Introduction to OOPS Programming

1. Introduction to C++

Q1: What are the key differences between Procedural Programming and Object-Oriented Programming (OOP)?
Procedural Programming (POP) organizes programs around functions (procedures). Data is separate from functions and is often global, which can make it less secure. It is best suited for small programs. In contrast, Object-Oriented Programming (OOP) organizes programs around objects that combine both data and methods together. This makes programs more modular, secure, and reusable. Key Points: - POP: Focus on functions, data is global, less secure, good for small programs. - OOP: Focus on objects, encapsulation ensures security, suitable for large programs. - POP Example: C language; OOP Example: C++, Java, Python.

Q2: List and explain the main advantages of OOP over POP.
Advantages of OOP: 1. Encapsulation: Data is hidden inside classes, only accessible through methods. 2. Reusability: Classes can be reused; inheritance reduces code duplication. 3. Modularity: Programs are divided into independent, manageable classes. 4. Polymorphism: Same function name works differently depending on the object. 5. Abstraction: Hides complex details and shows simple interfaces. 6. Maintainability: Easy to debug and update programs. Conclusion: OOP makes large programs easier to build, reuse, and maintain compared to POP.

Q3: Explain the steps involved in setting up a C++ development environment.
Steps: 1. Install a compiler like GCC, MSVC, or Turbo C++ for labs. 2. Install an IDE such as Code::Blocks, Dev-C++, Visual Studio, or Turbo C++. 3. Configure compiler paths if necessary. 4. Create a new project or .cpp file. 5. Write a program (e.g., Hello World). 6. Compile the program to check for errors. 7. Run the program to see output. 8. Debug if needed to fix mistakes. Example (Turbo C++ style): #include #include void main() { clrscr(); cout << "Hello, World!"; getch(); }

Q4: What are the main input/output operations in C++? Provide examples.
C++ provides input/output streams: 1. cout: For output. Example: cout << "Hello"; 2. cin: For input. Example: cin >> name; 3. endl: Inserts new line and flushes buffer. 4. cerr: For error messages. 5. clog: For logging/debugging. Example: #include #include void main() { clrscr(); char name[20]; int age; cout << "Enter name: "; cin >> name; cout << "Enter age: "; cin >> age; cout << "Hello " << name << ", Age: " << age; getch(); }

2. Variables, Data Types, and Operators

Q1: What are the different data types available in C++? Explain with examples.
Data types: - int: whole numbers. Example: int age = 20; - char: characters. Example: char grade = 'A'; - bool: true/false. Example: bool pass = true; - float/double: decimal numbers. Example: double pi = 3.14; - string: text data. Example: string name = "Harsh"; - Arrays, pointers, structures, and classes are advanced data types.

Q2: Explain the difference between implicit and explicit type conversion in C++.
Implicit conversion: Done automatically by compiler. Example: int a = 5; double b = 2.5; double c = a + b; (a becomes double). Explicit conversion: Done manually using cast. Example: int x = (int)3.7; (x becomes 3). Implicit is automatic, explicit gives programmer control.

Q3: What are different types of operators in C++? Provide examples of each.
Operators: - Arithmetic: +, -, *, /, %. Example: sum = a + b; - Relational: ==, !=, >, <, >=, <=. Example: if (a > b). - Logical: &&, ||, !. Example: if (x > 0 && y > 0). - Bitwise: &, |, ^, ~, <<, >>. - Assignment: =, +=, -=, *=. Example: a += 5; - Increment/Decrement: ++, --. Example: i++; -

Conditional (ternary): condition ? value1 : value2.

Q4: Explain the purpose and use of constants and literals in C++.
Literals: fixed values like 10, 3.14, 'A', "Hello". Constants: variables whose value cannot be changed. Declared with const keyword. Example: const double PI = 3.14159; Purpose: make code readable, prevent errors, easy maintenance.

