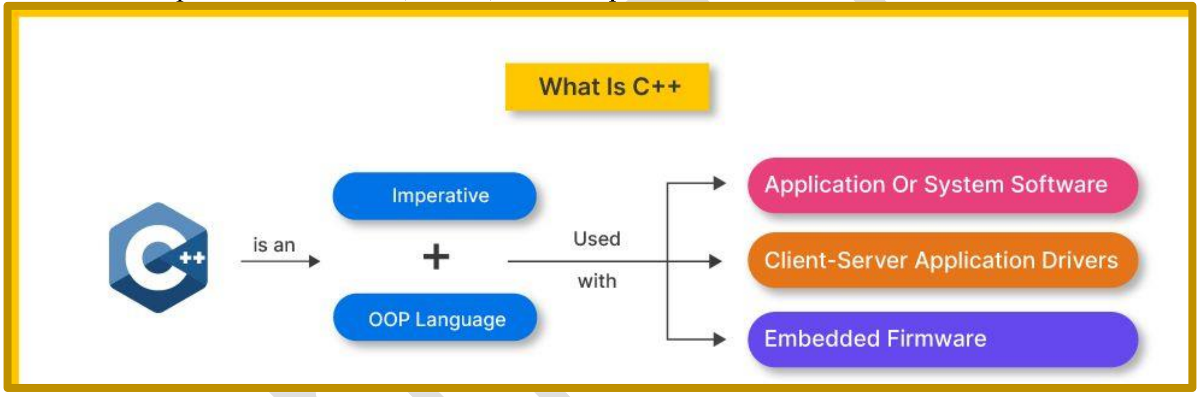
Basic C++ Notes

C++ ek programming language hai jo fast, powerful aur flexible programming ke liye use hoti hai. Yeh C language ka advanced version hai, isliye iska naam *C++* rakha gaya (C mein "++" ka matlab hota hai increment — yaani C ka upgraded version).
C++ kya karta hai?
C++ ka use karke hum:

- Games bana sakte hain
- High-performance software develop kar sakte hain
- Operating systems aur drivers likh sakte hain
- Large-scale applications banate hain, jaise banking systems ya 3D applications

Features (simple Hinglish explanation):

- Object-Oriented Programming (OOP): Isme classes aur objects use hote hain, jisse code organized aur reusable hota hai.
- Fast performance: C++ ka code bahut optimized hota hai, isliye ye bahut fast run hota hai.
- Low-level control: Aap memory ko directly control kar sakte ho (pointers ki madad se).
- Cross-platform: Windows, Linux, Mac sab par chal sakta hai.



Applications of C++

1. Game Development

- C++ ka performance bahut fast hota hai, isliye high-graphics games aur game engines (jaise Unreal Engine) C++ mein bante hain.
- Real-time rendering, physics engines, AI—sab C++ handle karta hai.

2. Operating Systems

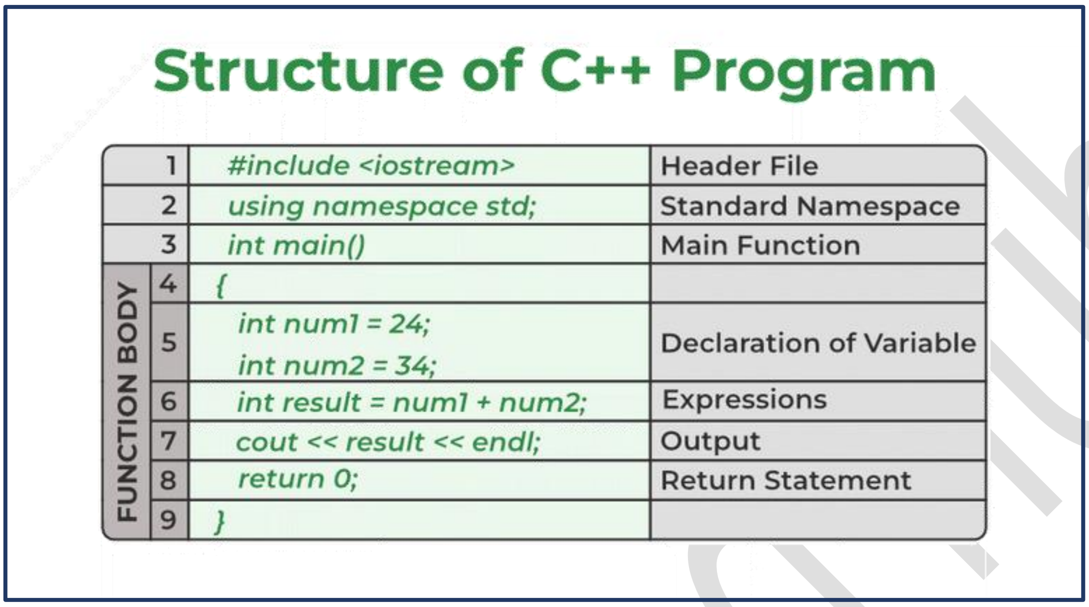
- Windows, macOS, Linux ke parts C++/C mein likhe gaye hain.
- Low-level hardware control aur high performance ki wajah se useful hai.

3. Software Development / NApplications

- Browsers (Firefox, Chrome ka engine), media players, database tools, Photoshop, MATLAB, etc. ke core parts C++ mein hote hain.
- Fast and reliable applications ke liye C++ choose kiya jata hai.

4. Embedded Systems

- Smartwatches, car systems, medical devices, home appliances—C++ inke firmware ya low-level systems mein use hota hai.



```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5   int num1 = 24;
    int num2 = 34;
6   int result = num1 + num2;
7   cout << result << endl;
8   return 0;
9 }
```

Difference between Cin & Cout

Feature	cin	cout
Meaning	Character Input	Character Output
Use	User se input lene ke liye	Screen par output dikhane ke liye
Direction of Data Flow	Keyboard → Program	Program → Screen
Operator Used	>> (extraction operator)	<< (insertion operator)
Header File	<iostream>	<iostream>
Belongs To	istream class	ostream class
Example	cin >> age;	cout << age;
Purpose	Values read karna	Values print karna

Difference Between C and C++

Feature	C	C++
Language Type	Procedural language (sirf functions par based)	Object-Oriented + Procedural (OOP + functions)
Programming Style	Top-down (pehle overall plan, phir parts)	Bottom-up (pehle chote parts, phir whole system)

Feature	C	C++
OOP Support	OOP features nahi hote	Classes, objects, inheritance, polymorphism sab hote hain
Data Security	Data hiding nahi hoti	Encapsulation ki wajah se data secure rehta hai
Input/Output	printf() / scanf()	cout / cin
Memory Management	malloc(), calloc(), free()	new / delete
Overloading	Function ya operator overloading nahi hoti	Function aur operator overloading hoti hai
Namespace	Namespace ka concept nahi	namespace available
Exception Handling	Error handling limited	Proper try-catch exception handling
Approach	Function-oriented	Object-

C++ Comments

C++ mein comments woh lines hoti hain jo compiler ignore kar deta hai.
Yeh sirf hum humans ke liye hoti hain — code ko samajhne, explain karne, aur maintain karne ke liye.
Comments likhne se:

- Code readable hota hai
- Future me changes karna easy hota hai
- Teamwork me dusre log code samajh pate hain
- Complex logic explain kiya ja sakta hai

Types of Comments in C++

- C++ me 2 types ke comments hote hain:

Single-line Comment (//)

- Ye comment **sirf ek line** ke liye hota hai.

```
#include <iostream>
using namespace std;
int main() {
   int a = 10;  // yaha variable a ko 10 assign kiya
   cout << a;   // a ka value print ho raha hai
}
```

Explanation

- // ke baad jo bhi likha hoga wo compiler ignore karega.
- Yaha humne comment diya ki variable a me 10 store kiya hai.

- Is tarah ka comment chhote-chhote explanations ke liye useful hota hai.

Multi-line Comment (/ … /)

- Ye comment multiple lines ke liye hota hai.

Syntax:

```
/*
  ye multi-line
  comment ka example hai
*/
```

```
#include <iostream>
using namespace std;
/*
   Ye program do numbers ka sum calculate karega.
   Isme hum user se input lenge.
*/
int main() {
    int x, y;
    cout << "Enter two numbers: ";
    cin >> x >> y;
    cout << "Sum = " << (x + y);
}
```

```
#include <iostream>
using namespace std;
/*
   Ye program do numbers ka sum calculate karega.
   Isme hum user se input lenge.
*/
int main() {
    int x, y;
    cout << "Enter two numbers: ";
    cin >> x >> y;
    cout << "Sum = " << (x + y);
}
```

C++ Variables

C++ me variables basically containers hote hain — matlab aise dabbe jisme hum data store kar sakte hain.
Example:

- Jaise aap paise rakhne ke liye wallet use karte ho
- Waise hi computer data store karne ke liye variables use karta hai

Variables ka naam hota hai, type hoti hai, aur value hoti hai.

C++ Variables Types

int (Integer Type)

- Is type me whole numbers store hote hain, jaise counting values.
- Decimal numbers isme allowed nahi hote.
- Positive aur negative dono values rakhta hai.

float (Floating-Point Type)

- Ye **decimal values** store karta hai.
- Accuracy kam hoti hai, isliye light calculations me use hota hai.
- Memory double se kam use karta hai.

double (Double Precision Floating-Point)

- Decimal numbers ke liye float se zyada accurate type hai.
- Scientific, financial aur precise calculations me useful hai.
- Memory float se zyada use karta hai.

char (Character Type)

- Ye ek single character store karta hai, jaise koi letter ya symbol.
- C++ is character ko internally number (ASCII) ke form me store karta hai.

string (Text Type)

- Ye text ya words store karta hai.
- Multiple characters store kar sakta hai.
- C++ me string ek class type hoti hai jo text ko handle karne ke liye powerful features deti hai.

bool (Boolean Type)

- Isme sirf true ya false store hota hai.
- Mostly conditions, decision making (if/else) me use hota hai.

long / long long (Extended Integer Types)

- Ye int jaisa hi hota hai, par bade range wale numbers store karta hai.
- long long bohot bade integers rakhne ke kaam aata hai (competitive programming me common).

unsigned Types

- Unsigned types me negative numbers store nahi hote.
- Sirf zero aur positive values store karta hai.
- Range normally double ho jaati hai kyunki negative part hata diya jata hai.
- Available in: unsigned int, unsigned long, unsigned char, etc.

wchar_t / char16_t / char32_t (Wide Character Types)

- Ye bade character sets store karne ke liye hote hain, jaise Unicode characters.
- Multi-language support me useful hote hain (Hindi, Chinese, Emoji etc.).

auto (Automatic Type Deduction)

- C++11 me introduce hua.
- Compiler automatically decide karta hai variable ka type based on value.
- Code ko clean aur easy banata hai.

Real-Life Example: Fruits kharidna

Imagine karo:

- Tum shop pe ja rahe ho aur apples aur oranges kharid rahe ho.
- Har fruit ka price hai.
- Tumhe total cost calculate karni hai.

Isme hum C++ variables use karenge.

```
#include <iostream>
using namespace std;
int main() {
    int apples = 5;
    int oranges = 3;
    double pricePerApple = 10.5;
    double pricePerOrange = 8.0;
    double totalCost = (apples * pricePerApple) + (oranges * pricePerOrange);
    cout << "Total cost of fruits = Rs " << totalCost << endl;
    return 0;
}
```

Identifiers

C++ me identifier basically kisi variable, function, class, array, ya kisi object ka naam hota hai.
Ye name compiler ko batata hai ki hum kis cheez ko refer kar rahe hain.

Scenario: Ek student ke marks calculate karna aur uska percentage nikalna.

```
#include <iostream>
using namespace std;
int main() {
   // Student details (identifiers)
   string studentName = "Rahul";   // Student ka naam
   int mathsMarks = 85;            // Maths ke marks
   int scienceMarks = 90;          // Science ke marks
   int englishMarks = 75;          // English ke marks
   // Total aur percentage calculate karna
   int totalMarks = mathsMarks + scienceMarks + englishMarks;
   float percentage = (totalMarks / 3.0);  // 3 subjects
   // Result display
   cout << "Student Name: " << studentName << endl;
   cout << "Total Marks: " << totalMarks << endl;
   cout << "Percentage: " << percentage << "%" << endl;
   return 0;
}
```

Explanation of Identifiers

Identifier	**Represents in real life**
studentName	Student ka naam
mathsMarks	Maths subject ke marks
scienceMarks	Science subject ke marks
englishMarks	English subject ke marks
totalMarks	Teen subjects ka total marks
percentage	Student ka percentage

Constants

C++ me constants wo values hoti hain jo program run hone ke baad change nahi hoti.
Matlab once assign kiya, phir usko modify nahi kar sakte.

Real-life Example: Grocery Billing with Discount & Tax
Sochiye aap ek grocery shop ka billing program bana rahe hain. Kuch cheezein aisi hongi jo fixed hongi — jaise discount rate ya tax rate — jise har bill me same use karna hai. Aise fixed values ko constants banaake use karte hain.

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

```
int main() {
    const float SALES_TAX_RATE = 0.18f;      // 18% sales tax — constant hai
    const float DISCOUNT_RATE = 0.05f;       // 5% discount — constant hai
    float priceOfItem;
    int quantity;
   cout << "Enter price of one item: ";
    cin >> priceOfItem;
    cout << "Enter quantity: ";
    cin >> quantity;
    float netPrice = priceOfItem * quantity;
    float discountAmount = netPrice * DISCOUNT_RATE;
    float priceAfterDiscount = netPrice - discountAmount;
    float taxAmount = priceAfterDiscount * SALES_TAX_RATE;
    float finalPrice = priceAfterDiscount + taxAmount;
    cout << "Net price (without discount & tax): " << netPrice << endl;
    cout << "Discount amount: " << discountAmount << endl;
    cout << "Price after discount: " << priceAfterDiscount << endl;
    cout << "Tax amount: " << taxAmount << endl;
    cout << "Final price to pay: " << finalPrice << endl;
    return 0;
}
```

Explanation

- SALES_TAX_RATE aur DISCOUNT_RATE ko constants declare kiya gaya hai, kyunki ye values fixed rehengi (shop me har item pe same tax aur discount rule hai).
- User se priceOfItem aur quantity liye ja rahe hain — ye variables hain, kyunki har item, har bill ke liye alag ho sakta hai.
- Fir hum net price, discount, tax, aur final price calculate kar rahe hain using constants + variables.

C++ User Input

- cin C++ ka predefined object hai jo input ke liye use hota hai.
- Technically, cin is an object of the class istream — i.e. istream ka ek instance hai.
- Ye object standard input stream (usually keyboard) se data read karne ke liye pre-set hota hai.

User Info Greeting

- Program se user se: naam aur age poochho.
- Fir output me print karo: “Hello <name>! You are <age> years old.”

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string name;
    int age;
    cout << "Enter your name: ";
    getline(cin, name);
   cout << "Enter your age: ";
    cin >> age;
    cout << "Hello " << name << "! You are " << age << " years old." << endl;
    return 0;
}
```

Explanation

Variables Declaration

- string name; → User ka naam store karega.
- int age; → User ki age store karega.

User Input

- getline(cin, name); → User se full name input leta hai (spaces ke saath).
- cin >> age; → User se integer input leta hai (age).

Output

- cout << "Hello " << name << "! You are " << age << " years old." << endl;
  → User ka greeting message display karta hai with name and age

output

```
Enter your name: Rahul Sharma
Enter your age: 21
Hello Rahul Sharma! You are 21 years old.
```

auto keyword kya hai?

- auto C++11 se introduce hua ek type specifier hai.
- Iska kaam hai compiler ko batana: "Mujhe variable ka type automatically dedho, main manually declare nahi karna chahta."
- Matlab, aapko variable ka type explicitly likhne ki zarurat nahi. Compiler input value ke type se type infer kar leta hai.

Real-Life Scenario: Shopping Cart Total Calculation

Scenario:

- Aap ek online shopping cart bana rahe ho.
- Har item ka price aur quantity alag hai.
- Total price calculate karna hai.
- Discount aur tax apply karna hai.
- Yahan auto keyword ka use karke type ko automatically infer karenge.

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

```
int main() {
    auto price1 = 500.0;
    auto price2 = 250.0;
    auto price3 = 130.0;
    auto quantity1 = 2;
    auto quantity2 = 1;
    auto quantity3 = 3;
    auto subtotal = price1*quantity1 + price2*quantity2 + price3*quantity3;
    auto discount = 0.10 * subtotal;   // 10% discount
    auto tax = 0.05 * (subtotal - discount); // 5% tax
    auto total = subtotal - discount + tax;
    cout << "Subtotal: Rs. " << subtotal << endl;
    cout << "Discount: Rs. " << discount << endl;
    cout << "Tax: Rs. " << tax << endl;
    cout << "Total Amount to Pay: Rs. " << total << endl;
    return 0;
}
```

Explanation

- auto keyword automatically variable ka type detect kar leta hai.
- price1, price2, price3 → double
- quantity1, quantity2, quantity3 → int
- subtotal, discount, tax, total → double
- Subtotal calculate karna = sum of (price × quantity)
- Discount = 10% of subtotal
- Tax = 5% on (subtotal - discount)
- Total = subtotal - discount + tax

Operator

Operator ek special symbol hai jo variables ya values par operations perform karta hai.

- Example: +, -, *, /, % etc.
- C++ me operators ko category wise divide kiya gaya hai.

C++ divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic Operators

- Arithmetic Operators mathematical calculations ke liye use hote hain.
- Ye numbers (integers, float, double) par operations karte hain.
- Common operators:
  +, -, *, /, %

List of Arithmetic Operators

Operator	**Meaning / Use**	**Example**
+	Addition (jodna)	5 + 3 = 8
-	Subtraction (ghatana)	5 - 3 = 2
*	Multiplication (guna)	5 * 3 = 15
/	Division (bhaag)	15 / 3 = 5
%	Modulus (remainder)	5 % 3 = 2

