CHAPTER-3

Classes and objects

Classes /specifying a class:

A **class** is a user-defined data type in object-oriented programming that acts as a **blueprint** for creating objects. It groups **data members** (variables) and **member functions** (methods) into a single unit. A class defines the properties and behaviors that the objects created from it will have.

Classes /specifying a class:

Key Features of a class

- A class is a template used to create multiple objects of the same type.
- It provides a way to bind data and functions together.
- It allows data hiding using access specifiers like private, public, and protected.
- It supports the concepts of encapsulation, abstraction, and reuse.
- A class behaves like a built-in data type — once defined, it can be used many times to create different objects.

Structure of class in c++

While defining a class, the class specification has two main parts:

- Class Declaration
- Function Definition

Prepared by Ayush Lamsal | |
layush788@gmail.com

Structure of class in c++

While defining a class, the class specification has two main parts:

- Class Declaration
- Class function Definition

Structure of class in c++

Class Declaration:

- This part defines the structure of the class.
- It includes the class name, data members, and function prototypes (without full code).
- It also specifies the access specifiers (public, private, protected) that define the scope and accessibility of members.
- The keyword class is used while declaring the class

Note:

The variable declared inside the class are known as a data member and the function declared are known as a member function

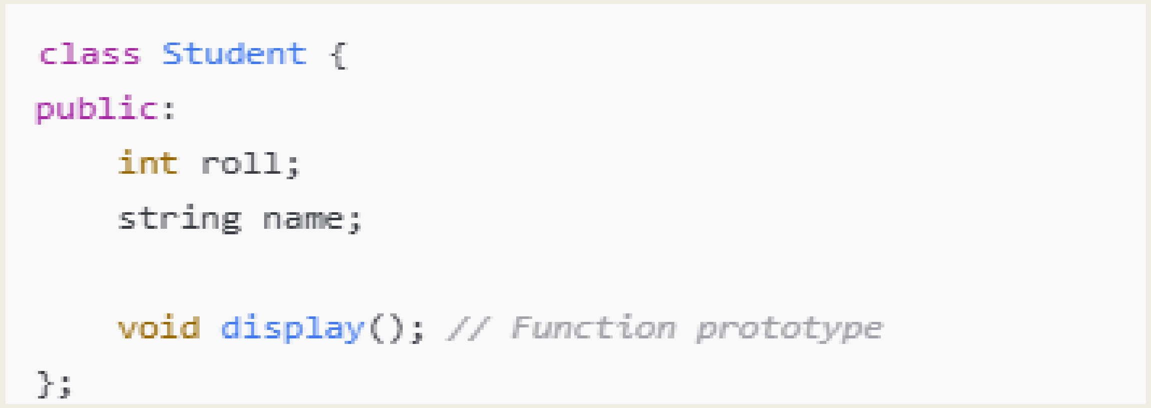
If the private and public is not specified than data member are private by default. The data member are usually declared as private and member function are declared as public

Prepared by Ayush Lamsal | |
layush788@gmail.com

Structure of class in c++

Class Declaration:

- This part defines the structure of the class.
- It includes the class name, data members, and function prototypes (without full code).
- It also specifies the access specifiers (public, private, protected) that define the scope and accessibility of members.
- The keyword class is used while declaring the class



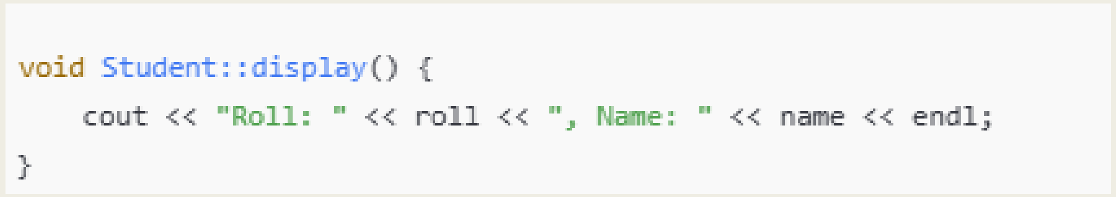
```cpp
class Student {
public:
    int roll;
    string name;

    void display(); // Function prototype
};
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

Structure of class in c++

Function Definition:

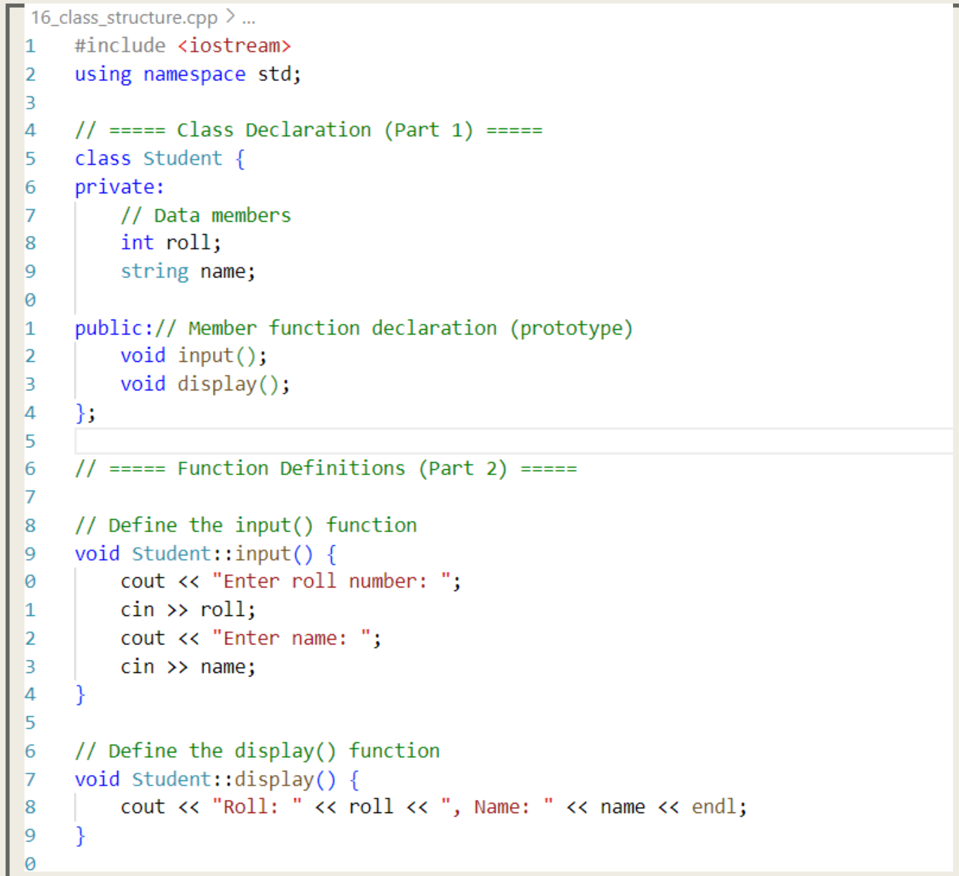
- This part provides the actual implementation of the functions that were declared in the class.
- It is usually written outside the class using the scope resolution operator (: :)



```
void Student::display() {
    cout << "Roll: " << roll << ", Name: " << name << endl;
}
```

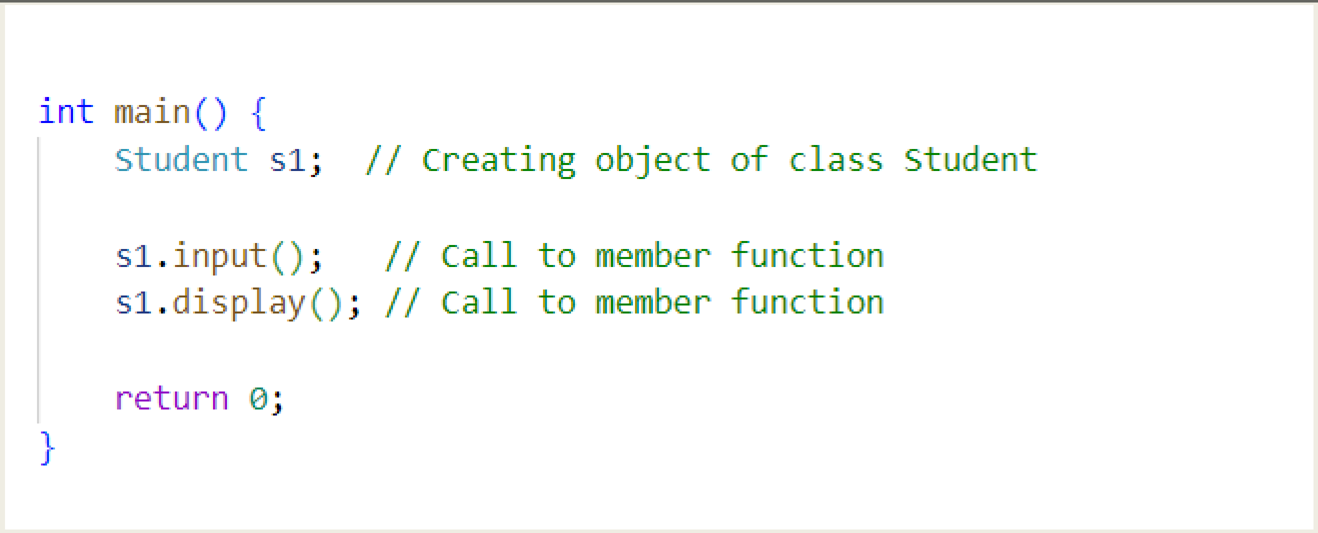
Prepared by Ayush Lamsal | |
layush788@gmail.com

Complete structure example: Class Declaration + Function Definition



16_class_structure.cpp > ...

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

// ===== Class Declaration (Part 1) =====
class Student {
private:
    // Data members
    int roll;
    string name;

public:// Member function declaration (prototype)
    void input();
    void display();
};

// ===== Function Definitions (Part 2) =====

// Define the input() function
void Student::input() {
    cout << "Enter roll number: ";
    cin >> roll;
    cout << "Enter name: ";
    cin >> name;
}

// Define the display() function
void Student::display() {
    cout << "Roll: " << roll << ", Name: " << name << endl;
}

