChannel(5);
    return 0;
}
```
Explanation

- Class TV: TV ka blueprint, internal details encapsulated.

- Private members (volume, channel): Internal state, user ko nahi dikhta → implementation hidden.
- Constructor (TV()): Initial volume aur channel set karta hai.
- Public methods (increaseVolume(), decreaseVolume(), changeChannel()): TV ke "buttons" jaise, interface for user.
- User interaction: myTV.increaseVolume() → user bas button dabata hai, internal logic hidden.
- Abstraction essence: "What to do" dikhao, "how it happens" hide karo.

Inheritance



Inheritance ek C++ feature hai jisme ek class (derived class) doosri class (base class) ki properties aur methods inherit kar sakti hai.

- Base Class (Parent): Original class, jiska code share kiya ja raha hai.
- Derived Class (Child): Nayi class, jo base class ki features ko use kar sakti hai aur apni additional features add kar sakti hai.
  Analogy:
- Base class → Car
- Derived class → SportsCar
- SportsCar me Car ki basic properties (wheels, engine) hoti hain, plus extra features (turbo, spoiler).

Single Inheritance

- Definition: Ek derived class sirf ek base class se inherit karti hai.
- Syntax: class Derived : public Base {}

Question:- Ek C++ program likho jisme base class Person me name aur age store ho.
Phir ek derived class Student banao jo Person se inherit kare aur usme rollNumber add ho.
Methods banao student details ko set aur show karne ke liye.
main() me user se input lo aur phir details display karo.

LEARNING HUB

SHAHABAD MARKANDA

📞CALL- 77000 90800

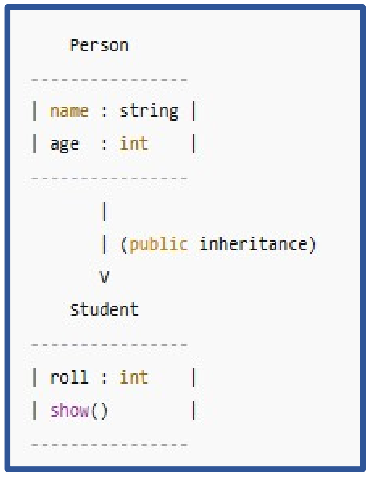
- **ALL COMPUTER COURSES**
- **ENGLISH LANGUAGE COURSES**
- **Brand Marketing & Custom Software Solution**
- **TUITION NURSERY TO 10TH (ALL SUBJECTS)**
  WEBSITE: https://learninghubshahabad.in

```
#include <iostream>
using namespace std;
class Person {
public:
    string name;
    int age;
};
class Student : public Person {
public:
    int roll;
    void show() {
        cout << "Name: " << name << "\nAge: " << age << "\nRoll: " << roll << endl;
    }
};
int main() {
    Student s;
    cout << "Enter name age roll: ";
    cin >> s.name >> s.age >> s.roll;
    cout << "\nStudent Details:\n";
    s.show();
    return 0;
}
```

```
Output
Student Details:
Name: Rahul
Age: 20
Roll: 101
```

Explanation

- #include <iostream>: Input/output ke liye library include ki.
- using namespace std;: Standard library ke names directly use kar sakte hain, std:: likhne ki zarurat nahi.
- class Person: Base class hai jisme do public members hain: name (string) aur age (int).
- class Student : public Person: Derived class hai jo Person se inherit karti hai.
  - Isme ek naya public member hai: roll (int).
  - show() function student ke details print karta hai.
- int main(): Program ka starting point.
  - Ek Student object s banaya.
  - User se name, age, aur roll input liya.
  - s.show() call karke details display ki.
- Ye code basic inheritance ko demonstrate karta hai, jahan Student Person ke members inherit karta hai.

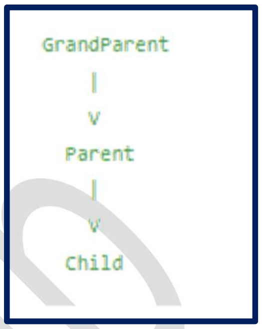


```
   Person
---------------
| name : string |
| age  : int    |
---------------
       |
       | (public inheritance)
       v
   Student
---------------
| roll : int    |
| show()        |
---------------
```

ADDRESS: EKTA VIHAR, NEAR ASHOKA PALACE, SHAHABAD MARKANDA CALL- 7700090800

Multi-Level Inheritance kya hai?

- Jab ek class dusri class se inherit karti hai, aur phir tisri class us derived class se inherit karti hai, to ise multi-level inheritance kehte hain.
- Matlab inheritance chain hoti ha



Device → Phone → Smartphone

- **Description:**

  1. Device class me brand aur price ho.
  2. Phone class Device se inherit kare aur memory ho.
  3. Smartphone class Phone se inherit kare aur OS ho.
  4. Program Smartphone details print kare.

```
#include <iostream>
using namespace std;
// Base Class
class Device {
public:
    string brand;
    float price;
    void setDeviceData(string b, float p) {
        brand = b;
        price = p;
    }
    void showDeviceData() {
        cout << "Brand: " << brand << endl;
        cout << "Price: $" << price << endl;
    }
};
// Derived Class 1
class Phone : public Device {
public:
    int memory; // in GB
    void setPhoneData(int m) {
        memory = m;
    }
    void showPhoneData() {
        cout << "Memory: " << memory << " GB" << endl;
    }
};
// Derived Class 2
```

```
Output
Brand: Apple
Price: $999.99
Memory: 128 GB
Operating System: iOS 17
```

```
class Smartphone : public Phone {
public:
    string OS;
    void setSmartphoneData(string os) {
        OS = os;
    }
    void showSmartphoneData() {
        cout << "Operating System: " << OS << endl;
    }
};
int main() {
    Smartphone s1;
    s1.setDeviceData("Apple", 999.99);
    s1.setPhoneData(128);
    s1.setSmartphoneData("iOS 17");
    cout << "Smartphone Details:" << endl;
    s1.showDeviceData();
    s1.showPhoneData();
    s1.showSmartphoneData();
    return 0;
}
```

Explanation

1. Classes:
   - Device → brand, price
   - Phone → memory (inherits Device)
   - Smartphone → OS (inherits Phone)
2. Flow:
   - Multi-Level Inheritance → Device → Phone → Smartphone
3. main() Function:
   - Smartphone s1 object banaya
   - set functions se data set kiya
   - show functions se details print ki

Multiple inheritance ka matlab hota hai ki ek class ek se zyada parent (base) classes se inherit karti hai. Isse derived class ko sabhi parent classes ke functions aur data members mil jaate hain.

Question: Ek Teacher class banao jisme subject ka info ho aur ek Researcher class jisme research topic ka info ho. Ek child class Prof banao jo dono inherit kare aur full information display kare.

```
#include <iostream>
using namespace std;
// Teacher class
class Teacher {
protected:
    string subject;
public:
    void setSubject(string s) {
        subject = s;
    }
};
// Researcher class
class Researcher {
protected:
    string researchTopic;
public:
    void setResearchTopic(string r) {
        researchTopic = r;
    }
};
// Child class using Multiple Inheritance
class Prof : public Teacher, public Researcher {
public:
    void displayInfo() {
        cout << "Subject Taught: " << subject << endl;
        cout << "Research Topic: " << researchTopic << endl;
    }
};
int main() {
    Prof p;
    p.setSubject("Computer Science");
    p.setResearchTopic("Artificial Intelligence");
    p.displayInfo();
    return 0;
}
```

Output

```
Subject Taught: Computer Science
Research Topic: Artificial Intelligence
```

Explanation

- #include <iostream> → Input/output ke liye use hota hai

- using namespace std; → std:: baar-baar likhne se bachata hai
- Teacher class → Subject ki information store karti hai
- subject → protected hai taaki child class access kar sake
- setSubject() → Subject set karne ke liye function
- Researcher class → Research topic ki information store karti hai
- researchTopic → protected hai taaki child class use kar sake
- setResearchTopic() → Research topic set karne ka function
- Prof class → Teacher aur Researcher dono ko inherit karti hai (multiple inheritance)
- displayInfo() → Subject aur research topic dono display karta hai
- main() function → Program execution start hota hai
- Prof p; → Prof class ka object banaya
- Data set karke displayInfo() call kiya

Hybrid Inheritance ka matlab hota hai do ya zyada types of inheritance ka combination ek hi program me use karna.

Example
- Teacher aur Staff dono Person hote hain
- Professor ek Teacher bhi hota hai aur Staff bhi
- Different inheritance ka mix ho gaya

Practical Question: University Staff System (Hybrid Inheritance)
Scenario:
University me kuch log Person hote hain. Kuch Employee hote hain, kuch Teacher hote hain. Ek Professor dono hota hai – Employee aur Teacher, aur saath me research bhi karta hai.
Requirements:
1. Class Person
   - Data: name (string), age (int)
   - Function: setPerson() → name aur age set kare
2. Class Employee (inherits from Person)
   - Data: salary (float)
   - Function: setSalary() → salary set kare
3. Class Teacher (inherits from Person)
   - Data: subject (string)
   - Function: setSubject() → subject set kare
4. Class Professor (inherits Employee aur Teacher)
   - Data: researchArea (string)
   - Functions:
     - setResearch() → research area set kare
     - displayInfo() → print kare name, age, salary, subject, research area

Task:
- Ek Professor object banao
- Sab details set karo (name, age, salary, subject, research area)
- displayInfo() call karke sab print karo

```
#include <iostream>
using namespace std;
// Base class
class Person {
protected:
    string name;
    int age;
public:
    void setPerson(string n, int a) {
        name = n;
        age = a;
    }
};
// Employee class inherits Person
class Employee : public Person {
protected:
    float salary;
public:
    void setSalary(float s) {
        salary = s;
    }
};
// Teacher class inherits Person
class Teacher : public Person {
protected:
    string subject;
public:
    void setSubject(string sub) {
        subject = sub;
    }
};
// Professor class inherits Employee and Teacher (Hybrid Inheritance)
class Professor : public Employee, public Teacher {
private:
    string researchArea;
public:
    void setResearch(string r) {
        researchArea = r;
    }
    void displayInfo() {
        // Ambiguity solve karne ke liye Person ka data Employee ke through access kiya
        cout << "Name: " << Employee::name << endl;
        cout << "Age: " << Employee::age << endl;
```

```
Output
Name: Dr. Sharma
Age: 45
Salary: 75000
Subject: Computer Science
```

```
        cout << "Salary: " << salary << endl;
        cout << "Subject: " << subject << endl;
        cout << "Research Area: " << researchArea << endl;
    }
};
int main() {
    Professor prof;
    // Set all details
    prof.setPerson("Dr. Sharma", 45);      // Person info
    prof.setSalary(75000);                 // Employee info
    prof.setSubject("Computer Science");   // Teacher info
    prof.setResearch("Artificial Intelligence"); // Professor info
    // Display all info
    prof.displayInfo();
   return 0;
}
```

Explanation

- Person → base class, name & age store karta hai, setPerson() se set hota hai
- Employee → Person se inherit, salary store karta hai, setSalary()
- Teacher → Person se inherit, subject store karta hai, setSubject()
- Professor → Employee + Teacher inherit, researchArea store karta hai, setResearch()
- displayInfo() → sab info print karta hai (name, age, salary, subject, research)
- main() → Professor object banaye, details set kare, display call kare

Hierarchical Inheritance tab hoti hai jab ek base class se multiple derived classes banti hain.

- Ek parent class hoti hai.
- Usse do ya usse zyada child classes inherit karte hain.

Question:
Ek company ke liye C++ program likhiye jisme hierarchical inheritance ka use ho. Program mein ye features honi chahiye:

1. Base Class: Employee
   - Data members: name (string), salary (float)
   - Member function: work() — ye print kare "Employee is working."
2. Derived Classes:
   - Manager
     - Function: conductMeeting() — ye print kare "Manager is conducting a meeting."
   - Developer
     - Function: writeCode() — ye print kare "Developer is writing code."
   - Intern
     - Function: assistTasks() — ye print kare "Intern is assisting in tasks."