```
#include <iostream>
using namespace std;
int main() {
   int a = 10, b = 3;
   cout << "Multiplication: " << a * b << endl; // 30
   cout << "Division: " << a / b << endl;      // 3 (integer division)
   return 0;
}
```

Explanation

- + → do numbers ko jodta hai
- - → do numbers me se ek ko ghata deta hai
- cout → screen par print karne ke liye
- endl → line break ke liye

Assignment Operators

Assignment operators variable me value assign karne ke liye use hote hain.

- Sabse basic operator hai =.
- Iske alawa shortcut operators hote hain jo arithmetic + assignment ko combine karte hain.

List of Assignment Operators

Operator	Meaning / Use	Example
=	Simple assignment	a = 5; → a me 5 store hoga
+=	Add and assign	a += 3; → a = a + 3
-=	Subtract and assign	a -= 2; → a = a - 2
*=	Multiply and assign	a *= 4; → a = a * 4
/=	Divide and assign	a /= 2; → a = a / 2
%=	Modulus and assign	a %= 3; → a = a % 3

```
#include <iostream>
using namespace std;
int main() {
    int balance = 5000;  // Initial balance
    balance += 2000; // Deposit Rs. 2000
    balance -= 1500; // Withdraw Rs. 1500
    balance *= 1;    // No change, just example
    balance /= 1;    // No change
    balance %= 1000; // Modulus, baki amount after dividing by 1000
    cout << "Current Balance: Rs. " << balance << endl;
    return 0;
}
```

Explanation

- int balance = 5000; → Initial balance set kiya 5000
- balance += 2000; → Deposit Rs. 2000, balance = 7000
- balance -= 1500; → Withdraw Rs. 1500, balance = 5500
- balance *= 1; → Multiply by 1, balance = 5500 (no change)
- balance /= 1; → Divide by 1, balance = 5500 (no change)
- balance %= 1000; → Modulus 1000, balance = 550
- cout << balance; → Print current balance, output = 550

Comparison Operators kya hain?

- Comparison Operators do values ko compare karte hain.
- Ye true (1) ya false (0) return karte hain.
- Mostly if-else aur loops me use hote hain.

List of Comparison Operators

Operator	Meaning	Example
==	Equal to (barabar hai)	a == b
!=	Not equal to (barabar nahi)	a != b
>	Greater than (bada hai)	a > b
<	Less than (chhota hai)	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

```
#include <iostream>
using namespace std;
int main() {
   int a = 10, b = 5;
   cout << (a == b) << endl;  // 0 (false)
   cout << (a != b) << endl;  // 1 (true)
   cout << (a > b) << endl;   // 1 (true)
   cout << (a < b) << endl;   // 0 (false)
   cout << (a >= b) << endl;  // 1 (true)
   cout << (a <= b) << endl;  // 0 (false)
   return 0;
}
```

Explanation

- int a = 10, b = 5; → Do variables a aur b declare kiye aur values assign ki.
- a == b → Check karta hai a aur b barabar hai ya nahi → Output: 0 (false)
- a != b → Check karta hai a aur b barabar nahi hai → Output: 1 (true)
- a > b → Check karta hai a bada hai b se → Output: 1 (true)
- a < b → Check karta hai a chhota hai b se → Output: 0 (false)
- a >= b → Check karta hai a bada ya barabar hai b ke → Output: 1 (true)
- a <= b → Check karta hai a chhota ya barabar hai b ke → Output: 0 (false)

Logical Operators kya hain?

- Logical Operators do ya zyada conditions ko combine karte hain.
- Result true (1) ya false (0) hota hai.
- Mostly if-else statements aur loops me use hote hain.

List of Logical Operators

Operator	Meaning (Hinglish)	Example
&&	AND (aur) → dono conditions true honi chahiye	(a > 5 && b < 10)
`		`
!	NOT (ulta) → condition ka opposite	!(a > 5)

```
#include <iostream>
using namespace std;
int main() {
    int a = 10, b = 5;
    cout << (a > 5 && b < 10) << endl;  // 1 (true)
    cout << (a > 5 && b > 10) << endl;  // 0 (false)
    cout << (a > 5 || b > 10) << endl;  // 1 (true)
    cout << !(a > 5) << endl;           // 0 (false)
    return 0;
}
```

Explanation

- int a = 10, b = 5; → Do variables a aur b declare kiye aur values assign ki.
- a > 5 && b < 10 → AND operator: dono conditions true hain → Output: 1 (true)
- a > 5 && b > 10 → AND operator: ek condition false hai → Output: 0 (false)
- a > 5 || b > 10 → OR operator: kam se kam ek condition true hai → Output: 1 (true)
- !(a > 5) → NOT operator: condition ka opposite → Output: 0 (false)

Bitwise Operators kya hain?

- Bitwise Operators numbers ke binary form (0s aur 1s) par operations karte hain.
- Ye integer values par use hote hain.
- Mostly low-level programming, flags, aur masks me use hote hain.

List of Bitwise Operators

Operator	Meaning (Hinglish)	Example
&	AND → dono bits 1 ho tab 1	5 & 3
`	`	OR → kam se kam ek bit 1 ho
^	XOR → sirf alag bits 1 ho	5 ^ 3
~	NOT → bits ka opposite	~5
<<	Left shift → bits ko left shift	5 << 1
>>	Right shift → bits ko right shift	5 >> 1

Scenario: Smart Home Lights Control

- **Assumption:** 4 lights hain aur har light ON/OFF flag me store hai.
- Binary representation use karte hain:

  - 0001 → Light 1 ON
  - 0010 → Light 2 ON
  - 0100 → Light 3 ON
  - 1000 → Light 4 ON

```
#include <iostream>
using namespace std;
int main() {
   int lights = 0;
   lights = lights | 1;
   lights = lights | 4;
   lights = lights & (~1);
   lights = lights ^ 2;
   cout << "Lights status (binary): " << lights << endl;
   return 0;
}
```

Explanation

- int lights = 0;
- Sab lights initially OFF hain. (Binary: 0000)
- lights = lights | 1;
- Light 1 ON kar di. (0000 → 0001)
- lights = lights | 4;
- Light 3 ON kar di. (0001 → 0101)
- lights = lights & (~1);
- Light 1 OFF kar di. (0101 → 0100)
- lights = lights ^ 2;
- Light 2 toggle kiya (agar OFF tha → ON, agar ON tha → OFF). (0100 → 0110)
- cout << lights;
- Final lights ka status print hua. (Binary 0110 → Light 2 & 3 ON, Light 1 & 4 OFF)

C++ String ka Meaning

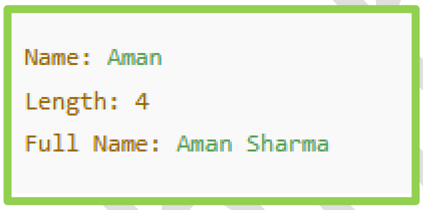
C++ mein string ek data type hai jo text ko store karne ke liye use hota hai. Matlab, agar aapko letters, words, ya sentences ko store karna hai, to aap string ka use karte ho.

- Example: "Hello", "C++ is fun", "123ABC" ye sab strings hain.
- String basically characters ka sequence hoti hai.

C++ String (string class)

- Ye modern C++ ka feature hai.
- Header file <string> use hoti hai.
- Ye dynamic hoti hai, matlab aap length ke baare mein tension nahi lete, string automatically adjust ho jati hai.
- Easy to use aur built-in functions milte hain jaise: .length(), .substr(), .find(), .append(), etc.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string name = "Aman";
    cout << "Name: " << name << endl;
    cout << "Length: " << name.length() << endl;
    name = name + " Sharma"; // string concatenate
    cout << "Full Name: " << name << endl;
    return 0;
}
```



```
Name: Aman
Length: 4
Full Name: Aman Sharma
```

C-style String (char array)

- Ye old style C programming ka method hai.
- Sirf char array ka use hota hai.
- Fixed size hoti hai, matlab agar 20 characters allocate kiye hain aur aap 25 type kar do, to problem ho sakti hai.
- Functions ke liye <cstring> header use hota hai, jaise strlen(), strcat(), strcmp(), etc.

Example:

```
#include <iostream>
#include <cstring> // C-string functions
using namespace std;
int main() {
    char name[20] = "Aman";
    cout << "Name: " << name << endl;
    cout << "Length: " << strlen(name) << endl;
    strcat(name, " Sharma"); // concatenate
    cout << "Full Name: " << name << endl;
    return 0;
}
```

String Operations

Operation	Method / Syntax	Example	Output	Explanation
Length	.length() / .size()	"Hello".length()	5	String ke characters ki total sankhya
Character Access	str[index]	str[0]	'H'	Kisi bhi index ka character access karna (0-based index)
Concatenation	+ operator	"Aman" + " " + "Sharma"	Aman Sharma	Do ya zyada strings ko jodna
Compare	.compare(other)	"Apple".compare("Banana")	Negative value	Strings ko compare karna; 0 = same, <0 = a<b, >0 = a>b
Find Substring	.find(substring)	"Hello World".find("World")	6	String me kisi word/character ka index dhundhna
Substring	.substr(start, length)	"Hello World".substr(6,5)	World	String ka ek part nikalna
Replace	.replace(start, length, new_string)	"Hello World".replace(6,5,"C++")	Hello C++	String ke part ko replace karna
Uppercase	Loop + toupper()	"Hello" → loop + toupper	HELLO	Sab characters ko capital me convert karna
Lowercase	Loop + tolower()	"HELLO" → loop + tolower	hello	Sab characters ko small letters me convert karna
Clear String	.clear()	str.clear()	"" (empty)	String ko khali kar dena
Check Empty	.empty()	"".empty()	true	Check karna ki string empty hai ya nahi
Insert	.insert(pos, "text")	"Hello".insert(5," World")	Hello World	String me naya text insert karna
Erase	.erase(start, length)	"Hello World".erase(5,6)	Hello	String ke part ko delete karna
Reverse	reverse(str.begin(), str.end())	"Hello"	olleH	String ke characters ko ulta karna
String → int	stoi(str)	stoi("123")	123 (int)	String ko integer me convert karna
Int → String	to_string(num)	to_string(123)	"123"	Integer ko string me

Operation	Method / Syntax	Example	Output	Explanation
				convert karna

```
#include <iostream>
#include <string>
using namespace std;
int main() {
   string name = "Aman";
   cout << "Name: " << name << endl;
   cout << "Length: " << name.length() << endl;
   return 0;
}
```

C++ Access Strings
C++ strings ek sequence of characters hoti hain, aur har character ka index hota hai. Index 0 se start hota hai.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
   string name = "Aman";
   cout << "First character: " << name[0] << endl;  // 'A'
   cout << "Second character: " << name[1] << endl; // 'm'
   cout << "Last character: " << name[name.length()-1] << endl; // 'n'
   return 0;
}
```

Explanation

- Humne ek string "Aman" li
- Uske pehle, second, aur last character print kiye
- Indexing 0 se start hoti hai
- Last character ke liye .length()-1 use kiya

C++-Style Strings (Character Arrays)

C++ mein do tarah ke strings hote hain:

1. C-Style Strings (Purane type, character arrays)
2. C++ Strings (std::string, modern and easy)

Yahaan hum C-Style Strings ke baare mein padh rahe hain.

C++-Style Strings (Character Arrays)

C++ mein do tarah ke strings hote hain:

1. **C-Style Strings** (Purane type, character arrays)

2. **C++ Strings** (std::string, modern and easy)

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
   char source[] = "Hello World";
   char destination[20];
   strcpy(destination, source);
   cout << destination;
}
```

<cmath> kya hai

- <cmath> C++ standard library ka ek header hai, jo math — yani ganit — ke common operations ke liye functions provide karta hai. Agar aapko square root, powers, trigonometry, rounding, etc. karna hai, toh aapko <cmath> include karna padta hai.
- Ye basically C ke math.h ka C++ version hai. math.h bhi same functions deta hai, lekin C++ mein “std” namespace ke saath <cmath> use karna zyada appropriate mana jata hai.
- Functions jo <cmath> deta hai, woh float, double, long double jaise numeric types ke saath kaam karte hain.

Kuch Common & Useful Functions in <cmath>

- Yeh kuch most-useful functions hain jo aap programming mein frequently dekhoge:

Function	Meaning
sqrt(x)	x ka square root return karta hai.
pow(x, y)	x ko y-th power tak raise karta hai (x^y).
abs(x) / fabs(x)	Number ka absolute value deta hai (negative ko positive bana deta hai).
ceil(x)	Agar x fractional ho, toh usko upar round karta hai (nearest integer jo x se ≥ ho
floor(x)	Agar x fractional ho, toh usko neeche round karta hai (nearest integer jo x se ≤ ho).
round(x)	x ko nearest integer pe round karta hai.
sin(x), cos(x), tan(x)	Trigonometric calculations — sine, cosine, tangent (angle in radians).
log(x)	Natural logarithm (base e) return karta hai.
log10(x)	Logarithm base 10 return karta hai.
exp(x)	Exponential function: e^x return karta hai (e ~ 2.71828...).
fmod(x, y)	Floating-point division ka remainder return karta hai. (jab aap float numbers ko divide karke remainder chahte ho)

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    double radius;
    cout << "Enter radius of circle: ";
    cin >> radius;
    double area = M_PI * pow(radius, 2);       // pow(radius, 2) = radius²
    double circumference = 2 * M_PI * radius;  // 2πr
    cout << "Area = " << area << endl;
    cout << "Circumference = " << circumference << endl;
    return 0;
}
```

Explanation

- User se circle ka radius input lega.
- Fir area (πr^2) calculate karega using pow() function.
- Fir circumference ($2 \pi r$) calculate karega.
- Console par area aur circumference print kareg

Right Triangle — Hypotenuse Calculation

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    double a, b;
    cout << "Enter perpendicular sides a and b: ";
    cin >> a >> b;
    double hypotenuse = sqrt(a*a + b*b);  // Pythagoras theorem
    cout << "Hypotenuse = " << hypotenuse << endl;
    return 0;
}
```

- Agar aapke paas ek right-angled triangle hai, uske do side-lengths a aur b pata hain.
- Teesri side — yaani longest side, jo 90° ke saamne hoti hai — usko hum hypotenuse kehte hain.
- Formula: ‘$hypotenuse^2 = a^2 + b^2$‘→ Matlab pehle a*a + b*b karo.
- Phir uska square-root lo → usse milta hai hypotenuse ka actual length.
- Code me sqrt(a*a + b*b) waise hi karta hai — $a^2 + b^2$ ka root nikalta hai.

Conditional Statement in C++

Definition:
Conditional statement ek aisa statement hai jo program ko decision lene me help karta hai. Matlab,

program ek condition check karta hai aur true ya false ke basis pe alag-alag kaam karta hai.

if Statement in C++

Definition:
if statement ek conditional statement hai jo program ko allow karta hai ki wo kisi condition ke true hone par hi kuch kaam kare.
Matlab: Agar condition true hai → code chalega, false hai → code skip ho jayega.

```
if (condition) {
   // code to execute if condition is true
}
```

- condition: Ye expression ya logical test hota hai jo true ya false return karta hai.
- code block: Ye { } ke andar likha code sirf tab execute hoga jab condition true ho.

Example 1: Weather Check
Imagine karo: Tum soch rahe ho ki umbrella le jaoge ya nahi.
Logic in real life:
- Agar barish ho rahi hai, toh umbrella le lo.
- Agar barish nahi ho rahi, toh umbrella nahi chahiye.

```
#include <iostream>
using namespace std;
int main() {
   bool barish = true;  // true matlab barish ho rahi hai
   if (barish) {
      cout << "Umbrella le lo." << endl;
   }
   return 0;
}
```
Explanation:
- barish = true hai → condition true → "Umbrella le lo." print hoga.
- Agar barish false hoti → code skip ho jata, aur umbrella nahi lete.

Traffic Signal
Logic:
- Agar traffic signal green hai → aage jao.
- Agar red hai → ruk jao.
```
#include <iostream>
using namespace std;
int main() {
   string signal = "red";
   if (signal == "green") {
       cout << "Go!" << endl;
   }
   if (signal == "red") {
       cout << "Stop!" << endl;
   }
   return 0;
}
```

Explanation:

- Signal red hai → "Stop!" print hoga.
- Aase hi or colors ke according statement print hogi.

if-else Statement in C++

Definition:
if-else statement ek conditional statement hai jo program ko allow karta hai ki wo do possible actions me se ek choose kare:

- Agar condition true → if block execute hoga
- Agar condition false → else block execute hoga
- Matlab: “Agar ye condition true hai, toh ye karo, warna ye karo.”

Syntax:

```
if (condition) {
    // code if condition is true
} else {
    // code if condition is false
}
```

Age Verification for Movie
Scenario:

- Agar age 18 ya zyada → movie dekh sakte ho
- Nahi → entry allowed nahi

```
#include <iostream>
using namespace std;
int main() {
   int age = 16;
   if (age >= 18) {
      cout << "Aap movie dekh sakte ho." << endl;
   } else {
      cout << "Sorry, entry allowed nahi hai." << endl;
   }
   return 0;
}
```

- Age 16 → "Sorry, entry allowed nahi hai." print hoga

Bank ATM Check
Scenario:

- Agar account me paisa hai → withdraw possible
- Nahi → withdraw nahi hoga

```
#include <iostream>
using namespace std;
int main() {
   double balance = 0;
   if (balance > 0) {
      cout << "Withdraw possible." << endl;
   } else {
      cout << "Insufficient balance." << endl;
   }
   return 0;
}
```

- Balance 0 → "Insufficient balance." print hoga

Shopping Discount
Scenario:

- Agar shopping amount 1000 se zyada → 10% discount milega
- Nahi → discount nahi milega

```
#include <iostream>
using namespace std;
int main() {
   double amount = 800;

   if (amount > 1000) {
      cout << "10% discount milega!" << endl;
```

```
  } else {
     cout << "Discount nahi milega." << endl;
  }
  return 0;
}
```

- Amount 800 → else block execute → "Discount nahi milega."

else if Statement in C++

- else if statement ka use multiple conditions check karne ke liye hota hai.
- Matlab:
- "Agar pehli condition false hai, toh doosri condition check karo, agar wo bhi false hai, toh teesri condition check karo… aur agar sab false hai, toh else execute karo."
- Ye basically decision-making for more than two choices ke liye use hota hai.

```
Syntax:
if (condition1) {
   // code if condition1 is true
} else if (condition2) {
   // code if condition2 is true
} else if (condition3) {
   // code if condition3 is true
} else {
   // code if all conditions are false
}
```

Temperature Check

Scenario:

- Temp >= 35 → "It's hot"
- Temp >= 25 → "Warm"
- Temp >= 15 → "Cool"
- Temp < 15 → "Cold"

```
#include <iostream>
using namespace std;
int main() {
  int temp = 28;
  if (temp >= 35) {
     cout << "It's hot" << endl;
  } else if (temp >= 25) {
     cout << "Warm" << endl;
  } else if (temp >= 15) {
     cout << "Cool" << endl;
  } else {
        cout << "Cold" << endl;
    }
    return 0;
}
```

- Temp 28 → "Warm" print

Online Shopping Delivery Charges
Scenario:

- Amount > 1000 → Free delivery
- Amount > 500 → 50 Rs delivery charges
- Amount <= 500 → 100 Rs delivery charges

```
#include <iostream>
using namespace std;
int main() {
    double amount = 650;
    if (amount > 1000) {
        cout << "Free delivery" << endl;
    } else if (amount > 500) {
        cout << "Delivery charges: 50 Rs" << endl;
    } else {
        cout << "Delivery charges: 100 Rs" << endl;
    }

    return 0;
}
```

- Amount 650 → "Delivery charges: 50 Rs" print

switch statement in C++

switch ek control statement hota hai jo multiple choices me se ek specific block ko run karta hai.
Jab aapke paas bahut saare if-else ho jaayein, tab switch code ko clean aur readable banata hai.

