Object Oriented Programming

Chapter 2: Objects & Classes

Exam-Oriented Questions & Answers

Section A: Short Answer Questions (2–3 Marks)

Q1. Define a class.

Ans. A class is a user-defined data type that acts as a blueprint for creating objects. It groups data members and member functions together under one name and controls their access through access specifiers.

Q2. Define an object.

Ans. An object is a runtime instance of a class. It is a variable of the class type that occupies memory for the class's non-static data members and can invoke the class's member functions.

Q3. What is the default access specifier of a class?

Ans. The default access specifier of a class in C++ is private. If no access specifier is mentioned, all members declared immediately after the class name are treated as private.

Q4. What is the default access specifier of a struct in C++?

Ans. The default access specifier of a struct in C++ is public, unlike a class, whose default is private.

Q5. What is the scope resolution operator? Why is it used?

Ans. The scope resolution operator (::) is used to define a member function outside the class body and to associate it with its class. It tells the compiler that the function being defined belongs to a specific class rather than being a global function.

Q6. When is a member function treated as inline by the compiler?

Ans. A member function is treated as inline (at the compiler's discretion) when it is defined inside the class body itself, rather than being declared inside the class and defined outside it.

Q7. When is memory allocated for the data members of a class?

Ans. Memory for data members is allocated only when an object of the class is created. The class declaration alone is just a template and does not consume memory for data members.

Q8. Is memory allocated separately for member functions of every object?

Ans. No. Unlike data members, member functions are not duplicated for each object. Only one copy of each member function exists and is shared by all objects of the class.

Q9. What is the this pointer?

Ans. The this pointer is a hidden pointer implicitly passed to every non-static member function. It points to the specific object that invoked the function, allowing the shared function code to operate on the correct object's data.

Q10. What is a static data member?

Ans. A static data member is a data member declared with the static keyword. Only a single copy of it exists, and this copy is shared by all objects of the class, rather than each object holding its own copy.

Q11. Where must a static data member be defined?

Ans. A static data member must be defined and optionally initialised outside the class, at file/global scope, using the class name with the scope resolution operator, e.g., int ClassName::count = 0;.

Q12. What is a static member function? Give one property.

Ans. A static member function is a member function declared with the static keyword, which can be called using the class name without creating an object. It can directly access only other static members of the class, since it has no this pointer.

Q13. How is an array of objects declared?

Ans. An array of objects is declared the same way as an array of a built-in type, using the class name as the type, e.g., Employee emp[5]; — this creates 5 independent objects, each accessed using an index.

Q14. Name the three ways an object can be passed to a function.

Ans. An object can be passed to a function by value (a copy is passed), by reference (the original object is passed via a reference), or by pointer (the address of the object is passed).

Q15. Which access specifier is used to achieve data hiding, and why?

Ans. The private access specifier is used to achieve data hiding, because private members can only be accessed by the member functions of the same class and are hidden from code outside the class.

Q16. What is the use of the protected access specifier?

Ans. The protected access specifier allows a member to be accessed by member functions of the class and by member functions of any class derived from it, while still keeping it inaccessible to the outside world — it is mainly used in inheritance.

Q17. Can a class contain an array as a data member? Give an example.

Ans. Yes. A class can contain an array as a data member, e.g., int marks[5]; inside a Student class, and every object of that class gets its own independent copy of the array.

Q18. State one key difference between a C structure and a C++ class.

Ans. In a C structure all members are public by default and it cannot hold member functions, whereas in a C++ class all members are private by default and it can hold both data members and member functions together.

Section B: Broad / Long Answer Questions (5–10 Marks)

QB1. What is a class? Explain how a class is specified in C++ with the help of a suitable example.

Ans. A class is a user-defined data type that combines data members and member functions into a single unit, serving as a blueprint from which objects are created. A class is specified using the keyword class, followed by the class name and a body enclosed in braces and terminated by a semicolon. Inside the body, members are grouped under access specifiers (private, public, protected).

```
class Rectangle
{
    private:
        float length, width;
    public:
        void setDimension(float l, float w)
        {
            length = l;
            width = w;
        }
        float getArea()
        {
            return length * width;
        }
};
```

Explanation: Here, length and width are private data members that cannot be accessed directly from outside, while setDimension() and getArea() are public member functions that form the interface used to set and read the object's data.

