Control Flow

Control flow determines the order in which instructions are executed. It allows programs to make decisions, repeat actions, and manage complex logic.

1 `if`, `else if`, and `else`

Use when you want the program to take different actions depending on a condition.

```
int score = 85;
if (score >= 90) {
    cout << "Grade: A";
} else if (score >= 80) {
    cout << "Grade: B";
} else {
    cout << "Grade: C or below";
}
```

2 `switch-case` Statement

Use when checking a variable against fixed values (e.g., menu options).

```
int option = 2;
switch (option) {
    case 1:
        cout << "Start game";
        break;
    case 2:
        cout << "Load game";
        break;
    default:
        cout << "Invalid option";
}
```

Each case runs until a break is reached. Without break, the program continues ("falls through") to the next case, which may cause unintended behavior.

3 `while` Loop

Use when you want to repeat something as long as a condition is true.

```
int count = 0;
while (count < 3) {
    cout << "Hello\n";
    count++;
}
```

4 `do-while` Loop

Use when you want to run the loop at least once.

```
int number;
do {
    cout << "Enter a positive number: ";
    cin >> number;
} while (number <= 0);
```

`cin` is used to take input from the user. It reads a value typed into the console and stores it in a variable.

5 `for` Loop

Use when the number of repetitions is known in advance.

```
for (int i = 1; i <= 5; i++) {
    cout << "Step " << i << "\n";
}
```

5.1 Nested Loops (Without Arrays)

Use when repeating something inside another repetition (e.g., patterns).

```
// Print a 3x3 square of stars
for (int row = 0; row < 3; row++) {
    for (int col = 0; col < 3; col++) {
        cout << "* ";
    }
    cout << "\n";
}
```

5.2 Control Statements

`break`, `continue`, and `return` help manage loop execution.

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) continue;   // Skip 3
    if (i == 5) break;      // Stop at 4
    cout << i << " ";
}
```

`continue` skips the rest of the loop body and moves to the next iteration.
`break` exits the loop entirely, even if the condition is still true.

5.3 Ternary Operator

Short form of an `if-else` expression. A ternary operator is a compact way to choose between two values.
Syntax: `condition ? value-if-true : value-if-false`

```
int x = 10, y = 20;
int max = (x > y) ? x : y;
cout << "Max is " << max;
```

5.4 Range-Based `for` Loop (C++11)

Used to loop over collections like strings or vectors.

```
string name = "C++";
for (char ch : name) {
    cout << ch << "\n";
}
```