Simple Explanation

- Ek variable ya expression ko switch me dete ho.
- Phir case ke through check hota hai ki value kis case se match ho rahi hai.
- Jo case match hota hai, uska code execute hota hai.
- break lagane se switch ruk jata hai (warna next case bhi chal sakta hai).
- default tab chalega jab koi case match nahi karta.

Syntax

```
switch (expression) {
    case value1:
        // code
        break;
    case value2:
        // code
        break;

    default:
        // code
}
```

Ques 1:- Aap ek coffee machine ka program bana rahe ho.
User ko 3 options diye gaye hain:

1. Espresso
2. Cappuccino
3. Latte

User ek number enter karega, aur machine uske according coffee ka naam print karegi.
Agar user galat number dale, to **"Invalid Choice"** print hona chahiye.

```
#include <iostream>
using namespace std;
int main() {
    int choice;
    cout << "Coffee Machine Menu:\n";
    cout << "1. Espresso\n";
    cout << "2. Cappuccino\n";
    cout << "3. Latte\n";
    cout << "Enter your choice: ";
    cin >> choice;
    switch (choice) {
        case 1:
            cout << "You selected: Espresso";
            break;
        case 2:
            cout << "You selected: Cappuccino";
            break;
        case 3:
            cout << "You selected: Latte";
            break;
        default:
            cout << "Invalid Choice";
    }
    return 0;
}
```

Simple Real-Life Understanding
Jaise real coffee machine me 3 buttons hote hain:

- Button 1 → Espresso
- Button 2 → Cappuccino
- Button 3 → Latte

Switch-case wahi cheez program me implement karta hai.
User jo bhi button (number) enter karega, uske hisaab se output milega.

Ques 2:- ATM machine ka program banao jisme user option choose kare:
1 = Check Balance
2 = Withdraw Money
3 = Deposit Money

```
#include <iostream>
using namespace std;
int main() {
   int choice;
   cout << "ATM Menu:\n";
   cout << "1. Check Balance\n";
   cout << "2. Withdraw Money\n";
   cout << "3. Deposit Money\n";
   cout << "Enter your choice: ";
   cin >> choice;
   switch(choice) {
      case 1:
         cout << "Your balance is: 5000 Rs";
         break;
      case 2:
         cout << "Withdrawal successful!";
         break;
      case 3:
         cout << "Amount deposited successfully!";
         break;
      default:
         cout << "Invalid Option!";
   }
   return 0;
}
```

Simple Real-Life Meaning
Just like real ATM:

- 1 dabao → Balance dikhega
- 2 dabao → Paise nikalne ka process
- 3 dabao → Paise deposit
- Galat button dabao → Error message

Wahi pura process program me switch-case ke through implement kiya gaya hai.

Ques 3: Ticket booking system design karo jisme user travel class choose kare:
1 (Sleeper), 2 (AC), 3 (General)..

```
#include <iostream>
using namespace std;
int main() {
    int type;
    cout << "Choose Ticket Type:\n";
    cout << "1. Sleeper\n";
    cout << "2. AC\n";
    cout << "3. General\n";
    cout << "Enter choice: ";
    cin >> type;
    switch(type) {
        case 1:
            cout << "You selected: Sleeper Ticket";
            break;
        case 2:
            cout << "You selected: AC Ticket";
            break;
        case 3:
            cout << "You selected: General Ticket";
            break;
        default:
            cout << "Invalid Selection";
    }
    return 0;
}
```

Explanation
Socho aap ek ticket window pe ho — waha ek board hai:
Choose Ticket Type:
1. Sleeper
2. AC
3. General
Enter your choice:

- Agar aap 1 likhte ho → teller bolega "You selected: Sleeper Ticket".
- Agar 2 likhte ho → "AC Ticket".

- Agar 3 → “General Ticket”.
- Agar aap kuch aur number likhte ho (jaise 5 ya 0) → “Invalid Selection” bolega.

Program basically wahi logic implement karta hai — user input ke hisaab se appropriate response print karta hai.

Loop

- Loop ek aisa control structure hai jo code ko repeat karta hai — ek ya zyada baar — jab tak koi condition true hai.
- Matlab: agar aap chahte ho ki “print 1 to 10”, ya “sab students ke marks print karo”, ya “user se input lo jab tak wo valid na ho” — aise tasks ke liye loop bahut useful hai.

Types of Loops in C++ (aur jab use karein)
C++ (aur bahut saare programming languages) me mainly **3** loop types hote hain.

Loop Type	Kab use karein / kis situation ke liye
for loop	Jab pata ho ki code ko kitni baar run karna hai (fixed number of iterations).
while loop	Jab iterations ka number pehle se pata na ho — ya loop chalana ho “jab tak condition true hai”.
do-while loop	Jab kam se kam ek baar code run karna ho, phir condition check karni ho. Agar condition true ho, loop fir se chalega.

for loop

- For-loop ek loop / repetition control statement hai jo ek block of code ko baar-baar (repeat) chalata hai.
- Jab aapko pata ho ki code ko kitni baar chalana hai (ya start-se-end tak count/iterate karna hai), for-loop bahut useful hai. In short: “for loop = repeat karna, fixed ya known number of times”.

Syntax of for-loop (in C++)

```
for (initialization; condition; update) {
   // code block (body) — jo baar-baar chalega
}
```

Yaha teen cheezein hoti hain: initialization, condition, update.

- **initialization** — ek variable ko start value dena (jaise int i = 0). Ye ek baar hota hai, loop ke shuru hone se pehle.
- **condition** — ye check karta hai ki loop chalna chahiye ya nahi. Agar condition true hogi, tab loop body chalega; agar false, to loop terminate ho jayega.
- **update (increment/decrement)** — har iteration ke baad loop variable update hota hai (jaise i++, ya i--, ya koi aur update).

Print “Hello World” 5 Times (for-loop ke saath)

```
#include <iostream>
using namespace std;
int main() {
    for (int i = 0; i < 5; i++) {
        cout << "Hello World\n";
    }
    return 0;
}
```

```
Output
Hello World
Hello World
Hello World
Hello World
Hello World
```

Print Even Numbers from 1 to N

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

int main() {
    int N;
    cout << "Enter a number: ";
    cin >> N;

    cout << "Even numbers from 1 to " << N << " are:\n";
    for (int i = 1; i <= N; i++) {
        if (i % 2 == 0) {
            cout << i << " ";
        }
    }
    cout << endl;
    return 0;
}
```

```
Output
Even numbers from 1 to 10 are: 2
4 6 8 10
```

Kya karta hai:

- User se N leta hai.
- 1 se N tak check karta hai kaunse numbers even hain (i % 2 == 0).
- Even numbers print karta hai.

Simple Sum of 5 Numbers

```
#include <iostream>
using namespace std;
int main() {
    int a, b, c, d, e;
    cout << "Enter 5 numbers:\n";
    cin >> a >> b >> c >> d >> e;
    int sum = a + b + c + d + e;
    cout << "Sum of those 5 numbers = " << sum << endl;
    return 0;
```

Input:
```
Enter 5 numbers:
2 4 6 8 10
```
Output:
```
Sum of those 5 numbers = 30
```

```
}
```

Explanation

- Socho: Aapko 5 alag amounts (jaise 5 alag cheezo ka kharcha) pata karna hai. Aap user se 5 amounts le rahe ho. Fir aap un sabka total nikal rahe hain.
- Ye program wahi karta hai: user 5 numbers dega → program unka sum karega → final sum print karega.

What is a while-loop

- While-loop ek aisa control-flow statement hai jo baar-baar (repeat) ek block of code chalata hai — jab tak koi condition (shart) true ho.
- Matlab agar aapko koi kaam tab tak repeat karna ho jab tak koi condition satisfy ho, lekin aap nahi jaante ki kitni baar — for loop ke bajay while loop useful hai.
- While-loop me pehle condition check hoti hai; agar wo true hai tab hi loop body chalega. Agar initial condition hi false ho, loop body ek baar bhi nahi chalega.

Syntax of while-loop (in C++)

```
while (condition) {
   // code block (loop body)
   // update statements (if needed)
}
```

- condition — Boolean expression ya koi logical check jo decide karega ki loop chalna chahiye ya nahi.
- Agar condition true hai → loop body executes. Uske baad control wapas condition check pe jata hai.
- Agar condition false ho jaati hai → loop terminate ho jata hai aur program next statement se continue karta hai.

Ques 1:- Socho, ek tank hai aur hume usme paani tab tak dalna hai jab tak tank full na ho jaye. Hum nahi jaante exactly kitna paani lagega, isliye hum tank ka current level check karte rahenge.

```
#include <iostream>
using namespace std;
int main() {
   int tankCapacity = 10; // tank ki maximum capacity (liters)
   int currentLevel = 0;  // abhi ka water level
   // Jab tak tank full nahi hota, paani add karte rahenge
   while (currentLevel < tankCapacity) {
      cout << "1 liter paani add kar rahe hain..." << endl;
      currentLevel++; // paani ka level 1 liter badh gaya
      cout << "Current water level: " << currentLevel << " liters" << endl;
   }
   cout << "Tank full ho gaya!" << endl;
```

```
Output
1 liter paani add kar rahe hain...
Current water level: 1 liters
1 liter paani add kar rahe hain...
Current water level: 2 liters
1 liter paani add kar rahe hain...
Current water level: 3 liters
1 liter paani add kar rahe hain...
Current water level: 4 liters
1 liter paani add kar rahe hain...
Current water level: 5 liters
1 liter paani add kar rahe hain...
Current water level: 6 liters
1 liter paani add kar rahe hain...
Current water level: 7 liters
1 liter paani add kar rahe hain...
Current water level: 8 liters
1 liter paani add kar rahe hain...
Current water level: 9 liters
```

```
    return 0;
}
```

Explanation

- currentLevel < tankCapacity → ye condition hai.
  - Jab tak ye true hai, loop chalega.
  - Jaise hi false ho jata hai (tank full), loop ruk jaata hai.
- currentLevel++ → iska matlab hai paani add kar rahe hain.
- Loop ke andar jo bhi statements hain, wo repeat hote rahenge jab tak
- condition false na ho jaye.

Ques 2:-Socho, app me login karna hai aur user galat password enter kare, to app baar-baar password maangega jab tak sahi password na mile.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string password;
    string correctPassword = "12345";
    // Loop tab tak chalega jab tak user sahi password na dale
    while (password != correctPassword) {
        cout << "Password enter karo: ";
        cin >> password;
        if (password != correctPassword) {
            cout << "Wrong password! Try again.\n";
        }
    }
    cout << "Login successful!" << endl;
    return 0;
}
```

Explanation

- while (password != correctPassword) → Jab tak password galat hai, loop chalega.
- Jaise hi password sahi ho jata hai, loop khatam ho jata hai.

Ques 3: - Socho, ATM me paise nikalna hai, par balance kam hai. ATM user se baar-baar amount enter karwayega jab tak valid amount na ho.

```
#include <iostream>
using namespace std;
int main() {
    int balance = 5000;
```

Output

```
Password enter karo: 1111
Wrong password! Try again.
Password enter karo: 2222
Wrong password! Try again.
Password enter karo: 12345
Login successful!
```

```
    int withdrawAmount = 0;
    cout << "ATM khula hai.\n";
    while (true) {
        cout << "Kitna paisa nikalna hai? ";
        cin >> withdrawAmount;
        if (withdrawAmount <= balance) {
            balance -= withdrawAmount;
            cout << "Transaction successful. Remaining balance: " << balance << endl;
            break; // loop stop kar do
        } else {
            cout << "Insufficient balance. Try again.\n";
        }
    }

    return 0;
}
```

Explanation

- while(true) → infinite loop, tab tak chalega jab tak break na ho.
- Agar paisa valid hai, transaction karte hi loop terminate ho jata hai.

do-while loop

do-while loop me code block kam se kam ek baar zaroor chalega, chahe condition false ho.
Uske baad condition check hoti hai. Agar true ho, loop repeat hota hai; agar false ho, loop ruk jata hai.

Syntax

```
do {
    // statements jo execute honge
} while (condition);
```

- do { ... } → code block pehle execute hota hai
- while(condition); → condition baad me check hoti hai

Difference between while aur do-while:

Feature	while loop	do-while loop
Execution	Condition check pehle hoti hai, shayad run hi na ho	Code pehle run hota hai, phir check hota hai
Minimum run	0 times	1 time
Use case	Jab code run na karna ho agar condition false ho	Jab code kam se kam 1 baar zaroor run karna ho

Ques1:- Socho, online shopping site pe user se poocha jaaye ki wo aur shopping karna chahta hai ya nahi.

```
#include <iostream>
using namespace std;
int main() {
    char choice;
    do {
        cout << "Item added to cart" << endl;
        cout << "Do you want to add more items? (y/n): ";
        cin >> choice;
    } while(choice == 'y' || choice == 'Y');

    cout << "Checkout time!" << endl;
    return 0;
}
```

Output

```
Item added to cart
Do you want to add more
items? (y/n): y
Item added to cart
Do you want to add more
items? (y/n): y
Item added to cart
Do you want to add more
items? (y/n): n
Checkout time!
```

Explanation

- Pehle item cart me add hota hai.
- Fir program poochta hai: “Aur add karna hai? (y/n)”
- Agar user y → fir se item add hota hai.
- Agar user n → loop ruk jaata hai.
- Last me “Checkout time!” print hota hai.

Ques 2: Playing a Game Until Win

- Player game khelta rahe tab tak jab tak jeet na jaaye.
- Real life: Kam se kam ek baar game khela jaayega.

```
#include <iostream>
using namespace std;
int main() {
    char win;
    do {
        cout << "Playing the game..." << endl;
        cout << "Did you win? (y/n): ";
        cin >> win;
    } while(win != 'y' && win != 'Y');
    cout << "Congratulations! You won the game!" << endl;
    return 0;
}
```

```
Output
Playing the game...
Did you win? (y/n): n
Playing the game...
Did you win? (y/n): n
Playing the game...
Did you win? (y/n): y
Congratulations! You
won the game!
```

Explanation

- Kam se kam ek baar game khela jaayega.
- User se poocha jaata hai: “Did you win?”
- Agar user y/Y → loop ruk jaata hai.
- Agar user n → loop repeat hota hai, game phir se khela jaata hai.

Ques 3: Playing a Video Game Until Level Complete

- Scenario: Player game khelta rahe jab tak level complete na ho jaaye.
- Real-life: Kam se kam ek attempt zaroor hoga.

```
#include <iostream>
using namespace std;
int main() {
    char levelComplete;
    do {
        cout << "Playing the level..." << endl;
        cout << "Is the level complete? (y/n): ";
        cin >> levelComplete;
    } while(levelComplete != 'y' && levelComplete != 'Y');
    cout << "Congratulations! You completed the level!" << endl;
```

```
Output
Playing the level...
Is the level complete?
(y/n): n
Playing the level...
Is the level complete?
(y/n): n
Playing the level...
Is the level complete?
(y/n): y
Congratulations! You
```

```
    return 0;
}
```

Explanation

- Kam se kam ek baar level khela jaayega.
- User se poocha jaata hai: "Is the level complete?"
- Agar user y/Y → loop ruk jaata hai.
- Agar user n → loop repeat hota hai, level fir se khela jaata hai.

For-Each Loop

For-each loop ka use tab hota hai jab hume array ya collection ke har element se kaam karna ho.

- Hume index ke baare me sochne ki zarurat nahi hoti.
- C++ me isko range-based for loop bhi kehte hain.

Syntax:

```
for (data_type variable : collection) {
    // variable ka use karo
}
```

- data_type variable → element ka type (int, string, etc.)
- collection → array, vector ya koi bhi iterable container

Ques 1: Checking All Items in a Shopping Cart

- Scenario: Shopping cart me jitne bhi items hain, unko print karna.

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<string> cart = {"Shirt", "Jeans", "Shoes"};
    for(string item : cart) {
        cout << "Item in cart: " << item << endl;
    }
    return 0;
}
```

Output

```
Item in cart: Shirt
Item in cart: Jeans
Item in cart: Shoes
```

Explanation

- #include <iostream> → Input/output ke liye use hota hai (cin, cout).
- #include <vector> → Vector (dynamic array) use karne ke liye.
- using namespace std; → std:: likhne ki zarurat nahi.
- int main() → Program yahan se start hota hai.
- vector<string> cart = {"Shirt", "Jeans", "Shoes"};
- cart ek vector of strings hai.
- Isme 3 items store hain: "Shirt", "Jeans", "Shoes".
- Real-life analogy: Shopping cart me items add kiye ja rahe hain.

- for(string item : cart)
- Ye for-each loop hai.
- Loop automatically cart ke har element ko pick karta hai.
- item variable har iteration me current element ko represent karta hai.
- cout << "Item in cart: " << item << endl;
- Har item ko print karta hai.

Ques 2:- Scenario: Ek store me kuch products available hain, jaise Laptop, Mobile, aur Tablet. Aapko C++ program likhna hai jo store ke sab available products ko ek-ek karke print kare.
Task:
1. Ek vector banaye jisme products ke names ho: "Laptop", "Mobile", "Tablet".
2. For-each loop ka use karke sab products print kare.
3. Output me har product ke aage likhe: "Available product: "

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
   vector<string> products = {"Laptop", "Mobile", "Tablet"};
   for(string product : products) {
      cout << "Available product: " << product << endl;
   }
   return 0;
}
```

```
Output
Available product: Laptop
Available product: Mobile
Available product: Tablet
```

Explanation
- #include <iostream> → Input/Output ke liye, jaise cout.
- #include <vector> → Vector use karne ke liye, jo ki dynamic array jaisa hota hai.
- using namespace std; → std:: likhne ki zarurat nahi.
- vector<string> products = {"Laptop", "Mobile", "Tablet"};
- products ek vector hai jisme 3 items store hain: Laptop, Mobile, Tablet.
- Real-life analogy: Store me available products list me store ho gaye hain.
- for(string product : products)
- Ye for-each loop hai.
- Loop automatically vector ke har element ko pick karta hai.
- product variable current element ko represent karta hai.
- cout << "Available product: " << product << endl;
- Loop ke andar har product print hota hai.

