Object Oriented Programming

Chapter 2: Objects & Classes

Engineering Notes — B.Tech / B.E. Course

2.1 Classes, Objects and Their Fundamentals

Introduction

Object Oriented Programming (OOP) treats data and the functions that operate on that data as a single unit called an object, instead of treating them separately as in procedural programming. A class is the blueprint or template from which objects are created, and an object is a runtime instance of a class. In C++, the class is the mechanism that brings together data members (attributes) and member functions (behaviour) under one name, and it also enforces control over how that data can be accessed from outside, through the principle of encapsulation.

Specifying a Class

A class is specified using the keyword class, followed by the class name and a body enclosed in curly braces, terminated by a semicolon. The body of the class contains the declaration of data members and the declaration/definition of member functions, grouped under access specifiers (private, public, protected) that control their visibility.

```
class ClassName
{
    private:
        // data members (hidden from outside)
        int data1;
        float data2;

    public:
        // member functions (interface to the outside world)
        void setData(int, float);
        void showData();
};
```

Key points about specifying a class:

- By default, all members of a class are private if no access specifier is mentioned.
- The class declaration only creates a new data type — no memory is allocated for data members until an object of that class is created.
- Data members are usually kept private to achieve data hiding, while member functions that form the interface are kept public.
- The closing brace of the class must be followed by a semicolon, unlike a normal function definition.

Defining Member Functions

Member functions can be defined in two ways: inside the class definition itself, or outside the class definition using the scope resolution operator (::).

(a) Inside the class definition

A function defined within the class body is treated as an inline function by the compiler (subject to compiler discretion), which can improve execution speed for small, frequently used functions.

```
class Sample
{
    int x;
    public:
        void setX(int a) { x = a; }      // defined inside
        void showX() { cout << x; }      // defined inside
};
```

(b) Outside the class definition

The function is only declared inside the class; its actual body is written outside using the scope resolution operator (::), which tells the compiler that the function belongs to that particular class.

```
class Sample
{
    int x;
    public:
        void setX(int a);   // declaration only
        void showX();
};

void Sample::setX(int a)     // definition outside
{
    x = a;
}
void Sample::showX()
{
    cout << x;
}
```

> *Note: The scope resolution operator (::) is essential outside the class because it distinguishes a member function of the class from an ordinary global function of the same name.*

Arrays Within a Class

A class can contain an array as one of its data members, just as it can contain simple variables. This is useful when an object needs to store a fixed collection of related values, such as marks of several subjects or a list of scores belonging to a single entity.

```
class Student
{
    int marks[5];      // array as a data member
    public:
        void getMarks();
        void showMarks();
};
```

- Each object of the class gets its own separate copy of the array.
- The array can be manipulated using loops inside member functions, exactly like any ordinary array.
- It is commonly used to model one-to-many relationships between an object and its attribute values.

Creating Objects

An object is an instance of a class. Once a class has been defined, objects can be created using a syntax similar to declaring a variable of a built-in type. Objects can be created at the time the class is defined, or later, wherever they are needed in the program.

```
class Item
{
    ...
};

Item obj1;              // simple object
Item obj2, obj3;        // multiple objects
Item *ptr = new Item(); // object created dynamically on the heap
```

Each object, once created, gets access to all the public member functions of the class and can invoke them using the dot (.) operator, for example obj1.setData();.

Memory Allocation for Objects

Memory for the data members of a class is not allocated when the class is declared — it is only a template. Memory is allocated only when an object of that class is actually created.

- Each object gets its own separate copy of the non-static data members, so changing the data of one object does not affect another object.
- Member functions, on the other hand, are not duplicated for every object. A single copy of each member function is created and shared by all objects of the class, since the code of a function is the same regardless of which object calls it.
- To distinguish which object's data a member function should operate upon during a shared call, the compiler internally passes a hidden pointer called the this pointer, which points to the object that invoked the function.
- The total memory occupied by an object is approximately the sum of the memory required for its individual non-static data members (ignoring alignment/padding effects and excluding memory for member functions).

