1. Discuss at least FIVE major Object-Oriented Principles (5 marks)

Object-Oriented Programming (OOP) is grounded on foundational principles designed to support modular development, enhance maintainability, and increase reusability of software systems. The following are five major OOP principles:

1. **Encapsulation**
   Encapsulation refers to the practice of bundling data (attributes) and the functions (methods) that operate on that data within a single unit known as a class. It restricts direct access to internal object states and forces interaction through well-defined interfaces. By using access specifiers such as private, protected, and public, encapsulation enhances data integrity and prevents unintended interference.
2. **Abstraction**
   Abstraction allows developers to hide complex implementation details and expose only the essential features required by the user. This is achieved through abstract classes, interfaces, and encapsulated methods. Abstraction reduces complexity, enhances clarity, and helps create scalable systems by focusing on what an object does rather than how it does it.
3. **Inheritance**
   Inheritance enables a new class (derived or child class) to acquire properties and behaviors from an existing class (base or parent class). It promotes code reuse, minimizes redundancy, and establishes hierarchical classification. Through inheritance, developers can extend existing functionality without modifying the original code, allowing easy implementation of specialized behavior.
4. **Polymorphism**
   Polymorphism refers to the ability of a function, method, or operator to exhibit different behaviors depending on the context or the type of object invoking it. It occurs in two forms: compile-time polymorphism (through function and operator overloading) and runtime polymorphism (through method overriding using virtual functions). Polymorphism improves flexibility and supports the implementation of dynamic systems.
5. **Modularity**
   Modularity organizes software into independent components (classes and objects) that can be developed, tested, and maintained separately. It enhances clarity and simplifies debugging by isolating functionality into cohesive units. Modularity also allows multiple developers to work concurrently on different parts of a system, leading to efficient project management.

---

2. Explain the TEN major features of Object-Oriented Programming (5 marks)

Object-Oriented Programming contains several distinguishing features that make it a powerful software development paradigm:

1. **Classes** – Classes serve as templates for creating objects. They define attributes (data members) and behaviors (member functions). They enable structured modelling of real-world entities.
2. **Objects** – An object is a specific instance of a class. It represents an entity with identifiable state and behavior. Objects interact through method calls and form the building blocks of OOP applications.
3. **Encapsulation** – Encapsulation ensures internal data protection by restricting direct access and exposing controlled interfaces. It is crucial for preventing unintended data manipulation and improving integrity.
4. **Abstraction** – Abstraction masks unnecessary details, offering simplified interaction with complex components. It promotes conceptual clarity and facilitates system design at higher levels.
5. **Inheritance** – Inheritance supports code reuse by enabling new classes to derive attributes and methods from existing classes. It minimizes duplication and provides a mechanism for defining general-to-specific relationships.
6. **Polymorphism** – Polymorphism introduces flexibility by allowing a single interface to have multiple implementations. It enables functions and operators to act differently based on context or object type.
7. **Message Passing** – Objects interact by sending messages, typically implemented as method calls. Each object responds based on its own implementation, ensuring decoupled and dynamic communication.
8. **Dynamic Binding** – Also known as late binding, dynamic binding resolves method calls during runtime. This supports runtime polymorphism and enhances flexibility in decision-making during execution.
9. **Data Hiding** – Data hiding is achieved through access specifiers and ensures that object attributes remain protected from unauthorized access. It strengthens security and preserves the correctness of object states.
10. **Reusability** – OOP facilitates reusability through inheritance, polymorphism, and modular design. Reusable components reduce development time, promote consistency, and simplify system extensions.

---

3. Discuss the major data types used in C++ programming (5 marks)

1. **Integer Types**
   These are used to store whole numbers, both positive and negative. They include:
   - `int`: Standard integer type.
   - `short`: A smaller integer type.
   - `long` and `long long`: Larger types used for long integers.