Nested Loop
- Jab ek loop ke andar aur ek loop hota hai, use nested loop kehte hain.
- Outer loop pehle run hota hai, aur inner loop har iteration ke liye run hota hai.
- Mostly use hota hai jab 2D patterns, tables, ya matrix banani ho.

Ques 1:- Classroom Seating Arrangement

- **Scenario: Ek classroom me 3 rows hain, aur har row me 4 seats hain.**
- **Program: Print row aur seat number.**

```
#include <iostream>
using namespace std;
int main() {
    for(int row = 1; row <= 3; row++) {         // Outer loop: rows
        for(int seat = 1; seat <= 4; seat++) {  // Inner loop: seats in row
            cout << "Row " << row << ", Seat " << seat << endl;
        }
    }
    return 0;
}
```

```
Output
Row 1, Seat 1
Row 1, Seat 2
Row 1, Seat 3
Row 1, Seat 4
Row 2, Seat 1
Row 2, Seat 2
Row 2, Seat 3
Row 2, Seat 4
Row 3, Seat 1
Row 3, Seat 2
Row 3, Seat 3
Row 3, Seat 4
```

Explanation

- for(int row = 1; row <= 3; row++)
  - Ye outer loop hai.
  - row variable 1 se 3 tak chalega → matlab 3 rows hain.
- for(int seat = 1; seat <= 4; seat++)
  - Ye inner loop hai jo outer loop ke har iteration me chalta hai.
  - seat variable 1 se 4 tak chalega → matlab har row me 4 seats hain.
- cout << "Row " << row << ", Seat " << seat << endl;
  - Ye line har seat ka position print karti hai.
  - Example: Row 2, Seat 3
- Loop ka flow:
  - Outer loop ka pehla iteration → row = 1
    1. Inner loop chalega → seat = 1,2,3,4 print honge
  - Outer loop ka dusra iteration → row = 2
    1. Inner loop chalega → seat = 1,2,3,4 print honge
  - Outer loop ka teesra iteration → row = 3
    1. Inner loop chalega → seat = 1,2,3,4 print honge
- return 0;
  - Program successfully finish ho gaya.

Ques 2- Multiplication Table (1 to 5)

- **Scenario: Table print karna 1 se 5 tak ke numbers ka.**

```
#include <iostream>
using namespace std;
int main() {
    for(int i = 1; i <= 5; i++) {        // Outer loop: table number
        for(int j = 1; j <= 10; j++) {   // Inner loop: multiply 1 to 10
```

```
Output
1 x 1 = 1
1 x 2 = 2
...
1 x 10 = 10
-----------------
2 x 1 = 2
2 x 2 = 4
...
5 x 10 = 50
-----------------
```

```
        cout << i << " x " << j << " = " << i*j << endl;
    }
    cout << "------------------" << endl;
  }
  return 0;
}
```

Explanation

- for(int i = 1; i <= 5; i++)
- Ye outer loop hai.
- i variable table number represent karta hai.
- Ye loop 1 se 5 tak chalega → matlab table 1 se 5 print honge.
- for(int j = 1; j <= 10; j++)
- Ye inner loop hai.
- j variable multiplier hai (1 se 10 tak).
- Har table ke liye 1 se 10 tak multiplication calculate hota hai.
- cout << i << " x " << j << " = " << i*j << endl;
- Ye line current table ka ek step print karti hai.
- Example: 2 x 3 = 6
- cout << "------------------" << endl;
- Ye line break deta hai har table ke baad, taaki tables alag dikhai dei

break

break ek control statement hai jo loop ya switch statement ko turant rok deta hai. Matlab jaise hi program break encounter karta hai, current loop ya switch se bahar nikal jaata hai.

Kahaan use hota hai?
break ka use mainly 2 jagah hota hai:

1. Loops mein – for, while, do-while
2. Switch statement mein

Example – Ticket Counter
Scenario:

- Ek bus stand pe log line me khade hain.
- Bus me sirf 5 seats available hain.
- Jaise hi 5 log board kar lete hain, counter line checking ko rok deta hai.

```
#include <iostream>
using namespace std;
int main() {
   int totalPeople = 10; // line me total log
   int seatsAvailable = 5;
   for (int person = 1; person <= totalPeople; person++) {
      cout << "Person " << person << " is boarding the bus." << endl;
      if (person == seatsAvailable) {
         cout << "Bus full! No more people can board." << endl;
         break; // bus full, loop stop
      }
   }
   return 0;
}
```

```
Output
Person 1 is boarding
the bus.
Person 2 is boarding
the bus.
Person 3 is boarding
the bus.
Person 4 is boarding
the bus.
Person 5 is boarding
the bus.
Bus full! No more
people can board.
```

Explanation:

- Loop normally 1 se 10 tak chalta.
- Lekin jaise hi 5th person board karta hai, break execute hota hai.
- Baaki log board nahi karte.

Scenario:

- School me 3 periods hain.
- Har period me students ko 10 times bell ring hoti hai (counting practice ke liye).
- Lekin agar 5th bell me emergency ho jaaye, teacher current period ka bell
- turant rok deta hai aur next period me chala jaata hai.

```
#include <iostream>
using namespace std;
int main() {
   for (int period = 1; period <= 3; period++) {
      cout << "Period " << period << " starts" << endl;
      for (int bell = 1; bell <= 10; bell++) {
         cout << "Bell ring " << bell << endl;
         if (bell == 5) { // emergency
            cout << "Emergency! Stop ringing bells for this period." << endl;
            break; // inner loop stop
         }
      }
      cout << endl;
```

```
Output
Period 1 starts
Bell ring 1
Bell ring 2
Bell ring 3
Bell ring 4
Bell ring 5
Emergency! Stop
ringing bells for
this period.

Period 2 starts
Bell ring 1
Bell ring 2
Bell ring 3
Bell ring 4
Bell ring 5
Emergency! Stop
ringing bells for
this period.
```

```
    }
    return 0;
}
```

Explanation

1. Outer loop 3 periods ke liye chalega aur har period ke start ko print karega.
2. Inner loop 1 se 10 tak bell rings simulate karta hai.
3. Jaise hi bell 5 hoti hai, emergency message print hota hai aur inner loop break se ruk jaata hai.
4. Next period ka loop continue hota hai, aur ye process har period ke liye repeat hota hai.

continue kya hai?

continue bhi ek control statement hai, lekin break se thoda alag kaam karta hai:

- continue loop ke current iteration ko skip kar deta hai aur next iteration pe chala jaata hai.
- Matlab, baki ka code us iteration me execute nahi hota, par loop poora terminate nahi hota.

Kahaan use hota hai?
continue sirf loops (for, while, do-while) me use hota hai.

Ques1:- 1 se 20 tak numbers print karo, lekin odd numbers skip karo using continue.
Idea:

- Loop 1–20
- Agar number odd ho (i % 2 != 0), to continue karo
- Sirf even numbers print honge

```
#include <iostream>
using namespace std;
int main() {
    // 1 se 20 tak loop chalega
    for (int i = 1; i <= 20; i++) {
        // Agar number odd hai to skip kar do
        if (i % 2 != 0) {
            continue; // odd number print nahi hoga
        }
        // Sirf even numbers print honge
        cout << i << " ";
    }

    return 0;
}
```

```
Output
2 4 6 8 10 12 14 16 18 20
```

Explanation

- Loop 1 se 20 tak chalega (for (i=1; i<=20; i++)).
- Agar number odd ho (i % 2 != 0), continue use karke us iteration skip karenge.

- Sirf even numbers print honge (cout << i).
- continue current iteration skip karta hai, loop ko exit nahi karta.

Ques 2: - Skip Multiples of 3 1 se 15 tak numbers print karo, multiples of 3 skip karo.

```
#include <iostream>
using namespace std;
int main() {
   // 1 se 15 tak numbers loop me
   for (int i = 1; i <= 15; i++) {
      // Agar number 3 ka multiple hai, to skip karo
      if (i % 3 == 0) {
         continue; // current iteration skip, next number par jao
      }
      // Baaki numbers print karo
      cout << i << " ";
   }
   return 0;
}
```

Explanation

- Loop 1 se 15 tak numbers ke liye chal raha hai.
- if (i % 3 == 0) check karta hai ki number 3 ka multiple hai ya nahi.
- Agar multiple hai → continue use karke us number ko skip karte hain.
- Baaki numbers print hote hain (cout << i).

Array

- Array ek group of values hota hai, jisme saare elements same data type ke hote hain.
- Ye contiguous (ek ke baad ek) memory locations me store hote hain.
- Socho ek table jisme 5 numbers store karne ho — instead of 5 alag variables, ek array me saare numbers store kar sakte ho.

Why Use Arrays?

- Ek hi naam ke under multiple values store hoti hain.
- Loop se easily traverse/print/process kar sakte ho.
- Same type data ko organize karna easy hota hai.

Array Initialization Methods

Way to Initialize	Syntax Example	Explanation
1) Full Initialization	int arr[5] = {10,20,30,40,50};	Sabhi elements ko values de di — array *exactly* isi order me fill hota hai.
2) Auto-Size Initialization	int arr[] = {1,2,3,4};	[] me size nahi likha — compiler values ki count se size decide karta hai.

Way to Initialize	Syntax Example	Explanation
3) Partial Initialization	int arr[5] = {5,10};	Sirf kuch values diye → baaki elements 0 se fill ho jayenge.
4) Zero Initialization	int arr[5] = {0};	Sabhi elements 0 se initialize ho jayenge (first explicitly, baaki by default zero).
5) Empty Braces Init	int arr[5] = {};	Is se bhi array ka har element 0 ho jayega.
6) Char Array with String	char name[] = "Hello";	Character array me string literal se init → arr = {'H','e','l','l','o','\0'} (null terminator).

Access the Elements of an Array

C++ mein array ke elements ko index number se access kiya jata hai.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
  string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
  cout << cars[0];
  return 0;
}
```

Change an Array Element

Array ek continuous memory block hota hai jisme hum multiple values store karte hain.
Har value ka apna index hota hai (0 se start hota hai).
Agar hume kisi specific element ko change karna ho, to bas uske index par new value assign kar dete hain.

```
#include <iostream>
using namespace std;
int main() {
    int numbers[5] = {10, 20, 30, 40, 50};
    // Index 2 (third element) ko change kar rahe hain
    numbers[2] = 100;
    cout << "Updated array: ";
    for(int i = 0; i < 5; i++) {
        cout << numbers[i] << " ";
    }
    return 0;
}
```

Types Of Array in C++

Array Type	Definition	Explanation	Example
1D Array (One-Dimensional)	Linear list of elements	Ek simple list jisme values ek line me hoti hain	int arr[5];
2D Array (Two-Dimensional)	Array of arrays (rows × columns)	Table/matrix jaisa structure	int arr[3][3];
Multi-Dimensional Array (3D/4D...)	More than 2 dimensions	Layers ya cube jaisi structure	int arr[2][3][4];
Static Array	Size fixed at compile time	Pehle se decide size, change nahi hota	int arr[10];
Dynamic Array	Size given at runtime using pointers	Program chalne ke time size decide hota hai	int* arr = new int[n];
Array of Objects	Stores multiple objects	Ek hi class ke multiple objects ek array me	Student s[10];

1D Array (Train Coaches Example)

Ek program banao jisme train ke 5 coaches ke naam ek 1D array me store karo (jaisey Engine, Sleeper, AC, General, Pantry) aur unko screen par print karo.

```
#include <iostream>
using namespace std;
int main() {
   // 1D array of 5 train coaches
   string coaches[5] = {"Engine", "Sleeper", "AC", "General", "Pantry"};
   cout << "Train ke coaches:\n";
   // Printing all coaches
   for(int i = 0; i < 5; i++) {
      cout << "Coach[" << i << "] = " << coaches[i] << endl;
   }

   return 0;
}
```

Output

```
Train ke coaches:
Coach[0] = Engine
Coach[1] = Sleeper
Coach[2] = AC
Coach[3] = General
Coach[4] = Pantry
```

Train Coaches 1D Array Program

Train Coaches 1D Array Program Explanation

- Step 1: #include <iostream> → Input/output ke liye library include ki.
- Step 2: using namespace std; → cout aur cin use karne ke liye.
- Step 3: string coaches[5] = {...}; → 1D array banaya jisme 5 train coaches ke naam store kiye.
    - Index 0 → Engine
    - Index 1 → Sleeper

  - Index 2 → AC
  - Index 3 → General
  - Index 4 → Pantry
- Step 4: for loop ka use karke array ke saare elements print kiye.
  - Loop i = 0 se i < 5 tak chalega.
  - cout << "Coach[" << i << "] = " << coaches[i]; → Har coach ka naam screen par print hoga.
- Step 5: Program end → return 0;

2D Array (Classroom Seating Example)

Ek program banao jisme classroom ki 4 rows aur 5 columns ka seating arrangement 2D array me store karna hai.
Har seat par baithne wale student ka roll number input lekar pura seating chart display karo.

```
#include <iostream>
using namespace std;
int main() {
   int rows = 4;
   int cols = 5;
   int seating[4][5];  // 2D array for classroom seating
   // Input roll numbers for each seat
   cout << "Enter roll numbers for 4x5 classroom seating:\n";
   for(int i = 0; i < rows; i++) {
      for(int j = 0; j < cols; j++) {
         cout << "Seat[" << i << "][" << j << "]: ";
         cin >> seating[i][j];
      }
   }
   // Display seating chart
   cout << "\nClassroom Seating Chart:\n";
   for(int i = 0; i < rows; i++) {
      for(int j = 0; j < cols; j++) {
         cout << seating[i][j] << "\t";  // \t for spacing
      }
      cout << endl;
   }
   return 0;
}
```

```
Output
Enter roll numbers for 4x5 classroom
seating:
Seat[0][0]: 101
Seat[0][1]: 102
Seat[0][2]: 103
Seat[0][3]: 104
Seat[0][4]: 105
...
Classroom Seating Chart:
101   102   103   104   105
106   107   108   109   110
111   112   113   114   115
116   117   118   119   120
```

Program Explanation

- Step 1: int seating[4][5]; → 2D array banaya jisme 4 rows × 5 columns ke seats store honge.
- Step 2: Nested for loop → Outer loop rows ke liye, inner loop columns ke liye.

- Step 3: cin >> seating[i][j]; → User se roll numbers input liye.
- Step 4: Display ke liye bhi nested loop use kiya.
- cout << seating[i][j] << "\t"; → tab spacing ke liye.
- Step 5: Har row ke baad endl; → next line par seating show ho.

Multi-Dimensional Array (School Building Example)

Ek program banao jisme school ki 3-floor building ka data store karna hai.
Har floor me 3 classrooms hain aur har classroom me 10 students hain. 3D array ka use karke har student ka roll number store karo aur display karo.

```
#include <iostream>
using namespace std;
int main() {
    int floors = 3;
    int classrooms = 3;
    int students = 10;
    int school[3][3][10]; // 3D array: floors × classrooms × students
    // Input roll numbers for each student
    for(int f = 0; f < floors; f++) {
        for(int c = 0; c < classrooms; c++) {
            cout << "Enter roll numbers for Floor " << f << ", Classroom " << c << ":\n";
            for(int s = 0; s < students; s++) {
                cout << "Student[" << s << "]: ";
                cin >> school[f][c][s];
            }
        }
    }
    // Display all roll numbers
    cout << "\nSchool Building Roll Numbers:\n";
    for(int f = 0; f < floors; f++) {
        for(int c = 0; c < classrooms; c++) {
            cout << "Floor " << f << ", Classroom " << c << ": ";
            for(int s = 0; s < students; s++) {
                cout << school[f][c][s] << " ";
            }
            cout << endl;
        }
        cout << endl;
    }
```

Output

```
Enter roll numbers for Floor 0, Classroom
0:
Student[0]: 101
Student[1]: 102
...
Floor 0, Classroom 0: 101 102 103 104 105
106 107 108 109 110
Floor 0, Classroom 1: 111 112 113 114 115
116 117 118 119 120
...
```

```
    return 0;
}
```

Program Explanation

- Step 1: int school[3][3][10]; → 3D array banaya (3 floors × 3 classrooms × 10 students).
- Step 2: Nested loops:
  - Outer loop → floors
  - Middle loop → classrooms
  - Inner loop → students
- Step 3: cin >> school[f][c][s]; → User se roll numbers input liye.
- Step 4: Display ke liye bhi nested loops ka use kiya, har classroom ka roll number ek line me print kiya.
- Step 5: Floors ke beech me extra endl → readability ke liye.

Static Array (Movie Hall Seats Example)

Ek program banao jisme movie hall ki 200 seats ka static array banao.
User se 10 booked seats ka number input lo, un seats ko "Booked" mark karo aur total available seats print karo.

```
#include <iostream>
using namespace std;
int main() {
    const int totalSeats = 200;
    int seats[totalSeats] = {0}; // 0 = available, 1 = booked
    cout << "Enter 10 booked seat numbers (1 to 200):\n";
    for(int i = 0; i < 10; i++) {
        int seatNumber;
        cin >> seatNumber;

        if(seatNumber >= 1 && seatNumber <= totalSeats) {
            seats[seatNumber - 1] = 1; // mark seat as booked
        } else {
            cout << "Invalid seat number! Try again.\n";
            i--; // repeat this iteration
        }
    }
    int availableSeats = 0;
    cout << "\nSeat Status:\n";
    for(int i = 0; i < totalSeats; i++) {
        if(seats[i] == 0) {
            availableSeats++;
        }
```

Outpt

```
Seat Status:
Total available seats: 190
```

```
    }
    cout << "Total available seats: " << availableSeats << endl;
    return 0;
}
```

Explanation

- Total seats = 200
- User ne 10 seats book ki → 200 - 10 = 190 available seats
- Program ne booked seats ko 1 mark kiya aur available seats count kiye.

Dynamic Array

(Wedding Hall Chairs Example) Program Question (Hinglish): User se poochho ki wedding me kitne guests aa rahe hain. Utni number of chairs ka dynamic array banao, har chair ko ek seat number do aur sari chairs ki numbering display karo.