```
#include <iostream>
#include <string>
```

```
using namespace std;
// Base class
class Employee {
public:
    string name;
    float salary;
    void work() {
        cout << "Employee is working." << endl;
    }
};
// Derived class 1
class Manager : public Employee {
public:
    void conductMeeting() {
        cout << "Manager is conducting a meeting." << endl;
    }
};
// Derived class 2
class Developer : public Employee {
public:
    void writeCode() {
        cout << "Developer is writing code." << endl;
    }
};
// Derived class 3
class Intern : public Employee {
public:
    void assistTasks() {
        cout << "Intern is assisting in tasks." << endl;
    }
};

int main() {
    // Manager object
    Manager m;
    m.work();           // Base class function
    m.conductMeeting(); // Derived class function
    cout << endl;
    // Developer object
    Developer d;
    d.work();          // Base class function
    d.writeCode();     // Derived class function
    cout << endl;
    // Intern object
```

```
Output
Employee is working.
Manager is conducting a
meeting.

Employee is working.
Developer is writing code.

Employee is working.
Intern is assisting in tasks.
```

```
    Intern i;
    i.work();          // Base class function
    i.assistTasks();   // Derived class function
    return 0;
}
```

Explanation

- Base Class Employee:
- Data members:
  - name → Employee ka naam store karne ke liye.
  - salary → Employee ka salary store karne ke liye.
- Member function:
  - work() → Print karta hai "Employee is working."
- Purpose: Common features jo sab employees me hote hain.
- Derived Class Manager:
- Inherits from Employee (class Manager : public Employee)
- Member function:
  - conductMeeting() → Print karta hai "Manager is conducting a meeting."
- Purpose: Manager ka apna specific behavior.
- Derived Class Developer:
- Inherits from Employee
- Member function:
  - writeCode() → Print karta hai "Developer is writing code."
- Purpose: Developer ka apna specific behavior.
- Derived Class Intern:
- Inherits from Employee
- Member function:
  - assistTasks() → Print karta hai "Intern is assisting in tasks."
- Purpose: Intern ka apna specific behavior.
- In main() function:
- Manager object m:
  - m.work() → Inherited function from Employee
  - m.conductMeeting() → Manager ka specific function
- Developer object d:
  - d.work() → Inherited function
  - d.writeCode() → Developer ka function
- Intern object i:
  - i.work() → Inherited function
  - i.assistTasks() → Intern ka function

Friend Function

C++ mein, friend function ek aisi function hoti hai jo class ka member nahi hoti, lekin class ke private aur protected members ko access kar sakti hai. Normally, private members sirf class ke apne functions access kar sakte hain, lekin friend function ek exception hai.

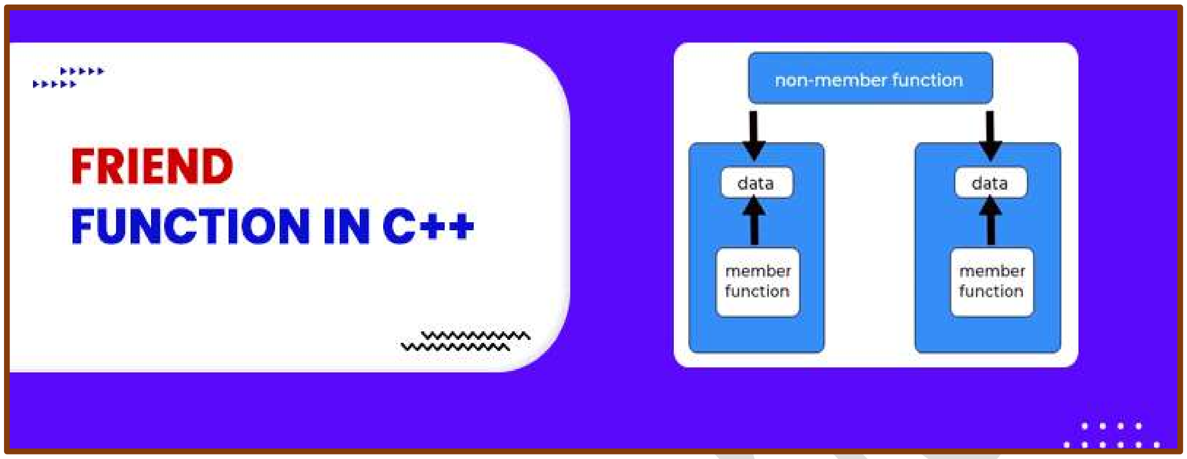
- Private/Protected Data Access: Class ke private/protected members ko class ke bahar se access karne ke liye.
- Operator Overloading: +, <<, >> jaise operators ko objects ke saath kaam karne ke liye.
- Multiple Classes Access: Ek function ko do ya zyada classes ke private members access karne ke liye.
- Non-Member Function Access: Function class ka member nahi hai, fir bhi private data ko access kar sakta hai.
- Encapsulation Support: Data ko secure rakhte hue, zarurat padne par external functions ko access dena.

Bank & Auditor Scenario

- Bank account ka balance private hai.
- Auditor function ko access dena hai balance check karne ke liye without making it a member function.

```
#include <iostream>
using namespace std;
// Class representing a bank account
class BankAccount {
private:
   double balance; // Private member, cannot be accessed normally
public:
   // Constructor to initialize balance
   BankAccount(double b) : balance(b) {}

   // Friend function declaration
   friend void auditor(BankAccount acc);
};
// Friend function definition
void auditor(BankAccount acc) {
   cout << "Auditor can see the balance: $" << acc.balance << endl;
}
int main() {
   // Create a bank account object
   BankAccount myAccount(7500.50);
   // Auditor checks the account balance
   auditor(myAccount); // Friend function can access private member
   return 0;
}
```



Scenario:
Company ke Employee class ka private member salary hai. HR department ko salaries ka report generate karna hai.
Question:
C++ program likho jisme ek friend function generateReport ho jo Employee objects ke private salary ko access karke print kare, without function ko class ka member banaye.

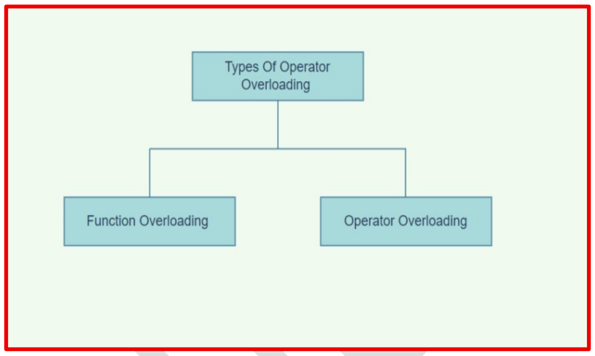
```
#include <iostream>
using namespace std;
// Employee class
class Employee {
private:
    string name;
    double salary; // Private member
public:
    // Constructor to initialize name and salary
    Employee(string n, double s) : name(n), salary(s) {}
    // Friend function declaration
    friend void generateReport(Employee e);
};
// Friend function definition
void generateReport(Employee e) {
    cout << "Employee Name: " << e.name << ", Salary: $" << e.salary << endl;
}
int main() {
    // Create Employee objects
    Employee emp1("Rahul", 5000.50);
    Employee emp2("Anita", 6200.75);
    // HR generates report using friend function
    generateReport(emp1);
    generateReport(emp2);
    return 0;
}
```

Explnation

- Class Employee: Employee ke name aur salary ko private rakhta hai.
- Constructor: Employee object create karte waqt name aur salary initialize karta hai.
- Friend Function generateReport():
  - Declared with friend keyword.
  - Private members name aur salary ko access kar sakta hai.
  - Class ka member nahi hai, phir bhi private data read kar sakta hai.
- In main():
  - Employee objects (emp1, emp2) create kiye gaye.
  - generateReport(emp1) aur generateReport(emp2) call karke private salary aur name print kiya.
- Key Concept: Friend function allow karta hai external function ko private data access karne ke liye bina class ka member banaye.

Operator Overloading kya hai?

- C++ me, operator overloading ka matlab hai ki aap *existing operators (+, -, , /, <<, >>, etc.) ko apne custom class ke objects ke liye redefine kar sakte ho.
- Normal operators jaise + numbers ke liye kaam karte hain. Lekin agar aapke paas ek class hai, jaise Complex, aur aap chahte ho ki + do complex numbers ko add kare, toh aap operator overloading use karenge.



Why We used Operator Overloading

- Addition of things: Jaise 2 apples + 3 apples = 5 apples. Objects ke liye + ka use karna easy ho jata hai.
- Comparison: Jaise "Kaun bada hai?" → > operator se objects compare kar sakte ho.
- Printing things nicely: Jaise cout << object se object ka data easy print ho jaye.
- Increment/Decrement: Jaise "+1 day" ya "-1 level" naturally objects me kar sakte ho.
- Combining things: Jaise 2 shopping carts ko merge karna → + se easily kar sakte ho.
- Real-world logic: Operators same meaning me use hote hain, bas objects ke liye adapt karte hain.

Member Function ke through Overloading

- Operator ko class ke andar function bana ke overload karte hain.
- Syntax:

ReturnType operatorSymbol (parameters)

- Example: Complex operator+ (const Complex &c)
- Rule: Left-hand side object class ka member hona chahiye.

Friend Function ke through Overloading

- Agar operator ko class ke bahar function me define karna ho, lekin private members access karne ho, toh friend function use karte hain.
- Syntax:

friend ReturnType operatorSymbol (ClassName &, ClassName &);

- Example: friend Complex operator+ (const Complex &c1, const Complex &c2);
- Rule: Yeh function non-member hai, lekin class ke private data ko access kar sakta hai.

Unary Operator Overloading

- Ek operator sirf **ek operand** ke liye kaam kare.
- Examples: ++obj, --obj, -obj, !obj

```
Complex operator-() {  // Unary minus
   Complex temp;
   temp.real = -real;
   temp.imag = -imag;
   return temp;
}
```

Binary Operator Overloading

- Ek operator do operands ke liye kaam kare.
- Examples: +, -, *, /, ==, !=

```
Complex operator+(const Complex &c) {  // Binary +
   Complex temp;
   temp.real = real + c.real;
   temp.imag = imag + c.imag;
```

```
    return temp;
}
```

Stream Operator Overloading

- << (output) aur >> (input) ko objects ke liye overload karte hain.
- Example:

```
friend ostream& operator<<(ostream &out, const Complex &c) {
    out << c.real << " + " << c.imag << "i";
    return out;
}
```

Member Function Overloading

Practical

Ek BankAccount class banao jo bank account ka balance store kare. + operator overload karo member function ke through, taaki amount deposit kiya ja sake.

Instructions:

1. Class me private variable balance rakho.
2. operator+ member function ke through implement karo jo integer amount ko balance me add kare.
3. display() function banake balance show karo.
4. Test karo account create karke aur + operator se deposit karke

```
#include <iostream>
using namespace std;
// BankAccount class
class BankAccount {
private:
    int balance;  // private variable to store balance
public:
    // Constructor to initialize balance
    BankAccount(int b = 0) {
        balance = b;
    }
    // Member function to overload + operator for deposit
    BankAccount operator+(int amount) {
        BankAccount temp;
        temp.balance = this->balance + amount;
        return temp;
    }

    // Display function
    void display() {
        cout << "Current balance: " << balance << endl;
    }
};
```

Output

```
Initial Current balance:
1000
After depositing 500:
Current balance: 1500
After depositing 300:
Current balance: 1800
```

```
int main() {
    // Create BankAccount object with initial balance
    BankAccount acc1(1000);
    cout << "Initial ";
    acc1.display();
    // Deposit 500 using overloaded + operator
    BankAccount acc2 = acc1 + 500;
    cout << "After depositing 500: ";
    acc2.display();
    // Deposit another 300
    BankAccount acc3 = acc2 + 300;
    cout << "After depositing 300: ";
    acc3.display();
    return 0;
}
```

Explanation:

1. balance private variable hai jo account ka balance store karta hai.
2. operator+ member function ke through integer amount add kiya ja raha hai aur naya BankAccount object return ho raha hai.
3. display() function se balance show kiya ja raha hai.
4. main() me initial balance, deposit 500, aur deposit 300 test kiya gaya hai.

Friend Function Overloading (Binary Operator)
Practical

Ek Cart class banao jo shopping cart ko represent kare. + operator overload karo friend function ke through taaki do carts merge ho sake.

Instructions:

1. Class me private variable items rakho (total items in cart).
2. friend Cart operator+(Cart c1, Cart c2) implement karo jo merged cart return kare.
3. display() function banake total items show karo.
4. Test karo by merging two carts using +.