```



```cpp
int main() {
    Student s1;  // Creating object of class Student

    s1.input();   // Call to member function
    s1.display(); // Call to member function

    return 0;
}
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

Accessing class member

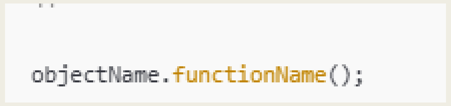
To access the data members and member functions of a class, we use the dot operator (.) with the object of the class.

Generally, the data members of a class are kept private to protect (hide) the internal state of the object from direct access or modification from outside.

Member functions are kept public so they can provide controlled access to the private data.

As we know, the private data of a class cannot be accessed directly from outside the class (including from main()). To access or modify private data, we use public member functions of the class. In main(), we cannot directly access private data, but we can call the public member functions, which internally access those private members.

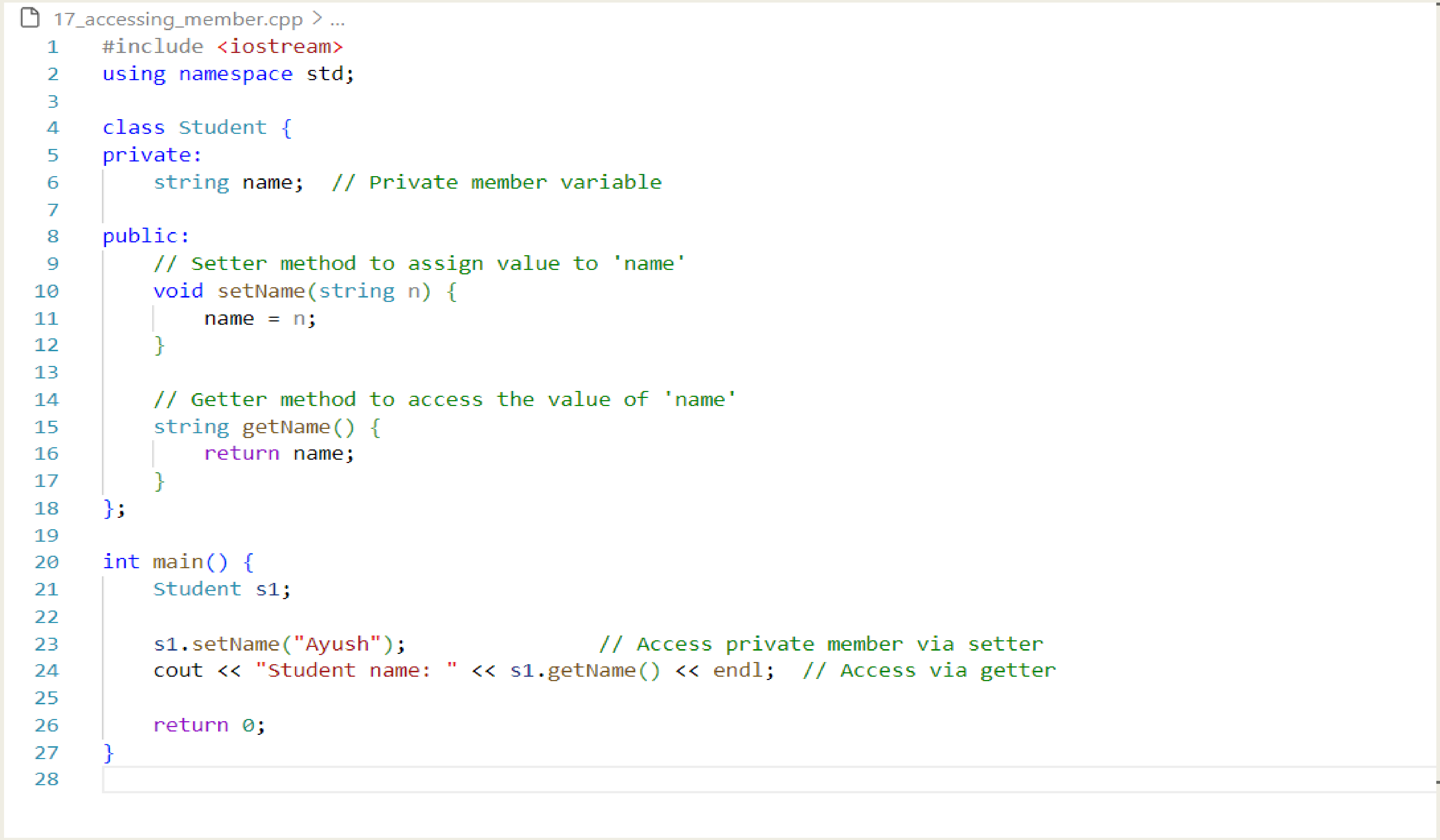
<u>Format for calling member function in main</u>



```
objectName.functionName();
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

Accessing member variable through method



17_accessing_member.cpp > ...

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

class Student {
private:
    string name;  // Private member variable

public:
    // Setter method to assign value to 'name'
    void setName(string n) {
        name = n;
    }

    // Getter method to access the value of 'name'
    string getName() {
        return name;
    }
};

int main() {
    Student s1;

    s1.setName("Ayush");                // Access private member via setter
    cout << "Student name: " << s1.getName() << endl;  // Access via getter

    return 0;
}

```

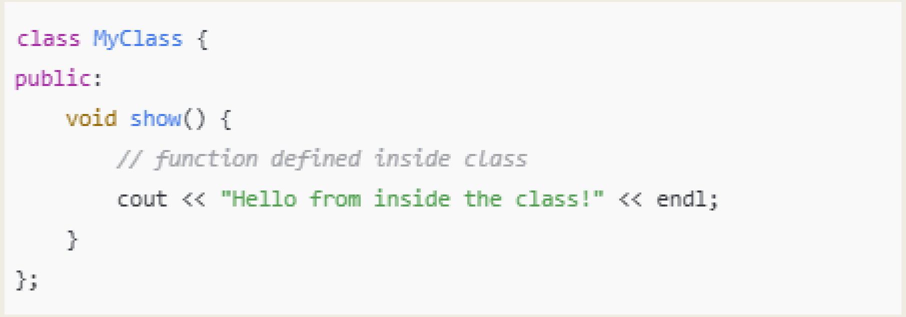
Defining a member function

Member functions can be defined in two places:

1. **Inside the class definition**
2. **Outside the class definition**

Inside the class definition

- When you define a member function inside the class, the function body is written right where it is declared.

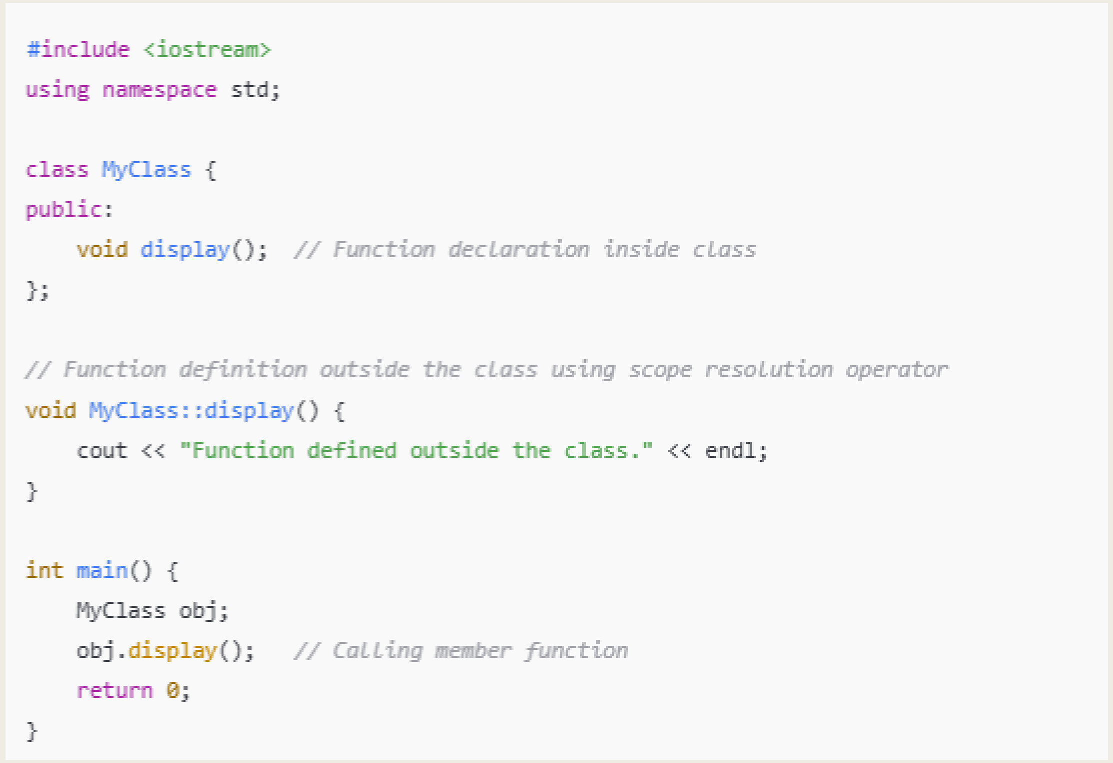


```cpp
class MyClass {
public:
    void show() {
        // function defined inside class
        cout << "Hello from inside the class!" << endl;
    }
};
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

Defining a member function

- When you declare a member function inside the class but define it outside, you need to specify that the function belongs to that class using the scope resolution operator : :
- This keeps the class declaration clean and separates interface from implementation.



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

class MyClass {
public:
    void display();  // Function declaration inside class
};

// Function definition outside the class using scope resolution operator
void MyClass::display() {
    cout << "Function defined outside the class." << endl;
}

int main() {
    MyClass obj;
    obj.display();   // Calling member function
    return 0;
}
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

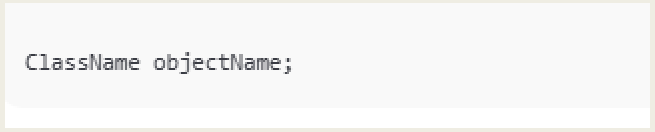
Defining a member function

Note:

Usually, small/simple functions like getters/setters are defined **inside** the class, while bigger or more complex functions are defined **outside** for better organization.