```
#include <iostream>
using namespace std;
int main() {
    int guests;
    cout << "Wedding me kitne guests aa rahe hain? ";
    cin >> guests;
    // Dynamic array creation
    int* chairs = new int[guests];
    // Assigning seat numbers
    for(int i = 0; i < guests; i++) {
        chairs[i] = i + 1; // seat numbers start from 1
    }
    // Display all chairs
    cout << "\nChairs numbering:\n";
    for(int i = 0; i < guests; i++) {
        cout << "Chair " << chairs[i] << endl;
    }
    // Free memory
    delete[] chairs;
    return 0;
}
```

Explanation

- int guests; → Guest count store karne ke liye variable.
- cin >> guests; → User se number of guests input liya.
- int* chairs = new int[guests]; → Dynamic array banaya, size run-time par decide hota hai.
- Loop (for) se chairs ko seat numbers assign kiye (chairs[i] = i + 1).
- Loop (for) se sabhi chairs display kiye (cout << "Chair " << chairs[i]).

- delete[] chairs; → Dynamic memory free ki.
- Concept: Dynamic array → size run-time par decide, guest count ke hisaab se chairs allocate aur display.

Array of Objects

(Students List Example) Program Question (Hinglish): Ek program banao jisme Student class ho — name, roll number aur marks ke saath. Array of objects ka use karke 5 students ka data input lo aur sab students ka full detail display karo.

```
#include <iostream>
using namespace std;
// Student class
class Student {
public:
    string name;
    int roll;
    int marks;
    // Function to input student details
    void input() {
        cout << "Enter name: ";
        cin >> name;
        cout << "Enter roll number: ";
        cin >> roll;
        cout << "Enter marks: ";
        cin >> marks;
    }
    // Function to display student details
    void display() {
        cout << "Name: " << name
             << ", Roll: " << roll
             << ", Marks: " << marks << endl;
    }
};
int main() {
    Student students[5]; // Array of 5 Student objects
    // Input details for 5 students
    cout << "Enter details of 5 students:\n";
    for(int i = 0; i < 5; i++) {
        cout << "\nStudent " << i+1 << ":\n";
        students[i].input();
    }
    // Display details
    cout << "\nStudents Details:\n";
    for(int i = 0; i < 5; i++) {
        students[i].display();
    }
    return 0;
}
```

Explanation

- class Student → Student class banayi jisme 3 members:
  - name → string, student ka name
  - roll → int, roll number
  - marks → int, marks
- void input() → Function, student ka data user se input lene ke liye.
- void display() → Function, student ka data screen par display karne ke liye.
- Student students[5]; → Array of 5 Student objects banaya.
- Loop (for) se 5 students ka data input liya using students[i].input().
- Loop (for) se 5 students ka data display kiya using students[i].display().
- Concept:
  - Array of objects → ek hi type ke multiple objects ek array me store kar sakte hain.
  - Object-oriented approach → data aur function ek sath manage hota hai.

Omit Array Size

- Omit Array Size ka matlab hai array declare karte waqt explicit size mention na karna.
- Ye sirf tab possible hai jab array initialize ho raha ho.
- Example: int arr[] = {1, 2, 3, 4}; → compiler automatically size 4 detect kar lega.
- Agar sirf declaration hai aur initialization nahi hai, size omit nahi kar sakte:
- int arr[];
- Char array me bhi size omit ho sakti hai: char name[] = "John"; → compiler size 5 (including \0) detect karega.
- Functions me array pass karte waqt size optional hoti hai: void printArray(int arr[], int n)
- Benefit: Future me elements add karna easy hai, bas initializer list update karna hai.
- Rule: Initialization mandatory hai agar array size omit karna hai.

Method	Size kaise handle hota hai	Example
Initialize at	Compiler initializer list se size ko automatically	int arr[] = {1, 2, 3, 4, 5};

Method	Size kaise handle hota hai	Example
declaration	deduce kar leta hai	
Function parameter	Size omit kar sakte hain, compiler ise pointer ke jaise treat karta hai	void printArray(int arr[], int n)
auto with initializer	Type aur elements {...} se deduce ho jate hain	auto arr = {1, 2, 3, 4};
std::vector	Dynamic size, declaration me size specify karne ki zarurat nahi	std::vector<int> vec = {1,2,3,4,5};
std::array	Template parameter me size chahiye, lekin code me repeat karne ki zarurat nahi	std::array<int,5> arr = {1,2,3,4,5};

Initialize at declaration
Question:
Ek week ke daily temperatures store karo aur print karo ki kaunsa din sabse zyada garam tha.
Constraints:

- Use int temps[] = {...}; aur size specify mat karo.
- Loop ka use karke maximum temperature find karo.

```
#include <iostream>
using namespace std;
int main() {
    // Daily temperatures of a week (size omitted, compiler deduces)
    int temps[] = {30, 32, 35, 33, 36, 31, 34};
    // Calculate size of array
    int n = sizeof(temps) / sizeof(temps[0]);
    // Find maximum temperature and corresponding day
    int maxTemp = temps[0];
    int maxDay = 0;
    for (int i = 1; i < n; i++) {
        if (temps[i] > maxTemp) {
            maxTemp = temps[i];
            maxDay = i;
        }
    }
    // Print result
    cout << "Sabse zyada garam din hai day " << maxDay + 1
         << " with temperature " << maxTemp << "°C" << endl;
    return 0;
}
```

Output
```
Sabse zyada garam din hai day 5
with temperature 36°C
```

Explanation

- Array Declaration: int temps[] = {30,32,35,33,36,31,34}; → Size compiler automatically deduce karta hai.

- Array Size: int n = sizeof(temps)/sizeof(temps[0]); → Total elements calculate karne ke liye.
- Initialize Max: maxTemp = temps[0]; maxDay = 0; → First element se start.
- Find Maximum: Loop through array → Agar current element > maxTemp, update maxTemp aur maxDay.
- Print Result: cout → Sabse zyada garam din aur temperature print kare.

Function parameter
Question: Ek function banao jo exam scores ka array le aur average score print kare.
Constraints:

- Function me array parameter ka size omit karo (int arr[]). Size n bhi function me pass karo. Array ke elements ka sum karke average calculate karo.

```
#include <iostream>
using namespace std;
// Function to calculate average
void printAverage(int arr[], int n) {
   int sum = 0;
   for(int i = 0; i < n; i++)
      sum += arr[i];
   double avg = (double)sum / n;
   cout << "Average score: " << avg << endl;
}
int main() {
   int scores[] = {80, 90, 75, 85, 95};
   int n = sizeof(scores)/sizeof(scores[0]);
   printAverage(scores, n);
   return 0;
}
```

Explanation

- printAverage(int arr[], int n) → Function me array (size omit) aur uska size pass hota hai.
- Loop se array ke sab elements ka sum calculate kiya.
- Average = sum / n → decimal ke liye (double) cast kiya.
- cout se average print kiya.
- main() me array scores[] declare kiya, size nikal ke function call kiya.

auto with initializer
Question: Grocery items ke prices ka list create karo aur total cost calculate karo. Constraints:

auto prices = {…}; use karo. Range-based for loop ka use karke total sum nikaalo.

```
#include <iostream>
using namespace std;
int main() {
   auto prices = {50, 100, 75, 120};  // initializer list
   int total = 0;
   for(auto price : prices)
      total += price;
   cout << "Total cost of groceries: " << total << endl;
   return 0;
}
```

Explanation

- auto prices = {50, 100, 75, 120};
- Compiler automatically type deduce karta hai → std::initializer_list<int>.
- int total = 0;
- Total cost store karne ke liye variable initialize kiya.
- for(auto price : prices)
- Range-based loop se array ke har element ko iterate kiya.
- Har price ko total me add kiya (total += price;).
- cout << "Total cost of groceries: " << total << endl;
- Final total cost screen par print kiya.

std::vector

Question: User se scores input lo jab tak user -1 enter na kare. Phir maximum score print karo. Constraints: vector<int> use karo taaki dynamic size handle ho. Loop me input lo aur vector me push_back karo. Maximum find karke print karo.

Explanation

- vector<int> scores;
- Dynamic array banaya jisme user ke scores store honge.
- Size declare karne ki zarurat nahi, vector automatically grow hota hai.
- while(cin >> input && input != -1)
- User se scores input lete raho.
- Jab user -1 enter kare, input loop stop ho jata hai.
- scores.push_back(input);
- Har valid input ko vector me add kiya jata hai.
- if(scores.empty())
- Check kiya ki vector me koi score enter hua ya nahi.

- Agar empty hai → "No scores entered" print karo.

std::array

Question: 5 students ke marks store karo aur highest marks wala student find karo.
Constraints: std::array<int,5> use karo. Loop se maximum marks nikaalo. Student index ke saath print karo.

```
#include <iostream>
#include <array>
using namespace std;
int main() {
   array<int, 5> marks = {85, 90, 78, 92, 88};
   int maxMarks = marks[0];
   int studentIndex = 0;

   for(int i = 1; i < marks.size(); i++) {
      if(marks[i] > maxMarks) {
         maxMarks = marks[i];
         studentIndex = i;
      }
   }
   cout << "Highest marks: " << maxMarks
      << " by student " << studentIndex + 1 << endl;
   return 0;
}
```

Explanation

- array<int, 5> marks = {85, 90, 78, 92, 88}; → Fixed-size array of 5 students' marks.
- maxMarks = marks[0]; studentIndex = 0; → Initial maximum marks aur student index set kiya.
- Loop for(int i = 1; i < marks.size(); i++) → Array ke elements check kiye.
- Agar marks[i] > maxMarks → update maxMarks aur studentIndex.
- cout → Highest marks aur student number print kiya (studentIndex + 1 for human-readable).

Static Array

- Size fixed hota hai compile time par.
  - Example: int arr[5]; → hamesha 5 elements ke liye allocate hota hai.
- Memory stack me allocate hoti hai.
- Size change nahi kar sakte runtime me.

- Fast access hota hai because stack memory fast hoti hai.
- Syntax:
- int arr[5] = {1,2,3,4,5};

Dynamic Array

- Size runtime me decide hota hai.
    - Example: User se input le ke array create karna.
- Memory heap me allocate hoti hai using new or vector.
- Size change kar sakte ho (especially std::vector).
- Thoda slower access compared to static array (heap memory).
- Syntax:
- int* arr = new int[n];   // Dynamic array

vector<int> vec;   // Saaf aur easy dynamic array

Static Array

```
#include <iostream>
using namespace std;
int main() {
    // Static array with fixed size
    int arr[5] = {10, 20, 30, 40, 50};
    cout << "Static Array Elements: ";
    for(int i = 0; i < 5; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    // Size of static array
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Size of static array: " << n << endl;
    return 0;
}
```

```
Output
Static Array Elements: 10
20 30 40 50
Size of static array: 5
```

Explanation:

- Size fixed hai → 5 elements.
- Memory stack me allocate hoti hai.
- Access fast hota hai.

Dynamic Array

```
#include <iostream>
using namespace std;
int main() {
    int n;
    cout << "Enter size of dynamic array: ";
    cin >> n;
    // Dynamic array on heap
    int* arr = new int[n];
    // Input elements
    cout << "Enter " << n << " elements: ";
    for(int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    // Print elements
    cout << "Dynamic Array Elements: ";
    for(int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    // Free memory
    delete[] arr;
    return 0;
}
```

Sample Input/Output:

```
Enter size of dynamic array: 4
Enter 4 elements: 5 10 15 20
Dynamic Array Elements: 5 10
15 20
```

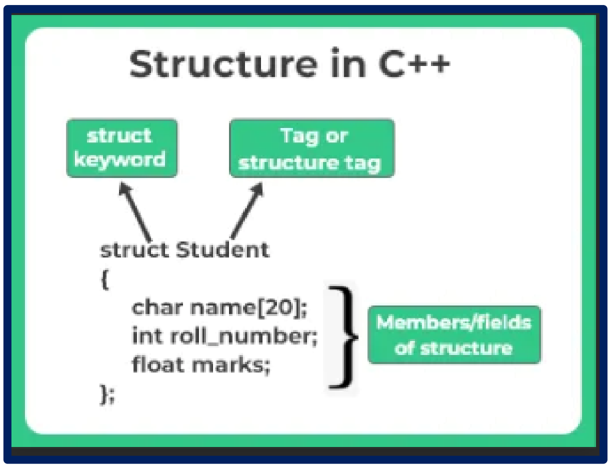
Explanation:

- Size runtime par decide hota hai (user input).
- Memory heap me allocate hoti hai.
- delete[] arr; → Memory free karna zaruri hai.

Structure

Definition:
Structure ek user-defined data type hai jo different types ke variables ko ek single unit me group karne ke liye use hota hai.

- Matlab, agar aapko ek entity ke multiple attributes ek saath store karne hain, toh structure best hai.
- Ek object ke jaise kaam karta hai lekin simple aur lightweight hai.



```
struct Student
{
    char name[20];
    int roll_number;
    float marks;
};
```

Syntax of Structure

```
struct Student {
    int rollNo;
    string name;
    float marks;
};
```

- struct Student → naya data type Student define kiya.
- rollNo, name, marks → members (fields) of the structure.
- Har member different type ka ho sakta hai.

Structure Variable Declare Karna

```
Student s1;  // ek student object
s1.rollNo = 101;
s1.name = "Rahul";
s1.marks = 88.5;
```

- s1 → Student structure ka instance.
- Dot operator (.) se members ko access ya update karte hain.

Structure Initialization

```
Student s2 = {102, "Anjali", 92.0};
```

- Initialization ke time order important hai: first rollNo, phir name, phir marks.

Array of Structures

- Agar multiple students ke data store karne hain:

```
Student students[3] = {
    {101, "Rahul", 88.5},
    {102, "Anjali", 92.0},
    {103, "Sanya", 79.5}
};
```

- Access karne ke liye loop:

```
for(int i = 0; i < 3; i++) {
    cout << students[i].name << " scored " << students[i].marks << endl;
}
```

Structure ke Fayde

- Related data ko ek unit me group karna easy hota hai.
- Different types ke data ek saath store kiye ja sakte hain.
- Code organized aur readable banta hai.
- Functions me structure ko pass by value ya reference kiya ja sakta hai.

Structure in Function

Function me structure pass kar sakte hain:

```
void printStudent(Student s) {
cout << s.rollNo << " " << s.name << " " << s.marks << endl;
}
```

Or pointer se pass karke memory efficient bhi bana sakte hain:

```
void printStudent(Student* s) {
cout << s->rollNo << " " << s->name << " " << s->marks << endl;
}
```

Ques 1: Student Record Management
Problem:
Ek program banao jo 3 students ka record store aur print kare.
Requirements:

1. Student ka roll number, name aur marks store karna hai.
2. struct ka use karke student ka custom data type banao.
3. User se roll number, name aur marks input lo.
4. Sabhi students ka data print karo in neat format.
5. Bonus: Highest marks wala student print karo.

```
#include <iostream>
using namespace std;
// Structure definition
struct Student {
   int rollNo;
   string name;
   float marks;
};
int main() {
   Student students[3];  // Array of 3 students
   // Input student details
   for(int i = 0; i < 3; i++) {
      cout << "Enter details for student " << i + 1 << ":\n";
      cout << "Roll No: ";
      cin >> students[i].rollNo;
      cin.ignore();  // Clear newline from buffer
      cout << "Name: ";
      getline(cin, students[i].name);
      cout << "Marks: ";
      cin >> students[i].marks;
      cout << endl;
   }
   // Print all student details
   cout << "\nStudent Records:\n";
```

```
    cout << "RollNo\tName\tMarks\n";
    for(int i = 0; i < 3; i++) {
        cout << students[i].rollNo << "\t"
             << students[i].name << "\t"
             << students[i].marks << endl;
    }
    // Find highest marks
    float maxMarks = students[0].marks;
    int topStudentIndex = 0;
    for(int i = 1; i < 3; i++) {
        if(students[i].marks > maxMarks) {
            maxMarks = students[i].marks;
            topStudentIndex = i;
        }
    }

    cout << "\nTopper: " << students[topStudentIndex].name
         << " with marks " << students[topStudentIndex].marks << endl;

    return 0;
}
```

Explanation

- struct Student → Roll number, name aur marks group karta hai.
- students[3] → Array of 3 student objects.
- Loop me user input liya for each student.
- Dot operator (.) se members access kiye.
- Print loop → sabhi students ka data neatly print kiya.
- Second loop → highest marks wala student find kiya aur print kiya.

Employee Record Management
Problem:
Ek program banao jo employees ka record store kare aur print kare.
Struct Members:

- int empId;
- string name;
- float salary;

```
#include <iostream>
using namespace std;
struct Employee {
    int empId;
    string name;
    float salary;
};
int main() {
    Employee emp[2];

    for(int i=0; i<2; i++){
        cout << "Enter Employee ID: ";
        cin >> emp[i].empId;
        cin.ignore();
        cout << "Enter Name: ";
        getline(cin, emp[i].name);
        cout << "Enter Salary: ";
        cin >> emp[i].salary;
    }
    cout << "\nEmployee Records:\n";
    for(int i=0; i<2; i++){
        cout << emp[i].empId << " " << emp[i].name << " " << emp[i].salary << endl;
    }
    return 0;
}
```

Explanation

- Ye program 2 employees ka data store karta hai
- struct Employee employee ki ID, name aur salary rakhta hai
- Loop se user se employee ka data input liya jata hai
- Dusra loop employee records ko display karta hai
- Program end mein successfully terminate ho jata hai

Enum kya hota hai?

- Enum (Enumeration) ek user-defined data type hota hai
- Iska use hum fixed values ka group banane ke liye karte hain
- Jab values limited aur predefined hoti hain, tab enum best hota hai

Enum kyu use karte hain?

- Code readable aur easy to understand ban jata hai
- Magic numbers (jaise 0,1,2) use karne se bachate hain
- Errors kam hote hain
- Fixed options ke liye perfect hota hai

Syntax

```
enum EnumName {
   value1,
   value2,
   value3
};
```

Difference Between enum and struct

Enum	Structure (struct)
enum ek user-defined data type hai	struct bhi user-defined data type hai
Fixed named constants ka group hota hai	Different data types ka collection hota hai
Sirf limited predefined values allow karta hai	Multiple aur different values store karta hai
Mostly decision making ke liye use hota hai	Mostly record / data storage ke liye use hota hai
Enum ke members usually integers hote hain	Struct ke members alag-alag data types ke ho sakte hain
Example: days, months, status	Example: employee, student, book
Memory sirf ek value ke liye allocate hoti hai	Memory sab members ke liye allocate hoti hai
Data change nahi hota (fixed set)	Data runtime pe change ho sakta hai

Ques 1:- Scenario: Tum ek traffic management system design kar rahe ho.
C++ mein enum use karke traffic light ke colors ko represent karo.
Tumhara program ye karna chahiye:

1. TrafficLight naam ka enum define karo jisme Red, Yellow, aur Green ho.
2. Enum type ka ek variable declare karo.
3. Variable ko koi color assign karo.
4. Conditional statements (if-else ya switch) ka use karke har color ke liye message print karo:
   - Red → "Stop"
   - Yellow → "Get Ready"
   - Green → "Go"

```
#include <iostream>
using namespace std;
// Enum define kiya
enum TrafficLight {
   Red,
   Yellow,
   Green
};
int main() {
   // Enum variable declare aur value assign ki
   TrafficLight signal = Red;
   // Conditional statements se message print kiya
   if(signal == Red) {
      cout << "Red Light: Stop";
   }
   else if(signal == Yellow) {
      cout << "Yellow Light: Get Ready";
   }
   else if(signal == Green) {
      cout << "Green Light: Go";
   }
   return 0;
}
```

Explanation

- enum TrafficLight → Traffic light ke fixed colors define karta hai
- signal variable enum type ka hai aur usko Red assign kiya
- if-else se check kiya kaunsa signal active hai aur message print kiya

Ques 2:- Tum ek e-commerce website ke liye order status manage kar rahe ho.
Enum ka use karke order ke status ko represent karo: Placed, Shipped, OutForDelivery, Delivered.
Program user ko status ke according message print kare.

