OBJECT ORIENTED PROGRAMMING

Chapter 1: Concept of Object Oriented Programming

Exam Oriented Questions & Answers — Set 2 (Short, Broad and Coding)

Section A: Short Answer Type Questions

The following questions carry 1-2 marks each. Answers are kept brief and to the point, suitable for quick revision before examinations.

Q1. Differentiate between a class and an object in one line.

Ans. A class is only a definition/blueprint, whereas an object is an actual instance of that class occupying memory at run time.

Q2. What is meant by reusability in OOP?

Ans. Reusability means the ability to reuse an already-written and tested class in a new program or a new class, mainly achieved through inheritance.

Q3. Why is OOP called a bottom-up approach?

Ans. OOP starts by building small, self-contained objects and then combines them to form the complete system, i.e. it builds from smaller units upward to the whole program.

Q4. Expand OOP and POP.

Ans. OOP stands for Object Oriented Programming, and POP stands for Procedure Oriented Programming.

Q5. Who is regarded as the father of C++?

Ans. Bjarne Stroustrup, who developed C++ at Bell Laboratories in the early 1980s, is regarded as the father of C++.

Q6. What is a stream in C++?

Ans. A stream is a sequence of bytes that flows from a source to a destination, used in C++ to handle input and output operations.

Q7. What does the keyword void mean in void main()?

Ans. void indicates that the main() function does not return any value to the operating system when the program finishes execution.

Q8. What is the purpose of the #include directive?

Ans. The #include directive tells the preprocessor to insert the contents of a specified header file into the program before compilation.

Q9. State one difference between cin and cout.

Ans. cin is an object of the istream class used to read input from the keyboard, whereas cout is an object of the ostream class used to display output on the screen.

Q10. Name two predefined stream objects in C++ other than cin and cout.

Ans. cerr (for unbuffered error messages) and clog (for buffered log messages) are two other predefined stream objects.

Q11. What is operator overloading?

Ans. Operator overloading is a form of polymorphism in which an existing operator, such as + or <<, is given additional meaning when used with user-defined data types.

Q12. What is function overloading?

Ans. Function overloading is a form of polymorphism in which two or more functions have the same name but different parameter lists within the same scope.

Q13. Which section of a C++ program is compulsory?
Ans. The main() function section is compulsory; every C++ program must contain exactly one main() function, since execution starts there.

Q14. What is meant by a top-down approach in POP?
Ans. A top-down approach means breaking a large problem into smaller sub-problems (functions) step by step, starting from the overall program and moving down to details.

Q15. Give one advantage of encapsulation.
Ans. Encapsulation protects data from unauthorised or accidental access by hiding it inside a class, improving data security.

Q16. Give one advantage of inheritance.
Ans. Inheritance avoids rewriting the same code by allowing a new class to directly reuse the properties and methods of an existing class.

Q17. What is a header file?
Ans. A header file is a file containing declarations of built-in functions, classes and objects, which is included in a program using #include so that they can be used.

Q18. Name the two types of comments used in C++.
Ans. Single-line comments, written using //, and multi-line comments, written between /* and */.

Section B: Broad / Long Answer Type Questions

The following questions carry 5-10 marks each and require descriptive, well-structured answers.

Q1. Discuss the important features of Object Oriented Programming and state why each feature is useful.

Ans.

Object Oriented Programming is built around several key features that together make software more secure, reusable and manageable. Encapsulation binds data and functions into a class and hides internal data from outside interference, which improves security and reduces accidental errors. Abstraction shows only the necessary details to the user and hides the complex implementation, making large systems easier to use and understand. Inheritance allows a new class to reuse the properties and behaviour of an existing class, which saves development time and reduces duplicate code.

Polymorphism allows the same function name or operator to behave differently depending on context, giving programs flexibility, for example through function overloading and operator overloading. Dynamic binding allows a function call to be resolved at run time rather than compile time, giving extra flexibility when working with related classes. Together, these features make OOP well suited to developing and maintaining large, complex, real-world software systems compared to the older procedural style.

Q2. Trace the evolution of programming paradigms from procedural programming to object oriented programming.

Ans.