Creating objects

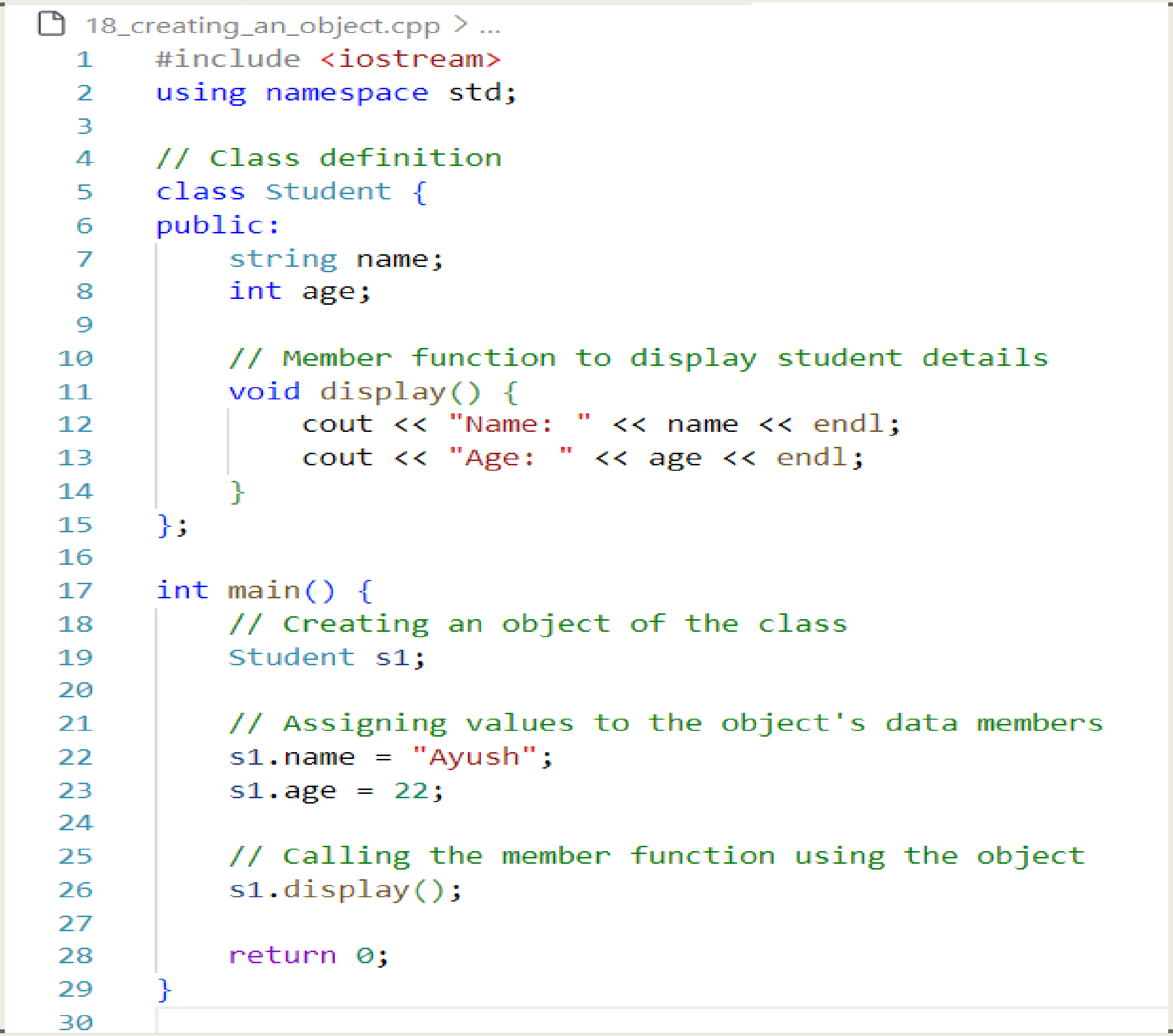
- An object is an instance of a class.
- It is created to use the properties (data members) and behavior (member functions) defined in the class.
- Declaring a class does not create any object. It only defines what kind of data and functions the object will contain.
- After a class is declared, we can create variables of that class type. These variables are called objects in C++.
- In C++, class variables are known as objects.
- To create an object form a class , you use the flowing syntax:



```
ClassName objectName;
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

Example of creating a object in the class


18_creating_an_object.cpp > ...

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

// Class definition
class Student {
public:
    string name;
    int age;

    // Member function to display student details
    void display() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};

int main() {
    // Creating an object of the class
    Student s1;

    // Assigning values to the object's data members
    s1.name = "Ayush";
    s1.age = 22;

    // Calling the member function using the object
    s1.display();

    return 0;
}
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

Access specifiers /Access modifiers in c++

- An **access specifier** in C++ defines the **scope** or **visibility** of **class members** (data members and member functions)**.**It controls **how and from where** the members of a class can be accessed**.**
- There are three types of access specifiers: **Public, Private , Protected**

Prepared by Ayush Lamsal | |
layush788@gmail.com

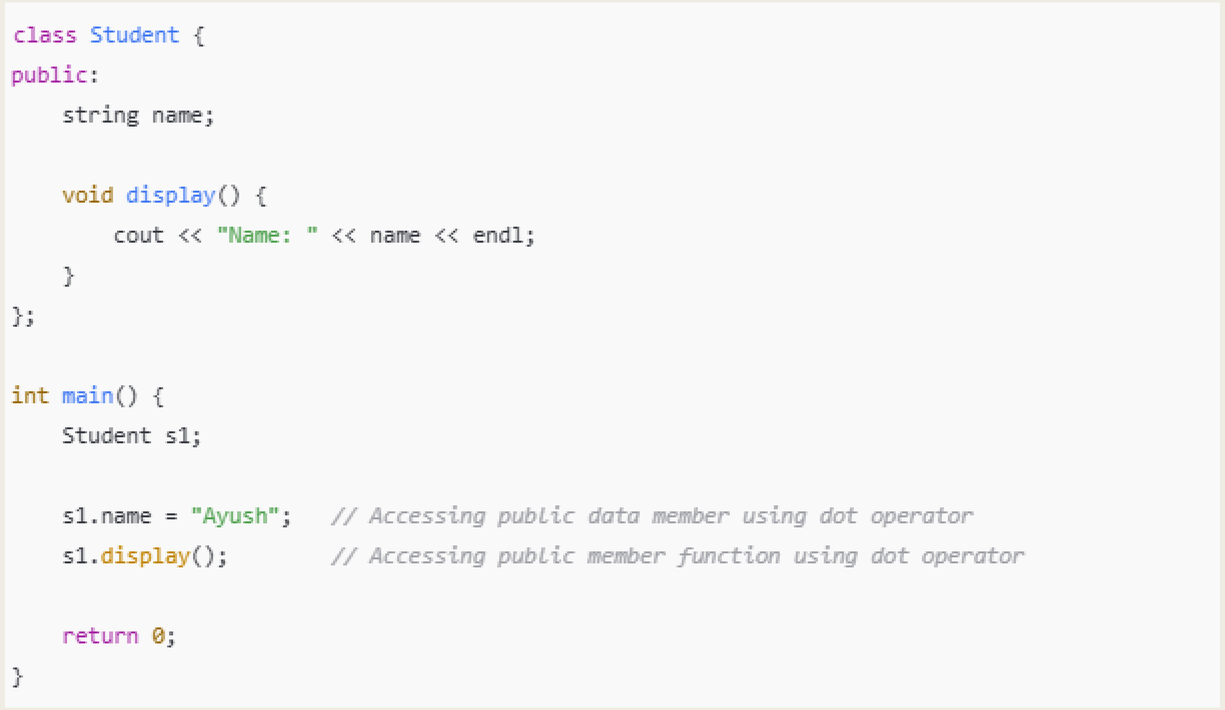
Public Access Specifiers

All the class members declared under the public access specifier are accessible from anywhere in the program.

The data members and member functions declared as public in a class can be accessed by other classes as well

The public members of a class can be accessed from anywhere in the program. They are accessed using access operator (.) with the object of the class.

Public Access Specifiers Example



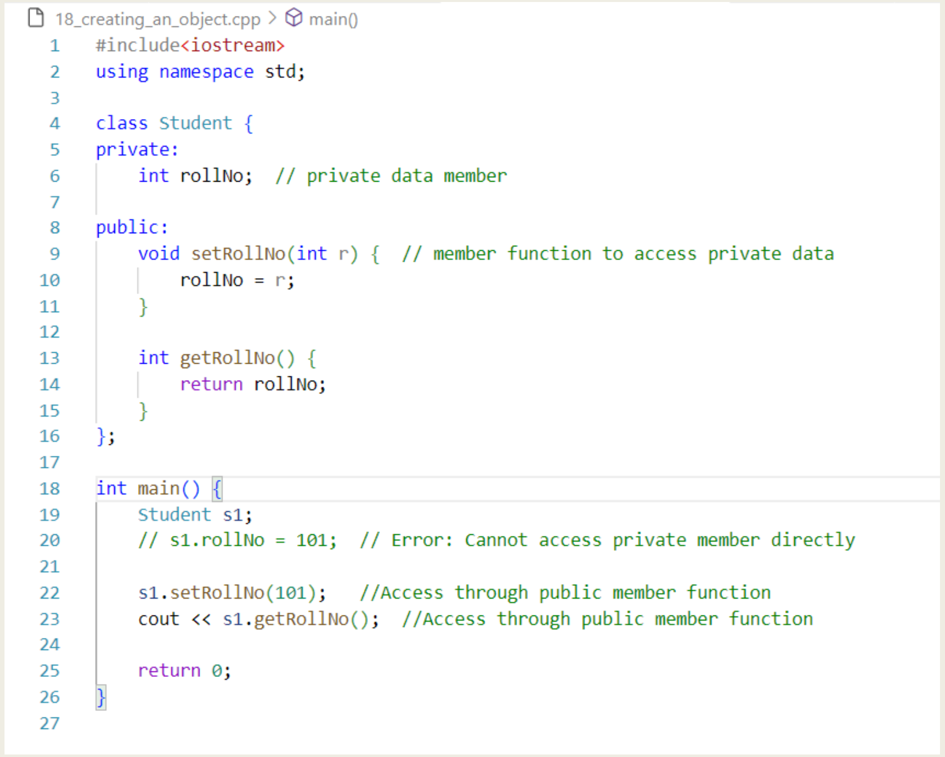
```
class Student {
public:
    string name;

    void display() {
        cout << "Name: " << name << endl;
    }
};

int main() {
    Student s1;

    s1.name = "Ayush";   // Accessing public data member using dot operator
    s1.display();        // Accessing public member function using dot operator

    return 0;
}
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

Private Access Specifiers

- The class members declared as private can be accessed only by the member functions inside the same class.
- They cannot be accessed directly by any object or function outside the class.
- Only member functions or friend functions (or friend classes) are allowed to access the private data members of a class.

Private Access Specifiers Example


18_creating_an_object.cpp > main()

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

class Student {
private:
    int rollNo;  // private data member

public:
    void setRollNo(int r) {  // member function to access private data
        rollNo = r;
    }

    int getRollNo() {
        return rollNo;
    }
};

int main() {
    Student s1;
    // s1.rollNo = 101;  // Error: Cannot access private member directly

    s1.setRollNo(101);   //Access through public member function
    cout << s1.getRollNo();  //Access through public member function

    return 0;
}

```

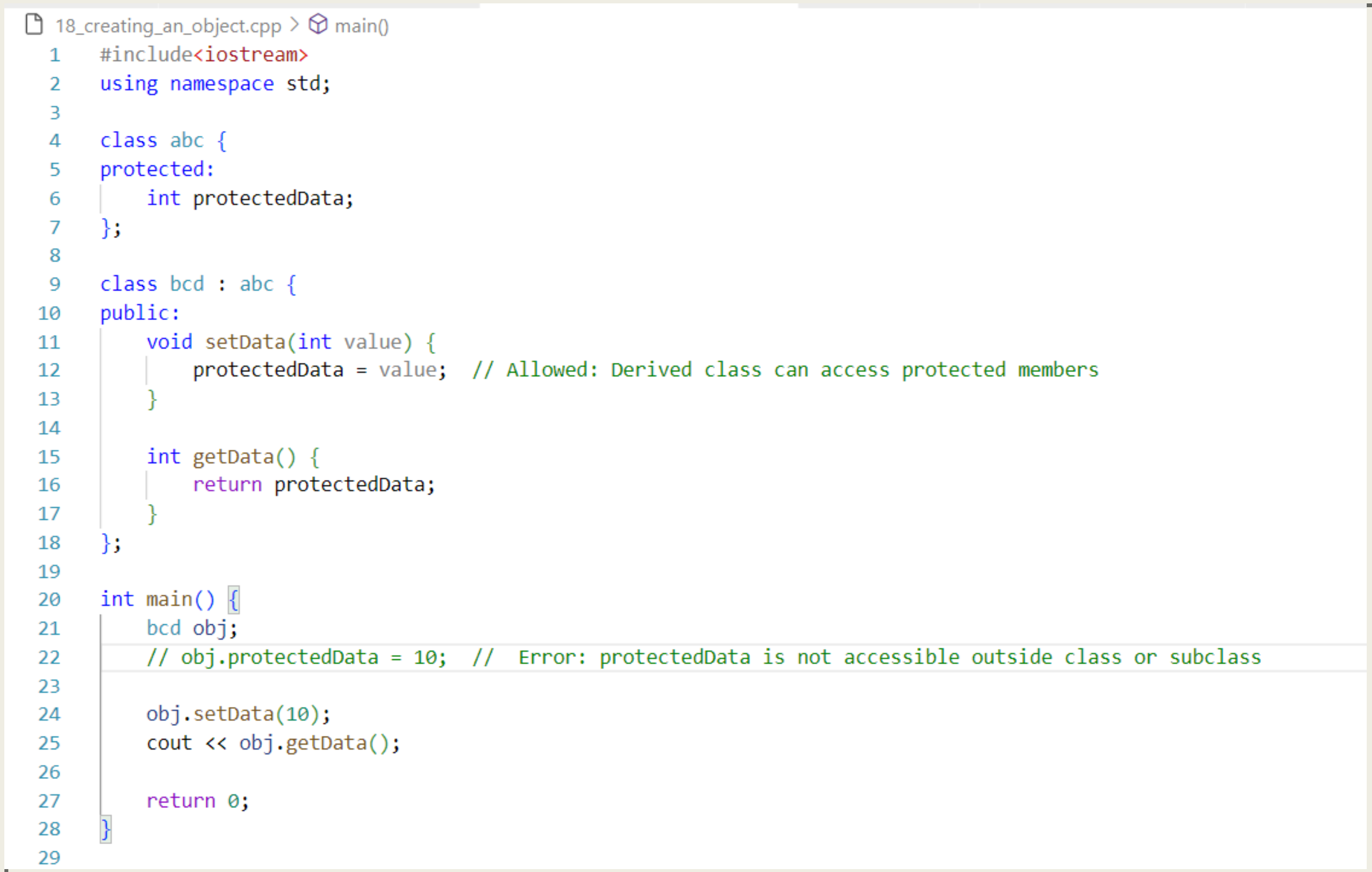
Protected Access specifiers

- **Protected Access Specifier**

- The **protected** access specifier is **similar to private**, but with one important difference.
- Members declared as **protected** are **not accessible outside the class** (just like private members).
- However, **protected members can be accessed by derived (sub) classes** that inherit from the base class.
- This allows subclasses to use or modify inherited protected members, while still preventing access from unrelated classes or code.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Protected Access specifiers


18_creating_an_object.cpp > main()

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

class abc {
protected:
    int protectedData;
};

class bcd : abc {
public:
    void setData(int value) {
        protectedData = value;  // Allowed: Derived class can access protected members
    }

    int getData() {
        return protectedData;
    }
};

int main() {
    bcd obj;
    // obj.protectedData = 10;  //  Error: protectedData is not accessible outside class or subclass

    obj.setData(10);
    cout << obj.getData();

    return 0;
}

```

Prepared by Ayush Lamsal | |
layush788@gmail.com

Some important question for Exam.

