Object Oriented programming

C++

@notbeexoul

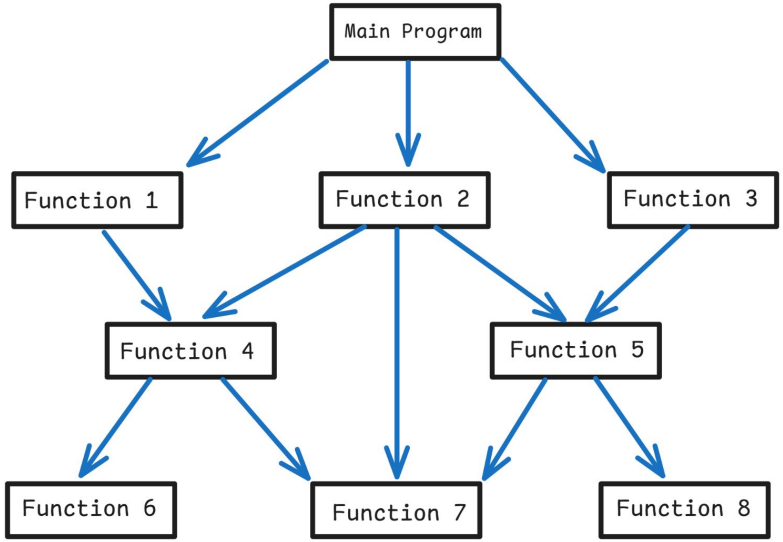
Procedure Oriented Programming

Procedure Oriented Programming (POP) is a programming paradigm centered around **functions or procedures**. It focuses on a **step-by-step approach** to solving problems by breaking them down into smaller tasks (functions).

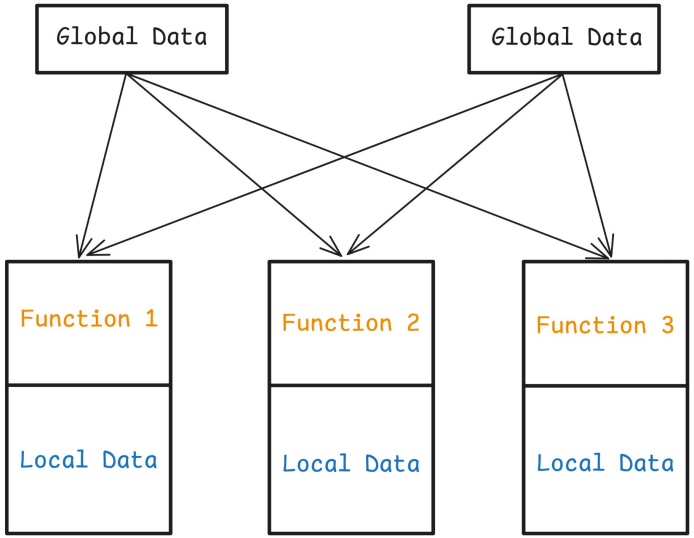
Key Features (in short):

- **Structured approach**: Program is divided into functions or procedures.
- **Top-down design**: Starts from the main function and breaks down into sub-functions.
- **Emphasizes functions**: Code is organized around procedures, not data.
- **Global data**: Functions share and modify global data.
- **Reusability**: Functions can be reused across programs.

Procedure-Oriented Programming



Typical Structure of Procedure- oriented Programming



Relationship of data and function in procedural programming

Object-Oriented Programming

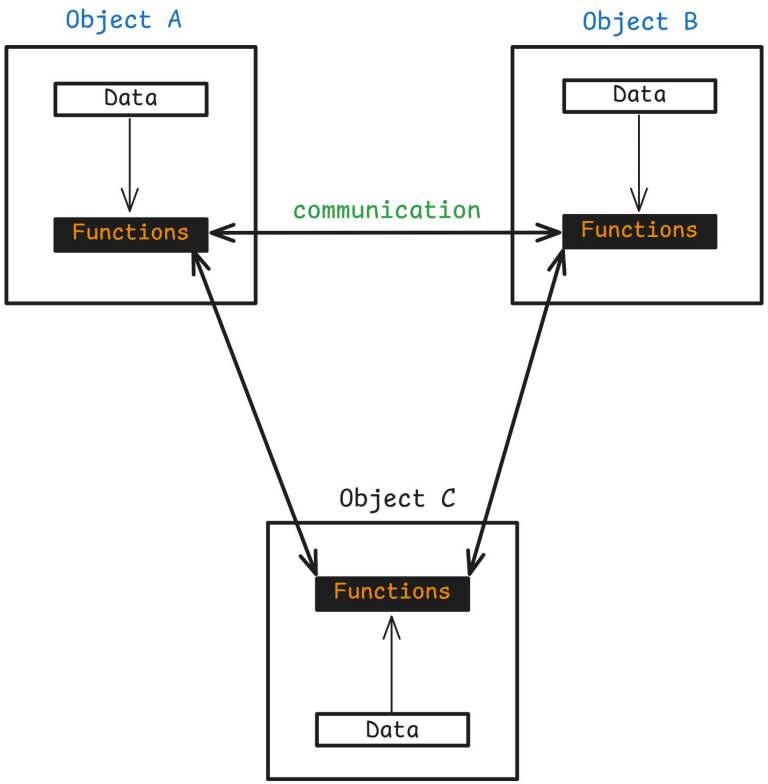
Object-Oriented Programming (OOP) is a programming paradigm based on the concept of **"objects"**, which contain **data (attributes)** and **functions (methods)** that operate on the data.