```
#include <iostream>
using namespace std;
// Cart class
class Cart {
private:
    int items;  // total items in cart
public:
    // Constructor
    Cart(int i = 0) {
        items = i;
    }
    // Friend function to overload + operator
```

```
Output
Cart1: Total items in cart:
3
Cart2: Total items in cart:
5
Merged Cart: Total items
in cart: 8
```

```
    friend Cart operator+(Cart c1, Cart c2);
    // Display function
    void display() {
        cout << "Total items in cart: " << items << endl;
    }
};
// Definition of friend function
Cart operator+(Cart c1, Cart c2) {
    Cart temp;
    temp.items = c1.items + c2.items;  // merge items
    return temp;
}
int main() {
    // Create two Cart objects
    Cart cart1(3);
    Cart cart2(5);
    cout << "Cart1: ";
    cart1.display();
    cout << "Cart2: ";
    cart2.display();
    // Merge carts using + operator
    Cart mergedCart = cart1 + cart2;
    cout << "Merged Cart: ";
    mergedCart.display();
    return 0;
}
```

Explanation:

1. items private variable hai jo cart me items store karta hai.
2. operator+ friend function ke through implement kiya gaya hai jo do carts ko merge karta hai aur naya Cart object return karta hai.
3. display() function total items show karta hai.
4. main() me cart1, cart2 create kiye aur merge test kiya.

Unary Operator Overloading

Practical Question: Ek Timer class banao jo seconds store kare. Prefix increment ++ operator overload karo taaki timer me 1 second add ho jaye. Instructions: Class me private variable seconds rakho. operator++() member function ke through implement karo. display() function se timer show karo. Test karo by incrementing timer multiple times.

```
#include <iostream>
using namespace std;
class Timer {
private:
    int seconds;  // Private variable to store seconds
public:
    // Constructor to initialize seconds
    Timer(int s = 0) {
        seconds = s;
    }
    // Prefix increment operator overload (++timer)
    Timer operator++() {
        seconds++;          // Increment seconds by 1
        return *this;       // Return updated object
    }
    // Display function to show current time
    void display() {
        cout << "Time: " << seconds << " seconds" << endl;
    }
};
int main() {
    // Create Timer object with initial 10 seconds
    Timer t(10);
    cout << "Initial ";
    t.display();
    // Increment timer using overloaded prefix ++
    ++t;
    cout << "After increment ";
    t.display();
    // Increment again
    ++t;
    cout << "After another increment ";
    t.display();
    return 0;
}
```

Output

```
Initial Time: 10 seconds
After increment Time: 11 seconds
After another increment Time: 12
seconds
```

Explanation:

1. seconds private variable me time store hota hai.
2. operator++() member function ke through prefix increment implement kiya gaya hai.
3. display() function se timer ka current value print hota hai.
4. main() me timer multiple times increment karke test kiya gaya hai

Binary Operator Overloading

Practical Question: Ek Vector class banao jo 2D vectors ko represent kare. + operator overload karo taaki do vectors ko add kiya ja sake. Instructions: Class me private variables x aur y rakho. operator+ member function ke through implement karo jo naya vector return kare. display() function se vector print karo (x, y) format me. Test karo addition of two vectors.

```
#include <iostream>
using namespace std;
class Vector {
private:
    int x, y;  // Private variables to store vector components
public:
    // Constructor to initialize vector
    Vector(int a = 0, int b = 0) {
        x = a;
        y = b;
    }
    // Overload + operator to add two vectors
    Vector operator+(Vector v) {
        Vector temp;
        temp.x = x + v.x;
        temp.y = y + v.y;
        return temp;  // Return new vector as result
    }
    // Display function to print vector
    void display() {
        cout << "(" << x << ", " << y << ")" << endl;
    }
};
int main() {
    // Create two vector objects
    Vector v1(2, 3);
    Vector v2(4, 1);
    cout << "Vector1: ";
    v1.display();

    cout << "Vector2: ";
    v2.display();
```

```
Output
Vector1: (2, 3)
Vector2: (4, 1)
Resultant Vector: (6,
4)
```

```
    // Add vectors using overloaded + operator
    Vector v3 = v1 + v2;
    cout << "Resultant Vector: ";
    v3.display();
    return 0;
}
```

Explanation:

1. x aur y private variables vector ke components store karte hain.
2. operator+ member function do vectors ko add karke naya vector return karta hai.
3. display() function vector ko (x, y) format me print karta hai.
4. main() me v1 aur v2 create karke + operator ke through addition test kiya gaya.

Stream Operator Overloading

Practical Question: Ek Complex class banao jo complex numbers store kare. << aur >> operators overload karo friend functions ke through taaki input/output easy ho jaye. Instructions: Class me private variables real aur imag rakho. >> operator overload karo input ke liye. << operator overload karo output ke liye a + bi format me. Test karo main() me.

```
#include <iostream>
using namespace std;
class Complex {
private:
    int real, imag;  // Private variables for real and imaginary parts
public:
    // Constructor
    Complex(int r = 0, int i = 0) {
        real = r;
        imag = i;
    }
    // Friend function to overload >> operator for input
    friend istream& operator>>(istream &in, Complex &c) {
        cout << "Enter real part: ";
        in >> c.real;
        cout << "Enter imaginary part: ";
        in >> c.imag;
        return in;
    }

    // Friend function to overload << operator for output
    friend ostream& operator<<(ostream &out, const Complex &c) {
        out << c.real << " + " << c.imag << "i";
        return out;
    }
```

```
Output
Enter real part: 3
Enter imaginary part: 4
You entered: 3 + 4i
```

```
};
int main() {
    Complex c1;
    // Input complex number using overloaded >> operator
    cin >> c1;
    // Output complex number using overloaded << operator
    cout << "You entered: " << c1 << endl;
    return 0;
}
```

Explanation:

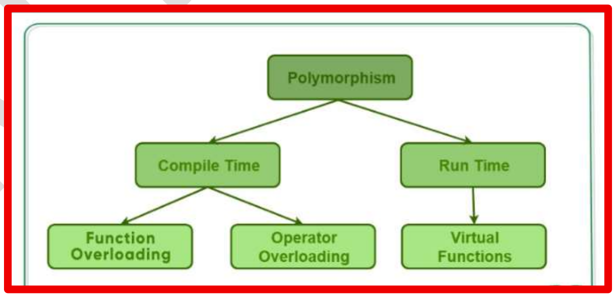
1. real aur imag private variables me complex number store hota hai.
2. >> operator friend function ke through input ke liye overload kiya gaya hai.
3. << operator friend function ke through output ke liye overload kiya gaya hai a + bi format me.
4. main() me input/output test kiya gaya hai.

Polymorphism

In C++, polymorphism means "one thing, many forms."
Literally:

- Poly = many
- Morph = forms

It allows a function or object to behave differently depending on context.



Types of Polymorphism in C++

1. Compile-timePolymorphism (Static Polymorphism)
   - The compiler decides which function to call at compile time.
   - Examples:
     - Function Overloading – same function name, different parameters.
     - Operator Overloading – giving operators new behavior for objects.
2. Run-time Polymorphism (Dynamic Polymorphism)
   - The decision of which function to call is made at run-time.
   - Usually done using Inheritance + Virtual Functions.

Function Overloading

```
#include <iostream>
using namespace std;
class Calculator {
public:
    // Function to add two integers
    int add(int a, int b) {
        return a + b;
    }
    // Function to add two doubles
    double add(double a, double b) {
        return a + b;
    }
};
int main() {
    Calculator calc;
    cout << "Integer add: " << calc.add(5, 3) << endl;     // calls int version
    cout << "Double add: " << calc.add(2.5, 3.5) << endl; // calls double version
}
```

Explanation

- #include <iostream> → allows input/output functions.
- using namespace std; → avoids writing std:: every time.
- class Calculator → defines a class with functions.
- Function Overloading:
- int add(int a, int b) → adds integers.
- double add(double a, double b) → adds doubles.
- Calculator calc; → creates an object of the class.
- calc.add(5,3) → calls integer add.
- calc.add(2.5,3.5) → calls double add.

Operator Overloading

```
#include <iostream>
using namespace std;
class Box {
public:
    int length;
    Box(int l) : length(l) {}

    // Overloading + operator to add lengths of two boxes
    Box operator+(const Box& b) {
```

```
Output
Combined length: 15
```

```
        return Box(length + b.length);
    }
};
int main() {
    Box b1(5), b2(10);
    Box b3 = b1 + b2;  // calls overloaded + operator
    cout << "Combined length: " << b3.length << endl;
}
```

Explanation

- #include <iostream>
    - Input/output ke liye library include karta hai (cout use karne ke liye).
- using namespace std;
    - std:: likhne ki zarurat nahi padti.
- class Box
    - Box naam ki ek class define ki gayi hai.
- public:
    - Iske andar ke members bahar se access ho sakte hain.
- int length;
    - Box ki length store karne ke liye variable.
- Box(int l) : length(l) {}
    - Constructor hai jo object bante hi length ko initialize karta hai.
- Box operator+(const Box& b)
    - + operator ko overload kiya gaya hai Box objects ke liye.
- const Box& b
    - Dusra Box reference ke through pass hota hai, taaki extra copy na bane.
- return Box(length + b.length);
    - Dono boxes ki length add karke naya Box return karta hai.
- int main()
    - Program yahin se start hota hai.
- Box b1(5), b2(10);
    - Do Box objects banaye gaye, lengths 5 aur 10 ke saath.
- Box b3 = b1 + b2;
    - Overloaded + operator call hota hai.
- cout << "Combined length: " << b3.length << endl;
    - Combined length screen par print hoti hai.
- Same interface, multiple implementations allow karta hai
- Same function name se different kaam kar sakte hain
- Code reusability badhata hai
- Code ko flexible aur extensible banata hai
- Runtime par decision lene mein help karta hai

- Function overriding support karta hai (base class pointer ke saath)
- Code ki maintainability improve hoti hai
- Code complexity reduce karta hai
- Real-world behavior ko easily represent karta hai
- Compile-time aur run-time polymorphism dono support karta hai

C++ Files

C++ files ka use program ko organize aur manage karne ke liye hota hai

- Large programs ko multiple files me divide karna easy hota hai

1. .cpp File (Source File)

- Actual program logic yahin likhi jaati hai
- Functions ki definitions hoti hain
- main() function usually isi file me hota hai
- Compiler is file ko compile karta hai

2. .h File (Header File)

- Declarations store karta hai (functions, classes, variables)
- Code ko reuse karne me help karta hai
- #include karke .cpp file me use hota hai
- Interface provide karta hai, implementation nahi

3. .hpp File

- .h ka hi advanced version hota hai
- Mostly C++ specific headers ke liye use hota hai
- Optional hota hai, .h bhi kaam karta hai

4. #include Statement

- Header file ko source file ke saath jodta hai
- Example: #include "file.h"

5. Header Guards

- Multiple inclusion se bachate hain
- Example:
- #ifndef FILE_H
- #define FILE_H
- // code
- #endif

6. Compilation Process

- .cpp file → Object file (.o)
- Object files → Executable file
- Header files directly compile nahi hote

1. ofstream (Output File Stream)

- Kaam: File create karna aur usmein data write karna.
- Agar file pehle se exist karti hai, toh wo overwrite ho jaati hai.
- Mostly jab humein data file mein save karna hota hai, tab use hota hai.

Example use:
Student details file mein store karna.

2. ifstream (Input File Stream)

- **Kaam:** File se **data read** karna.
- File pehle se exist honi chahiye, warna read nahi hoga.
- Jab humein file ka data program mein lana hota hai, tab use hota hai.

Example use:
File se marks ya records read karna.

3. fstream (File Stream)

- Kaam: Yeh ofstream + ifstream dono ka combo hai.
- Isse hum file create, read, aur write sab kuch kar sakte hain.
- Ek hi file par multiple operations ke liye useful hai.

Example use:
File mein data likhna aur baad mein wahi data read karn

ofstream – File create karna aur write karna

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ofstream file("student.txt");   // file create/open
    file << "Name: Rahul\n";
     file << "Age: 20\n";
    file << "Marks: 85\n";
    file.close();   // file band karna
    return 0;
}
```

