# 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.

Prepared by Ayush Lamsal | | layush788@gmail.com

# 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.

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

- 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

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

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 (: :)

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

Prepared by Ayush Lamsal ||
layush788@gmail.com

# Complete structure example: Class Declaration + Function Definition

```cpp
16_class_structure.cpp > ...
1   #include <iostream>
2   using namespace std;
3
4   // ===== Class Declaration (Part 1) =====
5   class Student {
6   private:
7       // Data members
8       int roll;
9       string name;
0
1   public:// Member function declaration (prototype)
2       void input();
3       void display();
4   };
5
6   // ===== Function Definitions (Part 2) =====
7
8   // Define the input() function
9   void Student::input() {
0       cout << "Enter roll number: ";
1       cin >> roll;
2       cout << "Enter name: ";
3       cin >> name;
4   }
5
6   // Define the display() function
7   void Student::display() {
8       cout << "Roll: " << roll << ", Name: " << name << endl;
9   }
0
```

```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.

### Format for calling member function in main

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

Prepared by Ayush Lamsal ||
layush788@gmail.com

# Accessing member variable through method

```cpp
17_accessing_member.cpp > ...
1    #include <iostream>
2    using namespace std;
3
4    class Student {
5    private:
6        string name;  // Private member variable
7
8    public:
9        // Setter method to assign value to 'name'
10       void setName(string n) {
11           name = n;
12       }
13
14       // Getter method to access the value of 'name'
15       string getName() {
16           return name;
17       }
18   };
19
20   int main() {
21       Student s1;
22
23       s1.setName("Ayush");              // Access private member via setter
24       cout << "Student name: " << s1.getName() << endl;  // Access via getter
25
26       return 0;
27   }
28
```

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;
    }
};
```

## 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;
}
```

# 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.

Prepared by Ayush Lamsal ||
layush788@gmail.com

## 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

```cpp
18_creating_an_object.cpp > ...
1    #include <iostream>
2    using namespace std;
3
4    // Class definition
5    class Student {
6    public:
7        string name;
8        int age;
9
10       // Member function to display student details
11       void display() {
12           cout << "Name: " << name << endl;
13           cout << "Age: " << age << endl;
14       }
15   };
16
17   int main() {
18       // Creating an object of the class
19       Student s1;
20
21       // Assigning values to the object's data members
22       s1.name = "Ayush";
23       s1.age = 22;
24
25       // Calling the member function using the object
26       s1.display();
27
28       return 0;
29   }
30
```

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.

Prepared by Ayush Lamsal ||
layush788@gmail.com

# Public Access Specifiers Example

```cpp
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.

Prepared by Ayush Lamsal ||
layush788@gmail.com

# Private Access Specifiers Example

```cpp
18_creating_an_object.cpp > main()
1    #include<iostream>
2    using namespace std;
3
4    class Student {
5    private:
6        int rollNo;  // private data member
7
8    public:
9        void setRollNo(int r) {  // member function to access private data
10           rollNo = r;
11       }
12
13       int getRollNo() {
14           return rollNo;
15       }
16   };
17
18   int main() {
19       Student s1;
20       // s1.rollNo = 101;  // Error: Cannot access private member directly
21
22       s1.setRollNo(101);   //Access through public member function
23       cout << s1.getRollNo();  //Access through public member function
24
25       return 0;
26   }
27
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

# 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

```cpp
18_creating_an_object.cpp > ⊘ main()
1    #include<iostream>
2    using namespace std;
3
4    class abc {
5    protected:
6        int protectedData;
7    };
8
9    class bcd : abc {
10   public:
11       void setData(int value) {
12           protectedData = value;  // Allowed: Derived class can access protected members
13       }
14
15       int getData() {
16           return protectedData;
17       }
18   };
19
20   int main() {
21       bcd obj;
22       // obj.protectedData = 10;  //  Error: protectedData is not accessible outside class or subclass
23
24       obj.setData(10);
25       cout << obj.getData();
26
27       return 0;
28   }
29
```

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

Prepared by Ayush Lamsal ||
layush788@gmail.com

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.

Prepared by Ayush Lamsal ||
layush788@gmail.com

# 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;
}
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

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

```cpp
19_Array_example.cpp > ...
1   #include <iostream>
2   using namespace std;
3
4   class Student {
5   public:
6       int marks[5]; // Array for 5 subjects
7
8       void inputMarks() {
9           for (int i = 0; i < 5; i++) {
10              cout << "Enter marks for subject " << i + 1 << ": ";
11              cin >> marks[i]; // Input mark one by one
12          }
13      }
14
15      void displayMarks() {
16          cout << "Marks: ";
17          for (int i = 0; i < 5; i++) {
18              cout << marks[i] << " ";
19          }
20          cout << endl;
21      }
22  };
23
24  int main() {
25      Student students[3]; // Array of 3 Student objects
26
27      for (int i = 0; i < 3; i++) {
28          cout << "\nEnter marks for Student " << i + 1 << ":\n";
29          students[i].inputMarks();
30      }
31
32      cout << "\nDisplaying Marks:\n";
33      for (int i = 0; i < 3; i++) {
34          cout << "Student " << i + 1 << ": ";
35          students[i].displayMarks();
36      }
37
38      return 0;
39  }
```

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

Prepared by Ayush Lamsal ||
layush788@gmail.com

## 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.

Prepared by Ayush Lamsal ||
layush788@gmail.com

# Simple example of array of object