   These types differ in size, range, and memory usage.
2. **Floating-Point Types**
   Used for representing real numbers that include fractional parts. They include:
   - `float` (single precision)
   - `double` (double precision)
   - `long double` (extended precision)

   These types support scientific notation and are essential for mathematical and statistical computations.
3. **Character Type (char)**
   The `char` type stores a single character (e.g., 'A', 'b', '5') using ASCII encoding. It occupies one byte and can hold numeric or symbolic values.
4. **Boolean Type (bool)**
   This type stores logical values: either `true` or `false`. Boolean values are widely used in control statements and decision-making logic.
5. **Void Type (void)**
   The void type represents the absence of a value. It is used primarily in functions that do not return data, and in pointers that must be type-neutral.
6. **Wide Character (wchar_t)**
   Used for representing larger character sets beyond ASCII, particularly useful in internationalized or Unicode-based applications.

---

4. State and explain at least 5 keywords used in C++ (5 marks)

1. **class** – Declares a user-defined type that encapsulates attributes and methods.
2. **public** – An access specifier that makes class members accessible from outside the class.
3. **private** – Restricts access to class members, allowing access only within the class.
4. **return** – Used within a function to specify the value to be sent back to the caller.
5. **new** – Allocates memory dynamically on the heap and returns a pointer to the allocated memory.

Each keyword serves a specific syntactic and semantic role in program structure and execution.

---

5. Discuss the control structures used in C++ programming (5 marks)

Control structures regulate the flow of execution in a program. C++ provides three categories:

1. **Selection Control Structures**
   These structures allow decisions to be made based on conditions.
   - **if statement**: Executes a block if the condition is true.
   - **if-else statement**: Provides an alternative block when the condition is false.
   - **switch statement**: Selects a code block based on the value of an expression, usually when handling multiple discrete cases.
2. **Iteration (Looping) Control Structures**
   These structures repeat a block of code until a certain condition is met.
   - **for loop**: Ideal for loops with a known number of iterations.
   - **while loop**: Continues executing as long as the condition remains true.
   - **do-while loop**: Executes the loop body at least once before evaluating the condition.
3. **Jump Control Structures**
   Used to alter the normal sequence of execution.
   - **break**: Terminates a loop or switch statement.
   - **continue**: Skips the current iteration and proceeds to the next.
   - **goto**: Transfers control to a labeled statement, though generally discouraged due to readability issues.

---

6. Write a function in C++ to add two integers and return the result (5 marks)

```
int addNumbers(int a, int b) {
    return a + b;
}
```

---

7. Discuss the concept of Polymorphism in OOP (5 marks)

Polymorphism is one of the central pillars of Object-Oriented Programming. It allows different classes to be treated as instances of the same parent class, while each may respond differently to the same method call.
There are two primary forms:

1. **Compile-Time Polymorphism**
   Achieved through:
   - **Function Overloading**: Multiple functions with the same name but different parameter lists.
   - **Operator Overloading**: Redefining standard operators (e.g., +, -, ==) for user-defined classes.

     Resolution occurs during compilation.
2. **Runtime Polymorphism**
   Implemented through:
   - **Method Overriding**: A derived class provides a specific implementation of a method in the base class.
   - **Virtual Functions**: Allow dynamic binding so that the method executed depends on the object type at runtime.

---

8. Encapsulation and data hiding (2 marks)
Encapsulation is the process of bundling data and the methods that manipulate that data within a class. Data hiding is a related concept that restricts direct access to object attributes by using access specifiers such as private and protected. Together, they safeguard data integrity, prevent unauthorized modification, simplify interface design, and support modular development.

---

9. Operator Overloading and Polymorphism (2 marks)
Operator overloading allows built-in operators to be redefined for user-defined data types. This enables intuitive operations on objects, such as adding two objects using the + operator. Operator overloading is a form of compile-time polymorphism because the compiler determines which operator function to invoke based on the type and number of operands. It enhances code readability and supports natural syntax usage for complex types.