Explanation

- student.txt file create hogi
- Student details file me write ho jayengi
- Agar file pehle se hogi, to overwrite ho jayegi

ifstream – File se data read karna

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ifstream file("student.txt");   // existing file open
    string line;
    while (getline(file, line)) {
        cout << line << endl;   // file ka data screen par print
    }
    file.close();   // file band
    return 0;
}
```

Kaam:

- student.txt file se data read karega
- Data console par display hoga
- File pehle se exist honi chahiye

fstream – Read + Write dono

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    fstream file;
    file.open("data.txt", ios::out | ios::in);
    // Write data
    file << "Hello File Handling\n";
    // Cursor wapas start par le jana
    file.seekg(0);
    // Read data
    string text;
    getline(file, text);
    cout << text << endl;
    file.close();
    return 0;
}
```

Kaam:

- File me likhna bhi
- Usi file se read bhi
- Ek hi file par multiple operations

C++ Errors kya hote hain?

Errors wo problems hoti hain jinke wajah se:

- Program compile nahi hota, ya
- Program run nahi hota, ya
- Program galat output deta hai

C++ me mainly 3 types ke errors hote hain

Compile-Time Errors

Ye errors compile karte time aate hain (program run hone se pehle).

Kyun aate hain?

- Syntax galat hota hai
- Rules of C++ follow nahi kiye jaate
- Spelling mistakes
- Semicolon ; bhool jana

Common examples:

- Semicolon missing

- Wrong keyword
- Variable declare nahi kiya

```
#include <iostream>
using namespace std;
int main() {
    int a = 10
    cout << a;
    return 0;
}
```

❌ Error: ; missing after int a = 10

Jab tak compile-time error fix nahi hoga, program run hi nahi hoga

Run-Time Errors

Ye errors program run hone ke baad aate hain.

Kyun aate hain?

- Galat logic jisse program crash ho jaye
- Invalid operations
- Memory related problems

Common examples:

- Divide by zero
- Array index out of range

```
int a = 10, b = 0;
cout << a / b;
```

❌ Error: Division by zero

Result:

- Program crash ho sakta hai
- Ya abnormal behavior dikhata hai

Logical Errors

Ye sabse dangerous hote hain

Program run bhi hota hai, error bhi nahi deta,
lekin output galat aata hai

Kyun aate hain?

- Logic galat hota hai
- Formula galat use hota hai
- Condition galat likhi hoti hai

Example:

```
int a = 10, b = 5;
cout << a + b * 2;
```

Output: 20
Agar aap expect kar rahe the (a + b) * 2 = 30
toh logic error hai (brackets nahi lagaye)

C++ Debugging kya hoti hai?

Debugging ka matlab hai: Program me jo errors (bugs) hain unko find karna, samajhna, aur fix karna.
Simple words me:
"Galti dhoondhna aur sudharna" = Debugging

Debugging kyun zaroori hai?

- Program crash ho raha ho
- Program galat output de raha ho
- Program expected result nahi de raha

Debugging ke Common Methods
Compiler Error Messages padhna

- Compiler hume batata hai:
  - Error kis line me hai
  - Kis type ka error hai

Example:
error: expected ';'
Matlab: semicolon missing hai

cout Statements ka use (Most Common)

- Variables ki value check karne ke liye cout lagate hain

```
cout << "Value of x: " << x << endl;
```

Isse pata chalta hai:

- Program kaunsa step tak pahucha
- Value galat kaha ho rahi hai

Step-by-Step Code Check karna

- Code ko line by line read karo
- Socho:
  - "Yeh line kya kaam kar rahi hai?"
  - "Yeh expected output de rahi hai ya nahi?"

Dry Run karna (Paper pe)

- Program ko man me ya paper par run karo
- Values ko step by step change hota hua likho

Ye logical errors pakadne me bahut helpful hota hai

Debugging Tools / Debugger
(Advanced but important)

- IDEs jaise:
  - Code::Blocks
  - Dev C++
  - Visual Studio

Features:

- Breakpoints lagana
- Line by line execution
- Variable values live dekhna

Debugging vs Errors

- **Errors** → Problem hoti hai
- **Debugging** → Us problem ko fix karne ka process

C++ Exception kya hoti hai?

Exception ek run-time problem hoti hai
jo program ke normal flow ko disturb kar deti hai.
Simple words me:
Jab program chal raha hota hai aur unexpected error aa jaata hai
Us error ko exception kehte hain

Exception Handling kyun zaroori hai?
Agar exception handle na karein to:

- Program crash ho sakta hai
- Program abruptly stop ho jaata hai

Exception handling se:

- Program safe tareeke se error handle karta hai
- Program crash hone ke bajay proper message deta hai

C++ me Exception Handling ke 3 main keywords
try

- Is block me risky code likha jaata hai
- Jahan exception aane ke chances hote hain

throw

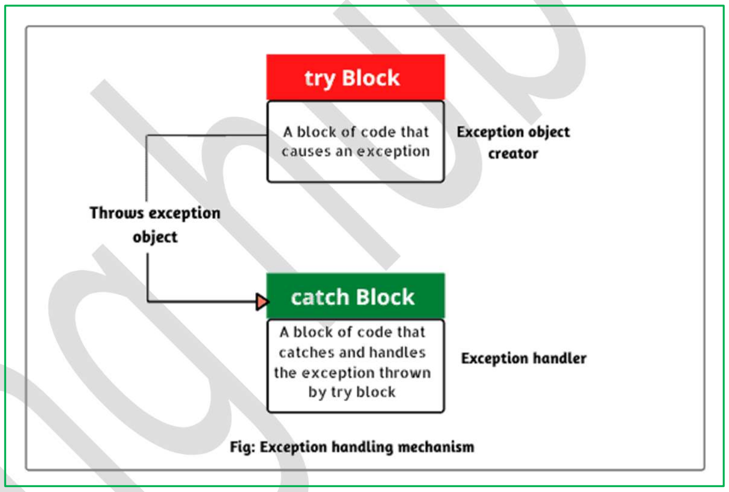
- Jab error detect hota hai, tab exception throw ki jaati hai

catch

- Ye exception ko catch karta hai aur handle karta hai

Exception Handling ka Flow

1. Program try block me jaata hai
2. Agar error aata hai → throw hota hai
3. Control turant catch block me chala jaata hai
4. Error handle hota hai
5. Program safely continue ya stop hota hai


try Block
A block of code that causes an exception
Exception object creator
Throws exception object
catch Block
A block of code that catches and handles the exception thrown by try block
Exception handler
Fig: Exception handling mechanism

try block

(Risky code ko try me rakhna)

```
#include <iostream>
using namespace std;
int main() {
    int a = 10, b = 0;
    try {
        cout << a / b;   // risky code
    }
    catch (...) {
        cout << "Error occurred";
    }
    return 0;
}
```

Yahan error aa sakta hai, isliye code try me hai.

throw

(Error detect hote hi exception throw karna)

```
#include <iostream>
using namespace std;
int main() {
    int age;
    cout << "Enter age: ";
    cin >> age;
    try {
        if (age < 18) {
            throw age;   // exception throw
        }
        cout << "You are eligible";
    }
    catch (int x) {
        cout << "Error: Age must be 18 or above";
    }
    return 0;
}
```

throw age; error ko catch block tak bhejta hai.

catch

(Exception ko handle karna)

```
#include <iostream>
using namespace std;
int main() {
    try {
        throw 10;   // direct exception
    }
    catch (int e) {
        cout << "Exception caught: " << e;
    }
    return 0;
}
```

catch exception ko pakad kar handle karta hai.

Short Summary

- **try** → jahan error aa sakta hai
- **throw** → error ko bhejna
- **catch** → error ko handle karna

Input Validation kya hai?

User se jo input liya jaa raha hai, usko check karna ki wo valid hai ya nahi.
Simple words me:

- Agar user galat value daale → program ko crash ya wrong output na mile
- Program sirf correct data accept kare

Kyun zaroori hai?

- User galat input de sakta hai
- Division by zero ya invalid operations se program crash ho sakta hai
- Data consistency maintain karne ke liye

Common Input Validation Techniques

Range Check

- Input allowed range me ho
- Example: Age must be 1–100

```
#include <iostream>
using namespace std;
int main() {
    int age;
    cout << "Enter your age (1-100): ";
    cin >> age;
    if(age >= 1 && age <= 100) {
        cout << "Valid age: " << age;
    } else {
        cout << "Invalid age!";
    }
    return 0;
}
```

```
Output
Enter your age (1-100): 25
Valid age: 25
```

Type Check

- Input correct type ka ho
- Example: User se number expected, na ki letter

```
#include <iostream>
using namespace std;
int main() {
    int num;
    cout << "Enter a number: ";
    if (!(cin >> num)) {   // check if input is not an integer
        cout << "Invalid input! Please enter a number.";
    } else {
```

```
Output
Enter a number: 42
You entered: 42
```

```
        cout << "You entered: " << num;
    }
    return 0;
}
```

Loop-based Validation

- User tab tak input de jab tak valid na ho
- Example: Age 1–100 tak check

```
#include <iostream>
using namespace std;
int main() {
    int age;
    do {
        cout << "Enter your age (1-100): ";
        cin >> age;
        if(cin.fail()) {       // type check
            cin.clear();       // clear error flag
            cin.ignore(1000,'\n'); // ignore wrong input
            age = -1;          // force invalid
        }
    } while(age < 1 || age > 100);
    cout << "Valid age entered: " << age;
    return 0;
}
```

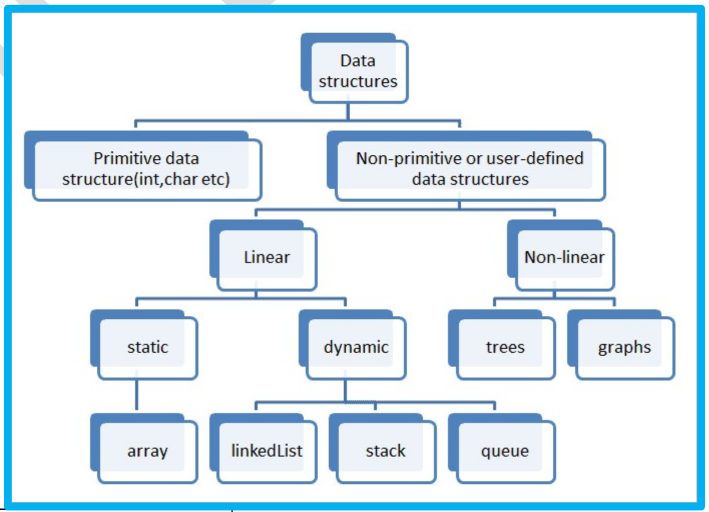
```
Output
Enter your age (1-100): 150
Enter your age (1-100): abc
Enter your age (1-100): -5
Enter your age (1-100): 30
Valid age entered: 30
```

C++ Data Structures kya hain?