Static Data Members and Static Member Functions

Static Data Members

A data member of a class can be declared static using the keyword static. A static data member has special properties:

- Only one copy of a static data member is created and shared by all objects of the class, regardless of how many objects exist.
- It is initialised to zero when the first object of the class is created, unless explicitly initialised.
- It must be defined and (optionally) initialised outside the class, in the file scope, because the class declaration alone does not allocate storage for it.
- It is commonly used to keep track of information common to all objects, such as a count of the number of objects created from a class.

```
class Counter
{
    static int count;    // declaration inside class
```

```
    public:
        Counter() { count++; }
};
int Counter::count = 0;   // definition + initialisation outside class
```

Static Member Functions

A member function declared static behaves like a class-level utility function rather than an object-level function:

- A static member function can be called using the class name and the scope resolution operator, without needing to create an object, e.g., ClassName::functionName();.
- A static member function can directly access only other static members (data or functions) of the class; it cannot directly access non-static (ordinary) data members, because it has no this pointer.
- It is generally used for tasks that are related to the class as a whole rather than to any one specific object, such as returning the value of a static data member.

Arrays of Objects

Just as an array of int or char can be declared, an array of objects of a user-defined class can also be declared. Each element of such an array is a separate, independent object of that class, and the class must have an accessible default constructor (if constructors are used) so that every element can be created automatically when the array is declared.

```
class Employee
{
    int id;
    public:
        void setId(int i) { id = i; }
        void showId() { cout << id; }
};

Employee emp[5];           // array of 5 objects
emp[0].setId(101);
emp[0].showId();
```

- Memory for all the objects in the array is allocated in one contiguous block, similar to an ordinary array.
- Individual objects of the array are accessed using an index along with the dot operator, e.g., emp[i].functionName();.

Objects as Function Arguments

Like variables of ordinary data types, an object of a class can also be passed as an argument to a function. There are three common ways of doing this:

1. Pass by value: A copy of the entire object is created and passed to the function. Any changes made to the object inside the function do not affect the original object, but this method can be inefficient in terms of memory and time for large objects.
2. Pass by reference: The function receives a reference to the original object rather than a copy. Any change made to the object inside the function directly affects the original object, and this method avoids the overhead of copying.

3. Pass by pointer: The address of the object is passed to the function. The function can access and modify the original object through the pointer using the arrow (->) operator.

```
void display(Item obj);         // pass by value
void display(Item &obj);        // pass by reference
void display(Item *obj);        // pass by pointer
```

> *Note: An object can also be returned from a function in the same three ways, and objects are used as function arguments exactly like objects of built-in types are used, except that member functions may be invoked on them inside the called function.*

2.2 Class Specifiers and Struct vs Class

Class Specifiers (Access Specifiers) and Their Uses

Access specifiers (also called visibility labels or access modifiers) determine which parts of a program can access the members of a class. C++ provides three access specifiers: private, public and protected. They may appear any number of times, in any order, inside a class body, and each one governs the members listed until the next specifier appears.

private

- Members declared private can be accessed only by the member functions and friend functions of the same class.
- They cannot be accessed directly by any code outside the class, including objects of the class in the calling program.
- This is the default access specifier for a class — if nothing is specified, all members are private.
- It is used to achieve data hiding, protecting the internal state of an object from unintended external interference.

public

- Members declared public can be accessed from anywhere in the program, both by member functions of the class and by code outside the class (through an object).
- Public members typically form the interface of the class — the set of operations through which the outside world interacts with the object.

protected

- Members declared protected behave like private members as far as the outside world is concerned — they cannot be accessed directly through an object.
- However, unlike private members, protected members can be directly accessed by member functions of any derived (child) class.
- This specifier is primarily useful in the context of inheritance, allowing controlled access to a base class's members from its derived classes.

Typical use in practice: data members are kept private or protected to enforce encapsulation, while functions meant to be called from outside are kept public, forming a controlled interface to the class.

Distinction between a C Structure (struct) and a C++ Class

Although a C++ struct and a C++ class are almost identical constructs internally, the traditional C structure and the C++ class differ significantly. The important points of distinction are summarised below.

