Introduction to C++

What is C++?

- C++ is a **general-purpose, high-level programming language**.
- It was developed as an extension of the **C language** to include **object-oriented features**.
- It supports both **procedural programming (like C)** and **object-oriented programming (OOP)**, so it is called a **multi-paradigm language**.
- C++ is widely used in:
    - **System programming** (Operating systems, compilers)
    - **Game development**
    - **Real-time systems**
    - **Competitive programming**

History of C++

- Developed by **Bjarne Stroustrup** at **Bell Labs** in **1979**.
- Originally called **"C with Classes"** (because it added OOP features to C).
- Officially renamed **C++ in 1983**.
- The name "C++" comes from the increment operator (++) in C, meaning "an improved version of C".
- Over the years, C++ evolved with different standards:
    - **C++98** (First standard version)
    - **C++03** (Bug fixes)
    - **C++11** (Major new features like auto, smart pointers, lambda functions)
    - **C++14, C++17, C++20, C++23** (modern updates with advanced features)

Simple C++ Program

```
#include <iostream>   // Header file for input/output

using namespace std; // Use standard namespace

int main() {
    cout << "Welcome to C++ Programming!"; // Print message on screen
    return 0; // End of program
}
```

Output:

```
Welcome to C++ Programming!
```

Note: So, C++ is basically an **enhanced C language** with the power of **object-oriented programming**, used for both **low-level system tasks** and **high-level application development**.

Variables in C++

- A variable is a **named memory location** that stores data.
- The value of a variable can **change during program execution**.

- Every variable in C++ has:
  - **Name** (identifier)
  - **Data type** (int, float, char, etc.)
  - **Value**

Syntax

```
data_type variable_name = value;
```

Example:

```
int age = 23;
float salary = 45000.50;
char grade = 'A';
```

Rules for Variable Declaration

1. Variable name can contain **letters, digits, and underscore (_)**.
   - Example: `marks1`, `student_name`
2. Variable name **must begin with a letter or underscore** (not a digit).
   - Wrong: `1age`
   - Correct: `age1`
3. No special characters allowed (except _).
4. Variable names are **case-sensitive**.
   - Example: `Age` and `age` are two different variables.
5. Variable name should **not be a C++ keyword**.
   - Wrong: `int float;`
6. Variable should have a **meaningful name**.
   - Good: `studentMarks`
   - Bad: `x`, `y` (unclear meaning)

Comments in C++

- Comments are **non-executable lines** in a program.
- Used to explain code, improve readability, or for temporary notes.
- Compiler **ignores** comments.

Types of Comments

1. **Single-line Comment**

```
// This is a single-line comment
cout << "Hello"; // Prints Hello
```

2. **Multi-line Comment**

```
/* This is a
   multi-line comment */
cout << "Welcome to C++";
```

Example Program Using Variables & Comments

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

int main() {
    int age = 20;        // Variable to store age
    float marks = 88.5; /* Variable to store marks */

    cout << "Age: " << age << endl;
    cout << "Marks: " << marks << endl;

    return 0;
}
```

Output:

```
Age: 20
Marks: 88.5
```

Data Types in C++

- A **data type** defines the type of data a variable can store.
- It tells the compiler **how much memory to allocate** and **what kind of operations** can be performed on that data.

Types of Data Types in C++

1. Basic (Primitive) Data Types

Data Type	Description	Size (approx.)	Example
`int`	Stores integers (whole numbers)	4 bytes	`int age = 23;`
`float`	Stores decimal numbers (single precision)	4 bytes	`float price = 99.5;`
`double`	Stores decimal numbers (double precision)	8 bytes	`double pi = 3.14159;`
`char`	Stores a single character	1 byte	`char grade = 'A';`
`bool`	Stores true/false values	1 byte	`bool isPass = true;`
`void`	Represents no value (used in functions)	0	`void display();`

2. Derived Data Types

- Created from basic data types.
- Examples: **Arrays, Functions, Pointers, References**

3. User-Defined Data Types

- Defined by programmers.
- Examples: **Class, Structure, Enum, Typedef**

Example Program

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

int main() {
    int age = 21;           // Integer type
    float marks = 88.75;    // Float type
    double pi = 3.14159;    // Double type
    char grade = 'A';       // Character type
    bool isPass = true;     // Boolean type

    cout << "Age: " << age << endl;
    cout << "Marks: " << marks << endl;
    cout << "Value of Pi: " << pi << endl;
    cout << "Grade: " << grade << endl;
    cout << "Pass Status: " << isPass << endl;

    return 0;
}
```

Output

```
Age: 21
Marks: 88.75
Value of Pi: 3.14159
Grade: A
Pass Status: 1
```

(**Note:** `true` is displayed as `1` and `false` as `0`)

Operators in C++

- An **operator** is a special symbol used to perform an operation on one or more operands (variables/values).
- Example: `+`, `-`, `*`, `/`, `>`

Types of Operators in C++:

1. Arithmetic Operators

Used for mathematical calculations.

Operator	Meaning	Example
+	Addition	`a + b`
-	Subtraction	`a - b`
*	Multiplication	`a * b`

Operator	Meaning	Example
/	Division	a / b
%	Modulus (Remainder)	a % b

2. Relational (Comparison) Operators

Used to compare two values (result is `true` or `false`).

Operator	Meaning	Example
==	Equal to	a == b
!=	Not equal to	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

3. Logical Operators

Used to combine relational expressions.

Operator	Meaning	Example
&&	Logical AND	(a > 5 && b < 10)
||	Logical OR	(a > 5 || b < 10)
!	Logical NOT	!(a > b)

4. Assignment Operators

Used to assign values to variables.

