MENTORBRO LEARNING LLP,
Email: mentorbroofficial@gmail.com
Ph no : + 91 8157867616

Mentor Bro Notes

🌟 C vs. C++: Comprehensive Comparison

C and C++ are two powerful programming languages, but they differ significantly. While **C** is great for procedural programming and low-level tasks, **C++** builds on it with **Object-Oriented Programming (OOP)** concepts, adding flexibility and reusability.

1️⃣ Basic Differences Between C and C++

Feature	C	C++
Paradigm	Procedural Programming 🛠️	Procedural + Object-Oriented Programming 🛠️🌀
Data Security	Low 🔓 (Global variables are common)	High 🔐 (Encapsulation with classes)
Code Reusability	Less ♻️ (No OOP)	More ♻️ (Inheritance, Polymorphism)
Function Overloading	❌ Not Supported	✅ Supported
Operator Overloading	❌ Not Supported	✅ Supported
Encapsulation	❌ No classes	✅ Uses classes and objects
Standard I/O	`printf()`, `scanf()` 📄	`cout`, `cin` 📖 (also supports C functions)
Memory Management	`malloc()`, `free()` 📂	`new`, `delete` 🗑️ (safer and faster)

Mentor Bro

✨ Theory:

- **C** follows a procedural paradigm, meaning it focuses on **step-by-step instructions**.
- **C++**, while retaining C's procedural roots, introduces OOP concepts like **classes**, **inheritance**, and **polymorphism**, making the code more reusable and secure.

2️⃣ Code Comparison: Hello World

📜 Hello World Program in C:

```c
#include <stdio.h>

int main() {
    printf("Hello, World!");
    return 0;
}
```

📜 Hello World Program in C++:

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, World!";
    return 0;
}
```

📌 **Key Difference**:

- **C** uses `printf()` from the `<stdio.h>` library.
- **C++** uses `cout` from the `<iostream>` library, providing a more **modern and flexible approach**.

3️⃣ Procedural vs Object-Oriented Programming (OOP)

Mentor Bro

C: Procedural Approach

- Focuses on **functions** and **data structures**.
- No encapsulation, meaning global variables are common.

Example: Struct in C

```c
#include <stdio.h>

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

int main() {
    struct Student s1 = {"Alice", 20};
    printf("Name: %s, Age: %d", s1.name, s1.age);
    return 0;
}
```

C++: OOP Approach

- Introduces **classes**, combining **data** and **functions**.
- Supports encapsulation, inheritance, and polymorphism.

Example: Class in C++

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

class Student {
public:
    string name;
    int age;

    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};
```

Mentor Bro

```
int main() {
    Student s1;
    s1.name = "Alice";
    s1.age = 20;
    s1.display();
    return 0;
}
```

📌 **Key Difference**:

- **C**: Uses `struct`, but it cannot have functions.
- **C++**: Introduces `class`, which encapsulates **data** and **methods**.

4️⃣ Memory Management

Efficient memory management is crucial for performance. Here's how **C** and **C++** differ:

C: Uses `malloc()` and `free()`

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *ptr = (int*) malloc(sizeof(int));
    *ptr = 10;
    printf("%d", *ptr);
    free(ptr);
    return 0;
}
```

C++: Uses `new` and `delete`

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

int main() {
```

Mentor Bro

```
    int *ptr = new int;
    *ptr = 10;
    cout << *ptr;
    delete ptr;
    return 0;
}
```

📌 **Key Difference**:

- **C**: `malloc()` and `free()` are **function-based**.
- **C++**: `new` and `delete` are **operator-based**, making them faster and safer.

5️⃣ Function Overloading

Function overloading allows multiple functions with the **same name** but different parameters.

C: No Function Overloading ❌

```
#include <stdio.h>

void add(int a, int b) {
    printf("Sum: %d", a + b);
}

/* ❌ Cannot define another function with the same name. */

int main() {
    add(5, 10);
    return 0;
}
```

C++: Supports Function Overloading ✅

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

class Math {
```

Mentor Bro

```
public:
    void add(int a, int b) {
        cout << "Sum: " << a + b << endl;
    }
    void add(double a, double b) {
        cout << "Sum: " << a + b << endl;
    }
};

int main() {
    Math obj;
    obj.add(5, 10);       // Calls the int version
    obj.add(3.5, 2.5);    // Calls the double version
    return 0;
}
```

📌 **Key Difference**:

- **C**: Does **not** support function overloading.
- **C++**: Allows function overloading, increasing flexibility.

6️⃣ Performance Comparison

Feature	C	C++
Execution Speed	Faster ⚡	Slightly slower (OOP overhead)
Compilation Time	Faster 🛠️	Slower (more features)
Code Readability	Less readable 📄	More readable (OOP) 📘

📌 **Summary**:

- **C** is better for **low-level, performance-critical tasks**.
- **C++** excels in **large, complex projects** with reusable components.

Mentor Bro

7️⃣ When to Use C vs. C++?

Use C When…	Use C++ When…
You need **low-level system programming** (OS, drivers).	You need **OOP concepts** (games, GUI, apps).
Performance is **critical**.	Code **reusability** is important.
You are working with **C-based legacy code**.	You need **modern programming features**.

8️⃣ Final Summary

Feature	C	C++
Programming Type	Procedural 🛠️	OOP + Procedural 🌀
Memory Management	`malloc()`, `free()` 📂	`new`, `delete` 🗑️
Encapsulation	❌ No classes	✅ Yes (Classes)
Operator Overloading	❌ No	✅ Yes
Function Overloading	❌ No	✅ Yes
Code Complexity	More manual effort ⚙️	More structured 💻
Execution Speed	Faster ⚡	Slightly slower 🌀

📌 **Verdict**:

- Choose **C** for **performance-critical, low-level programming**.
- Choose **C++** for **modern, scalable, and maintainable projects**.

MentorBro