Object Oriented Programming

Chapter 2: Objects & Classes

Coding Problems and Solutions

This set contains programming problems covering every concept of the chapter — specifying a class, member functions, arrays within a class, creating objects, memory allocation, static members, arrays of objects, objects as function arguments, and access specifiers. Each problem includes the full C++ program, its output, and a short explanation.

Problem 1: Specifying a class and defining member functions (inside vs outside the class)

Statement: Write a C++ program to define a class Circle with radius as a private data member. Define one member function inside the class and one outside the class using the scope resolution operator, to calculate area and circumference.

Solution:

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

class Circle
{
    private:
        float radius;
    public:
        void setRadius(float r) { radius = r; }   // defined inside class
        float area();                             // declared here, defined outside
        float circumference();
};

float Circle::area()                              // defined outside class
{
    return 3.14 * radius * radius;
}
float Circle::circumference()
{
    return 2 * 3.14 * radius;
}

int main()
{
    Circle c;
    c.setRadius(5);
    cout << "Area = " << c.area() << endl;
    cout << "Circumference = " << c.circumference();
    return 0;
}
```

Output: Area = 78.5, Circumference = 31.4

Explanation: setRadius() is defined inside the class body, so it is treated as inline. area() and circumference() are only declared inside the class and defined outside using Circle::, which requires the scope resolution operator to tell the compiler that these functions belong to the Circle class.

Problem 2: Arrays within a class

Statement: Write a program using a class Student that stores marks of 5 subjects as an array data member, and displays the total and average.

Solution:

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

class Student
{
    int marks[5];
    public:
        void readMarks()
        {
            int sample[5] = {80, 75, 90, 65, 70};
            for (int i = 0; i < 5; i++)
                marks[i] = sample[i];
        }
        void showResult()
        {
            int total = 0;
            for (int i = 0; i < 5; i++)
                total += marks[i];
            cout << "Total = " << total << endl;
            cout << "Average = " << (float)total / 5;
        }
};

int main()
{
    Student s;
    s.readMarks();
    s.showResult();
    return 0;
}
```

Output: Total = 380, Average = 76

Explanation: marks[5] is an array data member of the class. Since it is a normal (non-static) data member, every Student object created would get its own independent copy of this array.

Problem 3: Creating objects and memory allocation for objects

Statement: Write a program to create multiple objects of a class Account and show that each object holds its own independent copy of data (memory allocated per object).

Solution:

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

class Account
{
    int accNo;
    float balance;
    public:
        void setData(int no, float bal)
        {
            accNo = no;
            balance = bal;
        }
        void showData()
        {
            cout << "Acc No: " << accNo << "  Balance: " << balance << endl;
        }
};
```

```
int main()
{
    Account a1, a2;      // two separate objects
    a1.setData(1001, 5000);
    a2.setData(1002, 8000);

    a1.showData();
    a2.showData();
    return 0;
}
```

Output: Acc No: 1001 Balance: 5000, Acc No: 1002 Balance: 8000

Explanation: a1 and a2 are two separate objects of the class Account, and each gets its own memory for accNo and balance. Changing a2's data through setData() has no effect on a1's data, confirming that non-static data members are allocated independently for every object.

Problem 4: Static data member — counting objects created

Statement: Write a program using a static data member to keep count of how many objects of a class Product have been created so far.

Solution:

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

class Product
{
    static int count;   // declared inside class
    public:
        Product() { count++; }
        static void showCount()
        {
            cout << "Total products created: " << count << endl;
        }
};
int Product::count = 0;  // defined & initialised outside class

int main()
{
    Product p1;
    Product::showCount();

    Product p2, p3;
    Product::showCount();
    return 0;
}
```

Output: Total products created: 1, Total products created: 3

Explanation: count is static, so a single shared copy exists for the whole class. It is incremented in the constructor every time a new object is created, and showCount() — itself a static member function — is called directly using the class name Product:: without needing any object.

