Unit I

C++ Class Overview- Class Definition, Objects, Class Members, Access Control, Class Scope, Constructors and destructors, parameter passing methods, Inline functions, static class members, this
pointer, friend functions, Function Overloading, Operator Overloading,dynamic memory allocation and
de-allocation (new and delete), exception handling.

C++ Class Overview

1. **Class Definition**:
   - **Blueprint for Objects**: Defines the properties and behaviors of objects.
   - **Encapsulation**: Bundles data and methods together.
   - **Syntax**: `class ClassName { /* members and methods */ };`
2. **Objects**:
   - **Instances of Classes**: Created from class blueprints.
   - **Attributes and Methods**: Hold data and perform operations.
3. **Class Members**:
   - **Data Members**: Variables within a class.
   - **Member Functions**: Functions that operate on the class's data.
4. **Access Control**:
   - **Public, Private, Protected**: Keywords determining access levels to class members.
   - **Encapsulation**: Controlling access to data for better security and integrity.
5. **Class Scope**:
   - **Scope Resolution Operator** (::): Used to access class members.
   - **Namespace**: Defines the scope in which identifiers exist.
6. **Constructors and Destructors**:
   - **Constructor**: Initializes object properties when an object is created.
   - **Destructor**: Releases resources and performs cleanup when an object goes out of scope.
7. **Parameter Passing Methods**:
   - **Pass by Value, Pass by Reference**: Different ways to pass arguments to functions.
   - **Const Reference Parameters**: Prevents modification of original data.
8. **Inline Functions**:
   - **Performance Optimization**: Reduces function call overhead by inserting code directly.

    - **Defined in Header Files**: Typically used for short functions.
9. **Static Class Members**:
    - **Shared Among Objects**: Belongs to the class, not to any specific object instance.
    - **Accessed without Object Instance**: Accessed using the class name.
10. **This Pointer**:
- **Pointer to Object**: Refers to the current object within a class.
- **Used for Differentiation**: Helps distinguish between local and class members.

11. **Friend Functions**:
- **Access Private Members**: Functions that are granted access to a class's private and protected members.
- **Not Member Functions**: Not part of the class but has access to its members.

12. **Function Overloading**:
- **Multiple Definitions**: Defining multiple functions with the same name but different parameters.
- **Compile-Time Polymorphism**: Resolves the function call during compilation.

13. **Operator Overloading**:
- **Defining Operators for Custom Types**: Redefining the behavior of operators for user-defined classes.
- **Syntax**: `ReturnType operatorSymbol(Parameters);`

14. **Dynamic Memory Allocation and Deallocation**:
- **New and Delete Operators**: Used for allocating and deallocating memory dynamically.
- **Memory Management**: Avoiding memory leaks by deallocating memory appropriately.

15. **Exception Handling**:
- **Error Handling Mechanism**: Handles runtime errors or exceptional conditions.
- **Try-Catch Blocks**: Tries to execute a block of code and catches exceptions that occur.

C++ Class Overview

1. Class Definition, Objects, and Class Members:

Class Definition:

A class in C++ is a blueprint that defines the properties and behaviors of objects. It encapsulates data (attributes) and functions (methods) that operate on that data.

```
class Car {
public:
// Attributes
string brand;
int year;

// Methods
void displayInfo() {
cout << "Brand: " << brand << ", Year: " << year << endl;
}
};
```

Objects:

Objects are instances of a class. They represent real-world entities and have their own unique set of attributes and behaviors.

```
Car myCar; // Creating an object of the Car class

// Accessing attributes and methods of the object
myCar.brand = "Toyota";
myCar.year = 2022;
myCar.displayInfo(); // Output: Brand: Toyota, Year: 2022
```

2. Access Control and Class Scope:

Access Control:

C++ provides three access specifiers: public, private, and protected, which determine the access levels of class members.

```
class MyClass {
private:
int privateVar; // Accessible only within this class

public:
void setPrivateVar(int value) {
```

```
privateVar = value;
}
};
```

<u>Class Scope:</u>

The scope resolution operator (::) is used to define methods outside the class.

```
void MyClass::setPrivateVar(int value) {
privateVar = value;
}
```

<u>3. Constructors and Destructors:</u>

Constructors:

Constructors are special member functions invoked automatically when an object is created. They initialize object properties.

```
class Book {
public:
// Constructor
Book(string title, string author) {
this->title = title;
this->author = author;
}
};
```

Destructors:

Destructors are invoked when an object goes out of scope or is explicitly deleted. They perform cleanup tasks.

```
class Book {
public:
// Destructor
~Book() {
```

```
// Cleanup code
}
};
```

4. Parameter Passing Methods:

Pass by Value, Reference, and Pointer:

Different methods of passing parameters to functions.

```
void passByValue(int val) { /* ... */ } // Passed by value
void passByReference(int &val) { /* ... */ } // Passed by reference
void passByPointer(int *ptr) { /* ... */ } // Passed by pointer
```

5. Inline Functions:

Inline Functions:

inline keyword is used for small functions to avoid function call overhead.

```
inline int multiply(int a, int b) {
return a * b;
}
```

6. Static Class Members and this Pointer:

Static Members:

Static members belong to the class rather than instances. They are shared among all objects of the class.

```
class Example {
public:
static int count; // Static member
};
int Example::count = 0; // Initializing static member
```

this Pointer:

this refers to the current object within a class. It's used to access object members.

```
class Person {
private:
string name;

public:
void setName(string name) {
this->name = name;
}
};
```

7. Friend Functions:

Friend Functions:

Functions declared as friends can access private and protected members of a class.

```
class MyClass {
private:
int privateVar;

public:
friend void friendFunction(MyClass obj);
};
void friendFunction(MyClass obj) {

obj.privateVar = 10;

}
```

8. Function Overloading and Operator Overloading:

Function Overloading:

Defining multiple functions with the same name but different parameters.

```
void display(int val) { /* ... */ }

void display(double val) { /* ... */ }
```

Operator Overloading:

Redefining operators for user-defined types.

```
class Complex {
public:
Complex operator+(const Complex &other) {
// Define addition for Complex numbers
}
};
```

9. Dynamic Memory Allocation and Deallocation (new and delete):

new and delete:

Used for allocating and deallocating memory dynamically.

```
int *ptr = new int; // Allocation

delete ptr; // Deallocation
```

10. Exception Handling:

Try-Catch Blocks:

Used to handle exceptions and perform error management.

```
try {
// Code that might throw an exception
} catch (ExceptionType &e) {
// Handle exception
}
```