Contents

C++ Basics

1. What are the main features of C++?

C++ is an **object-oriented** programming language with features like:

- Encapsulation
- Polymorphism
- Inheritance
- Abstraction
- Strong type checking

---

2. What is the difference between C and C++?

Feature	C	C++
Programming Type	Procedural	Object-Oriented
Encapsulation	❌ No	✅ Yes
Function Overloading	❌ No	✅ Yes
Exception Handling	❌ No	✅ Yes (try-catch)

---

3. What are the different data types in C++?

- **Basic:** int, char, float, double, bool
- **Derived:** array, pointer, reference
- **User-defined:** struct, class, enum, union

---

4. What is the difference between class and struct?

- **class**: Members are **private** by default.
- **struct**: Members are **public** by default.

Example:

```
class Example {
private:
   int a;  // Private
};

struct ExampleStruct {
   int a;  // Public
};
```

---

5. What are pointers in C++?

A pointer stores the **memory address** of a variable.

Example:

```
int x = 10;
int *ptr = &x;
cout << *ptr;  // Output: 10
```

---

6. What is the difference between new and malloc()?

Feature	new	malloc()
Returns	Exact Data Type	void*
Calls Constructor	✅ Yes	❌ No
Syntax	int *p = new int;	int *p = (int*)malloc(sizeof(int));

---

7. What is a reference variable in C++?

A reference is an **alias** for another variable.

Example:

cpp

CopyEdit

int x = 10;

int &y = x; // y is a reference to x

y = 20; // Changes x to 20

---

8. What is the difference between == and = in C++?

- = **Assignment Operator**
- == **Comparison Operator**

Example:

int x = 10; // Assignment

if (x == 10) { cout << "Equal"; } // Comparison

---

9. What is the difference between function overloading and function overriding?

Feature	**Overloading**	**Overriding**
Where?	Same class	Derived class
Signature	Different	Same
Keyword	None	virtual

---

10. What is the difference between ++i and i++?

- ++i **(Pre-increment):** Increments first, then returns value.
- i++ **(Post-increment):** Returns value first, then increments.

---

11. What is a constructor in C++?

A **constructor** is a special function that initializes an object.

Example:

class Example {

```
public:
    Example() { cout << "Constructor called"; }
};
Example obj;  // Constructor automatically runs
```

---

12. What is a destructor in C++?

A **destructor** is a function that is automatically called when an object is destroyed.

Example:

```
class Example {
public:
    ~Example() { cout << "Destructor called"; }
};
```

---

13. What is the difference between public, private, and protected access specifiers?

Access	Scope
public	Accessible anywhere
private	Only accessible inside the class
protected	Accessible in derived classes

---

14. What is dynamic memory allocation in C++?

Dynamic memory allocation reserves memory during runtime using new and delete.

Example:

```
int *p = new int(10);  // Allocating memory
delete p;  // Freeing memory
```

---

15. What is an inline function?

An **inline function** replaces the function call with the actual function code.

Example:

```
inline void show() { cout << "Inline Function"; }
```

---

16. What is the difference between deep copy and shallow copy?

Type	Definition
Shallow Copy	Copies only object references
Deep Copy	Copies object data

---

17. What is the difference between const int* and int* const?

- **const int* p** → Pointer to a **constant integer** (can't change value).
- **int* const p** → **Constant pointer** (can't change address).

---

18. What is the this pointer?

The this pointer holds the **current object's address** inside a class.

Example:

```
class Example {
public:
   void show() { cout << this; }
};
```


---

19. What is friend function in C++?

A **friend function** can access private members of a class.

Example:

```
class Example {
   int x;
public:
   friend void show(Example);
};
void show(Example obj) { cout << obj.x; }
```

---

20. What is the difference between throw and throws?

- **throw** → Used to throw an exception.

- **throws** → Used in function signature to specify exceptions (only in Java).

Pointers

1. What is a pointer in C++?

A pointer is a variable that stores the **memory address** of another variable.