1. Program for calculating area and perimeter of rectangle by user input data using concept of class and object
2. Program to display general information of the student like account no. phone number , roll no using class and object
3. Program to covert Indian currency into Nepali currency using the concept of class and objects
4. Write a program to check whether to check whether the given number is prime or object using the concept of class and object

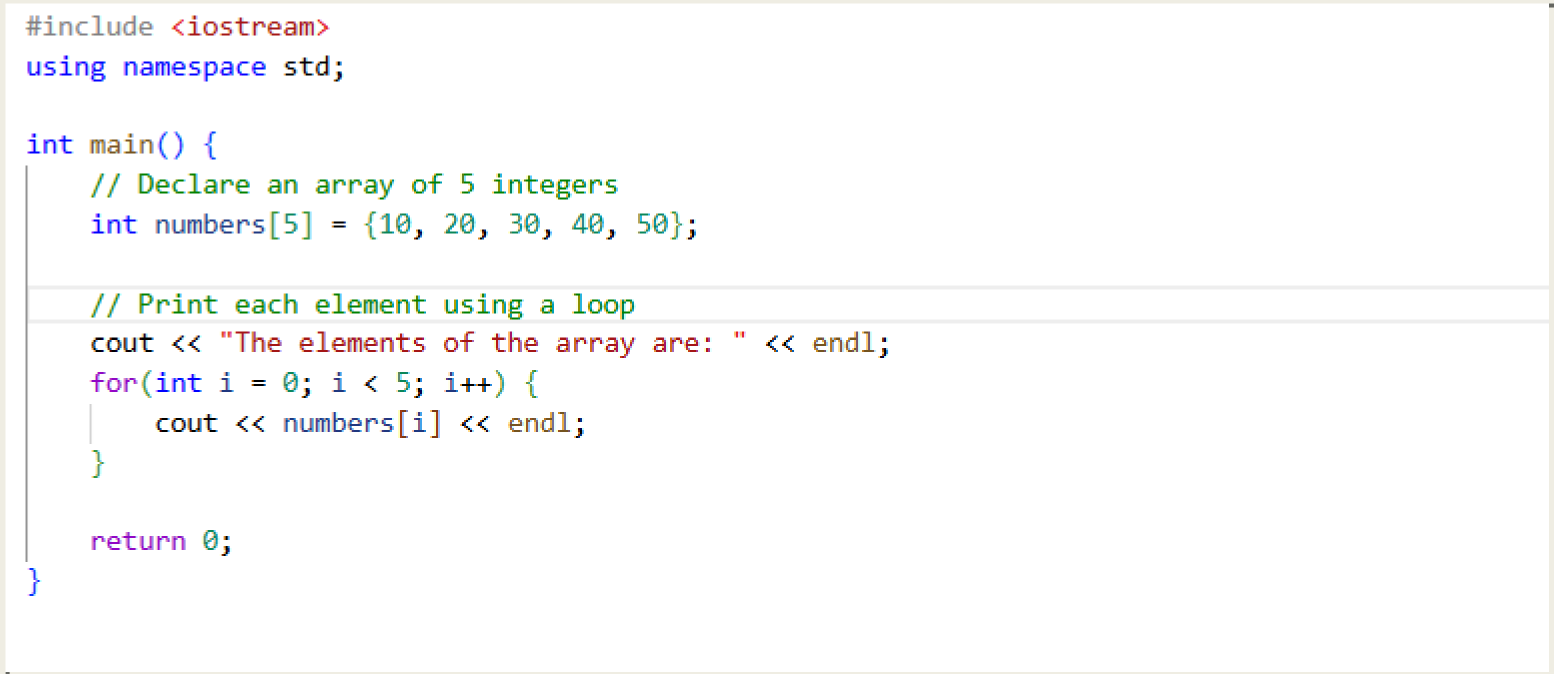
Arrays within a class

We can also declare array as a data member inside a class to store multiple values of the same types.

How to use Array inside a class

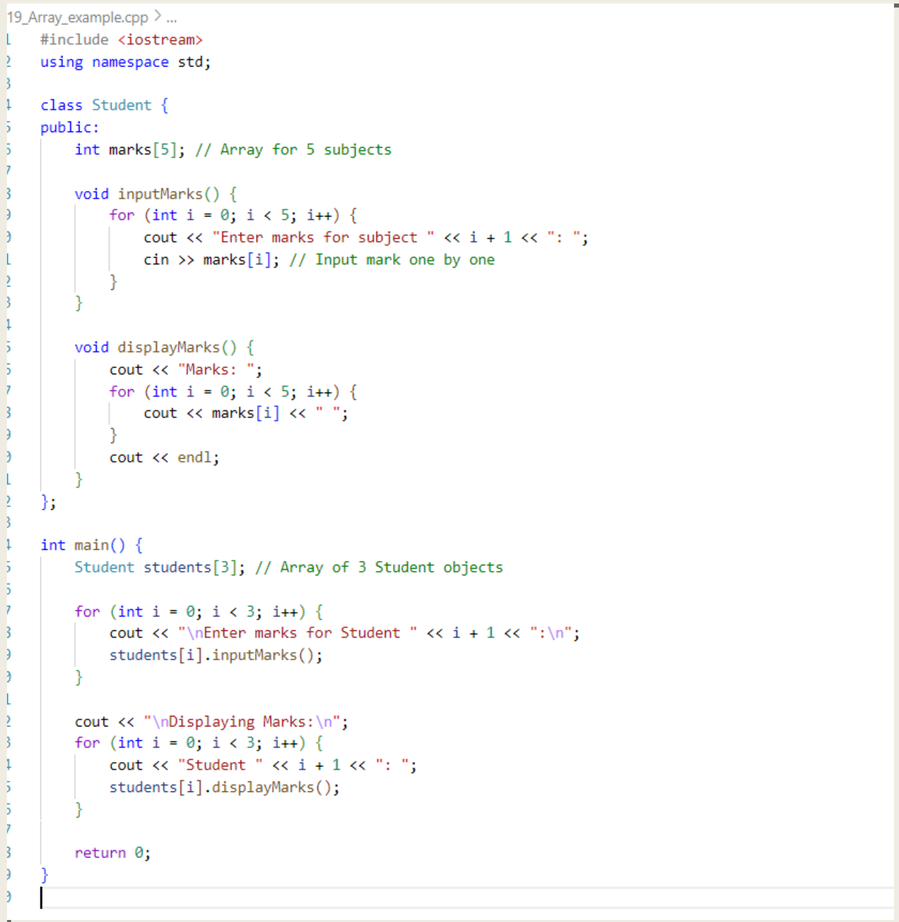
- Declare an array just like a normal variable inside the class.
- You can access or modify array elements using member functions.
- Arrays inside classes are useful to group related data together.

Simple example of Array



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

int main() {
    // Declare an array of 5 integers
    int numbers[5] = {10, 20, 30, 40, 50};

    // Print each element using a loop
    cout << "The elements of the array are: " << endl;
    for(int i = 0; i < 5; i++) {
        cout << numbers[i] << endl;
    }

    return 0;
}
```

Example: Store marks of 3 object and 5 subjects and using array inside a class


19_Array_example.cpp > ...

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

class Student {
public:
    int marks[5]; // Array for 5 subjects

    void inputMarks() {
        for (int i = 0; i < 5; i++) {
            cout << "Enter marks for subject " << i + 1 << ": ";
            cin >> marks[i]; // Input mark one by one
        }
    }

    void displayMarks() {
        cout << "Marks: ";
        for (int i = 0; i < 5; i++) {
            cout << marks[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    Student students[3]; // Array of 3 Student objects

    for (int i = 0; i < 3; i++) {
        cout << "\nEnter marks for Student " << i + 1 << ":\n";
        students[i].inputMarks();
    }

    cout << "\nDisplaying Marks:\n";
    for (int i = 0; i < 3; i++) {
        cout << "Student " << i + 1 << ": ";
        students[i].displayMarks();
    }

    return 0;
}
```

Note:

- By writing Student students[3];We are creating an array of objects, meaning three separate Student objects exist in memory.
- Each **object is independent and holds its own data and is accessed using index**
- specifically: Each Student object has an array marks[5] that stores marks for five subjects.
- The students[0], students[1], and students[2] instances are individual objects, each with their own marks array.

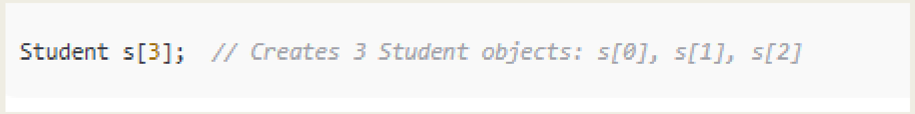
This is also an example for array of object

What is array object?

In **C++**, an **array of objects** is a collection of **multiple instances of a class** stored together in a single array.

Just like we can create an array of integers or floats, we can also create an array of class-type objects.

An array of objects means: A list of objects created from the same class, stored in an array format and accessed using indexes.

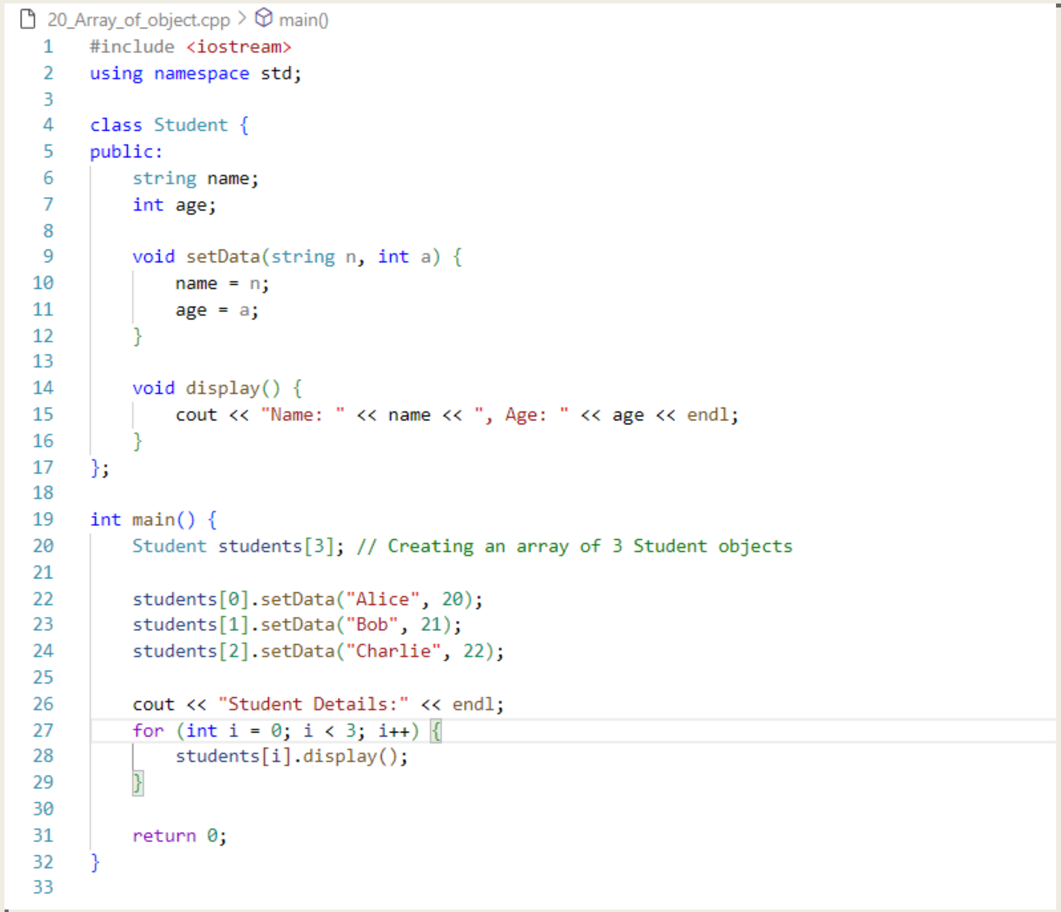


```
Student s[3];  // Creates 3 Student objects: s[0], s[1], s[2]
```

In **array of objects** in C++ is used when we need to store and manage **multiple objects** of the same class.

Instead of creating each object separately, we can use an array to hold many objects. This makes our program **clean**, **efficient**, and **easy to handle** using loops.