---

10. Write a C++ program to demonstrate inheritance (1 mark)

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

class Animal {
public:
   void eat() {
      cout << "Animal is eating" << endl;
   }
};

class Dog : public Animal {
public:
   void bark() {
      cout << "Dog is barking" << endl;
   }
};

int main() {
   Dog d;
   d.eat();   // inherited from Animal
   d.bark();  // specific to Dog
   return 0;
}
```

1. Discuss the TWO building blocks in Object-Oriented Programming (3 marks)
Object-Oriented Programming (OOP) is fundamentally built on two core building blocks: **classes** and **objects**.

1. **Classes**
   A class is a user-defined blueprint or template that defines the properties (attributes) and behaviors (methods) of entities within a system. It specifies the structure and operations that its objects will possess. Classes encapsulate data and functions into a single unit and define the form and functionality of objects without occupying memory until instantiated. They provide modularity, reusability, and a logical approach to system modeling.
2. **Objects**
   An object is an instance of a class. It represents a specific, tangible occurrence of a class with actual values assigned to its attributes. Objects interact with one another by sending messages (calling methods), thereby enabling system functionality. Because each object maintains its own state, OOP systems support modular, maintainable, and scalable designs. Objects embody real-world entities such as students, accounts, or vehicles within software applications.

---

2. Explain the basic principles of Object-Oriented Programming (3 marks)
OOP is governed by four foundational principles that enable robust and maintainable software development:

1. **Encapsulation**
   Encapsulation packages data (attributes) and functions (methods) into a single unit (class) while restricting direct access through access specifiers such as private, protected, and public. It enhances data integrity and creates clear interfaces.
2. **Abstraction**
   Abstraction focuses on exposing only essential features while hiding the underlying complexity. This allows developers and users to interact with a simplified representation of objects without concerning themselves with implementation details.
3. **Inheritance**
   Inheritance allows a new class (subclass) to acquire the properties and behaviors of an existing class (superclass). It supports hierarchical classifications, reduces redundancy, and promotes code reuse.
4. **Polymorphism**
   Polymorphism allows a single interface or method to perform different functions depending on the context or the type of object invoking it. It occurs through method overloading (compile-time) and method overriding (runtime), enhancing flexibility and extensibility.

---

3. Explain the functions of at least 5 keywords used in OOP (3 marks)

1. **class** – Declares a user-defined type that encapsulates data and functionality.
2. **public** – An access specifier that allows class members to be accessed from outside the class.
3. **private** – Restricts access to class members, making them accessible only within the class.
4. **protected** – Allows access within the class and derived classes, but not from outside.
5. **virtual** – Enables runtime polymorphism by allowing derived classes to override a base class function dynamically.

---

4. Discuss the different types of abstractions and their implementations (3 marks)

1. **Data Abstraction**
   Data abstraction focuses on representing essential attributes while hiding implementation details. It allows interaction with data through well-defined interfaces rather than direct manipulation.
   Implementation:
   - Achieved through classes and objects
   - Access specifiers such as private and protected
   - Getter and setter functions that control data access
2. **Control Abstraction**
   Control abstraction hides the internal logic of operations while presenting simplified interfaces. Users call methods without knowing the underlying algorithms.
   Implementation:
   - Functions and methods with clear signatures
   - Abstract classes that outline method prototypes
   - Interfaces that enforce method implementation in derived classes

---

5. How does OOP differ from the Procedural Programming paradigm? (3 marks)

OOP and procedural programming differ in structure, design philosophy, and application approach:

1. **Program Structure**
   - **OOP** structures programs around objects and classes that model real-world entities.
   - **Procedural programming** structures programs around procedures or functions that operate on data.
2. **Data Handling**
   - **OOP** binds data and functions together, enabling data hiding and encapsulation.
   - **Procedural programming** keeps data and functions separate, leading to potential unintended access and modifications.
3. **Reusability and Extensibility**
   - **OOP** supports inheritance and polymorphism, making it easy to extend and reuse code.
   - **Procedural programming** relies on functions and modules, which offer limited reusability without object structures.
4. **Flexibility and Maintenance**
   - **OOP** allows modular design, making systems easier to maintain and modify.
   - **Procedural programming** may become difficult to manage as complexity increases due to its linear flow.
5. **Application Domain**
   - **OOP** is ideal for large-scale, complex systems with dynamic requirements.
   - **Procedural programming** is suitable for small or simple programs with a clear sequential flow.