Problem 5: Static member function accessing only static members

Statement: Write a program that demonstrates why a static member function cannot directly access a non-static data member (conceptual/error-based question).

Solution:

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

```
class Test
{
    int x;              // non-static data member
    static int y;       // static data member
    public:
        static void show()
        {
            cout << y;      // OK - static function can access static data
            // cout << x;   // ERROR - cannot access non-static member here
        }
};
int Test::y = 100;

int main()
{
    Test::show();
    return 0;
}
```

Output: 100

Explanation: A static member function has no this pointer, since it is not tied to any particular object. It can therefore access only static members directly. Accessing x (a non-static member) inside show() would give a compile-time error, which is why that line is commented out above.

Problem 6: Arrays of objects

Statement: Write a program to declare an array of 4 objects of a class Book, set their titles and prices, and display the most expensive book.

Solution:

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

class Book
{
    string title;
    float price;
    public:
        void setBook(string t, float p) { title = t; price = p; }
        string getTitle() { return title; }
        float getPrice() { return price; }
};

int main()
{
    Book books[4];        // array of 4 objects
    books[0].setBook("C++ Basics", 350);
    books[1].setBook("Data Structures", 480);
    books[2].setBook("OOP Concepts", 420);
    books[3].setBook("Algorithms", 510);

    int maxIndex = 0;
    for (int i = 1; i < 4; i++)
        if (books[i].getPrice() > books[maxIndex].getPrice())
            maxIndex = i;

    cout << "Most expensive: " << books[maxIndex].getTitle()
         << " (Rs. " << books[maxIndex].getPrice() << ")";
```

```
    return 0;
}
```

Output: Most expensive: Algorithms (Rs. 510)

Explanation: books[4] creates four independent Book objects in contiguous memory. Each element is accessed with an index and the dot operator, exactly like elements of an ordinary array, and a simple loop is used to compare the price stored in each object.

Problem 7: Objects as function arguments — pass by value vs pass by reference

Statement: Write a program that passes an object of class Point to a function by value and to another function by reference, and shows how the results differ.

Solution:

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

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

void shiftByValue(Point p)        // copy received
{
    p.x += 5;
    p.y += 5;
}

void shiftByReference(Point &p)   // original received
{
    p.x += 5;
    p.y += 5;
}

int main()
{
    Point p1;
    p1.set(1, 1);

    shiftByValue(p1);
    cout << "After shiftByValue: ";
    p1.show();                     // unchanged

    shiftByReference(p1);
    cout << "After shiftByReference: ";
    p1.show();                     // changed
    return 0;
}
```

Output: After shiftByValue: (1,1), After shiftByReference: (6,6)

Explanation: shiftByValue() operates on a temporary copy of p1, so the original object in main() is unaffected. shiftByReference() operates on p1 itself through the reference parameter, so the change made inside the function is visible in main() after the call.

Problem 8: Objects as function arguments — pass by pointer

Statement: Write a program that passes an object of class Item to a function using a pointer, and updates its data through the pointer.

Solution:

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

class Item
{
    public:
        int code;
        float price;
};

void applyDiscount(Item *p, float percent)
{
    p->price = p->price - (p->price * percent / 100);
}

int main()
{
    Item it;
    it.code = 501;
    it.price = 1000;

    applyDiscount(&it, 10);   // address of object passed
    cout << "Code: " << it.code << "  New Price: " << it.price;
    return 0;
}
```

Output: Code: 501 New Price: 900

Explanation: The address of the object it is passed to applyDiscount() using &it, and inside the function the arrow operator (->) is used to access and modify the object's members through the pointer p, directly affecting the original object in main().

Problem 9: Access specifiers — private, public and protected in practice

Statement: Write a program to show that a private data member cannot be accessed directly from outside the class, and must be accessed through a public member function.

Solution:

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

class BankAccount
{
    private:
        float balance;         // hidden from outside
    public:
        void deposit(float amt) { balance += amt; }
        float getBalance() { return balance; }
};

int main()
{
    BankAccount acc;
    // acc.balance = 5000;    // ERROR: balance is private
    acc.deposit(5000);        // OK: deposit() is public
    cout << "Balance = " << acc.getBalance();
    return 0;
}
```

Output: Balance = 5000

Explanation: balance is private, so any attempt to access acc.balance directly in main() results in a compile-time error, as shown by the commented line. Instead, the object must go through the public functions deposit() and getBalance(), which together form the class's controlled interface — this is the essence of data hiding.