Operator	Meaning	Example
=	Assign value	a = 10
+=	Add and assign	a += 5 (same as a = a + 5)

Operator	Meaning	Example
`-=`	Subtract and assign	`a -= 5`
`*=`	Multiply and assign	`a *= 5`
`/=`	Divide and assign	`a /= 5`
`%=`	Modulus and assign	`a %= 5`

5. Increment and Decrement Operators

Operator	Meaning	Example
`++`	Increment by 1	`a++` (post), `++a` (pre)
`--`	Decrement by 1	`a--` (post), `--a` (pre)

6. Other Operators

- **Ternary Operator (`?:`)** → Short form of if-else

```
int result = (a > b) ? a : b;
```

- **Sizeof Operator (`sizeof`)** → Gives size of a data type

```
cout << sizeof(int);
```

- **Comma Operator (`,`)** → Separates expressions

```
int a = (2, 3); // a = 3
```

Example Program

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

int main() {
    int a = 10, b = 3;

    cout << "Arithmetic Operators:" << endl;
    cout << "a + b = " << a + b << endl;
    cout << "a - b = " << a - b << endl;
    cout << "a * b = " << a * b << endl;
    cout << "a / b = " << a / b << endl;
    cout << "a % b = " << a % b << endl;

    cout << "\nRelational Operators:" << endl;
    cout << "(a == b): " << (a == b) << endl;
    cout << "(a > b): " << (a > b) << endl;

    cout << "\nLogical Operators:" << endl;
```

```
    cout << "(a > 5 && b < 5): " << (a > 5 && b < 5) << endl;
    cout << "(a > 5 || b < 5): " << (a > 5 || b < 5) << endl;

    return 0;
}
```

Output

```
Arithmetic Operators:
a + b = 13
a - b = 7
a * b = 30
a / b = 3
a % b = 1

Relational Operators:
(a == b): 0
(a > b): 1

Logical Operators:
(a > 5 && b < 5): 1
(a > 5 || b < 5): 1
```

Input and Output in C++

- C++ provides input and output operations using the **iostream** library.
- Two main objects are used:
  - **`cout`** → For displaying (output) data on the screen.
  - **`cin`** → For taking (input) data from the user.

1. Output in C++ (`cout`)

- `cout` stands for **console output**.
- It uses the **insertion operator (<<)** to send data to the output screen.

Example:

```
int age = 20;
cout << "My age is " << age;
```

Output:

```
My age is 20
```

2. Input in C++ (cin)

- `cin` stands for **console input**.
- It uses the **extraction operator (>>)** to take input from the user.

Example:

```
int age;
cout << "Enter your age: ";
cin >> age;
```

```
cout << "You entered: " << age;
```

Output (User enters 25):

```
Enter your age: 25
You entered: 25
```

3. Common Output Manipulators

Manipulators are used to **format input and output**.

Manipulator	Description	Example
`endl`	Moves to a new line	`cout << "Hello" << endl << "World";`
`setw(n)`	Sets width of output (requires `<iomanip>`)	`cout << setw(5) << 12;`
`setprecision(n)`	Sets number of decimal places (requires `<iomanip>`)	`cout << setprecision(3) << 3.14159;`
`fixed`	Displays floating values in fixed-point notation	`cout << fixed << setprecision(2) << 3.14159;`

Example Program

```
#include <iostream>
#include <iomanip>   // for manipulators
using namespace std;

int main() {
    int roll;
    float marks;

    cout << "Enter Roll Number: ";
    cin >> roll;   // taking input

    cout << "Enter Marks: ";
    cin >> marks;  // taking input

    cout << "\n--- Student Details ---" << endl;
    cout << "Roll Number: " << roll << endl;
    cout << "Marks: " << fixed << setprecision(2) << marks << endl;

    return 0;
}
```

Sample Output

```
Enter Roll Number: 101
Enter Marks: 88.756

--- Student Details ---
Roll Number: 101
```

```
Marks: 88.76
```

Reference Variable in C++

- A **reference variable** is an **alias (another name)** for an already existing variable.
- It does not create new memory, it just provides another name for the same memory location.

Syntax

```
data_type &reference_name = existing_variable;
```

Example:

```
int a = 10;
int &ref = a; // ref is a reference to a

cout << a << endl;   // 10
cout << ref << endl; // 10

ref = 20;  // changing value using reference
cout << a << endl;   // 20
```

Typecasting in C++

Typecasting is converting one data type into another.

Types

1. **Implicit Typecasting (Type Conversion)**
   - Done automatically by the compiler.
   - Smaller data type → Larger data type.

   ```
   int x = 5;
   double y = x;  // int automatically converted to double
   ```

2. **Explicit Typecasting**
   - Done manually by the programmer.
   - Syntax: `(data_type) expression`

   ```
   double pi = 3.14;
   int val = (int) pi; // explicit conversion
   cout << val; // Output: 3
   ```

Constant Variable in C++

A **constant variable** is a variable whose value cannot be changed once it is assigned.

Example:

```
const float PI = 3.14159;
```

```
cout << PI << endl; // 3.14159

// PI = 3.2; Error: cannot modify constant
```

Operator Precedence in C++

- Operator precedence decides **which operator is executed first** when multiple operators appear in an expression.
- Operators with **higher precedence** are evaluated first.
- If precedence is same, then **associativity** decides execution order.

Operator Precedence Table

OPERATOR	TYPE	ASSOCIAVITY
() [] . ->		left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
|	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
||	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right