Simple example of array of object



20_Array_of_object.cpp > main()

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

class Student {
public:
    string name;
    int age;

    void setData(string n, int a) {
        name = n;
        age = a;
    }

    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Student students[3]; // Creating an array of 3 Student objects

    students[0].setData("Alice", 20);
    students[1].setData("Bob", 21);
    students[2].setData("Charlie", 22);

    cout << "Student Details:" << endl;
    for (int i = 0; i < 3; i++) {
        students[i].display();
    }

    return 0;
}
```

Memory Allocation for object

In C++, a **class** serves as a blueprint for creating objects. It contains **data members** (variables) and **member functions** (methods). Understanding how memory is allocated for these components is crucial for efficient programming.

Memory Allocation for Data Members

- When a class is defined, no memory is allocated for its data members.
- Memory for data members is allocated only when an object of the class is created.
- Each object has its own separate memory space for storing its data members.
- For example, if a class Book has data members title and price, and three objects book1, book2, and book3 are created, each object will have its own copy of title and price.
- This allows each object to store different values for its data members.

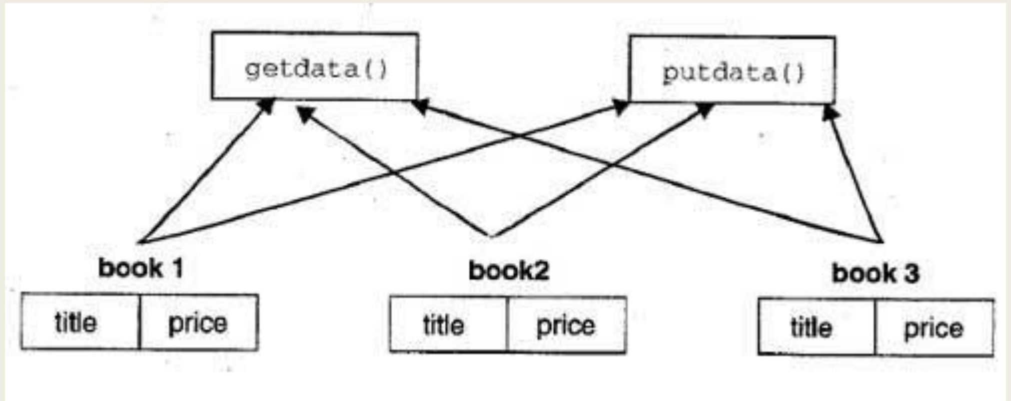
Memory Allocation for object

Memory Allocation for Member Functions

- Unlike data members, member functions do not consume memory for each object.
- Only one copy of each member function is stored in memory, regardless of how many objects are created.
- This means that all objects share the same code for member functions.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Memory Allocation for object



Memory Allocation for the Objects of the Class book

- Objects: book1, book2, book3
- Each has separate memory for data members (title, price)
- All share the same member function code {getdata() , putdata() }

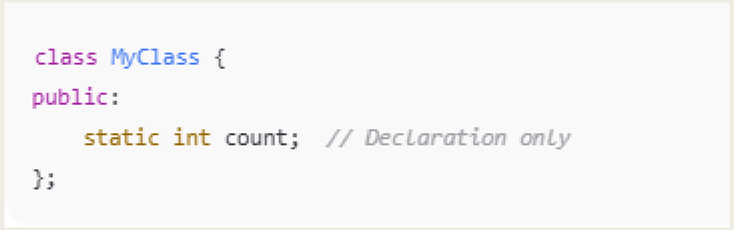
Static Data Member

- **Definition:**
- A **static data member** of a class is a variable that is **shared by all objects** of that class.
- Unlike regular data members, which have separate copies for each object, a static data member has **only one copy** that all objects access and share.
- So the static member in a class can be declared in two steps : **Declaration and Definition**

Static Data Member

- **Declaration**

- When you **declare** a static data member **inside the class**, you only tell the compiler that the class **has** this static member.
- This declaration does **not** allocate memory or assign a value.
- Syntax Inside the class:

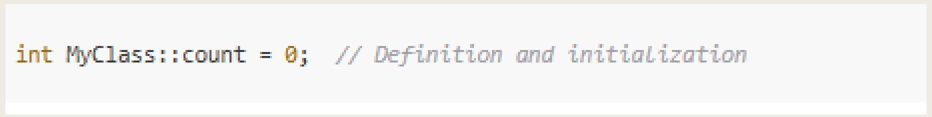


```
class MyClass {
public:
    static int count;  // Declaration only
};
```

Static Data Member

2. Definition

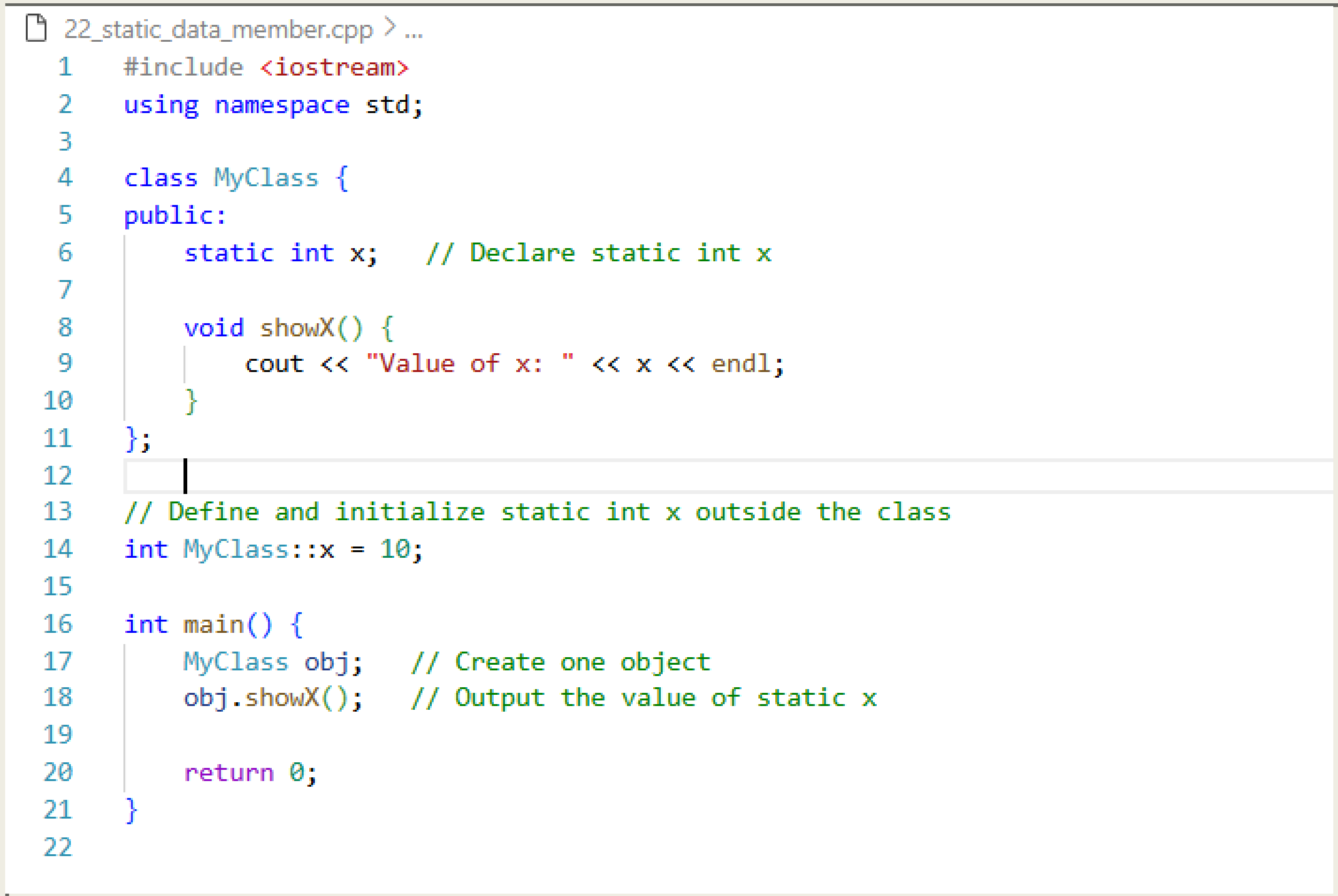
- The definition actually allocates memory for the static data member.
- It must be done outside the class using the scope resolution operator ::.
- You can also assign an initial value here.
- Syntax outside the class:



```
int MyClass::count = 0;  // Definition and initialization
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

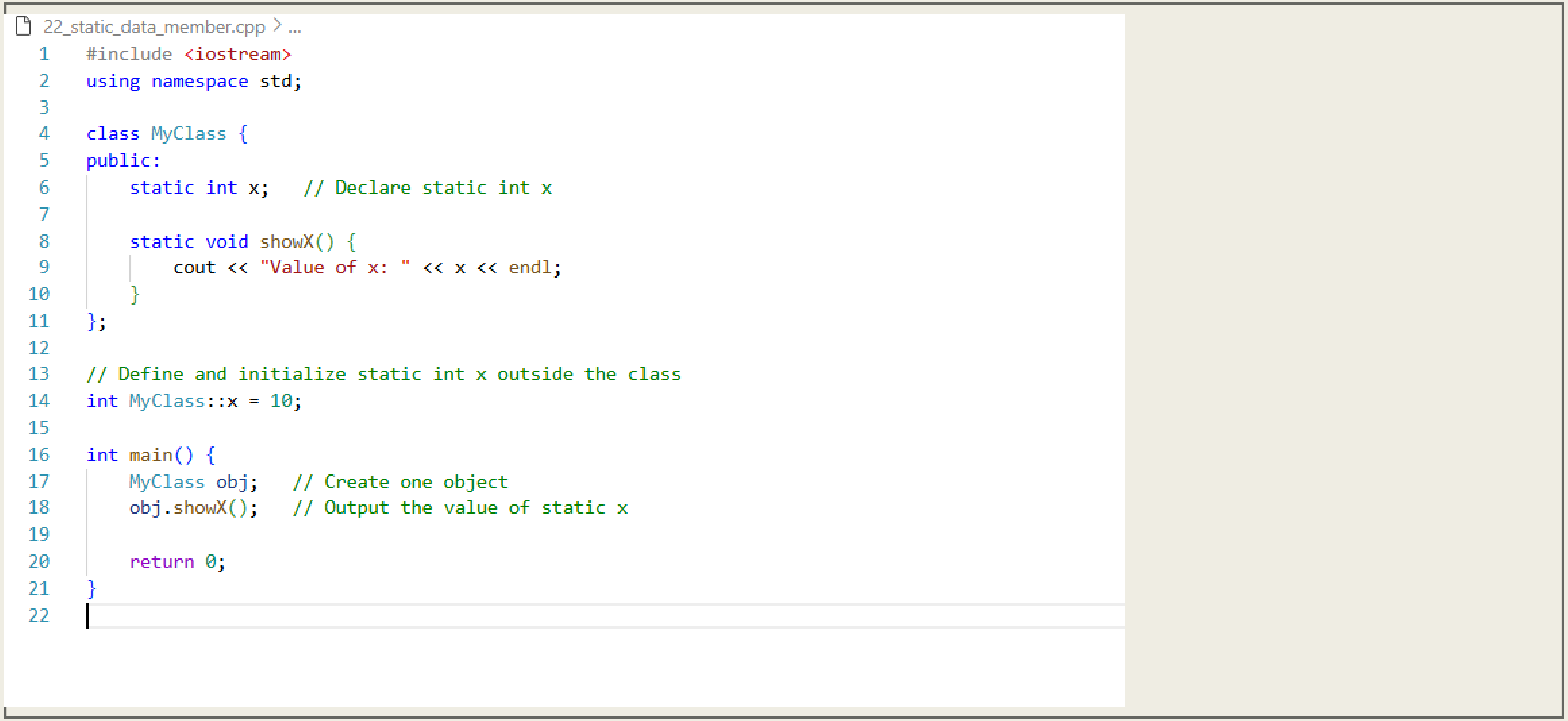
Simple Example



22_static_data_member.cpp > ...