Problem 10: protected access specifier used with inheritance

Statement: Write a program with a base class Person having a protected data member name, and a derived class Student that accesses it directly.

Solution:

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

class Person
{
    protected:
        string name;
    public:
        void setName(string n) { name = n; }
};

class Student : public Person
{
    int rollNo;
    public:
        void setRoll(int r) { rollNo = r; }
        void display()
        {
            cout << "Name: " << name << endl;   // direct access - protected
            cout << "Roll No: " << rollNo;
        }
};

int main()
{
    Student s;
    s.setName("Rahul");
    s.setRoll(21);
    s.display();
    return 0;
}
```

Output: Name: Rahul, Roll No: 21

Explanation: name is declared protected in the base class Person. It cannot be accessed directly through an object from outside (as with a private member), but it can be accessed directly inside the member function display() of the derived class Student, which is the defining feature of the protected access specifier.

Problem 11: Struct vs class — same layout, different default access

Statement: Write a program that defines a struct and a class with identical members to show that the only real functional difference in C++ is the default access specifier.

Solution:

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

struct PointStruct
{
```

```
    int x, y;               // public by default
    void show() { cout << "(" << x << "," << y << ")" << endl; }
};

class PointClass
{
    int x, y;               // private by default
    public:
        void set(int a, int b) { x = a; y = b; }
        void show() { cout << "(" << x << "," << y << ")" << endl; }
};

int main()
{
    PointStruct p1;
    p1.x = 3; p1.y = 4;      // OK: members are public
    p1.show();

    PointClass p2;
    // p2.x = 3;             // ERROR: x is private
    p2.set(3, 4);            // must use a public function
    p2.show();
    return 0;
}
```

Output: (3,4), (3,4)

Explanation: Both PointStruct and PointClass have identical members, but p1.x can be set directly because struct members are public by default, whereas p2.x cannot be accessed directly and must be set through the public function set(), because class members are private by default. This program isolates the one true functional difference between struct and class in C++.

Problem 12: Combined program — class with array data member, static member, and array of objects together

Statement: Write a program for a class Employee that stores a name and an array of monthly sales (3 months), uses a static member to count total employees, and is used through an array of objects.

Solution:

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

class Employee
{
    string name;
    float sales[3];          // array within class
    static int empCount;     // static data member
    public:
        Employee() { empCount++; }
        void setData(string n, float s0, float s1, float s2)
        {
            name = n;
            sales[0] = s0; sales[1] = s1; sales[2] = s2;
        }
        float totalSales()
        {
            return sales[0] + sales[1] + sales[2];
        }
        void show()
        {
            cout << name << " - Total Sales: " << totalSales() << endl;
```

```
        }
        static void showCount()
        {
            cout << "Total Employees: " << empCount << endl;
        }
};
int Employee::empCount = 0;

int main()
{
    Employee emp[2];         // array of objects
    emp[0].setData("Asha", 1000, 1200, 1500);
    emp[1].setData("Bimal", 900, 1100, 1000);

    for (int i = 0; i < 2; i++)
        emp[i].show();

    Employee::showCount();
    return 0;
}
```

Output: Asha - Total Sales: 3700, Bimal - Total Sales: 3000, Total Employees: 2

Explanation: This program brings several concepts together: sales[3] is an array data member with its own copy per object, empCount is a static data member shared across all Employee objects and incremented once per constructor call, emp[2] is an array of objects, and showCount() is a static member function called using the class name.