Key Features :

- **Encapsulation**: Bundles data and methods into a single unit (class).
- **Abstraction**: Hides complex details and shows only the necessary parts.
- **Inheritance**: One class can inherit properties and methods from another.
- **Polymorphism**: Same function can behave differently in different classes.

Advantages:

- Better **code organization**.
- Promotes **reusability** through inheritance.
- Easier to **maintain** and **extend**.
- Improves **security** through encapsulation.



Organization of data and function in OOP

Basic Concepts of Object-Oriented Programming

1. **Class** – Blueprint for creating objects (defines properties and methods).
2. **Object** – Instance of a class; real-world entity in code.
3. **Encapsulation** – Hides data and provides access through methods.
4. **Abstraction** – Shows essential features, hides complex details.
5. **Inheritance** – One class inherits features from another.
6. **Polymorphism** – Same method behaves differently for different objects.

Class

A **class** is a blueprint for creating objects. It defines the properties (data) and behaviors (methods) that the objects will have.

Think of a class as a template for a car: it describes the engine, color, and how it drives—but it's not a car yet.

```
class Car {
public:
    string brand;
    void drive() {
        cout << "Driving..." << endl;
    }
};

```

Object

An **object** is an instance of a class. It’s the actual thing you can work with in your code.

```
Car myCar;     // 'myCar' is an object of class Car
```

Encapsulation

This means **hiding internal details** and only exposing necessary parts. It protects data by keeping it private and accessing it through public methods.

```cpp
class BankAccount {
private:
    int balance;
public:
    void deposit(int amount) { balance += amount; }
    int getBalance() { return balance; }
};
```

Abstraction

It means **showing only essential details** and hiding the complexity.

📦 Like using a mobile: you see buttons (interface),
not the internal circuit (complexity).

Abstraction

Phones can do many things like:-

- Make a call
- Take Pictures
- Play Games

It does not show you the internal working of how things are done

@notbeexoul

Inheritance

It allows a class to **inherit** properties and methods from another class. Promotes code **reusability**.

```cpp
class Vehicle {
public:
    void move() { cout << "Moving...\n"; }
};

class Car : public Vehicle {
    // Inherits move() from Vehicle
};
```

Polymorphism

Poly = many, **morph** = forms. It allows the same method to behave differently based on the object.

- **Compile-time polymorphism** (Function overloading)
- **Run-time polymorphism** (Function overriding)

```cpp
class Animal {
public:
    virtual void sound() { cout << "Animal sound\n"; }
};

class Dog : public Animal {
public:
    void sound() override { cout << "Bark\n"; }
};

