# Modern Programming Principles & Practice

Feb-May 2025

sruthi.padathara@dorset.ie

# Object Oriented Programming Principles & Introduction C++

## Contents

- Console I/O Statements(cin, cout)
- Programs to perform various calculations
- Operators
- Programs to implement various operators
- Conditional Control Statements
- If-else , switch-case
- Implementing programs on conditional control statements
- Loops: While, do while, for
- Implementing programs on loops
- break, continue, goto keywords

# What is #include <iostream>?

- #include <iostream> is a **header file** that provides functionalities for standard input and output.
- It stands for "**Input-Output Stream**" (iostream).
- It defines input (cin), output (cout), error (cerr), and logging (clog) streams.
- It belongs to the **std (Standard) namespace** in C++.

## Why is #include <iostream> Needed?

### To Enable Input/Output Operations

- Without including `iostream`, you **cannot use** `cin`, `cout`, `cerr`, or `clog`.

### It Defines Stream Objects

- Provides predefined objects for handling text-based input and output.

### It Uses the Stream Buffering System

- Efficient and flexible I/O processing.

# Console I/O Statements(cin, cout)

## Handling Strings with cin and getline()

```
string fullName;
getline(cin, fullName);
```

getline() will capture the entire input line, including spaces.

Formatting with setw(), setprecision() (from <iomanip>)

```
#include <iostream>
#include <iomanip>  // Required for setw and setprecision
using namespace std;

int main() {
        double pi = 3.14159265;
        cout << setw(10) << "Pi: " << fixed << setprecision(2) << pi << endl;
        return 0;
}
```

setw(10): Sets width to 10 spaces.
fixed & setprecision(2): Limits decimal places.

## Programs to perform various calculations

1. Write a C++ program to perform various arithmetic operations, including **addition, subtraction, multiplication, and division**. The program should take two numbers as input from the user using the keyboard and display the result of each operation.
2. Write a C++ program to demonstrate the use of **arithmetic, relational, logical, bitwise, and assignment operators**. The program should take input from the user and display the results of different operations.

# Conditional Control Statements

Conditional control statements allow a program to make decisions based on conditions. These statements help in controlling the flow of execution.

**if Statement**
Used to execute a block of code **only if** a condition is true.
Example Use Case: Checking if a number is positive.

```
if (condition) {

        // Code to execute if condition is true
}
```

**if-else Statement**
Executes one block if the condition is true, otherwise executes another block.
Example Use Case: Checking if a person is eligible to vote.

```
if (condition) {
        // Code if condition is true
}
else {
        // Code if condition is false
}
```

**if-else if-else Ladder**
Used when there are **multiple conditions** to check.
Example Use Case : Assigning grades based on marks.

```
if (condition1) {
        // Code if condition1 is true
} else if (condition2)
{
        // Code if condition2 is true
} else {
        // Code if none of the conditions are true
 }
```

**Nested if Statement**
An if statement inside another if statement.
Example Use Case: Checking if a person is eligible for a loan based on multiple criteria.

```
if (condition1) {
        if (condition2) {
                // Code executes if both conditions are true
        }
 }
```

# switch Statement

Used for **multiple case-based** decisions, an alternative to if-else if.
Example Use Case: Displaying a day of the week based on user input.

```
switch(expression) {

        case value1: // Code for value1
                break;
        case value2: // Code for value2
                break;
        default: // Code if none of the cases match
}
```

1. Write a C++ program that takes a day of the week as input (e.g., "Monday", "Tuesday", etc.) and determines whether it is a **working day** or a **weekend (off day)**. Assume that **Saturday and Sunday** are off days, while the rest are working days.

   **Example Output:**
   *Enter the day of the week: Monday*
   *monday is a working day!*

2. Write a C++ program that asks the user to enter a **day of the week (1-7)**, where **1 represents Monday** and **7 represents Sunday**. The program should use a **switch statement** to determine whether the entered day is a **working day** or a **week off** and display the appropriate message.