Basis	C Structure (struct)	C++ Class
Default access	All members are public by default.	All members are private by default.
Contents	Can hold only data members (in traditional C); no functions.	Can hold both data members and member functions together.
Functions	Does not support member functions in C.	Supports member functions, including constructors and destructors.
Data hiding	No concept of data hiding; all data is openly accessible.	Supports data hiding through private and protected members.
Constructors/Destructors	Not supported in C.	Fully supported; used for automatic initialisation and cleanup.
Inheritance	Not supported.	Supported, along with polymorphism and other OOP features.
Purpose	Used purely to group related data of possibly different types.	Used to model real-world entities by binding data and behaviour together.

In C++ specifically, a struct is technically also allowed to have member functions, constructors, and inheritance, just like a class — the only real difference that remains in C++ is the default access specifier: members of a struct are public by default, whereas members of a class are private by default. The distinction described above is mainly meant to highlight the conceptual difference between the primitive C structure and the fully object-oriented C++ class.

Multiple Choice Questions (MCQs)

Answer keys are provided at the end of this section.

1. Which of the following best defines a class in C++?

a) A function that returns objects
b) A blueprint/template for creating objects
c) A single value of a built-in data type
d) A keyword used only for arrays

2. By default, the access specifier of members of a class (if none is mentioned) is:

a) public
b) protected
c) private
d) friend

3. Memory for data members of a class is allocated:

a) When the class is declared
b) When the compiler starts
c) When an object of the class is created
d) When the program is linked

4. A member function defined inside the class body is treated by the compiler as:

a) A virtual function
b) An inline function
c) A static function
d) A friend function

5. Which operator is used to define a member function outside the class?

a) Dot operator (.)
b) Arrow operator (->)
c) Scope resolution operator (::)
d) Ampersand (&)

6. Which of the following is TRUE about static data members?

a) A separate copy exists for every object
b) Only one copy is shared by all objects of the class
c) They must be initialised inside the class body
d) They cannot be used with static member functions

7. A static member function can directly access:

a) Only non-static data members
b) Only other static members of the class
c) Any member of any class
d) Only private members of another class

8. What must follow the closing brace of a class definition in C++?

a) A colon (:)
b) A semicolon (;)
c) Nothing
d) A pair of parentheses

9. Which access specifier allows access to derived classes but not to outside code?

a) private
b) public
c) protected
d) static

10. In C++, the default access specifier for a struct is:

a) private
b) protected
c) public
d) friend

11. If an array is declared as a data member inside a class, then:

a) It is shared by all objects like a static member
b) Each object gets its own separate copy of the array
c) It cannot be accessed by member functions
d) It must always be declared public

12. Which pointer is implicitly passed to a non-static member function to identify the calling object?

a) self pointer
b) this pointer
c) base pointer
d) null pointer

13. Which of the following statements about passing an object by reference to a function is correct?

a) A copy of the object is made
b) Changes inside the function do not affect the original object
c) Changes inside the function directly affect the original object
d) It is not allowed in C++

14. An array of objects requires the class to have:

a) Only static members
b) An accessible default constructor (if constructors are defined)
c) No member functions at all
d) Only public data members

15. Which of the following is NOT a valid way to pass an object to a function?

a) Pass by value
b) Pass by reference
c) Pass by pointer
d) Pass by static keyword

Answer Key

1-b, 2-c, 3-c, 4-b, 5-c, 6-b, 7-b, 8-b, 9-c, 10-c, 11-b, 12-b, 13-c, 14-b, 15-d

Broad / Long Answer Questions

1. 1. What is a class? Explain how a class is specified in C++ with the help of a suitable example.
2. 2. Differentiate between defining a member function inside the class and outside the class. Illustrate both methods with examples.
3. 3. Explain how objects are created in C++. Describe the different ways of creating an object with syntax.
4. 4. Discuss how memory is allocated for the data members and member functions of a class when objects are created.
5. 5. What are static data members? Explain their properties and demonstrate their use with a program that counts the number of objects created.
6. 6. What are static member functions? Explain how they differ from ordinary (non-static) member functions, with an example.
7. 7. Explain the concept of an array of objects with a suitable example. How is memory allocated for such an array?
8. 8. Describe the various ways in which an object can be passed as an argument to a function. Compare their advantages and disadvantages.
9. 9. Explain the role and use of the access specifiers private, public and protected in a class, with suitable examples.
10. 10. Bring out the distinction between a structure (struct) in C and a class in C++. Why is a class considered more powerful than a C structure?