```

Beginning with

C++

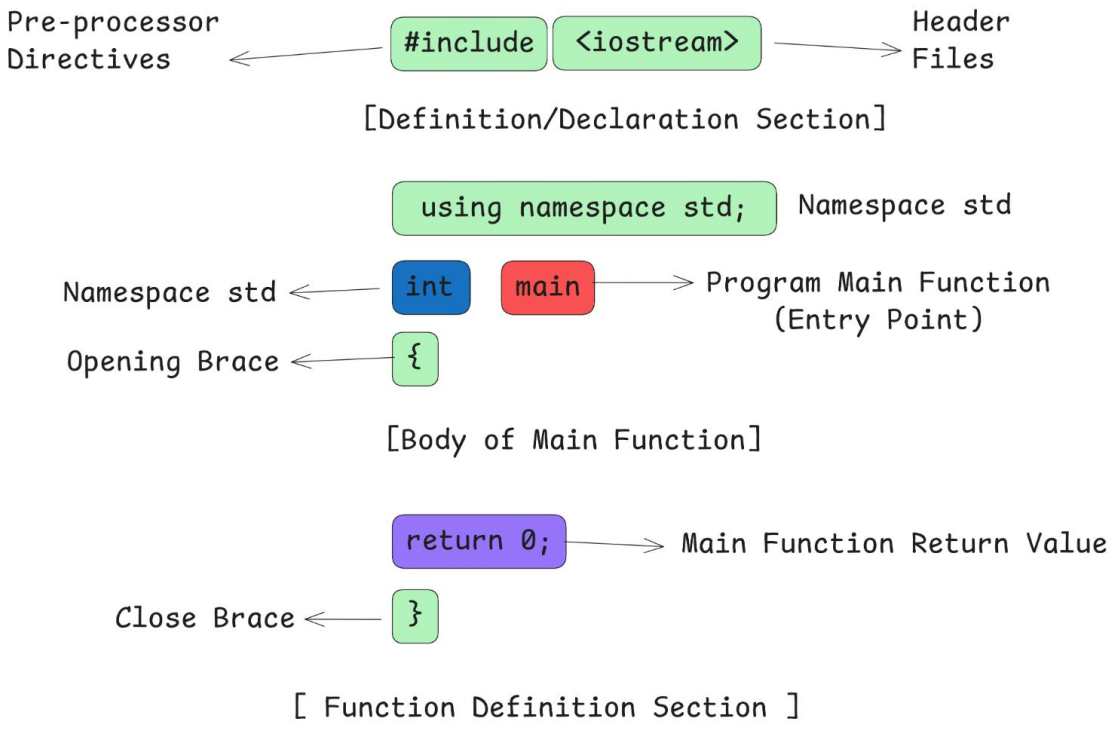
What is C++

C++ is a **general-purpose programming language** that supports both **procedure-oriented** and **object-oriented** programming. It was developed by **Bjarne Stroustrup** in the early 1980s as an extension of the C language.

Key Features of C++ :

- **Compiled** and **high-performance**
- Supports **Object-Oriented Programming** (classes, objects, inheritance, etc.)
- Allows **low-level memory manipulation** (like C)
- **Portable** and can be used across different platforms
- Rich **Standard Template Library (STL)** for data structures and algorithms

The basic Structure of a C++



@notbeexoul

The iostream File

The `iostream` file in C++ is a **header file** that provides functionalities for **input and output (I/O)** operations using **streams**.

Full form:

`iostream` = input/output stream

Purpose:

It allows you to use the standard **cin**, **cout**, **cerr**, and **clog** for input and output.

Some Header File:
1.<utility>
2.<string>
3.<sstream>

Namespace

A **namespace** in C++ is used to **organize code** and **avoid name conflicts**—especially when multiple libraries or programs have functions or variables with the same name.

Why Use Namespace?

To prevent **naming collisions** when different parts of code (or libraries) use the same names.