Example:

```
int x = 10;
int *ptr = &x;  // Pointer storing the address of x
cout << *ptr;  // Dereferencing, output: 10
```

---

2. What is the difference between a pointer and a reference?

Feature	Pointer	Reference
Stores	Address of a variable	Alias of a variable
Initialization	Can be initialized later	Must be initialized when declared
Reassignment	Can point to another variable	Cannot be reassigned

Example:

```
int a = 10;
int *p = &a;  // Pointer
int &r = a;   // Reference
```

---

3. How do you declare and use a pointer to a function in C++?

A function pointer stores the **address of a function** and can be used to call it dynamically.

Example:

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

```
void display() { cout << "Hello"; }

int main() {
   void (*funcPtr)() = display;  // Function pointer
   funcPtr();  // Calls display()
}
```

---

4. What are wild, dangling, and null pointers?

Type	Definition
Wild Pointer	Uninitialized pointer
Dangling Pointer	Points to deleted memory
Null Pointer	Points to nullptr

Example (Null Pointer):

```
int *ptr = nullptr;  // Safe pointer initialization
if (ptr == nullptr) { cout << "Null pointer"; }
```


---

5. How does pointer arithmetic work in C++?

Pointer arithmetic involves incrementing or decrementing pointers based on data types.

Example:

```
int arr[] = {10, 20, 30};
int *p = arr;
p++;  // Moves to next element (20)
cout << *p;  // Output: 20
```

---

6. What is a reference variable in C++?

A reference variable is an **alias** for another variable.

Example:

```
int x = 5;
int &y = x;  // y is another name for x
y = 10;  // Modifies x as well
```

---

7. What is a pointer to a reference?

A **pointer cannot store a reference**, but it can store the address of a referenced variable.

Example:

```
int x = 10;
int &ref = x;
int *ptr = &ref;  // Pointer to referenced variable
cout << *ptr;  // Output: 10
```

---

8. What is a double pointer (**)?

A **double pointer** stores the address of another pointer.

Example:


```
int x = 5;
int *p = &x;
int **pp = &p;  // Pointer to pointer
cout << **pp;  // Output: 5
```

---

9. What is const with pointers in C++?

Type	Meaning
const int *p	Pointer to a constant integer (cannot change value)
int *const p	Constant pointer (cannot change address)
const int *const p	Both constant pointer and constant value

Example:

```
const int a = 10;
```

```
const int *p = &a;  // Can't modify value
```

---

10. How does dynamic memory allocation work with pointers?

Memory can be allocated dynamically using new and deallocated using delete.

Example:

```
int *p = new int(5);  // Allocates memory
cout << *p;  // Output: 5
delete p;  // Free memory
```

Struct , Enum

1. What is the difference between struct, enum, and union in C++?

Feature	struct	enum	union
Stores	Multiple variables	Named constants	One variable at a time
Memory Allocation	Separate for each member	None (just names)	Shared among members
Default Access	Public	Public	Public
Example Use	Grouping related data	Defining named values	Saving memory

---

2. How do you define and use a struct in C++?

A **struct** is used to store multiple variables under one name.

Example:

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

struct Student {
    string name;
    int age;
```

```
};

int main() {
    Student s1 = {"John", 20};
    cout << s1.name << " is " << s1.age << " years old.";
}
```

💡 **Output:** John is 20 years old.

---

3. How does an enum work in C++?

An **enum (enumeration)** is a user-defined type that assigns names to integer values.

Example:

```
c
enum Color { RED, GREEN, BLUE };

int main() {
    Color c = GREEN;
    cout << c;  // Output: 1 (Enums are internally integers)
}
```


---

4. What is a union, and how does it differ from a struct?

A **union** is similar to a struct, but all members **share the same memory space**.

Example:

```
union Data {
    int i;
    float f;
};

int main() {
    Data d;
    d.i = 10;
    cout << d.i;  // Output: 10
```

```
    d.f = 3.14;
    cout << d.i;  // Now i holds garbage because f overwrote it
}
```

---

5. When should you use struct, enum, or union?

- **Use struct** when storing multiple independent variables.
- **Use enum** for defining named constants instead of #define.
- **Use union** when saving memory (e.g., embedded systems).

Exception Handling

1. What is exception handling in C++?

Exception handling allows a program to **handle runtime errors** using try, catch, and throw.

Example:


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

int main() {
    try {
        throw "An error occurred!";
    } catch (const char* msg) {
        cout << msg;  // Output: An error occurred!
    }
}
```

---

2. What is the difference between throw, try, and catch?

Keyword	Purpose
throw	Raises an exception
try	Defines a block that might throw an exception
catch	Handles the exception

---

3. Can we have multiple catch blocks in C++?

Yes, we can handle different types of exceptions with multiple catch blocks.

Example:

```
try {
   throw 10;
} catch (int e) {
   cout << "Integer Exception: " << e;
} catch (...) {
   cout << "Unknown Exception";
}
```


💡 **Output:** Integer Exception: 10

---

4. What is a generic catch block in C++?

A generic catch block (catch(...)) handles **any type of exception**.

Example:

```
try {
   throw "Error!";
} catch (...) {
   cout << "Unknown error occurred.";
}
```

💡 **Output:** Unknown error occurred.

---

5. What happens if an exception is not caught?

If no catch block handles the exception, the program **terminates abnormally**.

Example (Unhandled Exception):

```
int main() {
   throw 5;  // No catch block
   return 0;
}
```

💡 **Result:** Program crashes with terminate called after throwing an instance of 'int'.

Arrays Strings

1. What is an array in C++?

An array is a **collection of elements of the same data type** stored in **contiguous memory locations**.

2. How do you declare and initialize an array in C++?

Declaration:

```
int arr[5];  // Declares an array of 5 integers
```

Initialization:

```
int arr[5] = {1, 2, 3, 4, 5};
```

3. What is the difference between a static and dynamic array?

- **Static array:** Fixed size, declared at compile time.
- **Dynamic array:** Created at runtime using new.

4. How do you pass an array to a function?

Arrays are passed **by reference** using pointers.

```
void display(int arr[], int size);
```

5. What is a multi-dimensional array in C++?

A multi-dimensional array stores data in a **matrix-like format**.

int matrix[3][3]; // 2D array

6. What is a string in C++?

A string is a sequence of characters, stored using char arrays or std::string class.

7. What is the difference between C-style strings and std::string?

Feature	C-style String (char[])	std::string
Null-terminated	Yes ('\0')	No
Easy to manipulate	No	Yes
Requires manual memory management	Yes	No

8. How do you declare and initialize a C-style string?

```
char str[] = "Hello";  // Automatically adds '\0' at the end
```

9. How do you declare and initialize an std::string?

```
#include <string>
std::string str = "Hello";
```

10. How do you find the length of a string in C++?

- C-style string: strlen(str)
- std::string: str.length()

11. What are the different string manipulation functions in C++?

- strcpy() – Copy string
- strcat() – Concatenate
- strcmp() – Compare

12. How do you convert a string to an integer in C++?

Use stoi() for std::string or atoi() for C-style strings.

```
int num = stoi("123");
```

13. How do you compare two strings in C++?

Using == for std::string or strcmp() for C-style strings.

14. How do you concatenate strings in C++?

- C-style: strcat()
- std::string: + operator

std::string fullName = "John" + std::string(" Doe");

15. How do you convert a number to a string in C++?

Use to_string().

std::string str = std::to_string(100);

16. String functions ?

These functions require <cstring> header.