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

class MyClass {
public:
    static int x;   // Declare static int x

    void showX() {
        cout << "Value of x: " << x << endl;
    }
};

// Define and initialize static int x outside the class
int MyClass::x = 10;

int main() {
    MyClass obj;   // Create one object
    obj.showX();   // Output the value of static x

    return 0;
}
```

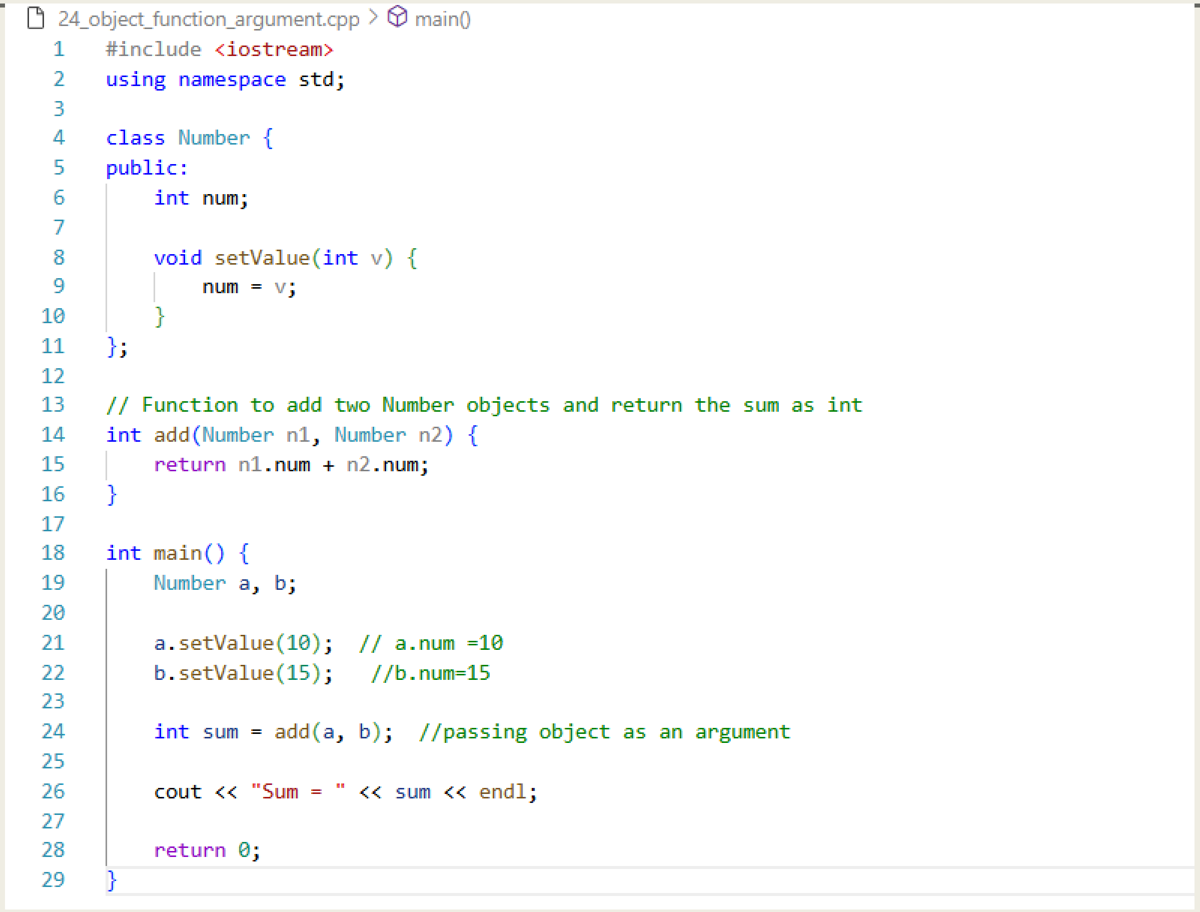
Same previous example with static data member with static member function



22_static_data_member.cpp > ...

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

class MyClass {
public:
    static int x;   // Declare static int x

    static void showX() {
        cout << "Value of x: " << x << endl;
    }
};

// Define and initialize static int x outside the class
int MyClass::x = 10;

int main() {
    MyClass obj;   // Create one object
    obj.showX();   // Output the value of static x

    return 0;
}

```

Object as function arguments



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

class Number {
public:
    int num;

    void setValue(int v) {
        num = v;
    }
};

// Function to add two Number objects and return the sum as int
int add(Number n1, Number n2) {
    return n1.num + n2.num;
}

int main() {
    Number a, b;

    a.setValue(10);  // a.num =10
    b.setValue(15);   //b.num=15

    int sum = add(a, b);  //passing object as an argument

    cout << "Sum = " << sum << endl;

    return 0;
}
```

What is constructor?

- A constructor is a special method that is present inside the class and gets automatically called when an object is created.
- It is used to initialize variable or allocate resource
- In C++, a constructor is a special member function that has the same name as the class and no return type not even a void
- Instead of using separate **set methods** to assign values to variables after creating an object, you can use a **constructor** to initialize those values automatically when the object is created.

Need of a constructor?

- When we create many objects of a class, like 100 Person objects, it becomes difficult to set values for each one manually. For example, if we want the age of every person to be 0 at the beginning, writing code to set age = 0 for every object is time-consuming and repetitive.
- To solve this, we use a constructor. A constructor is a special function inside the class that runs automatically when an object is created. So, if we write a constructor that sets age = 0, then every time we make a new Person object, its age will already be 0 without writing extra code.

Characteristics of constructor?

- Constructor should have the same name as the class.
- Constructor should be declared in the public section of the class.
- They are automatically called when the objects are created.
- They don’t have a return type, not even void, so they cannot return any value.
- Constructor cannot be inherited from another class.
- Constructor is always called once, when the object is created.
- Constructors are used for initializing the data members of the class.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Types of constructor

There are basically three types of constructor:

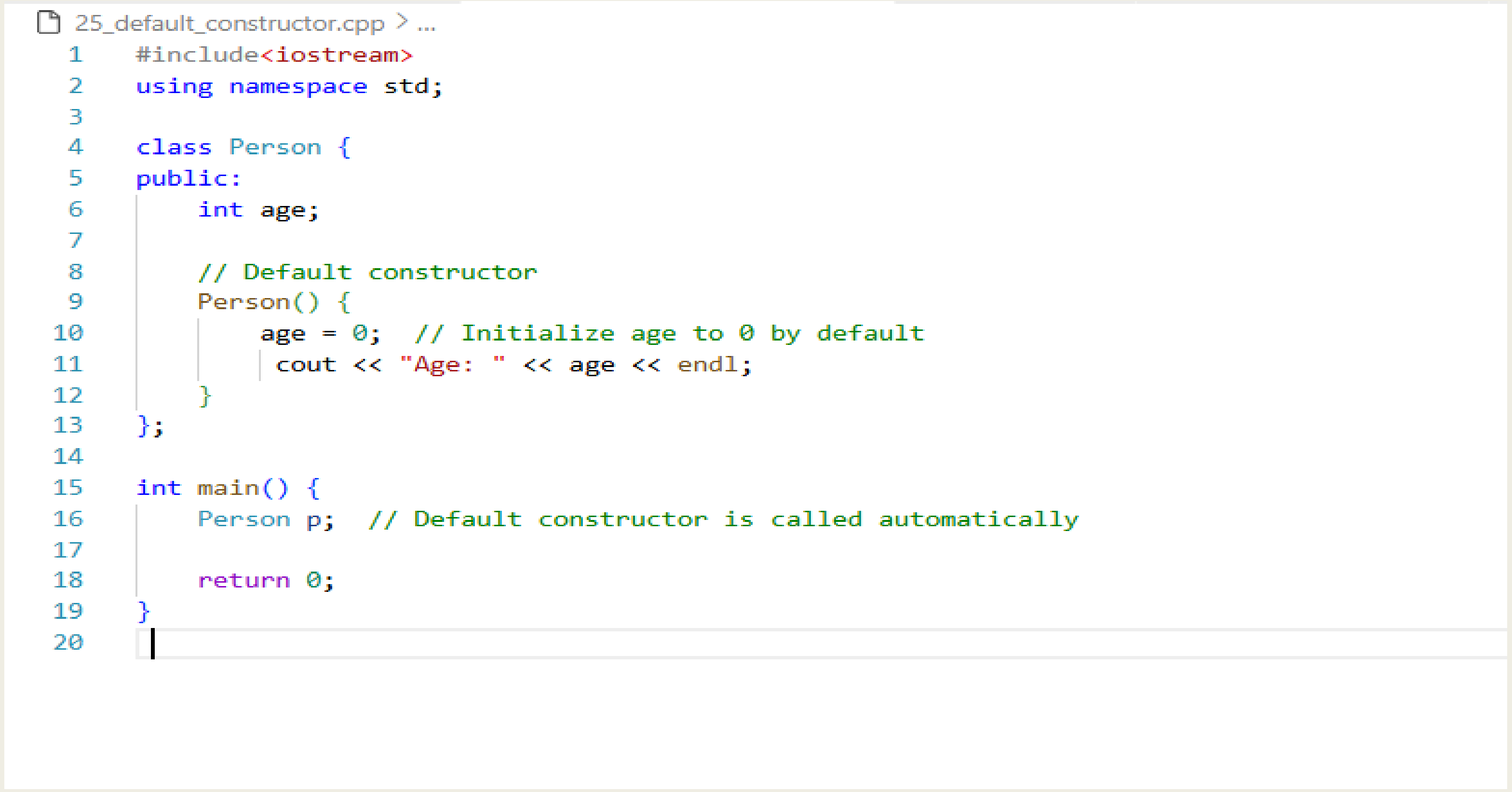
Default constructor

Parameterized constructor

Copy constructor

Default constructor

A **default constructor** is a constructor that **takes no arguments** and is used to **automatically initialize objects** with default values.



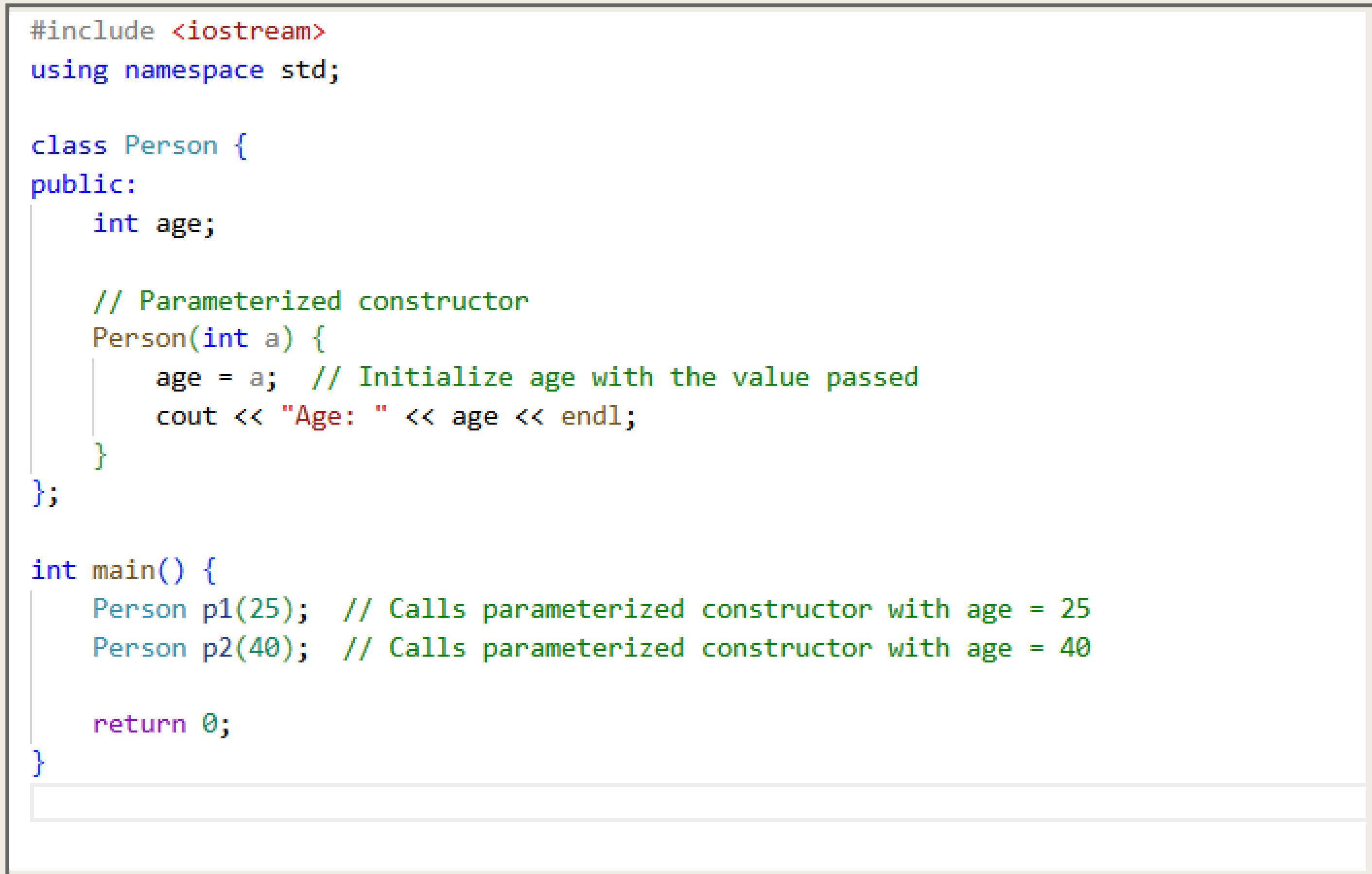
25_default_constructor.cpp > ...

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

class Person {
public:
    int age;

    // Default constructor
    Person() {
        age = 0;  // Initialize age to 0 by default
         cout << "Age: " << age << endl;
    }
};

int main() {
    Person p;  // Default constructor is called automatically

    return 0;
}

```

Parameterized constructor

- It is possible to **pass arguments to a constructor**. These arguments help **initialize an object** with specific values when it is created.
- To create a **parameterized constructor**, simply add parameters to the constructor just like any other function. Inside the constructor body, use these parameters to initialize the object's data members.