```cpp
20_Array_of_object.cpp > main()
1    #include <iostream>
2    using namespace std;
3
4    class Student {
5    public:
6        string name;
7        int age;
8
9        void setData(string n, int a) {
10           name = n;
11           age = a;
12       }
13
14       void display() {
15           cout << "Name: " << name << ", Age: " << age << endl;
16       }
17   };
18
19   int main() {
20       Student students[3]; // Creating an array of 3 Student objects
21
22       students[0].setData("Alice", 20);
23       students[1].setData("Bob", 21);
24       students[2].setData("Charlie", 22);
25
26       cout << "Student Details:" << endl;
27       for (int i = 0; i < 3; i++) {
28           students[i].display();
29       }
30
31       return 0;
32   }
33
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

# 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.
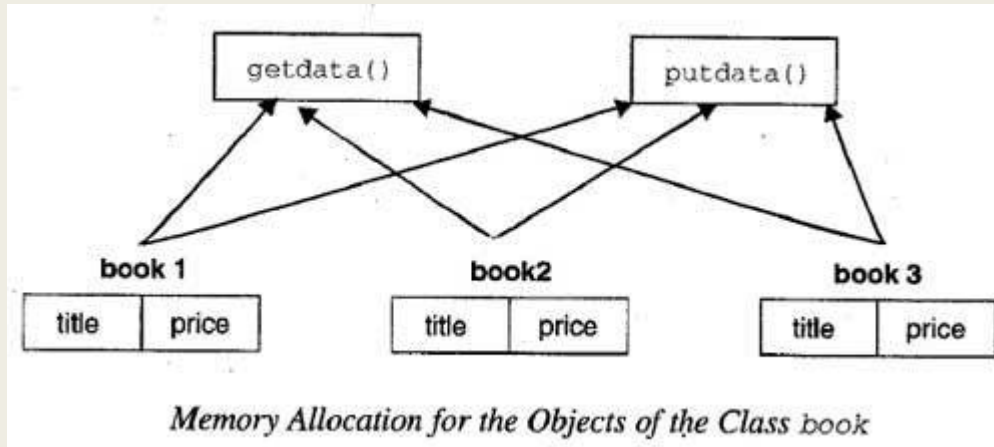
## Memory Allocation for Data Members

- 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.

Prepared by Ayush Lamsal ||
layush788@gmail.com

# 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() }

Prepared by Ayush Lamsal ||
layush788@gmail.com

## 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**

Prepared by Ayush Lamsal ||
layush788@gmail.com

## 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
};
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

## 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
```

# Simple Example

```cpp
22_static_data_member.cpp > ...
1   #include <iostream>
2   using namespace std;
3
4   class MyClass {
5   public:
6       static int x;    // Declare static int x
7
8       void showX() {
9           cout << "Value of x: " << x << endl;
10      }
11  };
12
13  // Define and initialize static int x outside the class
14  int MyClass::x = 10;
15
16  int main() {
17      MyClass obj;    // Create one object
18      obj.showX();    // Output the value of static x
19
20      return 0;
21  }
22
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

# Same previous example with static data member  with static member function

```cpp
22_static_data_member.cpp > ...
1    #include <iostream>
2    using namespace std;
3
4    class MyClass {
5    public:
6        static int x;    // Declare static int x
7
8        static void showX() {
9            cout << "Value of x: " << x << endl;
10       }
11   };
12
13   // Define and initialize static int x outside the class
14   int MyClass::x = 10;
15
16   int main() {
17       MyClass obj;    // Create one object
18       obj.showX();    // Output the value of static x
19
20       return 0;
21   }
22
```

# Object as function arguments

```cpp
24_object_function_argument.cpp  >  main()
1    #include <iostream>
2    using namespace std;
3
4    class Number {
5    public:
6        int num;
7
8        void setValue(int v) {
9            num = v;
10       }
11   };
12
13   // Function to add two Number objects and return the sum as int
14   int add(Number n1, Number n2) {
15       return n1.num + n2.num;
16   }
17
18   int main() {
19       Number a, b;
20
21       a.setValue(10);  // a.num =10
22       b.setValue(15);   //b.num=15
23
24       int sum = add(a, b);  //passing object as an argument
25
26       cout << "Sum = " << sum << endl;
27
28       return 0;
29   }
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

## 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.

Prepared by Ayush Lamsal || layush788@gmail.com

# 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.

Prepared by Ayush Lamsal ||
layush788@gmail.com

# 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

Prepared by Ayush Lamsal ||
layush788@gmail.com

# Default constructor

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

```cpp
25_default_constructor.cpp > ...
1    #include<iostream>
2    using namespace std;
3
4    class Person {
5    public:
6        int age;
7
8        // Default constructor
9        Person() {
10            age = 0;  // Initialize age to 0 by default
11            cout << "Age: " << age << endl;
12        }
13    };
14
15    int main() {
16        Person p;  // Default constructor is called automatically
17
18        return 0;
19    }
20    |
```

## 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.

Prepared by Ayush Lamsal ||
layush788@gmail.com

# Example of Parameterized constructor

```cpp
#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.

Prepared by Ayush Lamsal ||
layush788@gmail.com

# Example of copy constructor

```cpp
26_copy_constuctor.cpp > ...
1    #include <iostream>
2    using namespace std;
3
4    class Person {
5    public:
6        int age;
7
8        // Parameterized constructor
9        Person(int a) {
10           age = a;
11       }
12
13       // Copy constructor
14       Person(const Person &p) {
15           age = p.age;
16       }
17
18       void display() {
19           cout << "Age: " << age << endl;
20       }
21   };
22
23   int main() {
24       Person p1(20);      // Create object p1
25       Person p2(30);
26       Person p3 = p1;     // Copy constructor is called here
27
28       p1.display();
29       p2.display();
30       p3.display();
31
32       return 0;
33   }
34
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

# Characteristics of constructor

Prepared by Ayush Lamsal ||
layush788@gmail.com