Data Structures ek aisa way hai data ko organize karne ka, taaki:

- Data efficiently store ho
- Data fast access ho

C++ me common data structures:


Data structures
Primitive data structure(int,char etc)
Non-primitive or user-defined data structures
Linear
Non-linear
static
dynamic
trees
graphs
array
linkedList
stack
queue

Data Structure	Description
Array	Same type elements ka fixed-size collection

Data Structure	Description
Linked List	Nodes ka chain jisme data + pointer hota hai
Stack	LIFO (Last In First Out) structure
Queue	FIFO (First In First Out) structure
Deque	Double-ended queue, front & back se access possible
Tree	Hierarchical data, parent-child relationship
Graph	Nodes + edges, complex relations

STL (Standard Template Library) kya hai?
STL ek C++ library hai jo ready-made data structures aur algorithms provide karti hai.
STL Components

1. Containers → Data store karte hain
   - vector, list, deque, stack, queue, set, map, etc.
2. Algorithms → Common operations perform karte hain
   - sort(), reverse(), find(), accumulate(), etc.
3. Iterators → Containers ko traverse karne ke liye

Vector kya hai?

- Vector ek dynamic array hai
- Size automatically grow ya shrink ho sakta hai
- Random access possible (v[i])
- STL ka part hai → #include <vector>

Vector vs Array

Feature	**Array**	**Vector**
Size	Fixed	Dynamic
Memory	Static	Dynamic
Functions	Limited	push_back, pop_back, size, etc.

Create & Display Vector

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    // Vector create
    vector<int> v;
    // Elements add
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    // Display all elements
    cout << "Vector elements: ";
    for(int i = 0; i < v.size(); i++)
        cout << v[i] << " ";
    // Display first & last elements
    cout << "\nFirst element: " << v.front();
    cout << "\nLast element: " << v.back();
    return 0;
}
```

```
Output
Vector elements: 10 20
30
First element: 10
Last element: 30
```

Explanation

- #include <vector> → Vector use karne ke liye STL header include kiya
- vector<int> v; → Integer type ka empty vector create kiya
- v.push_back(10); → Vector ke end me 10 add kiya
- v.push_back(20); → Vector ke end me 20 add kiya
- v.push_back(30); → Vector ke end me 30 add kiya
- for(int i = 0; i < v.size(); i++) cout << v[i]; → Vector ke saare elements print kiye
- v.front() → Vector ka pehla element access kiya
- v.back() → Vector ka last element access kiya

Access a Vector

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v = {10, 20, 30, 40, 50};
    // Access by index
    cout << "First element: " << v[0] << endl;
    cout << "Second element: " << v.at(1) << endl;
    // Access front and back
    cout << "Front element: " << v.front() << endl;
    cout << "Back element: " << v.back() << endl;
    // Access using loop
    cout << "All elements: ";
    for(int i = 0; i < v.size(); i++)
        cout << v[i] << " ";
    cout << "\nUsing range-based loop: ";
    for(int x : v)
        cout << x << " ";
    return 0;
}
```

Explanation

- vector<int> v = {10, 20, 30, 40, 50}; → Vector create kiya aur initial elements assign kiye
- v[0] → First element access kiya using index
- v.at(1) → Second element access kiya using at(), safe method (out-of-range check karta hai)
- v.front() → Vector ka first element access kiya
- v.back() → Vector ka last element access kiya
- for(int i = 0; i < v.size(); i++) cout << v[i]; → Vector ke saare elements loop ke through print kiye
- for(int x : v) cout << x; → Range-based for loop se vector ke elements print kiye

at() Function

- Vector ke specific index ka element access karne ke liye use hota hai
- Safe method → agar index out of range ho → exception throw hota hai

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v = {10, 20, 30, 40, 50};
    cout << "Element at index 0: " << v.at(0) << endl; // first element
    cout << "Element at index 2: " << v.at(2) << endl; // third element
    // Example of out-of-range (commented to avoid crash)
    // cout << v.at(10) << endl; // Throws exception
    return 0;
}
```

Explanation

- #include <vector> → Vector use karne ke liye STL header include kiya
- vector<int> v = {10, 20, 30, 40, 50}; → Vector create kiya aur initial elements assign kiye
- v.at(0) → Vector ka first element access kiya (safe method, index check karta hai)
- v.at(2) → Vector ka third element access kiya
- // v.at(10) → Agar uncomment karte to, out-of-range exception throw hota (program crash se bachata hai)
- cout << ... << endl; → Elements print kiye

Vector Element Change Karne ke Tarike

1. **Using Index ([])**
   - Direct element ko index ke through access karke change kar sakte ho
   - Example: v[0] = 100; → first element 100 se replace ho jayega
   - Fast hai, lekin agar index invalid hai toh program crash ho sakta hai
2. **Using at() Function**
   - Safe method
   - Example: v.at(2) = 300; → third element 300 se replace ho jayega
   - Agar index out of range ho → exception throw karta hai, crash se bachata hai

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
   vector<int> v = {10, 20, 30, 40, 50};
   cout << "Original Vector: ";
   for(int x : v) cout << x << " ";
   // Modify all elements (multiply by 2)
   for(int i = 0; i < v.size(); i++) {
      v[i] = v[i] * 2;   // each element doubled
   }
   cout << "\nModified Vector: ";
   for(int x : v) cout << x << " ";
   return 0;
}
```

```
Output
Original Vector: 10 20 30 40 50
Modified Vector: 20 40 60 80 100
```

Explanation

- vector<int> v = {10, 20, 30, 40, 50};
  → Vector create kiya aur initial elements assign kiye
- for(int x : v) cout << x << " ";
  → Original vector elements print kiye using range-based for loop

- for(int i = 0; i < v.size(); i++) { v[i] = v[i] * 2; }
  → Loop ke through vector ke har element ko modify kiya
  → Har element ko 2 se multiply kiya (doubled)
- for(int x : v) cout << x << " ";
  → Modified vector elements print kiye

Remove Last Element: pop_back()

- v.pop_back(); → Vector ke last element ko remove karta hai
- No arguments needed → hamesha last element remove hota hai
- Vector ka size automatically decrease ho jata hai

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v = {10, 20, 30, 40, 50};
    cout << "Original Vector: ";
    for(int x : v) cout << x << " ";
    // Remove last element
    v.pop_back();
    cout << "\nVector after pop_back(): ";
    for(int x : v) cout << x << " ";
    return 0;
}
```

Output
```
Original Vector: 10 20 30 40 50
Vector after pop_back(): 10 20
30 40
```

Explanation:

- v.pop_back(); → Vector ka last element (50) remove ho gaya
- Vector ka size automatically 5 → 4 ho gaya
- Agar vector empty hota aur pop_back() call karte → undefined behavior

Vector Size

- Function: v.size()
- Kaam: Vector me kitne elements currently present hain, yeh batata hai
- Return type: size_t (unsigned integer)
- Vector ka size automatically update hota hai jab elements add ya remove hote hain

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v = {10, 20, 30};
    cout << "Original Vector size: " << v.size() << endl;
```

Explanation
```
Original Vector size: 3
After push_back(40), Vector
size: 4
After pop_back(), Vector size: 3
```

```
    // Add element
    v.push_back(40);
    cout << "After push_back(40), Vector size: " << v.size() << endl;
    // Remove last element
    v.pop_back();
    cout << "After pop_back(), Vector size: " << v.size() << endl;
    return 0;
}
```

Explanation:

- v.size() → current number of elements return karta hai
- Vector me elements add/remove karne par size automatically update hota hai

Check if Vector is Empty

- Function: v.empty()
- Kaam:
  - Agar vector me koi element nahi hai → true return karta hai
  - Agar vector me elements hain → false return karta hai
- Return type: bool

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<string> cars;
  cout << cars.empty();  // Outputs 1 (The vector is empty)
  return 0;
}
```

```
Output
0
```

Explanation

- #include <iostream>
- Ye header file input-output ke liye use hoti hai (jaise cout).
- #include <vector>
- Ye header file vector container use karne ke liye hoti hai.
- using namespace std;
- Isse hume baar-baar std:: likhne ki zarurat nahi padti.
- int main()
- Program execution yahin se start hota hai.
- vector<string> cars;
- Ye ek vector declare karta hai jisme string type ke elements store honge.
- Abhi vector empty hai, koi value add nahi ki gayi.
- cars.empty()
- Ye function check karta hai ki vector empty hai ya nahi.
- Agar empty ho → true (1 return karega)

- Agar empty na ho → false (0 return karega)
- cout << cars.empty();
- Kyunki vector empty hai, output hoga 1.
- return 0;
- Program successfully end ho gaya.

Loop Through a Vector

Vector ke har element ko loop (for / range-based for) ki help se ek-ek karke access karna hi loop through a vector kehlata hai.

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};
  for (int i = 0; i < cars.size(); i++) {
    cout << cars[i] << "\n";
  }
  return 0;
}
```

```
Output
Volvo
BMW
Ford
Mazda
```

Explanation

- #include <iostream>
- Input aur output ke liye use hota hai (cout).
- #include <vector>
- Vector container use karne ke liye header file.
- using namespace std;
- std:: baar-baar likhne se bachata hai.
- int main()
- Program execution yahin se start hota hai.
- vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};
- Ek vector banaya gaya hai jisme **car names (strings)** store hain.
- for (int i = 0; i < cars.size(); i++)
- for loop vector ke size tak chalega.
- cars.size() vector me total elements batata hai.
- cout << cars[i] << "\n";
- Har iteration me vector ka ek element print hota hai.
- return 0;
- Program successfully end ho jata hai.

C++ List

std::list kya hoti hai?

- std::list C++ STL ka ek container hai
- Ye internally Doubly Linked List par based hoti hai
- Har element (node) ke paas 3 cheezein hoti hain:
    1. Data
    2. Previous node ka address
    3. Next node ka address

Is wajah se elements memory mein alag-alag jagah par store hote hain (continuous nahi).

Memory Structure

- Vector → continuous memory
- List → non-continuous memory
- Iska matlab:
    - Vector fast access deta hai
    - List fast insertion & deletion deta hai

Direct Index Access

- List me list[i] kaam nahi karta
- Kyunki indexing support nahi hoti
- Elements access karne ke liye iterator ya loop use hota

Create a List

ka matlab hota hai C++ program mein std::list container ko declare aur initialize karna, taaki hum usme data store, add, remove aur manage kar sakein.

List kya hoti hai?

