

Object Oriented Programming with C++

Aakash

Roll Number - 01

Section: IoT (CSE-Semester III)

Submitted To: Mr. Sanjeev Kukreti

Date of Submission

September 2023

Assignment 1

***Q 1. What are the differences between procedural and object-oriented programming?***

Solution 1:

**Procedural programming** is a coding paradigm that stems from structured programming and relies on invoking procedures, which encompass functions, routines, or subroutines defining computational steps. This methodology adopts a sequential approach, deconstructing tasks into variables and routines via a series of instructions. In the execution of a program, a procedure can be invoked at any juncture, either by other procedures or by itself. Classic examples of procedural programming languages include ALGOL, COBOL, BASIC, PASCAL, FORTRAN, and C.

**Object-oriented programming** (OOP) is a design philosophy focusing on modeling software around objects or data instead of functions. It's a programming pattern centered on real-world entities, known as objects. Smalltalk was the first true object-oriented language.

* OOP is favored for large, complex software, offering concepts like abstraction, inheritance, polymorphism, and encapsulation.
* It effectively simulates real-world events and provides data hiding, enhancing security by encapsulating critical data.

Differences based on some basis can be discussed as follows:

|  |  |  |
| --- | --- | --- |
| Basis | Procedural Programming | Object Oriented Programming |
| Security | It is less than OPPs. | Data hiding is possible in OPPs due to abstraction, so it is more secure. |
| Approach | It follows a top-down approach. | It follows a bottom-up approach. |
| Data Movement | Data moves freely within the system from one function to another. | In OPPs, objects can move and communicate with each other via member functions. |
| Access modifiers | There are no access modifiers in procedural-oriented | The access modifiers in OPPs are named private, public, and protected. |
| Inheritance | Procedural programming does not have the concept of inheritance. | There is a feature of inheritance in OPPs. |
| Overloading | Overloading is not possible. | Function and operator overloading are possible. |
| Code reusability | Code is reusable using functions but does not provide features. | It offers code reusability using different features like inheritance. |

***Q 2. Write brief notes on concepts of object-oriented programming in C++.***

Solution 2: Object-oriented programming (OOP) in C++ is built on fundamental concepts that enable the creation and management of objects, enhancing code organization, reusability, and modularity. Here are the key concepts:

* **Classes and Objects:**

**Class:** A blueprint that defines the structure and behavior of objects. It contains member variables (data) and member functions (methods).

**Object:** An instance of a class, representing a real-world entity. Objects encapsulate data and behavior, following the class blueprint.

* **Encapsulation:**

Encapsulation restricts access to certain components within a class, typically through access specifiers (public, private, protected).

**Private:** Accessible only within the class.

**Public:** Accessible from outside the class.

**Protected:** Accessible within the class and its subclasses.

* **Inheritance**:

Inheritance allows a class to inherit properties and behaviors (methods) from another class. It promotes code reuse and the creation of a hierarchy of classes.

Types: Single, Multiple, Multilevel, Hierarchical, Hybrid inheritance.

* **Polymorphism**:

Polymorphism means "many forms" and allows objects to be treated as instances of their parent class. Achieved through function overloading and function overriding.

**Function Overloading:** Multiple functions with the same name but different parameters.

**Function Overriding:** Redefining a function in a derived class with the same signature as in the base class.

* **Abstraction**:

Abstraction involves showcasing only essential features of an object and hiding unnecessary details. It helps in reducing complexity and focusing on what an object does, not how it achieves its functionality.

* **Constructor and Destructor:**

**Constructor:** Special member function invoked automatically when an object is created, used for initializing object state.

**Destructor:** Special member function invoked automatically when an object goes out of scope or is explicitly deleted, used for cleanup operations.

* **Friend Functions and Classes:**

**Friend Functions:** Functions that grant access to private and protected members of a class.

**Friend Classes:** Entire classes can be declared as friends, providing access to private and protected members.

Understanding and effectively implementing these concepts in C++ forms the basis for successful object-oriented programming, facilitating better code organization, flexibility, and maintainability.

***Q 3. What are the differences between classes and structures?***

Solution 3: A structure is a grouping of variables of various data types referenced by the same name. A structure declaration serves as a template for creating an instance of the structure. The "struct" keyword indicates to the compiler that a structure has been declared. The "structure name" defines the name of the structure. Since the structure declaration is treated as a statement, it is often ended by a semicolon.

struct structure\_name

{

// Data members

};

A class in C++ is like a C structure in that it consists of a list of data members and a set of operations performed on the class. In other words, a class is the building block of Object-Oriented programming. It is a user-defined object type with its own set of data members and member functions that can be accessed and used by creating a class instance. A C++ class is like an object's blueprint.

class class\_name

{

// private data members and member functions.

Access specifier:

Data member;

Member functions (member list){ . . }

};

Differences based on some basis can be discussed as follows:

|  |  |  |
| --- | --- | --- |
| **Basis** | **Structure** | **Class** |
| Instance | The structure instance is called the 'structure variable'. | A class instance is called an 'object'. |
| Inheritance | It does not support inheritance. | It supports inheritance. |
| Memory Allocated | Memory is allocated on the stack. | Memory is allocated on the heap. |
| Nature | Value Type | Reference Type |
| Purpose | Grouping of data | Data abstraction and further inheritance. |
| Usage | It is used for smaller amounts of data. | It is used for a huge amount of data. |
| Null values | Not possible | It may have null values. |
| Requires constructor and destructor | Every structure has an implicit public constructor without parameters. | It may have all the types of constructors and destructors. |