QB2. Differentiate between defining a member function inside the class and outside the class, with examples.

Ans. A member function can be defined inside the class body, in which case it is treated by the compiler as an inline function; or it can be declared inside the class and defined outside it using the scope resolution operator (::), which is the preferred approach for larger functions to keep the class definition readable.

```
class Box
{
    int side;
    public:
        void setSide(int s) { side = s; }   // defined inside (inline)
        int getVolume();                    // only declared here
};

int Box::getVolume()                        // defined outside
{
    return side * side * side;
}
```

QB3. Explain memory allocation for objects, including how data members and member functions are stored.

Ans. When a class is declared, no memory is allocated — the class is only a template. Memory is allocated only when an object is created, and each object receives its own separate copy of the non-static data members. Member functions, however, are not duplicated per object; a single copy of the function code is

shared by all objects, and the compiler uses the hidden this pointer to make a shared function operate on the correct object's data whenever it is called through a particular object.

QB4. Explain static data members and static member functions with a program that counts the number of objects created.

Ans. A static data member has only one copy shared across all objects of a class, making it suitable for tracking information common to the whole class, such as an object counter. A static member function can be called using the class name, without an object, and can access only static members directly.

```
class Counter
{
    static int count;
    public:
        Counter() { count++; }
        static void showCount()
        {
            cout << "Objects created: " << count;
        }
};
int Counter::count = 0;

int main()
{
    Counter c1, c2, c3;
    Counter::showCount();   // called using class name
    return 0;
}
```

Output: Objects created: 3

Explanation: Every time the constructor runs for c1, c2 and c3, the single shared copy of count is incremented, so by the time showCount() is called it correctly reports 3 objects, even though showCount() was called without referring to any individual object.

QB5. Explain arrays of objects and objects as function arguments with suitable programs.

Ans. An array of objects is a collection of independent objects of the same class, stored contiguously in memory and accessed using an index. An object may also be passed to a function by value, by reference, or by pointer; passing by reference or pointer avoids copying overhead and allows the function to modify the original object.

```
class Point
{
    int x, y;
    public:
        void setPoint(int a, int b) { x = a; y = b; }
        void show() { cout << x << "," << y << " "; }
};

void movePoint(Point &p)     // object passed by reference
{
    p.setPoint(10, 20);      // modifies the original object
}

int main()
{
    Point pts[3];            // array of 3 objects
    for (int i = 0; i < 3; i++)
        pts[i].setPoint(i, i);
```

```
    for (int i = 0; i < 3; i++)
        pts[i].show();

    movePoint(pts[0]);
    pts[0].show();
    return 0;
}
```

Output: 0,0 1,1 2,2 10,20

QB6. Explain the access specifiers private, public and protected with examples, and state their typical use.

Ans. private members are accessible only within the class's own member/friend functions and are used to hide internal data. public members are accessible from anywhere in the program and form the class's interface. protected members behave like private members for the outside world but are additionally accessible to derived classes, which makes them useful in inheritance hierarchies.

QB7. Distinguish between a C structure and a C++ class. Why is a class regarded as more powerful?

Ans. A C structure can hold only data members and has all members public by default, with no support for member functions, constructors, data hiding, or inheritance. A C++ class can hold both data members and member functions together, defaults to private access, and fully supports data hiding, constructors/destructors and inheritance. Because it binds data and behaviour together while also controlling access, a class is far more powerful and forms the foundation of object-oriented programming, whereas a C structure is only a passive grouping of data.

QB8. Write a program that demonstrates arrays within a class, using a Student class that stores and displays marks for 3 subjects.