- List C++ STL ka ek container hai
- Ye Doubly Linked List par based hoti hai
- Elements order (sequence) mein store hote hain
- Memory continuous nahi hoti (vector se different)

```
#include <iostream>
#include <list>
using namespace std;
int main() {
   // Create a list
   list<int> numbers;
   // Add elements to the list
```

```
Output
5 10 20
```

```
    numbers.push_back(10);
    numbers.push_back(20);
    numbers.push_front(5);
    // Print the list
    for (int x : numbers) {
        cout << x << " ";
    }
    return 0;
}
```

Explanation

- #include <iostream>
- Output dikhane ke liye use hota hai (cout).
- #include <list>
- C++ STL ka list container use karne ke liye.
- using namespace std;
- std:: baar-baar likhne se bachata hai.
- int main()
- Program yahin se start hota hai.
- list<int> numbers;
- Ek empty list banayi gayi hai jo integer values store karegi.
- numbers.push_back(10);
- List ke end (last) mein 10 add karta hai.
- numbers.push_back(20);
- List ke end mein 20 add karta hai.
- numbers.push_front(5);
- List ke starting (front) mein 5 add karta hai.
- for (int x : numbers)
- Range-based for loop list ke har element par ek-ek karke chalta hai.
- cout << x << " ";
- List ke elements ko print karta hai.
- return 0;
- Program successfully end ho jata hai.

```
Output
5 10 20
```

Change a List Element in C++

- std::list mein direct indexing (list[i]) support nahi hota, isliye element change karne ke liye iterator ya loop use karna padta hai.

```
#include <iostream>
#include <list>
using namespace std;
```

```
Output
10 20 35 40
```

```
int main() {
    // Create a list
    list<int> numbers = {10, 20, 30, 40};
    // Use iterator to change 3rd element (30 → 35)
    auto it = numbers.begin();  // Iterator pointing to first element
    advance(it, 2);             // Move iterator 2 steps forward
    *it = 35;                   // Change value
    // Print updated list
    for (int x : numbers) {
        cout << x << " ";
    }
    return 0;
}
```

Explanation

- #include <iostream>
- Output ke liye use hota hai (cout).
- #include <list>
- C++ STL ka list container use karne ke liye.
- using namespace std;
- std:: baar-baar likhne se bachata hai.
- list<int> numbers = {10, 20, 30, 40};
- Ek integer list banayi gayi aur initial values set ki gayi hain.
- auto it = numbers.begin();
- it naam ka iterator banaya jo list ke first element par point karta hai.
- advance(it, 2);
- Iterator ko 2 steps aage move kiya → ab ye 3rd element (30) par point kar raha hai.
- *it = 35;
- Iterator ke current element ki value change kar di → 30 ko 35 se replace kar diya.
- for (int x : numbers)
- Range-based for loop list ke har element par chalta hai.
- cout << x << " ";
- List ke updated elements ko print karta hai.
- return 0;
- Program successfully end ho jata hai.

List Size in C++

C++ std::list ka size find karne ke liye size() function use hota hai. Ye list me total elements ki count return karta hai.

```
Output
Size of the list: 5
```

```
#include <iostream>
#include <list>
using namespace std;
int main() {
    // Create a list
    list<int> numbers = {10, 20, 30, 40, 50};
    // Get size of the list
    cout << "Size of the list: " << numbers.size() << endl;
    return 0;
}
```

Explanation

- #include <iostream>
  - Output ke liye use hota hai (cout).
- #include <list>
  - C++ STL ka list container use karne ke liye.
- using namespace std;
  - std:: baar-baar likhne se bachata hai.
- list<int> numbers = {10, 20, 30, 40, 50};
  - Ek integer list banayi aur initial elements assign kiye.
- numbers.size()
  - Ye function list ke total elements ko count karta hai.
  - Yahan list me 5 elements hain → size = 5
- cout << "Size of the list: " << numbers.size() << endl;
  - List ka size print karta hai.
- return 0;
  - Program successfully end ho jata hai.

Check if a List is Empty in C++

C++ std::list me empty() function use karke check karte hain ki list empty hai ya nahi.

- Agar list empty hai → true (1) return karega
- Agar list me elements hain → false (0) return karega

```
#include <iostream>
#include <list>
using namespace std;
int main() {
    // Create an empty list
    list<int> numbers;
    // Check if the list is empty
    if (numbers.empty()) {
        cout << "The list is empty." << endl;
```

```
Output
The list is empty.
The list is not empty.
```

```
    } else {
        cout << "The list is not empty." << endl;
    }
    // Add elements to the list
    numbers.push_back(10);
   // Check again
    if (numbers.empty()) {
        cout << "The list is empty." << endl;
    } else {
        cout << "The list is not empty." << endl;
    }
    return 0;
}
```

Explanation

- #include <iostream>
  - Output ke liye use hota hai (cout).
- #include <list>
  - C++ STL ka list container use karne ke liye.
- using namespace std;
  - std:: baar-baar likhne se bachata hai.
- list<int> numbers;
  - Ek empty integer list banayi gayi.
- numbers.empty()
  - Ye function check karta hai ki list empty hai ya nahi.
  - Agar empty hai → true (1) return karega
  - Agar empty nahi hai → false (0) return karega
- if (numbers.empty()) { ... } else { ... }
  - Condition check karke message print karte hain:
    - Pehli baar list empty hai → "The list is empty." print hoga
- numbers.push_back(10);
  - List me element add kiya → ab list empty nahi hai
- Dobara numbers.empty() check
  - Ab list empty nahi → "The list is not empty." print hoga
- return 0;
  - Program successfully end ho jata hai

Loop Through a List in C++ (Hinglish + Simple Explanation)

- C++ me std::list ko loop karke traverse karna hota hai tab hume elements ek-ek karke access karne padte hain kyunki list me direct indexing (list[i]) nahi hota.

```
#include <iostream>
#include <list>
using namespace std;
int main() {
   list<int> numbers = {10, 20, 30, 40, 50};
   // Loop through list using range-based for
   for (int x : numbers) {
      cout << x << " ";
   }
   return 0;
}
```

Explanation

- #include <iostream>
  → Ye line output ke liye hoti hai (jaise cout).
- #include <list>
  → Ye C++ ka list container use karne ke liye include kiya gaya header file hai.
- using namespace std;
  → Isse hume baar-baar std:: nahi likhna padta.
- list<int> numbers = {10, 20, 30, 40, 50};
  → Yahan humne ek integer list banayi aur usme 5 values initialize kiye.
  → List ek STL container hai jo elements ko sequence order me store karta hai.
- for (int x : numbers)
  → Ye range-based for loop hai — C++11 se available feature hai jo container jaise list ke sabhi elements par loop chalata hai. → Is loop me:
- int x har iteration me list ka current element value leta hai
- numbers wo container hai jiske elements par loop chal raha hai
- cout << x << " ";
  → Har element ko ek space ke sath print karta hai.
- return 0;
  → Program successfully end ho jata hai.

C++ Stack

Stack ek special type ka container hai jo Last-In-First-Out (LIFO) principle follow karta hai — jo last insert hota hai wahi pehle remove hota hai (jaise plate ka stack).

Create a Stack in C++

To create a stack in C++, we use the STL stack container from the <stack> header file. A stack follows LIFO (Last-In-First-Out) meaning the last item you push (add) is the first one you pop (remove).

```
#include <iostream>
```

```
#include <stack>
using namespace std;
int main() {
   // Create a stack of integers
   stack<int> myStack;
   // Add elements to the stack
   myStack.push(10);
   myStack.push(20);
   myStack.push(30);
   // Print and remove elements from the stack
   while (!myStack.empty()) {
      cout << myStack.top() << " ";  // Show top element
      myStack.pop();                 // Remove it
   }
   return 0;
}
```

```
Output
30 20 10
```

Explanation

- #include <stack>
  → Stack header include karta hai taaki hum stack use kar saken.
- stack<int> myStack;
  → Ye line ek stack of integers banati hai jisme hum numbers store karenge.
- myStack.push(10);
  → 10 ko stack ke top par add karta hai.
- myStack.push(20);
  → 20 bhi top par add karta hai (ab 20 top hai).
- myStack.push(30);
  → 30 ko top par add karta hai (ab 30 sabse upar).
- while (!myStack.empty())
  → Jab tak stack empty nahi hota, loop chalta hai.
- myStack.top()
  → Ye function current top element show karta hai bina remove kiye.
- myStack.pop()
  → Ye top element ko remove karta hai.

Get the Size of the Stack

In C++ STL, jab aapko stack me kitne elements (items) hain ye pata karna ho, to aap size() function use karte ho. Ye function stack me present elements ki total count return karta hai.

```
#include <iostream>
#include <stack>
```

```
using namespace std;
int main() {
    stack<int> st;
    st.push(10);
    st.push(20);
    st.push(30);
    cout << "Size of stack: " << st.size() << endl;
        return 0;
}
```

Explanation

- stack<int> st;
  → Ek stack of integers banaya gaya hai.
- st.push(10); st.push(20); st.push(30);
  → Stack me 3 elements add kiye gaye.
- st.size()
  → Ye function stack ke elements ki total count ko return karta hai — yahan 3.
- cout << st.size()
  → Stack ka size print karta hai.

Step-by-Step Explanation

- #include <stack>
  → Is header file se hum C++ ka stack container use kar paate hain.
- stack<int> st;
  → Yahan ek stack of integers banaya gaya hai jiska naam st hai.
- st.push(10);, st.push(20);, st.push(30);
  → Ye lines stack ke top par elements add karti hain.
  → Stack ka order ab: 10 (bottom), 20, 30 (top) — follow karta hai LIFO principle (Last In, First Out).
- st.top()
  → Ye function current top element ko return karta hai bina remove kiye. Yahan top element hoga 30.
- st.pop();
  → Ye line top element (30) ko stack se remove kar deti hai. Note: pop() value return nahi karta, bas remove karta hai.
- if (!st.empty()) { ... }
  → empty() check karta hai agar stack khaali hai ya nahi.
  → !st.empty() ka matlab hai *"stack empty nahi hai"*, isliye inside block me naya top print hoga.
- st.size()
  → Ye function stack me total elements count return karta hai.
- return 0;
  → Program normal tarike se end hota hai.

Change the Top Element

In C++ std::stack, you can directly change the top element by using the .top() function because it returns a reference to the element at the top — so you can modify it like a normal variable

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<int> st;
    st.push(10);
    st.push(20);
    st.push(30);
    cout << "Top element: " << st.top() << endl;
    st.pop();  // Remove top
    if (!st.empty()) {
        cout << "After pop, new top: " << st.top() << endl;
    }
    cout << "Size of stack: " << st.size() << endl;
    return 0;
}
```

```
Output
Tesla BMW Volvo
```

Explanation

- stack<string> cars;
  → Yahan ek stack banaya gaya hai jisme string values store hoti hain.
- cars.push("Volvo"); cars.push("BMW"); cars.push("Ford");
  → Ye lines elements top ke upar add kar rahi hain — stack ab is order me hai:
  Ford (top), BMW, Volvo.
- cars.top() = "Tesla";
  → .top() se hum top element ka reference le rahe hain aur uski value change kar rahe hain.
  → Ab top element "Tesla" ho jayega instead of "Ford".
- while (!cars.empty()) { ... }
  → Stack ke elements ko top se bottom tak print karta hai aur pop() se remove bhi karta hai.

Remove Elements from a Stack in C++

In C++ me stack se elements remove karne ke liye hum .pop() function use karte hain.
Stack ek LIFO (Last-In-First-Out) structure hota hai — matlab jo element sabse last add hua hai, wahi sabse pehle remove hota hai.

.pop() Function

- .pop() top element ko remove karta hai.
- Ye function koi value return nahi karta (sirf delete karta hai).

- Sirf top element remove hota hai, baki elements unchanged rehte hain

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<int> st;
    st.push(10);
    st.push(20);
    st.push(30);

    // Remove the top element
    st.pop();  // 30 remove
    st.pop();  // 20 remove
    // Now only 10 remains
    cout << "Current top: " << st.top() << endl;
    return 0;
}
```

```
Output
Current top: 10
```

Explanation

- stack<int> st;
  → Ek integer stack banaya.
- st.push(10); st.push(20); st.push(30);
  → Pehle 10, phir 20, phir 30 top par add hua.
  → Stack ab: 30 (top), 20, 10 (bottom).
- st.pop();
  → Top element 30 remove ho gaya.
  → Phir st.pop(); se 20 remove ho gaya.
  → Ab stack me sirf 10 bacha hai.
- st.top()
  → Ab new top element print hota hai → 10.

C++ Queue

Queue ek data structure hai jo FIFO (First-In-First-Out) principle follow karta hai —
jo pehle insert hua hota hai wo pehle remove hota hai, exactly waise jaise line me log queue.

Queue in C++ STL

C++ Standard Library me queue ko std::queue ke through use karte hain.