Early programming used machine language and assembly language, which were closely tied to hardware and very difficult to write and maintain. High-level procedural languages such as FORTRAN, COBOL and later C were then developed, which improved readability but still organised programs mainly as a sequence of function calls acting on shared, often global, data.

As applications became larger during the 1970s and 1980s, the drawbacks of procedural programming, particularly unprotected data and difficulty in reuse, became more serious, prompting the search for a better paradigm. This led to the development of Object Oriented Programming, first popularised by languages such as Simula and Smalltalk, and later made mainstream through C++ (an OOP extension of C) and subsequently Java, Python and C#. OOP represented a shift in focus from "functions acting on data" to "objects that carry their own data and behaviour together", which better matched the way real-world entities and large systems are naturally structured.

Q3. What are the advantages and limitations of Object Oriented Programming?

Ans.

Object Oriented Programming offers several advantages: it improves data security through encapsulation, allows reuse of existing, tested code through inheritance, provides flexibility through polymorphism, models real-world entities more naturally, and generally makes large software projects easier to design, extend and maintain over time.

However, OOP also has some limitations. Object oriented programs can be slightly slower and require more memory than equivalent procedural programs because of the additional structure needed to support classes and objects. Designing a good class hierarchy requires careful planning, and a poorly designed hierarchy can make a program harder to understand rather than easier. In addition, OOP has a steeper learning curve for beginners compared to straightforward procedural programming, and not every small or simple problem genuinely benefits from being modelled using classes and objects.

Q4. Explain, with suitable examples, any five points of difference between object oriented languages and object based languages.

Ans.

- Inheritance support: Object oriented languages such as C++ and Java support inheritance, allowing one class to derive from another; object based languages such as Ada generally do not support this feature.
- Polymorphism support: Object oriented languages support polymorphism through overloading and overriding; object based languages typically lack full support for polymorphism.
- Class hierarchy: Object oriented languages allow multi-level class hierarchies to be built; object based languages usually work with independent, standalone objects without a hierarchy.
- Code reusability: Object oriented languages offer high reusability through inheritance; object based languages offer comparatively limited reusability since classes cannot be extended.
- Examples: C++, Java, Python and C# are object oriented languages, whereas Ada, early Visual Basic and JavaScript's basic object model are object based languages.

In summary, both categories support encapsulation and the grouping of data with related functions as objects, but only fully object oriented languages provide the additional power of inheritance and polymorphism.

Q5. Explain the stream class hierarchy in C++ (ios, istream, ostream and iostream) with a suitable diagram description.

Ans.

C++ input/output is built around a hierarchy of stream classes defined in the header file iostream.h. At the top of this hierarchy is the class ios, which defines the basic properties and formatting flags common to all streams. From ios, two important classes are derived: istream, which handles input operations, and ostream, which handles output operations.

The predefined object cin is an object of the istream class and is used with the extraction operator (>>) to read input from the keyboard. The predefined object cout is an object of the ostream class and is used with the insertion operator (<<) to send output to the screen. The class iostream is derived from both istream and ostream (through multiple inheritance) and therefore supports both input and output operations together; objects such as fstream, used for file handling, are built upon this same hierarchy.

Q6. What are manipulators in C++? Explain the use of endl and setw() with examples.

Ans.

Manipulators are special functions in C++ that can be used along with the insertion (<<) or extraction (>>) operators to format input or output. They allow a programmer to control the appearance of output, such as line spacing, field width and number of decimal places, without writing extra formatting code.

The endl manipulator inserts a newline character into the output stream and also flushes the stream buffer, so cout << "Line 1" << endl << "Line 2"; prints each string on its own line. The setw() manipulator, declared in iomanip.h, sets the minimum field width for the next value to be printed, which is useful for aligning columns of numbers or text, for example cout << setw(10) << 25; right-aligns the number 25 within a field of 10 characters.

Q7. Explain the concept of cascading of insertion and extraction operators with an example.

Ans.

Cascading refers to the ability to chain multiple insertion or extraction operators together in a single statement, since each use of << or >> returns a reference to the same stream object, allowing another << or >> to be applied immediately afterward. For output, a statement such as cout << "Sum = " << a + b << endl; displays a text label, a computed value, and a newline, all in a single chained statement.

