Introduction to C++ (Beginner Friendly Notes)

---

1. WHY C++ AFTER C?

C is a powerful programming language.
But as software became bigger and more complex, C had some limitations:

- No support for Object-Oriented Programming (OOP)
- Code becomes hard to manage in large projects
- Memory management is manual and risky

So C++ was created.

👉 C++ = C + Object-Oriented Features + Improvements

C++ keeps everything from C
AND adds modern features to make programming easier and safer.

---

2. C++ vs C – What's New?

C++ introduced many new features.

🔷 Major Additions in C++

- `cout` and `cin` (easy input/output)
- Function overloading
- Default arguments
- References
- Classes & Objects (OOP)
- Better memory management

The good news:

Almost all C code works in C++
So learning C++ after C is easy.

---

3. INPUT & OUTPUT IN C++

In C, we use:

```
printf()
scanf()
```

In C++, we use:

```
cout    → output
cin     → input
```

🔷 Step 1: Include iostream

```
#include <iostream>
```

This allows us to use `cout` and `cin`.

🔷 Output Example

```
std::cout << "Hello";
```

<< is called the insertion operator.

It sends data to the screen.

🔷 Input Example

```
int age;
std::cin >> age;
```

>> takes input from user and stores it in a variable.

🔷 Complete Example

```
#include <iostream>

int main() {
    int age;
    std::cout << "Enter your age: ";
    std::cin >> age;
    std::cout << "You are " << age << " years old.\n";
    return 0;
}
```

🔷 What is `using namespace std;` ?

Instead of writing:

```
std::cout
std::cin
```

You can write:

```
using namespace std;

cout << "Hello";
```

It is just a shortcut.

---

4. FUNCTIONS IN C++

Functions in C++ are more powerful than in C.

C++ adds:

- Default arguments
- Function overloading
- References

Let’s understand each one.

🔷 4.1 Default Arguments

Default arguments allow you to give a value to a parameter.

If the user does not pass a value, the default value is used.

Example:

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

void welcome(string name = "Guest") {
    cout << "Welcome, " << name << "!\n";
}

int main() {
    welcome();          // Uses default value
    welcome("Sara");    // Uses given value
}
```

Output:

```
Welcome, Guest!
Welcome, Sara!
```

This makes functions flexible and easier to use.

🔷 4.2 Function Overloading

In C, you cannot use the same function name twice.

In C++, you can.

As long as parameters are different.

Example:

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

void print(int x) {
    cout << "Integer: " << x << "\n";
}

void print(double x) {
    cout << "Double: " << x << "\n";
}

int main() {
    print(5);     // Calls int version
    print(5.0);   // Calls double version
}
```

Same name
Different parameters
C++ automatically chooses the correct one.

This is called Function Overloading.

🔷 4.3 References (Very Important Concept)

In C, to modify a variable inside a function, we use pointers.

In C++, we can use references.

A reference is another name for the same variable.

Example:

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

void increment(int &x) {
    x = x + 1;
}

int main() {
    int num = 10;
    increment(num);
    cout << num;
}
```

Output:

```
11
```

Here:

- `&x` means x is a reference
- Changes affect the original variable
- No need for pointers

References make code cleaner and safer.

---

5. Dynamic Memory in C++

In C, we use:

```
malloc()
free()
```

In C++, we use:

- `new` → allocate memory
- `delete` → free memory

🔷 Example:

```
int* ptr = new int;  // allocate memory
*ptr = 10;

delete ptr;          // free memory
```

Advantages over C:

- No type casting needed
- Safer
- Works well with classes (important for OOP)

---

6. Bitwise Operators in C++

Bitwise operators work on binary numbers (bits).

Every number in computer is stored in binary.

Example:
5 in binary = 0101
3 in binary = 0011

🔷 Common Bitwise Operators

Operator	Meaning
&	AND
|	OR

Operator	**Meaning**
^	XOR
~	NOT
<<	Left Shift
>>	Right Shift

🔷 Example:

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

int main() {
    int a = 5;   // 0101
    int b = 3;   // 0011

    cout << (a & b);  // Output: 1
}
```

Explanation:

```
0101
0011
-----
0001  → 1
```

---

7. Dynamic Programming (Concept Only)

Dynamic Programming (DP) is:

A problem-solving technique used to solve complex problems efficiently.

It works by:

- Breaking problems into smaller subproblems
- Solving each subproblem once
- Storing results to avoid repetition

DP is not a C++ feature.

But C++ is commonly used to implement DP because it is fast and efficient.

✅ Summary

In this topic, we learned:

- Why C++ was created
- How C++ improves C
- How to use cin and cout
- Advanced function features
- Dynamic memory in C++
- Bitwise operators
- Basic idea of Dynamic Programming