```
#include <iostream>
using namespace std;
// Enum define kiya
enum OrderStatus {
   Placed,
   Shipped,
   OutForDelivery,
   Delivered
};
int main() {
   // Enum variable declare aur value assign ki
   OrderStatus order = Shipped;
   // Conditional statements se message print kiya
   if(order == Placed) {
       cout << "Your order has been placed successfully.";
   }
   else if(order == Shipped) {
       cout << "Your order is on the way!";
   }
   else if(order == OutForDelivery) {
       cout << "Your order is out for delivery.";
   }
   else if(order == Delivered) {
       cout << "Your order has been delivered.";
   }
   return 0;
}
```

Explanation

- enum OrderStatus → order ke fixed statuses define karta hai
- order variable enum type ka hai aur Shipped assign kiya
- if-else se current order status check karke message print kiya

Enum in a Switch Statement

```
#include <iostream>
using namespace std;
enum Level {
  LOW = 1,
  MEDIUM,
  HIGH
};

int main() {
  enum Level myVar = MEDIUM;
  switch (myVar) {
   case 1:
     cout << "Low Level";
     break;
   case 2:
     cout << "Medium level";
     break;
   case 3:
     cout << "High level";
     break;
 }
 return 0;
}
```

Explanation

- #include <iostream> → Screen pe print karne ke liye.
- using namespace std; → std:: likhne ki zarurat nahi.
- enum Level { LOW=1, MEDIUM, HIGH };
- Level ek enum type hai.
- LOW = 1, MEDIUM = 2, HIGH = 3.
- Enum ka use fixed choices ke liye hota hai.
- enum Level myVar = MEDIUM;
- myVar enum variable hai.
- Isme MEDIUM assign kiya (value = 2).
- switch(myVar)
- Dekhta hai myVar ki value kya hai.
- Agar 1 → "Low Level"
- Agar 2 → "Medium level" (yeh print hoga)
- Agar 3 → "High level"
- cout → Message screen pe print karega.
- return 0; → Program khatam ho gaya.

Reference

Reference ek alias (nickname) hota hai kisi existing variable ka.

- Matlab, reference ke through original variable ko directly access aur modify kar sakte ho.

- Reference memory mein original variable ka address hold karta hai, lekin alagalag memory allocate nahi hoti.

Syntax

datatype &refVar = originalVar;

- & → reference symbol
- refVar → reference variable (alias)
- originalVar → jis variable ka reference banaya ja raha hai

Example:

int a = 10;
int &ref = a; // ref ab a ka alias hai

- Ab ref aur a same memory location ko point karte hain.

Important Points about References

1. Reference hamesha initialize karna padta hai.
2. int &r; // Error, initialize nahi kiya
3. Reference ko dusre variable se baad mein change nahi kar sakte.
4. Reference ka use mostly function arguments aur return values mein hota hai.
5. Reference original variable ko modify kar sakta hai.

Why We Used Reference

- Original variable ko directly modify karne ke liye
- Function arguments efficiently pass karne ke liye (pass by reference)
- Function se reference return karne ke liye
- Syntax simple aur clean rakhne ke liye
- Memory efficient (copy create nahi hoti)

Types Of Reference

Type of Reference	Description	Syntax / Example	Use Case
Lvalue Reference	Normal reference, existing variable ko refer karta hai	int a = 10; int &ref = a;	Original variable ko modify karne ke liye
Rvalue Reference	Temporary values ya rvalues ko refer karta hai (C++11 onwards)	int &&rref = 5;	Move semantics, efficient resource handling
Const Reference	Value ko modify nahi kar sakte, read-only access	const int &ref = a;	Large objects ko copy kiye bina read-only access
Function Reference (Pass by	Function arguments ko reference ke through pass	void increment(int &x) { x++; }	Original variable function ke andar modify

Type of Reference	Description	Syntax / Example	Use Case
Reference)	karta hai		karne ke liye
Reference to Pointer	Pointer ka reference, pointer ko modify kar sakte hain	int *ptr = &a; int *&refPtr = ptr;	Pointer manipulation without copying

Lvalue Reference

Bank account balance ko update karna, jahan original balance change ho. Coder

```
#include <iostream>
using namespace std;
// Function to deposit amount using reference
void deposit(double &balance, double amount) {
   balance += amount; // original balance update ho raha hai
}
// Function to withdraw amount using reference
void withdraw(double &balance, double amount) {
   if(amount <= balance)
      balance -= amount;
   else
      cout << "Insufficient balance!" << endl;
}
int main() {
   double accountBalance = 5000.0; // Original balance
   cout << "Initial Balance: " << accountBalance << endl;
   deposit(accountBalance, 2000); // Deposit 2000
   cout << "After Deposit: " << accountBalance << endl;
   withdraw(accountBalance, 1500); // Withdraw 1500
   cout << "After Withdrawal: " << accountBalance << endl;
   return 0;
}
```

Output
```
Initial Balance: 5000
After Deposit: 7000
After Withdrawal: 5500
```

Explanation

- double &balance → Reference hai, original accountBalance ko point karta hai
- deposit aur withdraw function me balance directly update ho raha hai
- Extra copy create nahi hoti → memory efficient

Rvalue Reference

Large data transfer, jaise image processing me temporary buffer handle karna.

```
#include <iostream>
#include <vector>
using namespace std;
// Function to process image using Rvalue Reference
void processImage(vector<int> &&imageBuffer) {
    cout << "Processing image of size: " << imageBuffer.size() << endl;
    // Example: simple manipulation
    for (auto &pixel : imageBuffer) {
        pixel = pixel / 2; // dummy processing
    }
    cout << "Processing done!" << endl;
}
int main() {
    // Large image buffer (temporary)
    processImage(vector<int>(1000000, 255)); // 1 million pixels with value 255
    return 0;
}
```

Explanation

- vector<int> &&imageBuffer → Rvalue reference hai, temporary image data ko handle karta hai
- vector<int>(1000000, 255) → 1 million pixels ka temporary buffer create hua
- Rvalue reference se copy avoid hoti hai, direct buffer process hota hai → fast aur memory efficient
- Yeh technique image processing, large file handling, or move semantics me use hoti hai

Const Reference

Reading user data from database without changing it.

```
#include <iostream>
#include <string>
using namespace std;
// Function to display user data (read-only)
```

```
void displayUserData(const string &username, const int &age) {
    cout << "Username: " << username << endl;
    cout << "Age: " << age << endl;

    // username = "NewName"; // Error! Cannot modify const reference
    // age = 25; // Error! Cannot modify const reference
}
int main() {
    string name = "Rahul";
    int age = 25;
    // Passing variables as const reference
    displayUserData(name, age);
    return 0;
}
```

Explanation

- const string &username aur const int &age → const references, read-only access
- Function ke andar original variables change nahi ho sakte
- Efficient hai → large objects copy nahi hote
- Real-life use: Reading data from database, logs, or configuration files without modifying original data

Function Reference (Pass by Reference)

```
#include <iostream>
using namespace std;
// Function to update item quantity
void updateQuantity(int &quantity, int addItems) {
    quantity += addItems; // Original variable update hota hai
}
int main() {
    int itemQuantity = 5; // Initial quantity

    cout << "Initial Quantity: " << itemQuantity << endl;
    updateQuantity(itemQuantity, 3); // Add 3 items
    cout << "After Adding 3 Items: " << itemQuantity << endl;
    updateQuantity(itemQuantity, -2); // Remove 2 items
    cout << "After Removing 2 Items: " << itemQuantity << endl;
    return 0;
}
```

Explanation

- int &quantity → Reference hai, original itemQuantity ko point karta hai
- updateQuantity function ke through directly quantity update ho rahi hai
- Pass by reference se extra copy create nahi hoti → memory efficient

Reference to Pointer

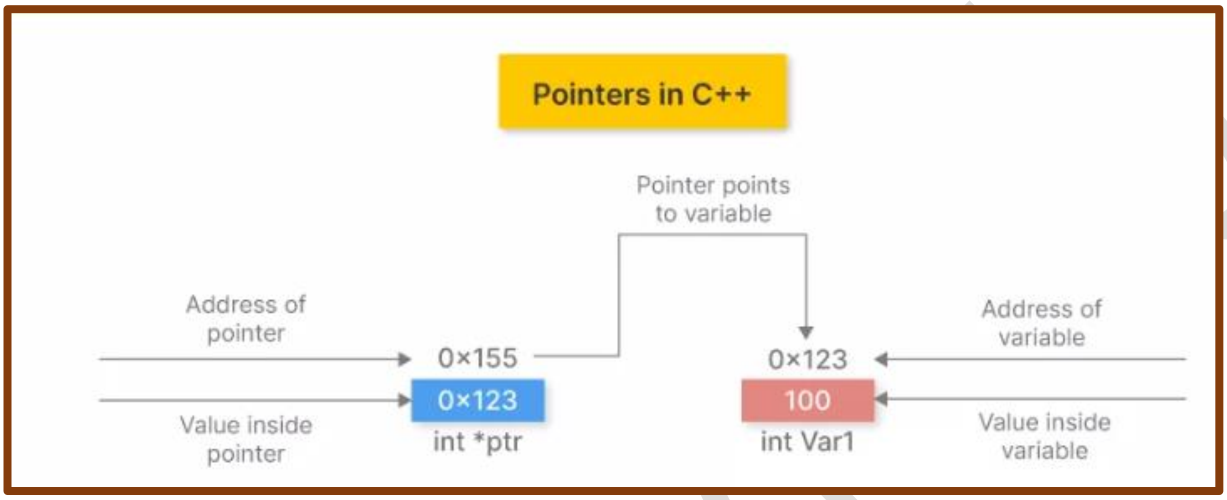
```
#include <iostream>
using namespace std;
// Function to allocate new memory and update pointer
void allocateMemory(int *&ptr, int size) { // Reference to pointer
   ptr = new int[size]; // Pointer update ho raha hai
   for(int i = 0; i < size; i++) {
      ptr[i] = i + 1; // Initialize array
   }
}
int main() {
   int *arr = nullptr; // Initial pointer
   cout << "Allocating memory..." << endl;
   allocateMemory(arr, 5); // Allocate array of size 5
   cout << "Array elements: ";
   for(int i = 0; i < 5; i++) {
      cout << arr[i] << " ";
   }
   cout << endl;
   // Free the allocated memory
   delete[] arr;
   return 0;
}
```

Explanation

- int *&ptr → Reference to pointer, original pointer ko modify kar sakte hain
- allocateMemory function ke through pointer ko new memory allocate karke update kiya
- Original pointer arr ab new memory block ko point karta hai
- delete[] arr; → Memory free karna, memory leak avoid karne ke liye

Pointer kya hai?

- Pointer ek variable hai jo dusre variable ka memory address store karta hai.
- Matlab pointer yeh batata hai ki variable memory me kahan stored hai.
- Pointer ke through hum original variable ko access aur modify kar sakte hain.



Syntax

datatype *pointerName;
datatype → pointer jis type ke variable ka address store karega
* → pointer declare karne ke liye
pointerName → pointer ka naam
Example:
int a = 10;
int *ptr = &a; // ptr me a ka address store hua

Pointer Operators

1. **Address-of Operator &**
   - Variable ka **memory address** dene ke liye
   - int a = 5;
   - int *ptr = &a; // ptr me a ka address
2. **Dereference Operator ***
   - Pointer ke through variable ki value access karne ke liye
   - cout << *ptr; // prints 5
   - *ptr = 20; // a ki value change ho gayi

Types Of Pointers

Pointer Type	Description	Syntax / Example	Use Case / Real-Life Example
Null Pointer	Pointer jo kisi variable ka address store nahi karta	int *ptr = nullptr;	Initialize karne ke liye, accidental access se bachane ke liye
Void Pointer	Generic pointer, kisi bhi data type ka address store kar sakta hai	void *ptr; int a = 10; ptr = &a;	Generic functions me data type ke bina pointer pass karna
Dangling Pointer	Pointer jo delete ya free hone ke baad memory point karta hai	int *ptr = new int(5); delete ptr;	Avoid karna, memory release ke baad use nahi karna
Wild Pointer	Pointer jo garbage memory address point karta hai	int *ptr; // uninitialized	Initialize karna nullptr ya valid address se
Pointer to Pointer	Pointer jo dusre pointer ka address store karta hai	int a = 10; int *ptr = &a; int **pptr = &ptr;	Dynamic 2D arrays, complex data structures
Function Pointer	Pointer jo function ka address store karta hai	void greet(){}; void (*ptr)() = greet; ptr();	Callbacks, dynamic function calls
Const Pointer / Pointer to Const	Pointer ya value ko modify nahi kar sakte	const int *ptr1 = &a; int *const ptr2 = &a;	Read-only access ya pointer fixed rakhna

Null Pointer

- **Bank account pointer initialize karna, jab account abhi create nahi hua ho.**

```
#include <iostream>
using namespace std;
int main() {
   // Null pointer example - bank account pointer initialize
   int *accountPtr = nullptr; // abhi koi account assigned nahi
   // Check if account exists
   if(accountPtr == nullptr) {
      cout << "No bank account exists yet." << endl;
   }
   // Ab account create karte hain
   int accountBalance = 5000;
   accountPtr = &accountBalance; // pointer ab account ko point karega
   cout << "Bank account created." << endl;
```

Output
```
No bank account exists
yet.
Bank account created.
Account balance: 5000
```

```
    cout << "Account balance: " << *accountPtr << endl;
    // Pointer ke through balance update karna
    *accountPtr += 2000; // deposit
    cout << "After  deposit, account balance: " << *accountPtr << endl;
    return 0;
}
```

Explanation

- int *accountPtr = nullptr; → Null pointer, abhi account assigned nahi hai
- if(accountPtr == nullptr) → Check karta hai ki pointer valid hai ya nahi
- accountPtr = &accountBalance; → Pointer ab original variable ko point karta hai
- *accountPtr → Pointer ke through variable ki value access aur modify kar sakte hain.

Void Pointer

- Function me alag-alag type ke user data (age, salary) print karna.

```
#include <iostream>
using namespace std;
// Function to print different types of data using void pointer
void printData(void *data, char type) {
    if(type == 'i') {
        cout << "Integer value: " << *((int*)data) << endl;
    }
    else if(type == 'f') {
        cout << "Float value: " << *((float*)data) << endl;
    }
    else {
        cout << "Unknown type!" << endl;
    }
}
int main() {
    int age = 25;          // User age
    float salary = 50000.5; // User salary
    // Passing data to function via void pointer
    printData(&age, 'i');      // Print integer
    printData(&salary, 'f');  // Print float
    return 0;
}
```

```
Output
Integer value: 25
Float value: 50000.5
```

Explanation

- void *data → Generic pointer, kisi bhi type ka address store kar sakta hai
- *((int*)data) → Void pointer ko integer pointer me cast karke value read karna
- printData(&age, 'i') → Function me address aur type pass kiya
- Function ke through different types ke data ko print kiya

Dangling Pointer

- Memory free hone ke baad pointer use karna, jaise deleted file ka pointer.

```
#include <iostream>
using namespace std;
int main() {
    // Dangling pointer example - deleted file pointer
    int *fileData = new int(100); // Memory allocate (like creating file data)
    cout << "File data before delete: " << *fileData << endl;
    // Memory free karna (file delete karna)
    delete fileData;
    // Dangling pointer - ab fileData invalid ho gaya
    // cout << *fileData << endl; // Dangerous! Accessing dangling pointer
    // Safe practice - pointer ko nullptr set karo
    fileData = nullptr;
    if(fileData == nullptr) {
        cout << "Pointer is now safe (nullptr) after deleting memory." << endl;
    }
    return 0;
}
```

Explanation

- int *fileData = new int(100); → Memory allocate kiya (jaise file create karna)
- delete fileData; → Memory free kiya (file delete)
- Ab pointer dangling ho gaya, direct access karna crash kar sakta hai
- fileData = nullptr; → Pointer safe banaya

Wild Pointer

- **Uninitialized pointer ko access karna, jaise unknown location ko access karna.**

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

int main() {
    // Wild Pointer example - uninitialized pointer
    int *ptr; // Wild pointer (garbage address)

    // Dangerous! Accessing wild pointer can crash the program
    // *ptr = 10; // Uncommenting this line may cause runtime error

    // Safe practice: pointer ko nullptr se initialize karo
    ptr = nullptr;

    if(ptr == nullptr) {
        cout << "Pointer is safe now (nullptr), no unknown memory access." << endl;
    }

    return 0;
}
```

Explanation

- int *ptr; → Wild pointer, random garbage address.
- *ptr = 10; → Dangerous, program crash ho sakta hai.
- ptr = nullptr; → Pointer ko safe initialize kiya.
- if(ptr == nullptr) → Pointer check kiya, ab safe hai.