3. Control Flow Statements

Q1: What are conditional statements in C++? Explain the if-else and switch statements.
Conditional statements let programs take decisions. if-else: checks conditions. Example: if (marks >= 50) cout << "Pass"; else cout << "Fail"; switch: checks multiple values of one variable. Example: switch(choice) { case 1: cout << "Option 1"; break; case 2: cout << "Option 2"; break; default: cout << "Invalid"; }

Q2: What is the difference between for, while, and do-while loops in C++?
for: used when number of iterations is known. Example: for(int i=0;i<10;i++). while: repeats as long as condition is true. Example: while(x < 5). do-while: executes at least once before checking condition. Example: do { cout << x; } while(x < 5);

Q3: How are break and continue statements used in loops? Provide examples.
break: exits loop immediately. Example: if(x==5) break; continue: skips current iteration and goes to next. Example: if(x%2==0) continue;

Q4: Explain nested control structures with an example.
Nested control structures are placing one inside another. Example: Nested for-loops to print triangle pattern: for(int i=1;i<=5;i++){ for(int j=1;j<=i;j++) cout << "*"; cout << endl; }

4. Functions and Scope

Q1: What is a function in C++? Explain function declaration, definition, and calling.
Function: block of code that performs a task. Declaration: tells compiler function exists. int add(int, int); Definition: contains function body. int add(int a,int b){return a+b;} Calling: using the function. cout << add(2,3);

Q2: What is the scope of variables in C++? Differentiate between local and global scope.
Scope defines where a variable can be accessed. Local: declared inside function, accessible only there. Global: declared outside functions, accessible everywhere. Example: int g=10; //global void main(){ int x=5; //local }

Q3: Explain recursion in C++ with an example.
Recursion: function calling itself. Example factorial: int fact(int n){ if(n==0) return 1; return n*fact(n-1); }

Q4: What are function prototypes in C++? Why are they used?
Prototype: declares function before use. Example: int sum(int,int); Used to allow calling functions before their definition.

5. Arrays and Strings

Q1: What are arrays in C++? Explain the difference between single-dimensional and multi-dimensional arrays.
Array: collection of same data type. 1D array: linear list. Example: int a[5] = {1,2,3,4,5}; 2D array: table. Example: int b[2][3] = {{1,2,3},{4,5,6}};

Q2: Explain string handling in C++ with examples.
C++ supports C-style strings and string class. C-style: char name[20] = "Hello"; String class: string s="World"; cout << s.length(); String class is easier and safer.

Q3: How are arrays initialized in C++? Provide examples of both 1D and 2D arrays.
1D: int a[5] = {1,2,3,4,5}; Partial: int b[5] = {1,2}; //rest 0 2D: int m[2][3] = {{1,2,3},{4,5,6}};

Q4: Explain string operations and functions in C++.
String operations: - Concatenation: s1 + s2 - Length: s.length() - Substring: s.substr(0,3) - Find: s.find("lo") Example: string s="Hello"; cout << s.substr(1,3);

6. Introduction to Object-Oriented Programming

Q1: Explain the key concepts of Object-Oriented Programming (OOP).
Key concepts: 1. Encapsulation: hiding data inside class. 2. Abstraction: showing essential features only. 3. Inheritance: reuse of code from base to derived class. 4. Polymorphism: same name different behaviors. These features make OOP powerful and reusable.

Q2: What are classes and objects in C++? Provide an example.
Class: blueprint for objects. Contains data and methods. Object: instance of class. Example: class Person { char name[20]; int age; public: void setData(){...} }; Person p1; //object

Q3: What is inheritance in C++? Explain with an example.
Inheritance: mechanism of reusing code from base class. Example: class Person { public: char name[20]; }; class Student : public Person { int roll; }; Here Student inherits name from Person.

Q4: What is encapsulation in C++? How is it achieved in classes?
Encapsulation: binding data and functions together. Achieved using private data members and public methods. Example: class Bank { private: int balance; public: void deposit(int amt){ balance+=amt; } };