Function	Purpose	Example
strlen(str)	**Returns the length of a string (excluding \0)**	**strlen("Hello") → 5**
strcpy(dest, src)	**Copies src to dest**	**strcpy(dest, "Hello")**
strncpy(dest, src, n)	**Copies n characters from src**	**strncpy(dest, "Hello", 3) → Hel**
strcat(dest, src)	**Appends src to dest**	**"Hi" + "There" → "HiThere"**
strncat(dest, src, n)	**Appends n characters from src**	**"Hi" + "There" (3) → "HiThe"**
strcmp(str1, str2)	**Compares two strings (returns 0 if equal)**	**strcmp("abc", "abc") → 0**
strncmp(str1, str2, n)	**Compares n characters of two strings**	**strncmp("apple", "apron", 2) → 0**
strchr(str, ch)	**Finds first occurrence of a character**	**strchr("hello", 'l') → "llo"**
strrchr(str, ch)	**Finds last occurrence of a character**	**strrchr("hello", 'l') → "lo"**
strstr(str1, str2)	**Finds first occurrence of str2 in str1**	**strstr("hello world", "world") → "world"**

---

2. Functions for std::string

These functions require <string> header.

Function	Purpose	Example
s.length() / s.size()	**Returns length of string**	**"Hello".length() → 5**
s.empty()	**Checks if the string is empty**	**"" .empty() → true**

Function	Purpose	Example
s.append(str)	Appends a string to another	"Hi".append(" There") → "Hi There"
s.insert(pos, str)	Inserts str at position pos	"Hi".insert(2, " There") → "Hi There"
s.erase(pos, len)	Removes characters from position pos	"Hello".erase(2,2) → "Heo"
s.replace(pos, len, str)	Replaces substring	"Hello".replace(1, 2, "i") → "Hillo"
s.find(str)	Finds first occurrence of str	"Hello".find("l") → 2
s.rfind(str)	Finds last occurrence of str	"Hello".rfind("l") → 3
s.substr(pos, len)	Extracts substring	"Hello".substr(1,3) → "ell"
s.compare(str)	Compares two strings (returns 0 if equal)	"abc".compare("abc") → 0
s.at(pos)	Returns character at index pos	"Hello".at(1) → 'e'
s.front() / s.back()	Returns first/last character	"Hello".front() → 'H'
s.c_str()	Converts std::string to C-style string	"Hello".c_str()
s.append(str, pos, len)	Appends substring from another string	"Hi".append("There", 1, 2) → "Hih"
s.find_first_of(str)	Finds first occurrence of any char from str	"hello".find_first_of(

New and Delete

1. What is the purpose of new and delete in C++?

- new is used to dynamically allocate memory on the **heap**.
- delete is used to **deallocate** memory that was allocated using new.

Example:

```
int* ptr = new int(10);  // Allocates memory for an integer
delete ptr;  // Frees the allocated memory
```

---

2. What is the difference between new/delete and malloc/free?

Feature	new/delete	malloc/free
Returns	Typed pointer	void* (requires typecasting)
Initializes memory	Yes (new int(5))	No (must use memset)
Calls constructor/destructor	Yes	No
Use in C/C++	C++ only	C and C++

---

3. What is the difference between delete and delete[]?

- **delete** is used for a **single object** allocated with new.
- **delete[]** is used for an **array** allocated with new[].

Example:

```
int* p1 = new int(10);
delete p1;  // Correct

int* p2 = new int[5];
delete[] p2;  // Correct (deletes entire array)
```


💡 **Using delete instead of delete[] on an array causes undefined behavior.**

---

4. What happens if you delete the same pointer twice?

If delete is called twice on the same pointer, it leads to **undefined behavior** (double free error).

Example (Unsafe Code):

```
int* ptr = new int;
delete ptr;
delete ptr;  // Undefined behavior!
```

💡 **Solution:** Always set the pointer to nullptr after deleting.

```
delete ptr;
ptr = nullptr;
```

---

5. How do you allocate and deallocate memory for a class object using new?

Example with Constructor & Destructor:

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

class Student {
public:
    Student() { cout << "Constructor called!\n"; }
    ~Student() { cout << "Destructor called!\n"; }
};

int main() {
    Student* s = new Student();
    delete s;  // Calls destructor before freeing memory
}
```

💡 **Output:**

```
Constructor called!
Destructor called!
```