Pointer to Pointer (int **p)

- Imagine ek remote (pointer) jo TV ko control karta hai.
- Ab ek super-remote (pointer to pointer) hai jo pehle remote ko control karta hai.
- Matlab: Super-remote → normal remote → TV.

```
#include <iostream>
using namespace std;
int main() {
    int tvChannel = 5;          // TV ka channel (normal variable)
    int *remote = &tvChannel;   // Normal remote (pointer) jo TV ko control karta hai
    int **superRemote = &remote; // Super-remote (pointer to pointer) jo normal remote ko
control karta hai
    cout << "TV channel via normal remote: " << *remote << endl;       // 5
    cout << "TV channel via super remote: " << **superRemote << endl; // 5
    // Change TV channel using normal remote
    *remote = 10;
    cout << "After changing via normal remote: " << tvChannel << endl; // 10
    // Change TV channel using super remote
    **superRemote = 15;
    cout << "After changing via super remote: " << tvChannel << endl; // 15
    // Change remote to point to another channel
    int newChannel = 20;
    *superRemote = &newChannel; // Super-remote ab naye channel ko point karega
    cout << "TV channel via super remote now: " << **superRemote << endl; // 20
    return 0;
}
```

Explanation

- tvChannel → TV ka current channel.
- remote → TV ko control karta hai.
- superRemote → Remote ko control karta hai.
- *remote ya **superRemote se hum actual TV channel change kar sakte hain.
- Pointer to pointer matlab super-remote → remote → TV.

Function Pointer

- Imagine ek manager (function pointer) jo decide karta hai ki kaunsa employee (function) kaam karega.
- Manager ko call karte hi wo decide karta hai kaunsa function execute hoga.

```
#include <iostream>
using namespace std;
// Simple function
void greet() {
    cout << "Hello!" << endl;
}
int main() {
    // Function pointer declaration and initialization
    void (*funcPtr)() = &greet;
    // Calling function using function pointer
    funcPtr(); // Output: Hello!
    // Function pointer can also be assigned again (if multiple functions exist)
    // Example: void (*funcPtr2)() = &anotherFunction;
    return 0;
}
```

Explanation

- greet() → Normal function jo "Hello!" print karta hai.
- void (*funcPtr)() = &greet; → funcPtr ek pointer hai jo greet function ko point karta hai.
- funcPtr(); → Function pointer ke through greet() call karna.

- Real-life analogy: Manager (funcPtr) decide karta hai kaunsa employee (function) kaam karega.

Const Pointer / Pointer to Const
Pointer to const: Tum kisi ki diary padh sakte ho (read-only), likh nahi sakte.
Const pointer: Tum hamesha ek hi diary ko hold karoge, par usme likh sakte ho.

```
#include <iostream>
using namespace std;
int main() {
   int x = 5, y = 10;
   const int *p = &x; // p pointer se value change nahi kar sakte
   cout << "Initial value via pointer to const: " << *p << endl; // 5
   p = &y; // allowed, pointer ab y ko point karega
   cout << "Value via pointer to const after pointing to y: " << *p << endl; // 10

   // *p = 20; // NOT allowed, value change nahi kar sakte
   // ---------------------------
   // Const pointer
   // ---------------------------
   int * const cp = &x; // cp hamesha x ko point karega, lekin value change ho sakti
   cout << "Initial value via const pointer: " << *cp << endl; // 5
   *cp = 20; // allowed, value change ho gayi
   cout << "Value via const pointer after changing: " << *cp << endl; // 20
   // cp = &y; // NOT allowed, pointer ko point nahi kar sakte
   return 0;
}
```

Explanation
Pointer to const (const int *p)
- Value ko change nahi kar sakte (*p = ... not allowed).
- Pointer ko dusre address pe point karna allowed hai (p = &y;).
- Analogy: Kisi ki diary padh sakte ho (read-only), likh nahi sakte.

2. Const pointer (int * const cp)
   - Pointer ko dusre address pe move nahi kar sakte (cp = &y; not allowed).
   - Value ko change karna allowed hai (*cp = ...).
   - Analogy: Hamesha ek diary ko hold karoge, par usme likh sakte ho.

Memory Address
Memory Address ek unique number hai jo computer ki memory location ko identify karta

hai.

- Har variable, object, ya array ka apna ek memory address hota hai.
- Isse data computer ke RAM me kahan stored hai pata chalta hai.

Address-of Operator &

- C++ me & operator ka use karke variable ka memory address le sakte ho.

Syntax / Example:

```
int a = 10;
cout << &a; // a ka memory address print hoga
```

Ques 1:- Ek integer variable num create karo. Uska memory address pointer ke through print karo aur pointer se uski value modify karo.

```
#include <iostream>
using namespace std;
int main() {
   int num = 10; // Integer variable
   int *ptr = &num; // Pointer me num ka address store
   cout << "Original value of num: " << num << endl;
   cout << "Memory address of num: " << ptr << endl; // pointer ke through address print
   // Pointer ke through value modify karna
   *ptr = 25;
   cout << "Modified value of num using pointer: " << num << endl;
   return 0;
}
```

C++ new and delete keyword

new keyword

- Purpose: Dynamic memory allocate karne ke liye, runtime pe.
- Jab aapko pata nahi kitna memory chahiye compile time pe, tab new use karo.
- new automatically memory heap se allocate karta hai.

```
#include <iostream>
using namespace std;
int main() {
    // Single integer ke liye dynamic memory allocate karna
    int *ptr = new int;   // int ke liye memory allocate ki
    *ptr = 42;            // value store ki
    cout << "Value stored in dynamically allocated memory: " << *ptr << endl; // Output: 42
    // Array ke liye dynamic memory
    int *arr = new int[3]; // 3 integers ke liye memory allocate ki
    arr[0] = 10;
    arr[1] = 20;
    arr[2] = 30;
    cout << "Values in dynamically allocated array: ";
    for(int i = 0; i < 3; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Explanation

- new int → Heap me ek integer ke liye memory allocate ki
- new int[3] → Heap me array allocate ki
- Value assign kar sakte ho jaise normal variables me

delete keyword

- **Purpose:** Dynamic memory free karne ke liye, jo new se allocate hui ho.
- delete use karna zaroori hai, nahi toh memory leak ho jayega.

```
#include <iostream>
using namespace std;
int main() {
    // Dynamic memory allocate karna
    int *ptr = new int;   // int ke liye memory allocate ki
    *ptr = 50;
    cout << "Value before deleting: " << *ptr << endl; // Output: 50
    // Memory free karna
    delete ptr;  // single variable ke liye
    // Array ke liye memory free karna
    int *arr = new int[3]; // dynamic array
    arr[0] = 1; arr[1] = 2; arr[2] = 3;
    cout << "Array values: ";
    for(int i = 0; i < 3; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    delete[] arr; // array ke liye
    return 0;
}
```

Explanation

- delete ptr; → Single variable ki memory free ki
- delete[] arr; → Array ki memory free ki
- Memory leak avoid karne ke liye delete zaroor use karo

Function kya hai

- Function ek block of code hai jo ek specific task perform karta hai.
- Ek tarah se mini-program jise baar-baar call kar sakte ho.

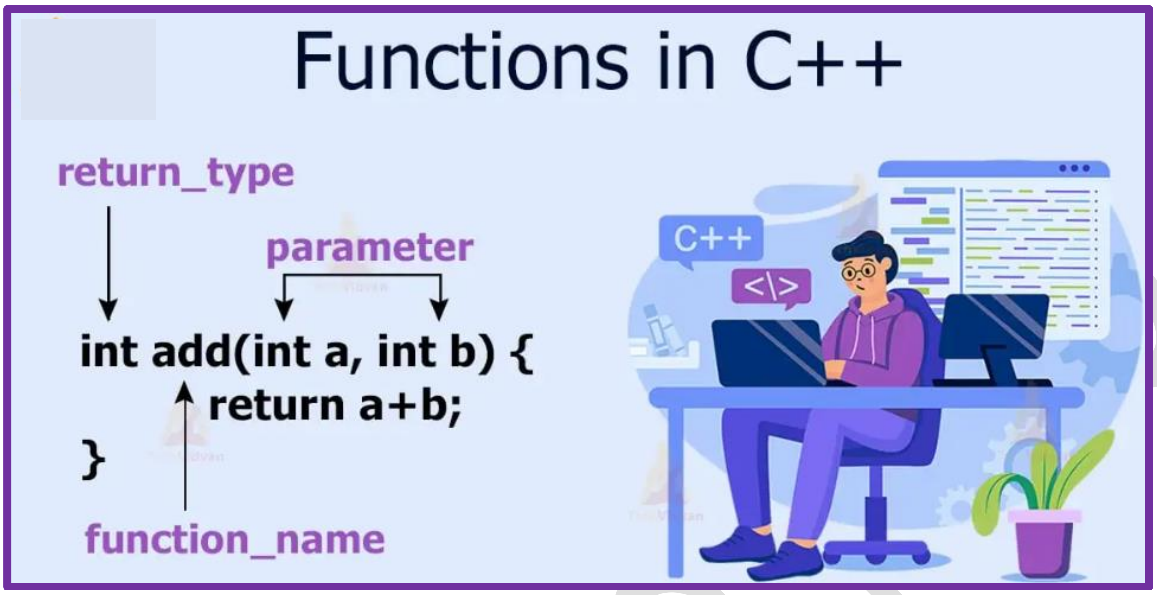
Benefits of using functions

- **Code Reusability:** Ek function ko multiple jagah call kar sakte ho.
- **Modularity:** Program ko small parts me divide karna easy ho jata hai.
- **Readability:** Program readable aur manageable ban jata hai.
- **Debugging easy:** Agar error hai, sirf function check karo.

Function Syntax

```
return_type function_name(parameters) {
    // function body
}
```

- return_type → function kya return karega (int, void, etc.)
- function_name → unique identifier
- parameters → input values jo function ko diye jate hain



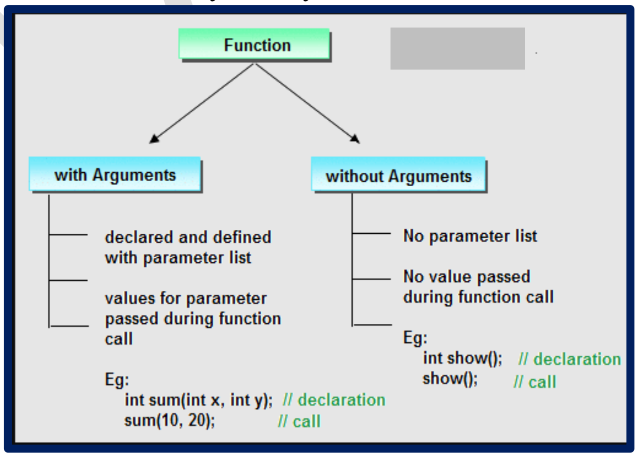
```
int add(int a, int b) {
    return a+b;
}
```

Parameter

- Definition: Function ke definition me jo variable likha jata hai, use parameter kehte hain.
- Ye function ke input ko hold karta hai.
- Parameter = "Box jo function ke andar item receive karne ke liye ready rakha hai."

Argument

- Definition: Function call karte waqt jo actual value pass karte hain, use argument kehte hain.
- Ye function me parameter ke through use hoti hai.
- Argument = "Item jo box (parameter) me dala ja raha hai."


Function
with Arguments
without Arguments
declared and defined with parameter list
values for parameter passed during function call
No parameter list
No value passed during function call
Eg:
Eg:

```
int sum(int x, int y);  // declaration
sum(10, 20);            // call
```

```
int show();   // declaration
show();       // call
```

Function Declaration (Prototype) kya hai?

- Definition:
  Function declaration ek statement hai jo compiler ko bataata hai ki program me ye function exist karta hai, lekin body abhi define nahi ki gayi.
- Purpose: Compiler ko pehle pata chal jata hai ki function ka name, return type, aur parameters kya hai.
- Function declaration = "Announcement ki ek function aayega, lekin kaam abhi bataya nahi."

Syntax

return_type function_name(parameter_list);

- Microwave oven me "Cook" button ek function jaisa hai.
- Button press karo → specific task (food cook karna) complete hota **hai.write code**

```
#include <iostream>
using namespace std;
// Function definition
void cookFood() {
    cout << "Cooking food..." << endl;
    cout << "Food is ready!" << endl;
}
int main() {
    char button;
    cout << "Press 'c' to Cook: ";
    cin >> button;
    if (button == 'c' || button == 'C') {
        // Function call
        cookFood();
    } else {
        cout << "Invalid button!" << endl;
    }
    return 0;
}
```

Explanation:
1. cookFood() function ek specific task perform karta hai — yahan food cook karna simulate karte hain.
2. main() me user se input lete hain — jaise "Cook" button press karna.
3. Agar input c hai, to cookFood() call hota hai aur task complete hota hai.

Function Definition kya hoti hai?
Function definition wo jagah hoti hai jahan:
- Function ka naam
- Return type
- Parameters
- Aur function ke andar ka code

likha hota hai.

Example 1: Simple Function

```
#include <iostream>
using namespace std;
void greet() {
   cout << "Hello, Welcome!" << endl;
}
```

Explanation

- void → function kuch return nahi karega
- greet → function ka naam
- () → koi parameter nahi
- {} → function ka actual kaam (print karna)

Call by Value

- Function ko original variable ki copy pass hoti hai.
- Function ke andar value change karne se original variable change nahi hota.

Example

```
#include <iostream>
using namespace std;
void addTen(int x) { // x ki copy pass hui
   x = x + 10;
   cout << "Inside function: " << x << endl;
}
int main() {
   int a = 5;
   addTen(a);
   cout << "In main: " << a << endl; // original 'a' unchanged
   return 0;
}
```

Explanation

- int x → Function me copy pass hui, original a safe hai
- x = x + 10; → Sirf copy ki value change hui
- cout << "Inside function: " << x; → Function ke andar 15 print hoga
- cout << "In main: " << a; → Original a unchanged, 5 print hoga

Call by Reference

- Function ko original variable ka reference (address) pass hota hai.
- Function ke andar value change karne se original variable bhi change hota **hai**.

Example

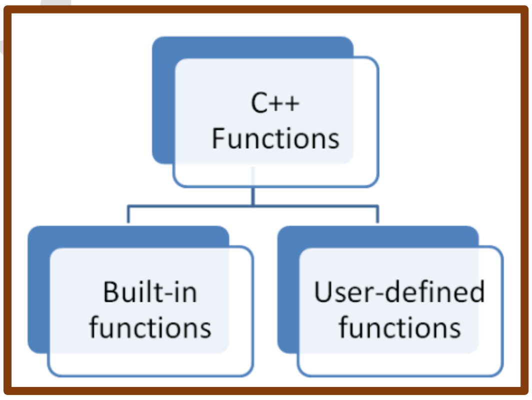
```
#include <iostream>
using namespace std;
void addTen(int &x) { // reference pass kiya
    x = x + 10;
    cout << "Inside function: " << x << endl;
}
int main() {
    int a = 5;
    addTen(a);
    cout << "In main: " << a << endl; // original 'a' changed
    return 0;
}
```

Explanation

- int &x → Function me original variable ka reference pass hua
- x = x + 10; → Original variable a bhi change ho gaya
- cout << "Inside function: " << x; → Function ke andar 15 print hoga
- cout << "In main: " << a; → Original a bhi changed, 15 print hoga

Types of Functions in C++

Type	Function Type	Description	Example
1	Library (Built-in) Functions	Pehle se bane hue functions jo C++ libraries me hote hain	sqrt(), cout, strlen()
2	User-Defined Functions	Programmer khud banata hai apni need ke hisaab se	add(), display()



Common Library Functions

<iostream>	cout	Output print karta hai
<iostream>	cin	Input leta hai
<cmath>	sqrt()	Square root nikalta hai
<cmath>	pow()	Power calculate karta hai
<cstring>	strlen()	String ki length batata hai

<cstring>	strcpy()	String copy karta hai
<cstdlib>	rand()	Random number generate karta hai
<ctime>	time()	Current time deta hai

cout – Output print karta hai

```
#include <iostream>
using namespace std;
int main() {
   cout << "Hello World";
   return 0;
}
```

cin – Input leta hai

```
#include <iostream>
using namespace std;
int main() {
   int a;
   cin >> a;            // user se input
   cout << a;
   return 0;
}
```

sqrt() – Square root nikalta hai

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
   int x = 16;
   cout << sqrt(x);
   return 0;
}
```

pow() – Power calculate karta hai

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
   cout << pow(2, 3);     // 2 ki power 3
   return 0;
}
```

strlen() – String ki length batata hai

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
    char name[] = "Hello";
    cout << strlen(name);
    return 0;
}
```

strcpy() – String copy karta hai

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
    char s1[] = "C++";
    char s2[10];
    strcpy(s2, s1);
    cout << s2;
    return 0;
}
```

rand() – Random number generate karta hai

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main() {
    cout << rand();
    return 0;
}
```

time() – Current time deta hai

```
#include <iostream>
#include <ctime>
using namespace std;
int main() {
    cout << time(0);
    return 0;
}
```

Types of User-Defined Functions

No Arguments, No Return Value

No.	Function Type	Arguments	Return Value	Explanation (Hinglish)	Example
1	No Arguments, No Return Value	❌	❌	Na input leta hai, na output deta hai	void show()
2	With Arguments, No Return Value	✅	❌	Input leta hai par value return nahi karta	void sum(int a,int b)
3	No Arguments, With Return Value	❌	✅	Input nahi leta par value return karta	int getNum()
4	With Arguments, With Return Value	✅	✅	Input bhi leta hai aur output bhi deta hai	int add(int a,int b)

```
#include <iostream>
using namespace std;
void show() {
   cout << "Hello";
}
int main() {
   show();
   return 0;
}
```

Explanation

- #include <iostream> → Output ke liye iostream header file use hoti hai
- using namespace std; → std:: likhne ki zarurat nahi padti
- void show() → Ek user-defined function hai jo koi value return nahi karta
- cout << "Hello"; → Screen par Hello print karta hai
- int main() → Program yahin se start hota hai
- show(); → show() function ko call kiya gaya hai
- return 0; → Program successful end hone ka signal deta hai

With Arguments, No Return Value

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

```
void sum(int a, int b) {
    cout << a + b;
}
int main() {
    sum(5, 3);
    return 0;
}
```

Explanation

- #include <iostream> → Input/output ke liye iostream header file use hoti hai
- using namespace std; → std:: likhne se bachne ke liye
- void sum(int a, int b) → User-defined function jo do arguments leta hai aur koi value return nahi karta
- int a, int b → Function ke input parameters hain
- cout << a + b; → Dono numbers ka sum print karta hai
- int main() → Program execution yahin se start hota hai
- sum(5, 3); → Function call, jisme 5 aur 3 pass kiye gaye hain
- return 0; → Program successful end hota hai

No Arguments, With Return Value

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

Explanation

- #include <iostream> → Input/output ke liye iostream library include ki
- using namespace std; → std:: baar-baar likhne ki zarurat nahi
- int getNumber() → User-defined function jo koi argument nahi leta aur integer value return karta hai
- return 10; → Function 10 value return karta hai
- int main() → Program execution yahin se start hota hai

- cout << getNumber(); → Function call kiya aur jo value return hui (10) usko print kiya
- return 0; → Program successful end hota hai

With Arguments, With Return Value

```
#include <iostream>
using namespace std;
int add(int a, int b) {
   return a + b;
}
int main() {
   cout << add(4, 6);
   return 0;
}
```

Explanation

- #include <iostream> → Input/output ke liye iostream library include ki
- using namespace std; → std:: baar-baar likhne ki zarurat nahi
- int add(int a, int b) → User-defined function jo do arguments leta hai aur integer value return karta hai
- return a + b; → Dono input numbers ka sum return karta hai
- int main() → Program execution yahin se start hota hai
- cout << add(4, 6); → Function call kiya aur jo value return hui (10) usko print kiya
- return 0; → Program successful end hota hai

C++ Functions – Pass By Reference

C++ me Pass By Reference ka matlab hai ki function ko original variable ka address pass kiya jata hai, na ki uski copy. Iska fayda ye hai ki function ke andar ki changes original variable par reflect hoti hain.