Example of Parameterized constructor



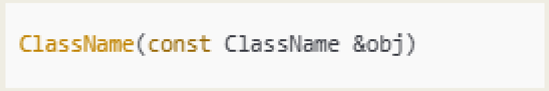
```
#include <iostream>
using namespace std;

class Person {
public:
    int age;

    // Parameterized constructor
    Person(int a) {
        age = a;  // Initialize age with the value passed
        cout << "Age: " << age << endl;
    }
};

int main() {
    Person p1(25);  // Calls parameterized constructor with age = 25
    Person p2(40);  // Calls parameterized constructor with age = 40

    return 0;
}
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

Note: In a **parameterized constructor**, you can assign **different values to different objects** by passing different arguments when creating those objects.

Copy constructor

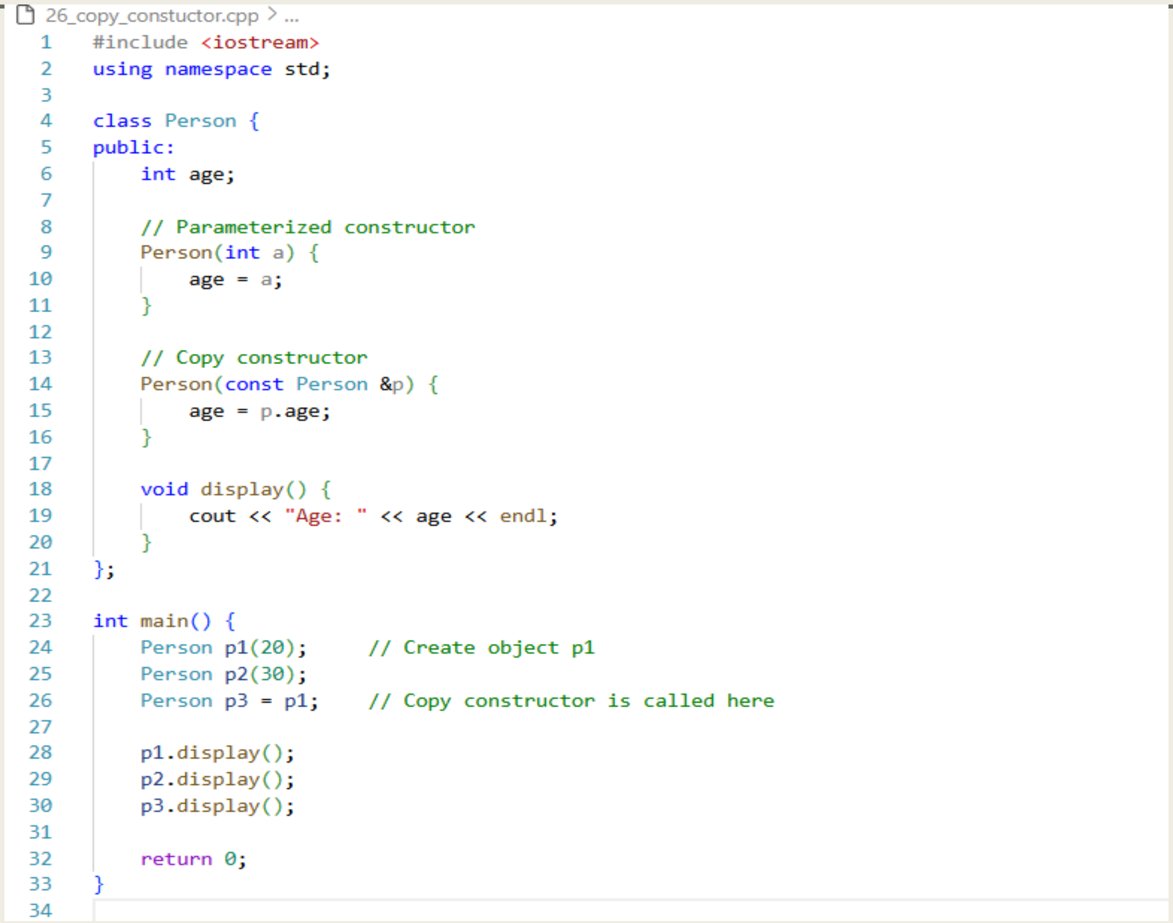
Copy Constructor is a special constructor in C++ that initializes a new object using the contents of an existing object of the same class.



```
ClassName(const ClassName &obj)
```

In a copy constructor, const is written because we don’t want to change the original object p. We only want to read its values and copy them to the new object.

Example of copy constructor


26_copy_constuctor.cpp > ...

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

class Person {
public:
    int age;

    // Parameterized constructor
    Person(int a) {
        age = a;
    }

    // Copy constructor
    Person(const Person &p) {
        age = p.age;
    }

    void display() {
        cout << "Age: " << age << endl;
    }
};

int main() {
    Person p1(20);      // Create object p1
    Person p2(30);
    Person p3 = p1;     // Copy constructor is called here

    p1.display();
    p2.display();
    p3.display();

    return 0;
}

```

Destructor

- A **destructor** is a special member function in object-oriented programming (like in C++) that is **automatically called when an object is destroyed.**
- It is mainly used to **free resources** like memory, close files, or clean up anything that the object was using during its lifetime.

When an object is created, the **constructor** is called automatically.
The constructor **executes** and can **reserve memory**, **open a file**, or **initialize variables** for the object.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Some property of destructor

- Same name as the class but starts with a tilde (~)

→ Example: ~MyClass()

- No return type (not even void)
- Takes no parameters

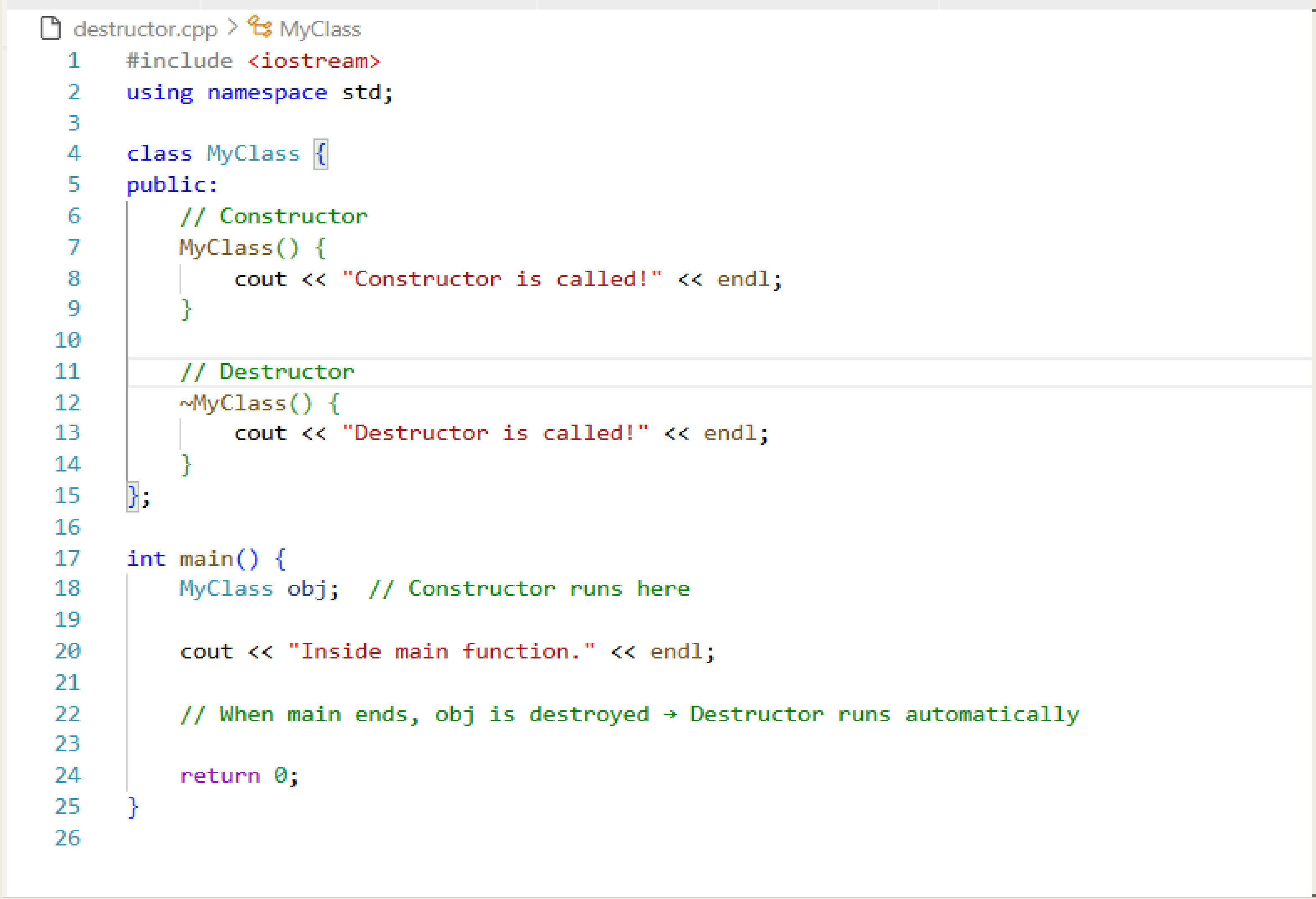
→ You cannot pass arguments to a destructor.

- Called automatically

→ When the object goes out of scope (no longer in use)

Note: The destructor is called automatically after the closing curly brace } of main() function — when the object goes out of scope.

What is destructor?