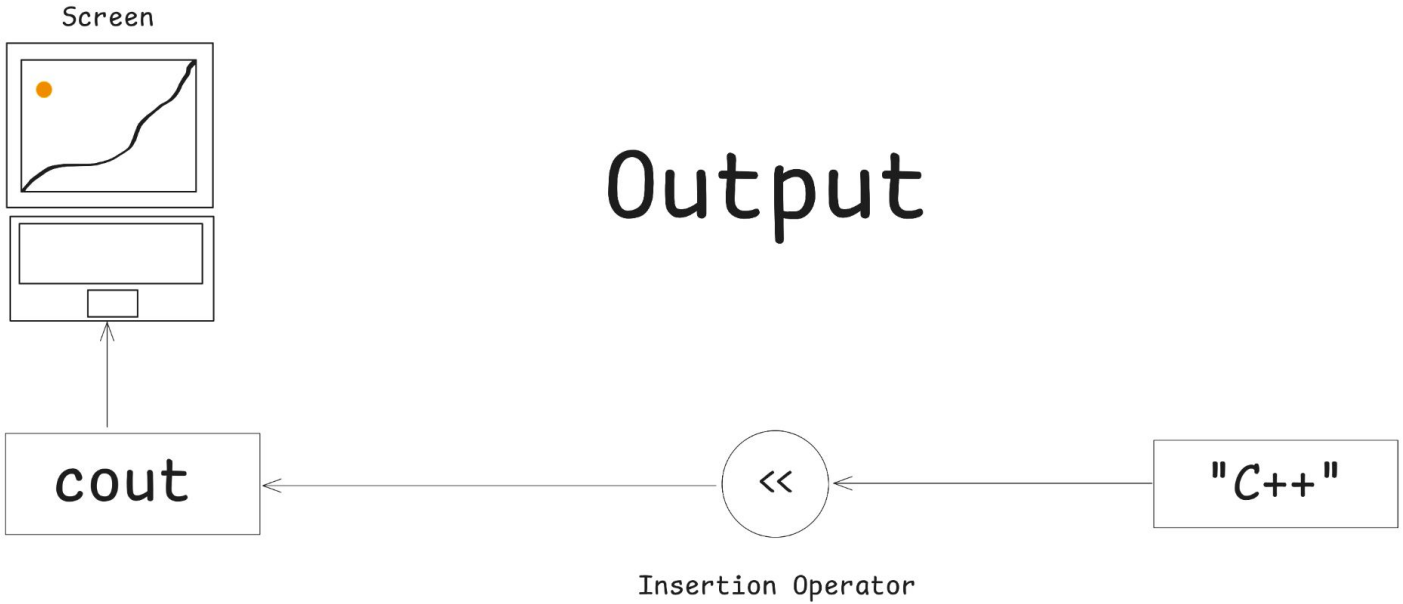
```
using namespace std;
```

This line allows you to use `cout`, `cin`, etc., without writing `std::cout`.

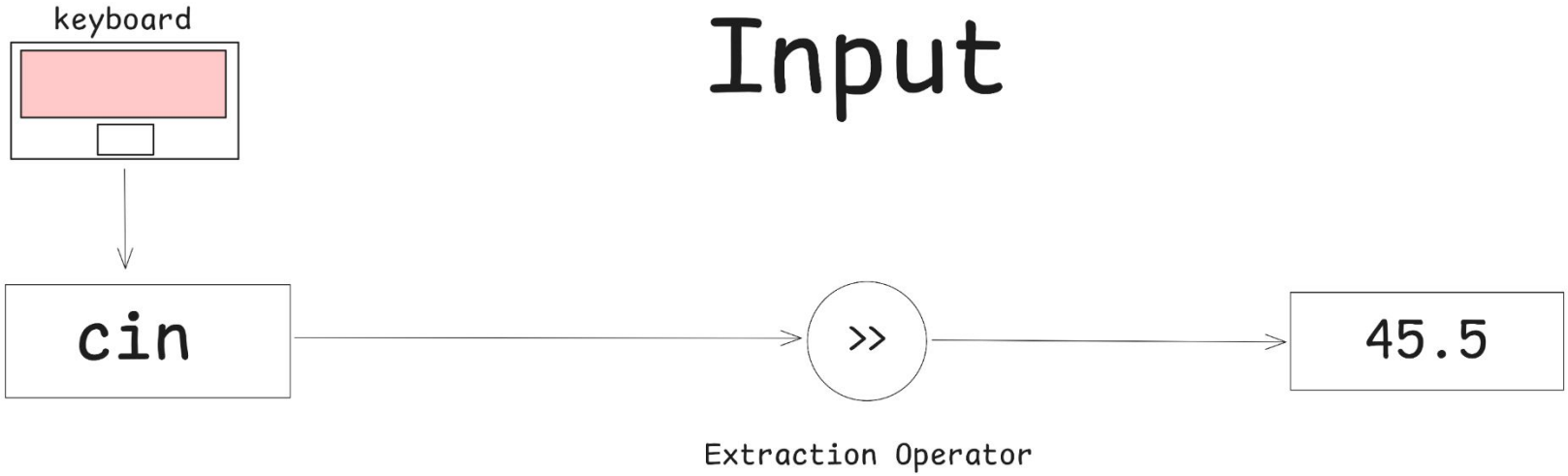
```cpp
#include <iostream>    //include header file
using namespace std;
int main()
{

    cout << "C++ is better than C \n";  //C++ statement
    return 0;
}
```

Input and Output

C++ handles input and output using **streams**, provided by the `iostream` header.



Input



```
cout<<"output";
cout<<"output"<<endl;
cin<<a;
cin>>b;
```

Cin can read only one word and therefore we cannot use names with blank spaces.

Thank you

Shiva Raj Paudel (Beexoul)

Gmail: notbeexoul@gmail.com

Website: shivarajpaudel.com.np