1. Introduction to Flow Control in C++

Flow control in C++ refers to the order in which statements of a program are executed. By default, a C++ program follows **sequential execution**, meaning statements run one after another from top to bottom. However, real-world problems require programs to **make decisions**, **repeat actions**, and **choose different paths** based on conditions.

Flow control statements allow programmers to control the execution path of a program. These statements help in implementing logic, decision-making, and branching. The most commonly used flow control statements in C++ are:

- if statement
- if-else statement
- else-if ladder
- nested if-else

2. Need for Decision Making in Programs

In real life, decisions are made continuously:

- If it rains, take an umbrella
- If marks ≥ 40, student passes
- If password is correct, allow login

Similarly, programs must also take decisions based on conditions. Without decision-making statements, a program would always behave the same way regardless of input.

Decision-making allows:

- Execution of different code blocks
- Logical problem solving
- Dynamic program behavior
- User-based output generation

C++ provides **conditional statements** to handle such decision-making efficiently.

3. Types of Flow Control Statements in C++

Flow control statements in C++ are broadly divided into:

1. **Selection Statements**
    - if
    - if-else
    - else-if ladder
    - nested if-else
    - switch (not covered here)
2. **Iteration Statements**
    - for

    - while
    - do-while
3. **Jump Statements**
    - break
    - continue
    - goto
    - return

This topic focuses mainly on **selection statements using if-else**.

4. The if Statement

The if statement is the simplest form of decision-making in C++. It executes a block of code **only when a condition is true**.

Syntax

```
if (condition)
{
   statements;
}
```

Explanation

- The condition is evaluated first.
- If the condition is **true (non-zero)**, the statements inside the if block are executed.
- If the condition is **false (zero)**, the statements are skipped.

Example

```
int age = 20;
if (age >= 18)
{
   cout << "Eligible to vote";
}
```

5. Working of the if Statement

The working of an if statement follows these steps:

1. Program reaches the if statement
2. Condition is evaluated
3. If condition is true → execute if block
4. If condition is false → skip if block
5. Control moves to the next statement

Important Points

- Condition can be relational or logical
- Curly braces {} are optional for single statements but recommended

- Zero is treated as false, non-zero as true

6. The if-else Statement

The if-else statement provides **two alternative paths** of execution. It executes one block when the condition is true and another block when the condition is false.

Syntax

```
if (condition)
{
   true block;
}
else
{
   false block;
}
```

Example

```
int number = 5;
if (number % 2 == 0)
{
   cout << "Even number";
}
else
{
   cout << "Odd number";
}
```

7. Working of the if-else Statement

Steps involved:

1. Condition is evaluated
2. If condition is true → if block executes
3. If condition is false → else block executes
4. Only **one block** executes at a time

Advantages

- Clear decision-making
- Improves program logic
- Eliminates ambiguity

8. Flowchart of if-else Statement

A **flowchart** is a diagrammatic representation of program logic.

Flowchart Explanation

1. Start
2. Evaluate condition
3. If condition is true → execute true block
4. If condition is false → execute false block
5. End

Flowcharts help beginners understand the program flow visually and reduce logical errors.

9. The else-if Ladder

The `else-if` ladder is used when **multiple conditions** need to be checked.

Syntax

```
if (condition1)
{
   statement1;
}
else if (condition2)
{
   statement2;
}
else if (condition3)
{
   statement3;
}
else
{
   default statement;
}
```

Example

```
int marks = 75;



if (marks >= 90)

   cout << "Grade A";

else if (marks >= 75)

   cout << "Grade B";

else if (marks >= 50)

   cout << "Grade C";

else

   cout << "Fail";
```

Key Points

- Conditions are checked top to bottom
- First true condition executes
- Remaining conditions are skipped

10. Nested if-else Statement

Nested if-else means placing one if-else inside another.

Syntax

```
if (condition1)
{
   if (condition2)
   {
      statement;
   }
   else
   {
      statement;
   }
}
else
{
   statement;
}
```

Example

```
int a = 10, b = 20;

if (a > b)
{
   cout << "a is greater";
}
else
{
   cout << "b is greater";
}
```

Uses

- Complex decision-making
- Multi-level conditions
- Real-world logical problems

11. Common Errors in if-else Statements

Some common mistakes include:

- Using = instead of ==
- Missing curly braces
- Incorrect logical conditions
- Deep nesting causing confusion

Example of Error

```
if (x = 5)   // Wrong
```

Correct

```
if (x == 5)
```

12. Best Practices for Using if-else

- Use proper indentation
- Keep conditions simple
- Avoid unnecessary nesting
- Use meaningful variable names
- Comment complex logic
- Good practices improve **readability**, **maintainability**, and **debugging**.

13. Advantages of if-else Statements

- Easy to understand
- Flexible decision-making
- Widely used in real applications
- Foundation for advanced logic

14. Applications of if-else in C++

- Login authentication
- Result calculation
- Menu-driven programs
- Game logic
- Input validation
- Banking and billing systems

15. Conclusion

Flow control using **if and if-else statements** is a fundamental concept in C++. It allows programs to make decisions and execute different blocks of code based on conditions. Understanding if-else is essential for building logical, efficient, and real-world applications.

A strong foundation in flow control helps learners progress smoothly to advanced programming concepts.