➤ **FIFO – First In, First Out**

- Push (enqueue) → new element last/back me add hota hai
- Pop (dequeue) → element front se remove hota hai

- front() → queue ka first element return
- back() → queue ka last element return
- empty() → check karta hai queue khaali hai ya nahi
- size() → total elements count karta hai

```
#include <iostream>
#include <queue>
using namespace std;
int main() {
    queue<int> q;
    // Add (push) elements
    q.push(10);
    q.push(20);
    q.push(30);
    // Print front and back
    cout << "Front: " << q.front() << endl;
    cout << "Back: " << q.back() << endl;
    // Remove (pop) front element
    q.pop();
    // Print new front
    cout << "Front after pop: " << q.front() << endl;
    return 0;
}
```

Explanation

- queue<int> q;
  → Ek integer queue banayi.
- q.push(10); q.push(20); q.push(30);
  → Elements 10, 20, 30 queue me tail/back se add hue.
- q.front()
  → Queue ka pehla element (10) return karega.
- q.back()
  → Queue ka last element (30) return karega.
- q.pop()
  → Front element (10) queue se remove ho jayega.
- Phir q.front() ab 20 return karega kyunki 10 remove ho gaya.

Why We used

- FIFO (First-In-First-Out) order follow karta hai — pehle jo element add hota hai, wahi pehle remove hota hai. Ye real-life lines jaisa behavior hota hai.
- Jab order matter karta ho, matlab pehle aaye ko pehle serve karna ho (jaise ticket line), to queue perfect hai.
- Task scheduling me use hota hai — CPU jobs ya printer jobs ko arrival ke order me process karne ke liye.
- Buffering aur data streaming ke liye queues use hote hain — ek side se data add hota hai aur dusri side se process ho jata hai.
- Graph algorithms, jaise Breadth-First Search (BFS) me nodes ko level-by-level process karne ke liye queue use hota hai.
- Networking me packet handling ya web requests order me process karne ke liye queues sahi choice hote hain.
- Queue ka use resource allocation me bhi hota hai — shared resources (CPU, printer) ko fair way me allocate karna easy hota hai.
- Code readability improve hoti hai — queue ka naam dekhte hi pata chal jata hai ki order maintain karna hai (FIFO)

C++ Deque

- Deque C++ ka ek special container hai jo elements ko dono ends se add aur remove karne deta hai — front aur back dono se. Isko "deck" (deque) bola jata hai.

Why Use Deque?

- Jab hume front aur back dono se efficient insert/delete chahiye.
- Jab queue se zyada flexible behavior chahiye (FIFO + stack like insert at front).
- Agar simple queue (sirf front se delete aur back se add) enough nahi hai.

Function	**Meaning**
push_back(x)	Element x ko end (back) me add kare.
push_front(x)	Element x ko start (front) me add kare.
pop_back()	Back se element remove kare.
pop_front()	Front se element remove kare.
front()	Deque ka first element return kare.
back()	Deque ka last element return kare.
size()	Total elements count return kare.

Function	Meaning
empty()	Bataata hai deque empty hai ya nahi.

```
#include <iostream>
#include <deque>
using namespace std;

int main() {
    deque<int> dq;
    dq.push_back(10);    // Add at back
    dq.push_front(5);    // Add at front
    dq.push_back(15);
    cout << "Front: " << dq.front() << endl;  // 5
    cout << "Back: " << dq.back() << endl;    // 15
    dq.pop_front();      // Removes 5
    dq.pop_back();       // Removes 15
    cout << "After pops, front: " << dq.front() << endl;  // 10
    cout << "Size: " << dq.size() << endl;               // 1
    return 0;
}
```

Explanation (Hinglish)

- #include <deque>
  → C++ ke deque container ko use karne ke liye required header file.
- deque<int> dq;
  → Ek deque of integers banaya gaya hai jiska naam dq hai. Ye container double-ended queue hota hai jisme elements ko front aur back dono se add/remove kiya ja sakta hai.
- dq.push_back(10);
  → Value 10 ko back (end) mein add karta hai.
- dq.push_front(5);
  → Value 5 ko front (start) mein add karta hai.
- dq.push_back(15);
  → Value 15 ko back mein add karta hai. Ab deque ka order hai: 5, 10, 15.

C++ Sets kya hoti hain?

- Set C++ Standard Template Library (STL) ka ek associative container hai.

- Iska purpose unique elements ko store karna hota hai — yani duplicate values allowed nahi hoti.
- Set me values automatically sorted order me store hoti hain (default ascending).
- Internally set ek balanced binary search tree (jaise Red-Black tree) ki tarah implement hota hai, isliye operations (insert, search, delete) efficient (O(log n)) hote hain.
- Set me aap indexing (set[i]) use nahi kar sakte — values ko unke value key ke through access karte hain.
- Ek baar element insert ho gaya toh usko direct modify nahi kar sakte — pehle remove karke phir new value insert karna padta hai.

why we use C++ Sets

- Duplicate values nahi chahiye → Set me sirf unique elements store hote hain.
- Automatically sorted order chahiye → Set elements by default ascending order me rehte hain.
- Fast search/insert/delete chahiye → Set me yeh operations efficiently (O(log n)) hote hain.
- Sorted aur organized data chahiye → Useful jab order important ho (like unique sorted list).
- Duplicates ko automatically ignore karna hai — same value dobara add nahi hoti.

```
#include <iostream>
#include <set>
using namespace std;
int main() {
    // Create a set of integers
    set<int> s;
    // Insert elements
    s.insert(30);
    s.insert(10);
    s.insert(20);
    s.insert(20);  // duplicate, will NOT be added
    // Print all elements (sorted & unique)
    cout << "Set elements: ";
    for (int x : s) {
        cout << x << " ";
    }
    cout << endl;
    //Remove an element
    s.erase(10);
    // Print updated set
    cout << "After erase: ";
```

```
Output
Set elements: 10 20 30
After erase: 20 30
```

```
    for (int x : s) {
        cout << x << " ";
    }
    cout << endl
return 0;
}
```

Explanation

- set<int> s; → Ek set of integers banaya (jo unique aur sorted data rakhega).
- s.insert(...) → Values add karta hai; agar duplicate value hai to set use ignore karta hai.
- Loop ke through set ke elements sorted order me print hote hain.
- s.erase(10); → Value 10 set se delete kar di; agar value nahi milti to kuch nahi hota

Map kya hai?

C++ me map ek container hai jo key-value pairs store karta hai.

- Key: Unique identifier (jaise dictionary me word)
- Value: Key ka corresponding data (jaise word ka meaning)

Why we use C++ maps

- Store key-value pairs → Data ko ek unique key ke saath store karte hain.
- Fast lookup/search → Kisi specific key ka value quickly mil jata hai (O(log n)).
- Unique keys → Ek hi key duplicate nahi ho sakti; agar same key aaye to value replace ho jata hai.
- Automatically sorted keys → Map me keys ascending order me stored hoti hain.
- Easy iteration → Keys aur values ko loop ke through easily access kar sakte hain.
- Frequency/count problems → Characters, numbers, ya items ka count nikalne me helpful.
- Dictionary-like problems → Jab word aur meaning, student aur marks, ya id aur data store karna ho.

```
#include <iostream>
#include <map>
using namespace std;
int main() {
    // 1. Creating a map of student names and their marks
    map<string, int> marks;
    // 2. Inserting key-value pairs
```

```
Output
Alice ka marks: 95

All students ke marks (sorted
by name):
Alice => 95
Bob => 90
Charlie => 92

Frequency of characters in
'hello':
e => 1
h => 1
```

```
    marks["Alice"] = 95;
    marks["Bob"] = 88;
    marks["Charlie"] = 92;
    // 3. Accessing value using key
    cout << "Alice ka marks: " << marks["Alice"] << endl;
    // 4. Updating a value for an existing key
    marks["Bob"] = 90; // Bob's marks updated from 88 to 90
    // 5. Iterating through the map
    cout << "\nAll students ke marks (sorted by name):" << endl;
    for(auto it : marks) {
        cout << it.first << " => " << it.second << endl;
    }
    // 6. Frequency counting example
    string word = "hello";
    map<char, int> freq;
    for(char c : word) {
        freq[c]++; // count each character
    }
    cout << "\nFrequency of characters in '" << word << "':" << endl;
    for(auto it : freq) {
        cout << it.first << " => " << it.second << endl;
    }
    return 0;
}
```

Explanation

- Map → Key-value pairs store karta hai (jaise dictionary).
- Keys unique hoti hain → Same key pe value update ho jati hai.
- Keys automatically sorted hoti hain → Iteration me ascending order me milti hain.
- Fast lookup → Direct key se value access possible.
- Use cases → Student marks, word frequency, counting problems, dictionary type data.

size() function ka use

- C++ map me size() function hota hai jo map me total elements (key-value pairs) batata hai.

```
#include <iostream>
#include <map>
using namespace std;
int main() {
    map<string, int> marks;
    marks["Alice"] = 95;
    marks["Bob"] = 88;
    marks["Charlie"] = 92;
    cout << "Map ka size: " << marks.size() << endl;
    return 0;
}
```

Explanation

- #include <iostream> & #include <map> → Input/output aur map use karne ke liye.
- map<string, int> marks; → Student names (key) aur marks (value) store karne ke liye map create kiya.
- marks["Alice"] = 95;, marks["Bob"] = 88;, marks["Charlie"] = 92; → Map me key-value pairs insert kiye.
- marks.size() → Map me total elements (unique keys) ka count return karta hai.
- cout << "Map ka size: " << marks.size(); → Output: Map ka size: 3
- Note: Agar existing key update ho jaye, size increase nahi hota.

Iterator kya hota hai?

Iterator ek object hota hai jo container ke elements ko point karta hai aur unke through move karne me help karta hai.
Simple words me:
👉 Iterator = pointer jaisa object
Jaise pointer array ke element ko point karta hai, waise hi iterator:

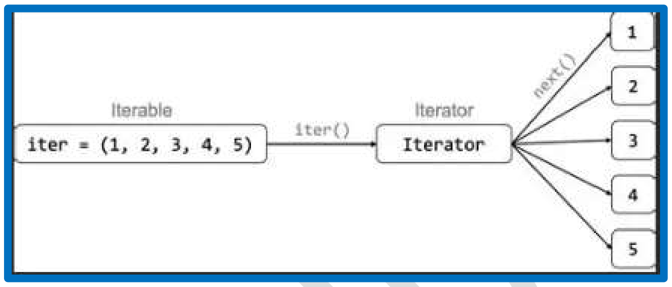
- vector
- list
- set
- map

ke elements ko point karta hai.

2. Iterator ki zarurat kyun hoti hai?
Different containers data ko alag-alag tareeke se store karte hain:

- vector → continuous memory
- list → linked list
- set → tree structure

Iterator ek common way deta hai elements ko access karne ka, bina ye soche ki data internally kaise stored hai.



```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  // Create a vector called cars that will store strings
  vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};
  // Create an iterator named it
  vector<string>::iterator it;
  // Use the iterator to loop through the vector
  for (it = cars.begin(); it != cars.end(); ++it) {
    cout << *it << "\n";
  }
  return 0;
}
```

Explanation

- #include <vector> → vector container use karne ke liye
- vector<string> cars → strings store karne wala vector banaya
- {"Volvo","BMW","Ford","Mazda"} → vector ke initial elements
- vector<string>::iterator it; → iterator declare kiya
- cars.begin() → first element ko point karta hai
- cars.end() → last element ke baad ka position
- it != cars.end() → jab tak last se aage na pahunch jao
- ++it → next element pe move karta hai
- *it → current element ki value access karta hai
- cout << *it → har car name print hota hai
- Iterator vector ke har element ko one by one access karta hai.

Iterate in Reverse

Iterate in Reverse ka matlab hota hai container ke elements ko last se first tak access karna.