Key Points

Syntax: void func(int &x) → & ka matlab hai reference

- Function me original variable ke sath kaam hota hai, copy nahi
- Memory me extra copy create nahi hoti → efficient
- Agar function me value change karni ho → Pass By Reference use karte hain

Example Program

```
#include <iostream>
using namespace std;
void addTen(int &x) { // Reference pass kiya
    x = x + 10;       // Original variable change hoga
}
int main() {
    int a = 5;
    addTen(a);        // 'a' ka reference function me pass hua
    cout << a;        // Original 'a' print hoga
    return 0;
}
```

Explanation (Hinglish)

- int &x → Original variable a ka reference liya gaya
- x = x + 10; → Original a ko modify kiya
- addTen(a); → Function call ke time pe **address pass hua**, copy nahi
- cout << a; → Original variable 15 print hua

C++ Pass Array to a Function

C++ me jab aap array ko function me pass karte ho, toh array ka starting address pass hota hai, pura array nahi.

- Matlab, function ke andar agar aap array ke elements change karte ho, toh original array bhi change ho jata hai.
- Ye pass by reference jaisa behavior hai.

Situation:

Maan lo aap ek school me teacher ho aur students ke marks ka record maintain kar rahe ho. Aapko ek function likhna hai jo average marks calculate kare. Scenario

- Student marks ek array me store hain.
- Function me array pass karenge aur average return karenge.

```
#include <iostream>
using namespace std;
// Function to calculate average marks
float calculateAverage(int marks[], int size) {
    int sum = 0;
    for(int i = 0; i < size; i++) {
        sum += marks[i];  // array elements add kar rahe
    }
    return (float)sum / size;
}
int main() {
    int studentMarks[5] = {80, 90, 70, 85, 95};  // students ke marks
    float avg = calculateAverage(studentMarks, 5);  // array aur size pass kiya
    cout << "Average Marks: " << avg << endl;
    return 0;
}
```

Explanation

- studentMarks array me 5 students ke marks hain.
- calculateAverage function ko array aur size pass kiya.
- Function me marks[i] se sab elements ko add kiya aur average return kiya.
- Result me original array change nahi hota, bas uska data use hota ha

Function Overloading kya hai?

Function Overloading ka matlab hai ek hi naam ke multiple functions create karna, lekin unke parameters alag hone chahiye (type, number ya order). Compiler decide karta hai ki kaunsa function call hoga arguments ke hisaab se.

```
#include <iostream>
using namespace std;
// Function 1: do sum of 2 integers
int add(int a, int b) {
    return a + b;
}

// Function 2: do sum of 3 integers
int add(int a, int b, int c) {
    return a + b + c;
}
// Function 3: do sum of 2 doubles
double add(double a, double b) {
    return a + b;
}
```

```
int main() {
   cout << add(5, 10) << endl;       // calls add(int, int)
   cout << add(5, 10, 15) << endl;   // calls add(int, int, int)
   cout << add(5.5, 10.5) << endl;   // calls add(double, double)
   return 0;
}
```

Explanation

- Function Overloading = Same function name, different parameters (number/type/order).
- Return type sirf alag hone se kaam nahi karta.
- Compiler automatically decide karta hai kaunsa function call hoga.
- Example from your code:
- add(5,10) → calls add(int,int) → 15
- add(5,10,15) → calls add(int,int,int) → 30
- add(5.5,10.5) → calls add(double,double) → 16

Without function overloding

```
#include <iostream>
using namespace std;
// 2 integers ka sum
int addTwoInt(int a, int b) {
   return a + b;
}
// 3 integers ka sum
int addThreeInt(int a, int b, int c) {
   return a + b + c;
}
// 2 doubles ka sum
double addTwoDouble(double a, double b) {
   return a + b;
}
int main() {
   cout << addTwoInt(5, 10) << endl;       // 15
   cout << addThreeInt(5, 10, 15) << endl; // 30
   cout << addTwoDouble(5.5, 10.5) << endl; // 16
   return 0;
}
```

Explanation

- Function overloading ke bina, har combination ke liye alag function name banana padta hai.
- Code thoda long aur repetitive ho jata hai.

Variable Scope kya hai?

Variable Scope ka matlab hai kahan-kahan se ek variable access kiya ja sakta hai.
Yaani, ek variable kaunse part of program mein visible ya usable hai.

Types of Variable Scope

Local Scope

- Jo variable function ke andar declare hota hai, wahi function ke andar hi accessible hota hai.
- Function ke bahar use nahi kiya ja sakta.

```
#include <iostream>
using namespace std;
void func() {
   int x = 10;  // local variable
   cout << x << endl; // OK
}
int main() {
   // cout << x << endl; // Error! x yahan accessible nahi
   func();
   return 0;
}
```

Global Scope

- Jo variable function ke bahar declare hota hai, woh pure program ke andar kahin se bhi accessible hota hai.

```
#include <iostream>
using namespace std;
int x = 100; // global variable
void func1() {
   cout << x << endl; // OK
}
int main() {
   cout << x << endl; // OK
```

```
    func1();
    return 0;
}
```

Block Scope

- Jo variable {} braces ke andar declare hota hai, woh sirf us block ke andar hi accessible hota hai.
- Block = function, loop, if statement, etc.

```
#include <iostream>
using namespace std;
int main() {
    if(true) {
        int y = 50; // block scope
        cout << y << endl; // OK
    }
    // cout << y << endl; // Error! y block ke bahar accessible nahi
    return 0;
}
```

Function Scope (for static variables)

- Agar variable ko static declare kiya, to uska value function ke multiple calls ke beech retain hota hai.
- Still function ke bahar accessible nahi.

```
#include <iostream>
using namespace std;
void counter() {
    static int count = 0; // static local variable
    count++;
    cout << count << endl;
}
int main() {
    counter(); // 1
    counter(); // 2
    counter(); // 3
    return 0;
}
```

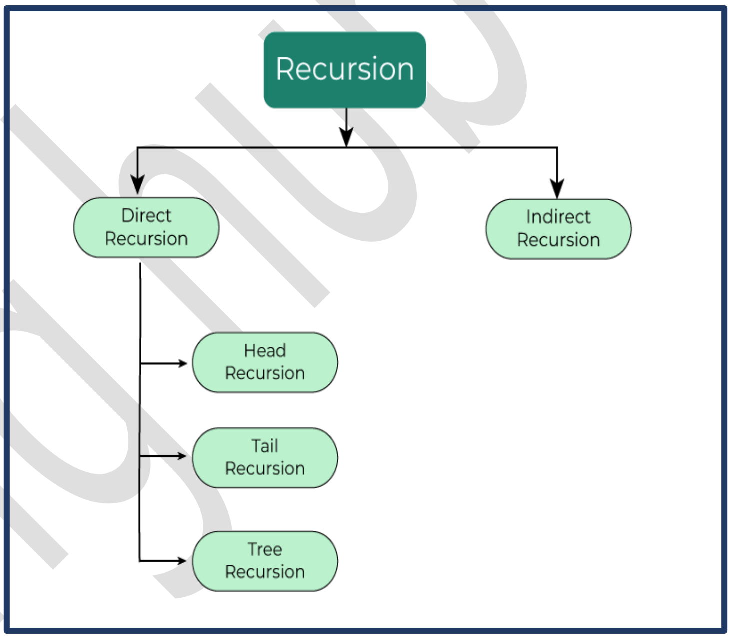
Summary

- Local variable → sirf function ke andar usable
- Global variable → pure program me accessible
- Block variable → sirf {} ke andar usable
- Static variable → function ke andar retain hota hai, par bahar access nahi

Recursion

Recursion ka matlab hai ek function ka khud ko call karna.

- Ye problem ko chhote-chhote parts me todkar solve karta hai.
- Har recursive function me ek base case hona zaruri hai, warna infinite loop ho jayega.



Structure of a Recursive Function

```
returnType functionName(parameters) {
    if (base_condition) {
        // base case
        return some_value;
    } else {
        // recursive case
        return
functionName(modified_parameters);
    }
}
```

Direct Recursion kya hai?

- Direct Recursion tab hoti hai jab ek function khud ko hi call karta hai.
- Yani, function ka naam apne hi body ke andar repeat hota hai.
- Ye sabse common type of recursion hai.

```
#include <iostream>
using namespace std;
void show(int n) {
    if (n == 0)
        return;
    cout << n << " ";
    // Direct recursive call
    show(n - 1);
}
int main() {
    show(5);
    return 0;
}
```

Explanation

- void show(int n) → ek function jo n se 1 tak numbers print karta hai.
- Base case: if (n == 0) return; → jab n 0 ho jaye, recursion stop ho jata hai.
- Recursive call: show(n - 1); → function apne aap ko call karta hai n-1 ke saath.

Head Recursion kya hai?

- Head Recursion me recursive call function ke start me hoti hai, aur processing (jaise printing ya calculation) uske baad hoti hai.
- Matlab, function pehle khud ko call karta hai, aur waapas aate waqt kaam karta hai.

```
void functionName(int n) {
    if (base_case) return;
    functionName(n - 1); // recursive call first (head)
    // processing after recursion
}
```

Print 1 to n using Head Recursion

```
#include <iostream>
using namespace std;
void printNumbers(int n) {
    if (n == 0) return;        // base case
    printNumbers(n - 1);       // head recursion
    cout << n << " ";          // processing after recursion
}
int main() {
    printNumbers(5);
    return 0;
}
```

Explanation:

1. printNumbers(5) → calls printNumbers(4) first
2. printNumbers(4) → calls printNumbers(3) first
3. … recursion goes on until base case n==0
4. Processing starts while returning → prints 1, 2, 3, 4, 5

Tail Recursion kya hai?

- Tail Recursion me recursive call function ke end me hoti hai, aur koi processing uske baad nahi hoti.
- Matlab, function apne kaam ko pehle karta hai, aur phir khud ko call karta hai.
- Tail recursion memory efficient hoti hai, kyunki compiler optimization kar sakta hai.

```
#include <iostream>
using namespace std;
void tail(int n) {
   if (n == 0)
      return;
   cout << n << " ";
   // Recursive call after processing
   tail(n - 1);
}
int main() {
   tail(5);
   return 0;
}
```

Code Explanation:

- void tail(int n) → ek function jo n se 1 tak numbers print karta hai.
- Base case: if (n == 0) return; → jab n 0 ho jaye, recursion stop ho jata hai.
- Processing first: cout << n << " "; → pehle current number print hota hai.
- Recursive call at the end: tail(n - 1); → function apne aap ko call karta hai n-1 ke saath.

Tree Recursion kya hai?

- Jab ek function apne aap ko ek se zyada baar call karta hai ek single call ke andar, use Tree Recursion kehte hain.
- Naam "Tree" isliye hai kyunki function calls ka structure tree ki tarah dikhta hai: ek root call ke multiple branches bante hain.

```
#include <iostream>
using namespace std;
void tree(int n) {
   if (n == 0)
      return;
   cout << n << " ";
   // Two recursive calls
   tree(n - 1);
   tree(n - 1);
}
int main() {
   tree(3);
   return 0;
}
```

Nested Recursion kya hai?

- Jab ek function ke andar recursion ke argument me hi function call hota hai, to use Nested Recursion kehte hain.
- Matlab, function ka call itself ke andar ek aur call ka argument ban jata hai.

```
#include <iostream>
using namespace std;
int nested(int n) {
   if (n > 100)
      return n - 10;
   // Recursive call inside another recursive call
   return nested(nested(n + 11));
}
int main() {
   cout << nested(95);
   return 0;
}
```

Explanation

1. Base case: $n > 100 \rightarrow$ return n - 10
2. Recursive case: $n \leq 100 \rightarrow$ function apne argument me hi apne aap ko call karta hai $\rightarrow$ nested recursion.
3. Example: nested(95) → multiple nested calls → final result 91
4. Yeh McCarthy 91 function ka example hai:
   - $n \leq 100 \rightarrow 91$ return
   - $n > 100 \rightarrow n - 10$ return

Lambda Function kya hai?

- Lambda function ek anonymous (naam nahi wala) function hota hai, jo inline likha ja sakta hai.
- Ye usually short operations ke liye use hota hai jahan alag function define karna unnecessary ho.
- Syntax modern C++ (C++11 onwards) me available hai.

Basic Syntax

[capture](parameters) -> return_type { body }

- capture: outer scope ke variables ko access karne ke liye
- parameters: function ke arguments
- return_type: (optional) function ka return type
- body: function ka kaam

Simple Example

```
#include <iostream>
using namespace std;
int main() {
   // Lambda function to add two numbers
   auto add = [](int a, int b) -> int {
      return a + b;
   };
   cout << add(5, 3); // Output: 8
   return 0;
}
```

- auto add → variable me lambda store kiya
- [](int a, int b) -> int { return a + b; } → anonymous function

Suppose humare paas students ke marks ka list hai aur hume descending order me sort karna hai. Lambda function se ye kaam bahut easy ho jata hai.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
   vector<int> marks = {85, 92, 76, 98, 64};
   // Lambda function to sort in descending order
   sort(marks.begin(), marks.end(), [](int a, int b) {
      return a > b; // descending
```

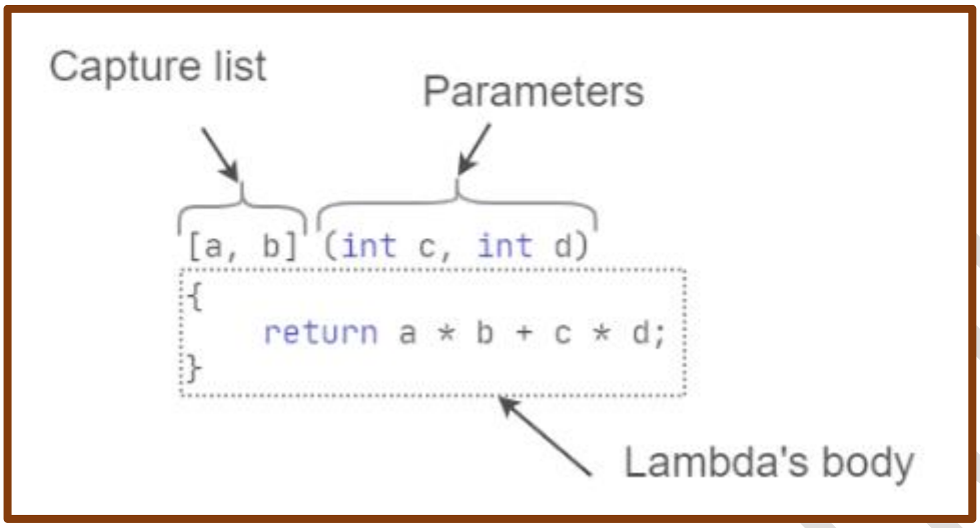
```
    });

    cout << "Sorted marks (descending): ";
    for (int m : marks) {
        cout << m << " ";
    }
    return 0;
}
```

Explanation

- vector<int> marks = {85, 92, 76, 98, 64};
  → Students ke marks ka list.
- sort(marks.begin(), marks.end(), [](int a, int b){ return a > b; });
  → Lambda function use karke marks ko descending order me sort kiya.
  → [ ] → lambda ka syntax, (int a, int b) → parameters, { return a > b; } → body.
- for (int m : marks) cout << m << " ";

Why We used Lamba Function

- Short & Inline Functions: Extra function define kiye bina small operations perform kar sakte hain.
- STL ke saath Easily Use: sort, for_each, count_if jaise algorithms me custom behavior define karna easy.
- Capture Outer Variables: Outer scope ke variables ko directly [ ] se access kar sakte hain.
- Readable & Concise Code: Code compact aur readable ho jata hai, extra boilerplate function ki zarurat nahi.
- Anonymous Functions: Naam ke bina function create kar sakte hain, temporary usage ke liye perfect.



```
[a, b] (int c, int d)
{
    return a * b + c * d;
}
```