Similarly, for input, a statement such as cin >> a >> b >> c; reads three separate values typed by the user (usually separated by spaces or the Enter key) into the three variables a, b and c in a single statement. Cascading makes C++ programs more compact and readable compared to writing a separate cout or cin statement for every single value.

Q8. Compare procedural programming languages and object oriented programming languages, giving real-world type examples of programs suited to each.

Ans.

Procedural languages such as C are well suited to small, linear tasks where the sequence of steps is the main concern, for example a simple program to calculate the average of a list of numbers or to convert temperature from Celsius to Fahrenheit; here, the data is simple and does not need to be protected or extended later.

Object oriented languages such as C++ or Java are better suited to larger, evolving systems that model multiple interacting real-world entities, for example a college management system with Student, Course and Faculty as separate classes, or a banking system with Account, Customer and Transaction classes. In such systems, the ability to protect data (encapsulation), reuse common features (inheritance) and extend behaviour later (polymorphism) makes the object oriented approach far more maintainable than a purely procedural one.

Section C: Coding Type Questions & Answers

The following questions require writing a small C++ program based on program structure and console input/output using cin and cout, as covered in this chapter.

Q1. Write a C++ program to accept the length and width of a rectangle and calculate its area and perimeter.

Ans.

```
#include <iostream.h>
void main()
{
    float length, width, area, perimeter;
    cout << "Enter length and width of rectangle: ";
    cin >> length >> width;
    area = length * width;
    perimeter = 2 * (length + width);
    cout << "Area = " << area << endl;
    cout << "Perimeter = " << perimeter;
}
```

Output:

```
Enter length and width of rectangle: 8 5
Area = 40
Perimeter = 26
```

Q2. Write a C++ program to swap the values of two variables using a third (temporary) variable.

Ans.

```
#include <iostream.h>
void main()
{
    int a, b, temp;
    cout << "Enter values of a and b: ";
    cin >> a >> b;
    temp = a;
    a = b;
    b = temp;
    cout << "After swapping: a = " << a << ", b = " << b;
}
```

Output:

```
Enter values of a and b: 5 9
After swapping: a = 9, b = 5
```

Q3. Write a C++ program to calculate the simple interest for a given principal, rate and time.

Ans.

```
#include <iostream.h>
void main()
{
    float principal, rate, time, interest;
    cout << "Enter principal, rate and time: ";
```

```
    cin >> principal >> rate >> time;
    interest = (principal * rate * time) / 100;
    cout << "Simple Interest = " << interest;
}
```

Output:

```
Enter principal, rate and time: 10000 5 2
Simple Interest = 1000
```

Q4. Write a C++ program that uses the setw() manipulator to display two numbers in aligned columns.

Ans.

```
#include <iostream.h>
#include <iomanip.h>
void main()
{
    int a, b;
    cout << "Enter two numbers: ";
    cin >> a >> b;
    cout << "Number 1: " << setw(6) << a << endl;
    cout << "Number 2: " << setw(6) << b;
}
```

Output:

```
Enter two numbers: 7 154
Number 1:      7
Number 2:    154
```

Q5. Write a C++ program to accept the marks of a student in five subjects and display the total and average marks.

Ans.

```
#include <iostream.h>
void main()
{
    int m1, m2, m3, m4, m5, total;
    float average;
    cout << "Enter marks in five subjects: ";
    cin >> m1 >> m2 >> m3 >> m4 >> m5;
    total = m1 + m2 + m3 + m4 + m5;
    average = total / 5.0;
    cout << "Total Marks = " << total << endl;
    cout << "Average Marks = " << average;
}
```

Output:

```
Enter marks in five subjects: 80 75 90 85 70
Total Marks = 400
Average Marks = 80
```

Note: The above programs use void main() and the iostream.h/iomanip.h headers, following the older Turbo C++ style commonly used at the diploma/polytechnic level. In standard modern C++ (ISO C++), the equivalent

structure uses #include <iostream>, #include <iomanip>, the statement using namespace std;, and int main() with a return 0; statement at the end.