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    // Create a vector of car names
    vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};
    // Iterate in reverse using reverse_iterator
    for (auto it = cars.rbegin(); it != cars.rend(); ++it) {
        cout << *it << endl;
    }
    return 0;
}
```

Short Explanation:

- rbegin() → last element se start
- rend() → first element ke pehle rukta hai
- auto → reverse iterator automatically handle
- *it → element print karta hai

Iterate Through Other Data Structures

C++ me different data structures hote hain aur un sabko iterators se traverse kiya ja sakta hai.

Iterate through List

List bidirectional hoti hai (aage–peeche move)

```
#include <iostream>
#include <list>
using namespace std;
int main() {
    list<int> nums = {10, 20, 30};
    for (auto it = nums.begin(); it != nums.end(); ++it) {
        cout << *it << " ";
    }
}
```

Explanation

- #include <list> → list container use karne ke liye
- list<int> nums → integer values store karne wali list banayi
- {10, 20, 30} → list ke elements
- auto it → iterator automatically detect hota hai
- nums.begin() → first element ko point karta hai
- nums.end() → last element ke baad ka position
- it != nums.end() → jab tak list khatam na ho
- ++it → next element pe move karta hai
- *it → current element ki value access karta hai
- cout << *it → element print karta hai

Iterate through Set

Set sorted hota hai, duplicates allow nahi

```
#include <iostream>
#include <set>
using namespace std;
int main() {
    set<int> s = {40, 10, 30};
    for (auto it = s.begin(); it != s.end(); ++it) {
        cout << *it << " ";
    }
}
```

Explanation

- #include <set> → set container use karne ke liye
- set<int> s → integers store karne wala set banaya
- {40, 10, 30} → elements insert kiye (unordered input)
- auto it → iterator ka type automatically detect hota hai
- s.begin() → smallest element ko point karta hai
- s.end() → last element ke baad ka position
- it != s.end() → jab tak set khatam na ho
- ++it → next element pe move karta hai
- *it → current element ki value access karta hai
- cout << *it → element print karta hai

Iterate through Map
Map me key + value pair hota hai

```
#include <iostream>
#include <map>
using namespace std;
int main() {
    map<int, string> m = {
        {1, "Apple"},
        {2, "Banana"},
        {3, "Mango"}
    };
    for (auto it = m.begin(); it != m.end(); ++it) {
        cout << it->first << " : " << it->second << endl;
    }
}
```

Explanation

- #include <map> → map container use karne ke liye
- map<int, string> m → key-value pair store karne wala map
- {1, "Apple"}, {2, "Banana"}, {3, "Mango"} → map ke elements
- auto it → iterator type automatically detect ho jata hai
- m.begin() → smallest key wale element ko point karta hai
- m.end() → last element ke baad ka position
- it != m.end() → jab tak map traverse na ho jaye
- ++it → next element pe move karta hai
- it->first → current element ki key
- it->second → current element ki value
- cout << it->first << " : " << it->second → key-value pair print karta hai

Iterate through Deque

Deque vector jaisa hota hai (front + back)

```
#include <iostream>
#include <deque>
using namespace std;
int main() {
    deque<int> d = {1, 2, 3};
    for (auto it = d.begin(); it != d.end(); ++it) {
        cout << *it << " ";
    }
}
```

Explanation

- #include <deque> → deque container use karne ke liye
- deque<int> d → integers store karne wala deque banaya
- {1, 2, 3} → deque ke initial elements
- auto it → iterator ka type automatically detect ho jata hai
- d.begin() → first element ko point karta hai
- d.end() → last element ke baad ka position
- it != d.end() → jab tak deque traverse na ho jaye
- ++it → next element pe move karta hai
- *it → current element ki value access karta hai
- cout << *it → element print karta hai

C++ Algorithm kya hai?

- Algorithm ek predefined function hai C++ STL me (<algorithm> header)

- Containers (vector, list, set, map) ke saath iterators ke through kaam karta hai
- Common tasks perform karne ke liye use hota hai without writing manual loops

Common Examples:
sort(), reverse(), find(), count(), accumulate(), max_element()

Algorithm use karne ke fayde
- Fast aur optimized
- Kam code → easy to read
- Har container ke saath kaam karta hai jo iterator support karta hai
- Manual loop bugs avoid hote hain

Algorithm ke Categories
a) Non-modifying algorithms
- Container ko **change nahi karte**
- Examples: count(), find(), all_of(), any_of(), none_of()

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
   vector<int> v = {10, 20, 30, 20};
   int c = count(v.begin(), v.end(), 20);
   cout << "20 occurs " << c << " times";
}
```

Output: 20 occurs 2 times

Explanation
- #include <iostream>, <vector>, <algorithm> → Input/output, vector, aur algorithms ke liye libraries.
- using namespace std; → std:: likhne ki zarurat nahi.
- vector<int> v = {10, 20, 30, 20}; → Vector banaya aur elements daale.
- count(v.begin(), v.end(), 20); → Vector me 20 kitni baar hai, count karta hai.
- cout << "20 occurs " << c << " times"; → Result print karta hai.

b) Modifying algorithms
- Container ke elements ko change karte hain
- Examples: sort(), reverse(), rotate(), replace()

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
int main() {
```

Output: 1 2 3 4

```
    vector<int> v = {3, 1, 4, 2};
    sort(v.begin(), v.end());   // Sort in ascending
    for (int x : v) cout << x << " ";
}
```

Explanation:

- #include <algorithm> → sort() function ke liye.
- #include <vector> → Vector use karne ke liye.
- vector<int> v = {3, 1, 4, 2}; → Vector v banaya aur numbers store kiye.
- sort(v.begin(), v.end());
  - Vector ke elements ko ascending order me sort karta hai.
  - v.begin() → vector ka start
  - v.end() → vector ka end (last ke baad)
  - Result: {1, 2, 3, 4}
- for (int x : v) cout << x << " ";
  - Ye vector ke har element ko print karta hai space ke saath.

c) Searching & Counting

- find() → element ka iterator return karta hai
- binary_search() → element exist karta hai ya nahi (container sorted hona chahiye)
- count_if() → condition satisfy karne wale elements count kare

```
#include <iostream>
#include <vector>
#include <algorithm> // for find, count, count_if
using namespace std;
int main() {
    vector<int> v = {10, 20, 30, 20, 40, 50};
    //  count() → count specific element
    int c = count(v.begin(), v.end(), 20);
    cout << "20 occurs " << c << " times" << endl;

    //  find() → find first occurrence of element
    auto it = find(v.begin(), v.end(), 30);
    if(it != v.end()) {
        cout << "Found 30 at position: " << distance(v.begin(), it) << endl;
    } else {
        cout << "30 not found" << endl;
    }
    // count_if() → count elements satisfying a condition
```

```
20 occurs 2 times
Found 30 at position: 2
Number of even elements: 4
```

```
    int even_count = count_if(v.begin(), v.end(), [](int x){ return x % 2 == 0; });
    cout << "Number of even elements: " << even_count << endl;
    return 0;
}
```

Explanation

- vector<int> v = {10, 20, 30, 20, 40, 50};
- Vector banaya aur elements daale.
- count(v.begin(), v.end(), 20);
- Ye check karta hai ki vector me 20 kitni baar aaya hai.
- c = 2
- Output: 20 occurs 2 times
- find(v.begin(), v.end(), 30);
- Ye vector me first occurrence of 30 find karta hai.
- Agar 30 mile, to iterator it us position ko point karega.
- distance(v.begin(), it) → ye calculate karta hai ki 30 vector me kaunsi position pe hai (0-based index).
- Output: Found 30 at position: 2
- count_if(v.begin(), v.end(), [](int x){ return x % 2 == 0; });
- count_if vector ke elements me check karta hai condition ke according.
- Yaha lambda function [](int x){ return x % 2 == 0; } → har element check karega ki wo even hai ya nahi.
- Even numbers: 10, 20, 30, 20, 40, 50 → 6 elements.

d) Set Operations (sorted containers ke liye)

- set_union(), set_intersection(), set_difference()
- Sorted ranges pe kaam karte hain

```
#include <iostream>
#include <vector>
#include <algorithm> // for set_union, set_intersection, set_difference
using namespace std;
int main() {
    // Two sorted vectors
    vector<int> A = {1, 3, 5, 7};
    vector<int> B = {3, 4, 5, 6};
    // set_union → all unique elements from both
    vector<int> uni(A.size() + B.size());
    auto it = set_union(A.begin(), A.end(), B.begin(), B.end(), uni.begin());
    uni.resize(it - uni.begin());
    cout << "Union: ";
    for (int x : uni) cout << x << " ";
    cout << endl;
    // set_intersection → elements present in both
    vector<int> inter(min(A.size(), B.size()));
    it = set_intersection(A.begin(), A.end(), B.begin(), B.end(), inter.begin());
    inter.resize(it - inter.begin());
    cout << "Intersection: ";
    for (int x : inter) cout << x << " ";
    cout << endl;

    // set_difference → elements in A not in B
    vector<int> diff(A.size());
    it = set_difference(A.begin(), A.end(), B.begin(), B.end(), diff.begin());
    diff.resize(it - diff.begin());
    cout << "Difference (A-B): ";
    for (int x : diff) cout << x << " ";
    cout << endl;
    return 0;
}
```

Explanation

1. Vectors A and B → A = {1,3,5,7}, B = {3,4,5,6} (sorted required).
2. set_union → Sab unique elements dono vectors ka combine karta hai.
    - Result: 1 3 4 5 6 7
3. set_intersection → Sirf common elements dono vectors me.
    - Result: 3 5
4. set_difference(A-B) → A ke elements jo B me nahi hain.
    - Result: 1 7
5. resize(it - vector.begin()) → Actual size adjust karne ke liye after set operation.

e) Min, Max, Swap, Accumulate

```
#include <iostream>
#include <algorithm>
#include <numeric>
#include <vector>
using namespace std;
int main() {
    vector<int> v = {10, 20, 30};
    cout << "Max: " << *max_element(v.begin(), v.end()) << endl;
    cout << "Sum: " << accumulate(v.begin(), v.end(), 0);
```

Output:

```
Max: 30
Sum: 60
```

```
}
```

Explanation:

1. vector<int> v = {10, 20, 30};
   - Vector banaya aur elements store kiye.
2. max_element(v.begin(), v.end())
   - Ye vector ka maximum element find karta hai.
   - Return value iterator hoti hai, isliye * lagana padta hai.
   - Output: Max: 30
3. accumulate(v.begin(), v.end(), 0)
   - Ye vector ke elements ka sum calculate karta hai.
   - 0 → initial value (sum start from 0)

f) Sorting & Partitioning

- sort() → ascending/descending
- stable_sort() → relative order preserve karta hai
- partial_sort() → first k elements sort karna
- partition() → condition ke basis pe separate k

```
#include <iostream>
#include <vector>
#include <algorithm> // for sort
using namespace std;
int main() {
  // Vector with unsorted elements
  vector<int> v = {4, 2, 5, 1, 3};
  //  Ascending sort
  sort(v.begin(), v.end()); // default sort is ascending
  cout << "Ascending: ";
  for (int x : v) {
    cout << x << " ";
  }
  cout << endl;
  //  Descending sort using lambda function
  sort(v.begin(), v.end(), [](int a, int b) {
    return a > b; // return true if a should come before b
  });
  cout << "Descending: ";
  for (int x : v) {
    cout << x << " ";
  }
  cout << endl;
```

```
Output
Ascending: 1 2 3 4 5
Descending: 5 4 3 2 1
```

```
    return 0;
}
```

Explanation:

1. vector<int> v = {4, 2, 5, 1, 3};
   - Vector banaya aur unsorted elements store kiye.
2. sort(v.begin(), v.end());
   - Ye default ascending order me sort karta hai.
   - Output: Ascending: 1 2 3 4 5
3. sort(v.begin(), v.end(), [](int a, int b){ return a > b; });
   - Lambda function ke through descending order sort kiya.
   - Logic: agar a > b true hai, to a pehle aayega.
   - Output: Descending: 5 4 3 2 1
4. for (int x : v)
   - Vector ke elements ko print karne ke liye range-based for loop use kiya.