   **Example Output:**
   Enter a day number (1 for Monday, 7 for Sunday): 3
   It's a Working Day.

# Loops: While, do while, for

Loops are used to execute a block of code repeatedly as long as a given condition is true. In C++, we have three main types of loops: **while**, **do-while**, and **for**.

**while Loop**
The while loop checks the condition before executing the code block. If the condition is **true**, the code inside the loop is executed. If the condition is **false** initially, the code inside the loop is never executed.

```
while (condition) {
        // Code to execute as long as condition is true
}
```

**Flow:**

1. **Condition** is checked first.
2. If **true**, the loop executes the code block.
3. The condition is checked again after each iteration.

## do-while Loop

The do-while loop executes the code block **at least once**, and then checks the condition after each iteration. Even if the condition is **false** initially, the loop will run **once**.

```
do {
        // Code to execute
} while (condition);
```

**Flow:**

1. The code block is executed first.
2. The condition is checked after executing the code block.
3. If **true**, it repeats. If **false**, the loop stops.

## for Loop

The for loop is used when the number of iterations is known beforehand. It allows initialization, condition-checking, and incrementing/decrementing in a single line.

```
for (initialization; condition; increment/decrement) {
        // Code to execute on each iteration
}
```

**Flow:**

1. **Initialization** is executed once at the beginning.
2. **Condition** is checked before each iteration.
3. **Increment/Decrement** happens at the end of each iteration.

Write a C++ program that asks the user to enter a number and then displays all the numbers from **1 to that number** using a loop. The program should use different types of loops (i.e., while, do-while, and for) to print the numbers.

**Example Output:**

Using while loop:
1 2 3 4 5

Using do-while loop:
1 2 3 4 5

Using for loop:
1 2 3 4 5

# break, continue, goto keywords

In C++, the break, continue, and goto keywords control the flow of the program during looping or conditional execution.

**break Keyword**

The break statement is used to **terminate** a loop or switch statement prematurely, and control is passed to the next statement after the loop or switch.

**Usage:**

- **Inside loops**: It can be used to exit the loop before the condition becomes false.
- **Inside switch-case**: It terminates the case block, exiting the switch statement.

*break;*

Use Case: Exiting a loop when a specific condition is met.

## continue Keyword

The continue statement is used to **skip the current iteration** of a loop and proceed to the next iteration. The code following the continue inside the loop is not executed for that iteration.

**Usage:**

- **In loops**: It forces the loop to skip the current iteration and move to the next iteration of the loop.

*continue;*

**Use Case:**

Skipping an iteration in a loop when a specific condition is met (e.g., skipping even numbers in a loop).

## goto Keyword

The goto statement is used to transfer control to a **specific label** in the program. It allows the program to jump to a different part of the code, marked with a label.

**Usage:**

- **In any part of the program**: It can be used to jump to any labeled line in the program, making control flow less predictable.
- **Avoid excessive use**: It's generally discouraged as it can make code harder to follow and maintain.

*goto label_name;*

And you define the label like this:

```
label_name:
    // Code to jump to
```

Use Case:
Jumping out of nested loops.

Write a C++ program that demonstrates the use of the following keywords:

1. **break**: Exit a loop when a specific condition is met.
2. **continue**: Skip an iteration in a loop when a condition is met.
3. **goto**: Jump to a specific labeled section of the program to demonstrate control transfer.

- The program should display the numbers **1 to 10** and perform the following:
    1. Use a for loop and **break** the loop when the number reaches **6**.
    2. Use another for loop and **continue** the iteration when the number is even.
    3. Use the **goto** keyword to iterate a few times and print the values until a condition is met (i.e., when the variable reaches 3).

**Expected Output:**

Using break:
1 2 3 4 5

Using continue:
1 3 5 7 9

Using goto:
1 2 3
End of program.