Ans. Program:

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

class Student
{
    int marks[3];
    public:
        void getMarks()
        {
            for (int i = 0; i < 3; i++)
                marks[i] = (i + 1) * 10;   // sample values
        }
        void showMarks()
        {
            int total = 0;
            for (int i = 0; i < 3; i++)
            {
                cout << "Subject " << i + 1 << ": " << marks[i] << endl;
                total += marks[i];
            }
            cout << "Total: " << total << endl;
        }
};

int main()
{
    Student s;
    s.getMarks();
    s.showMarks();
```

```
    return 0;
}
```

Output: Subject 1: 10, Subject 2: 20, Subject 3: 30, Total: 60

Section C: Coding / Program-Based Questions

QC1. Write a C++ program to define a class Box with length, breadth and height as private data members, and member functions to input values and calculate volume.

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

class Box
{
    private:
        float length, breadth, height;
    public:
        void setDimensions(float l, float b, float h)
        {
            length = l;
            breadth = b;
            height = h;
        }
        float getVolume()
        {
            return length * breadth * height;
        }
};

int main()
{
    Box b1;
    b1.setDimensions(2.0, 3.0, 4.0);
    cout << "Volume = " << b1.getVolume();
    return 0;
}
```

Output: Volume = 24

QC2. Write a program using a static data member to count and display the total number of objects created of a class Item.

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

class Item
{
    static int count;
    public:
        Item() { count++; }
        static int getCount() { return count; }
};
int Item::count = 0;

int main()
{
    Item i1, i2;
    Item *i3 = new Item();
    cout << "Total items: " << Item::getCount();
    return 0;
}
```

Output: Total items: 3

QC3. Write a program to demonstrate an array of objects using a class Employee that stores and displays the id and salary of 3 employees.

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

class Employee
{
    int id;
    float salary;
    public:
        void setData(int i, float s)
        {
            id = i;
            salary = s;
        }
        void showData()
        {
            cout << "ID: " << id << "  Salary: " << salary << endl;
        }
};

int main()
{
    Employee emp[3];
    emp[0].setData(101, 25000);
    emp[1].setData(102, 30000);
    emp[2].setData(103, 28000);

    for (int i = 0; i < 3; i++)
        emp[i].showData();
    return 0;
}
```

Output: ID: 101 Salary: 25000, ID: 102 Salary: 30000, ID: 103 Salary: 28000

QC4. Write a program that passes an object as a function argument by value and by reference, and shows the difference in output.

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

class Counter
{
    public:
        int value;
        Counter(int v) { value = v; }
};

void byValue(Counter c)       // copy is modified, original unaffected
{
    c.value += 10;
}

void byReference(Counter &c)  // original object is modified
{
    c.value += 10;
}

int main()
{
    Counter obj(5);
    byValue(obj);
```

```
    cout << "After pass by value: " << obj.value << endl;

    byReference(obj);
    cout << "After pass by reference: " << obj.value << endl;
    return 0;
}
```

Output: After pass by value: 5, After pass by reference: 15

Explanation: byValue() receives a copy of obj, so modifying it inside the function has no effect on the original object, and the value stays 5. byReference() receives a reference to the actual object, so the modification made inside the function is reflected in main() as well, changing value to 15.

QC5. Write a program to demonstrate the use of the this pointer inside a member function.

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

class Number
{
    int value;
    public:
        void setValue(int value)
        {
            this->value = value;   // 'this' distinguishes member from parameter
        }
        void show()
        {
            cout << "Value = " << this->value;
        }
};

int main()
{
    Number n;
    n.setValue(50);
    n.show();
    return 0;
}
```

Output: Value = 50

Explanation: Since the parameter of setValue() has the same name as the data member (value), this->value is used to explicitly refer to the object's data member, while the plain name value refers to the parameter.

QC6. Predict the output of the following program and justify your answer.

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

class Demo
{
    static int total;
    int id;
    public:
        Demo() { id = ++total; }
        void show() { cout << "Object id: " << id << endl; }
};
int Demo::total = 0;

int main()
{
    Demo d1, d2, d3;
    d1.show();
```

```
    d2.show();
    d3.show();
    return 0;
}
```

Ans. Output: Object id: 1, Object id: 2, Object id: 3

Explanation: total is static, so it is shared across all objects and keeps incrementing every time the constructor runs, while id is a non-static data member and therefore holds a separate value (the total at the time of that object's creation) for each individual object.