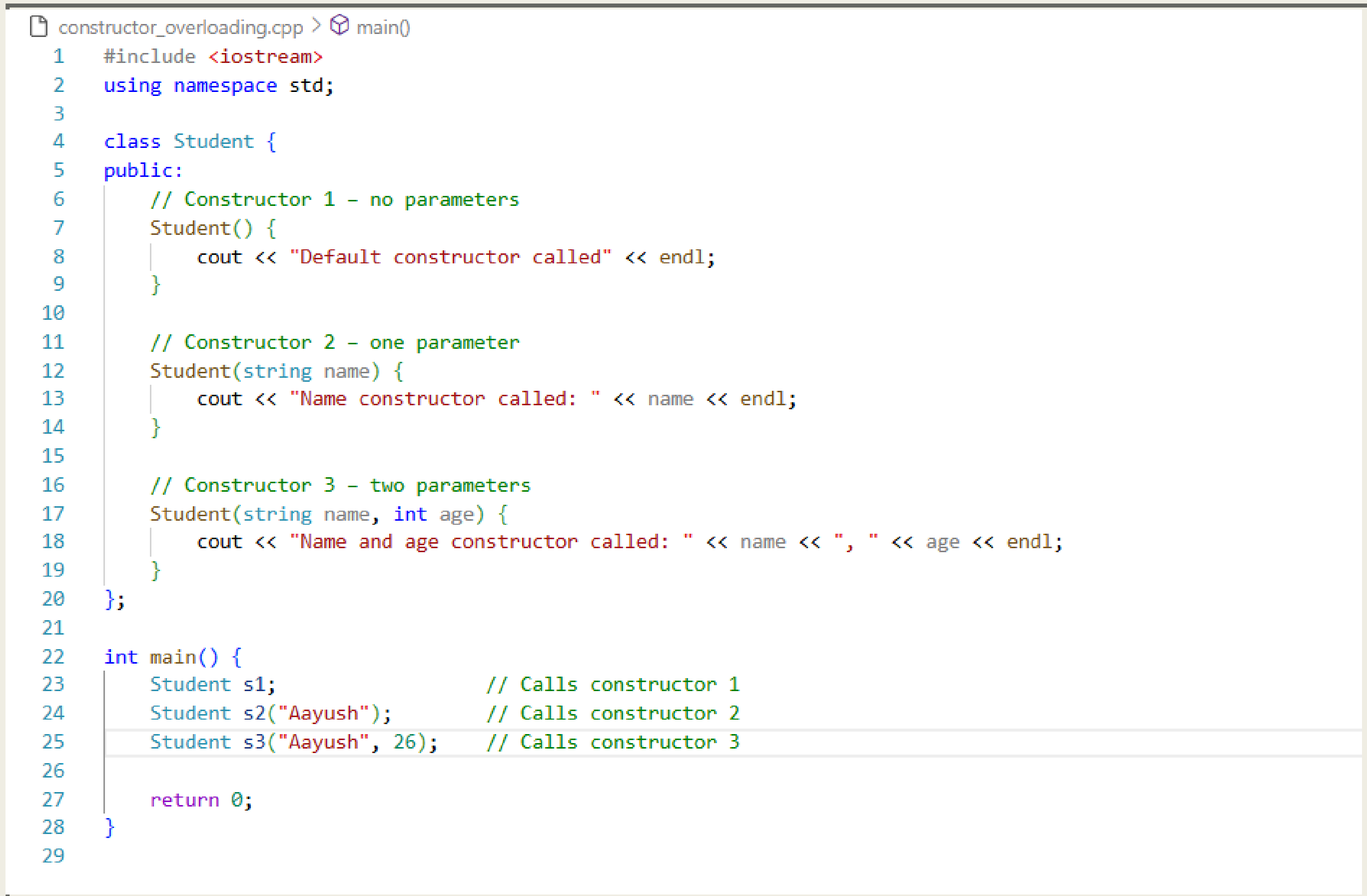


```cpp
#include <iostream>
using namespace std;

class MyClass {
public:
    // Constructor
    MyClass() {
        cout << "Constructor is called!" << endl;
    }

    // Destructor
    ~MyClass() {
        cout << "Destructor is called!" << endl;
    }
};

int main() {
    MyClass obj;  // Constructor runs here

    cout << "Inside main function." << endl;

    // When main ends, obj is destroyed → Destructor runs automatically

    return 0;
}
```

What is constructor overloading (Overloaded constructor)

- Constructor overloading means creating multiple constructors in the same class, but with different parameters.
- It allows you to initialize objects in different ways, depending on how many or what type of arguments we pass.

What is constructor overloading (Overloaded constructor)


constructor_overloading.cpp > main()

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

class Student {
public:
    // Constructor 1 - no parameters
    Student() {
        cout << "Default constructor called" << endl;
    }

    // Constructor 2 - one parameter
    Student(string name) {
        cout << "Name constructor called: " << name << endl;
    }

    // Constructor 3 - two parameters
    Student(string name, int age) {
        cout << "Name and age constructor called: " << name << ", " << age << endl;
    }
};

int main() {
    Student s1;                 // Calls constructor 1
    Student s2("Aayush");       // Calls constructor 2
    Student s3("Aayush", 26);   // Calls constructor 3

    return 0;
}

```

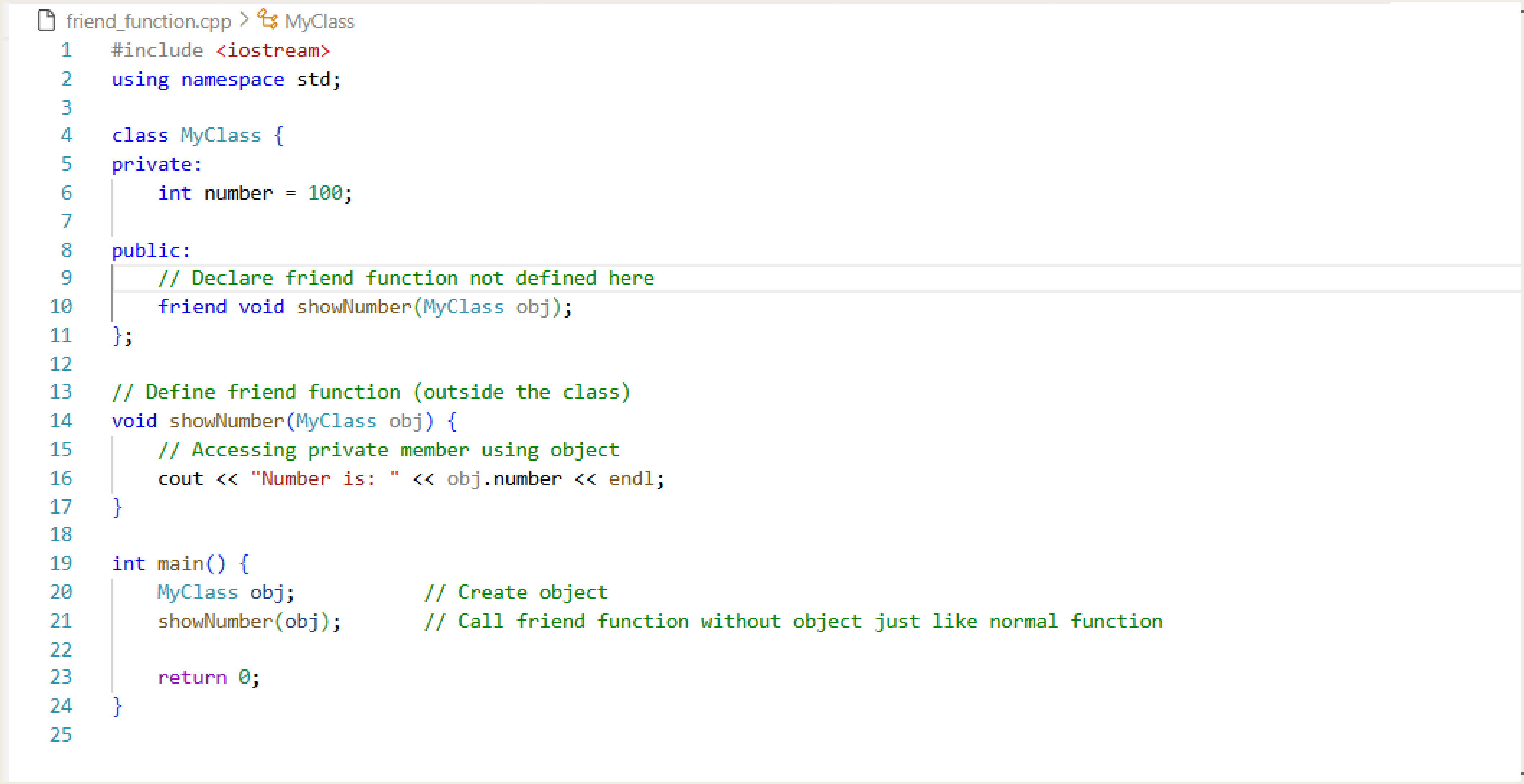
What is friend function?

- Normally, the private section of a class can be accessed only through member functions of the same class.
- A friend function is not a member of the class, but the class allows it to access its private and protected data by declaring it as a friend.

Key point:

- Even though the friend function is declared inside the class, it is not a member of that class, and we cannot access it using the object of that class.
- A **friend function** can **access the private and protected members** of a class, **but only through an object** of that class which is passed as an argument

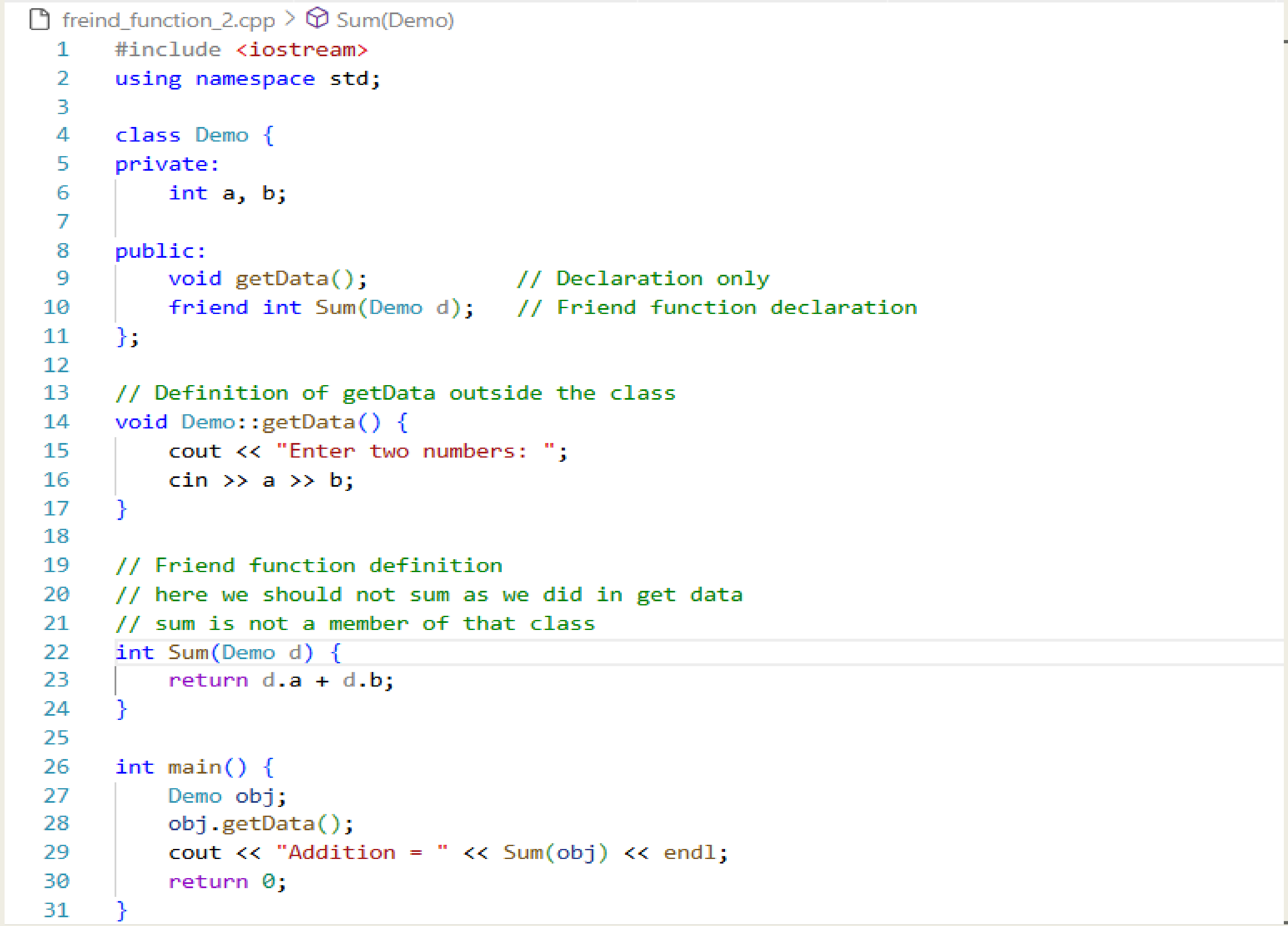
Simple example of friend function



friend_function.cpp > MyClass

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

class MyClass {
private:
    int number = 100;

public:
    // Declare friend function not defined here
    friend void showNumber(MyClass obj);
};

// Define friend function (outside the class)
void showNumber(MyClass obj) {
    // Accessing private member using object
    cout << "Number is: " << obj.number << endl;
}

int main() {
    MyClass obj;            // Create object
    showNumber(obj);        // Call friend function without object just like normal function

    return 0;
}
```

Simple example of friend function


freind_function_2.cpp > Sum(Demo)

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

class Demo {
private:
    int a, b;

public:
    void getData();            // Declaration only
    friend int Sum(Demo d);    // Friend function declaration
};

// Definition of getData outside the class
void Demo::getData() {
    cout << "Enter two numbers: ";
    cin >> a >> b;
}

// Friend function definition
// here we should not sum as we did in get data
// sum is not a member of that class
int Sum(Demo d) {
    return d.a + d.b;
}

int main() {
    Demo obj;
    obj.getData();
    cout << "Addition = " << Sum(obj) << endl;
    return 0;
}
```

The End

Prepared by Ayush Lamsal | |
layush788@gmail.com