Unit 3: Constructors, Destructors, Operator Overloading, Type Conversion, Virtual Functions, and Polymorphism

Syllabus :
Constructors and Destructors: Need for constructors and destructors, copy constructor, dynamic constructors, explicit constructors, destructors, constructors and destructors with static members, initializer lists.
Operator Overloading and Type Conversion: Overloading operators, rules for overloading operators, overloading of various operators, type conversion - basic type to class type, class type to basic type, class type to another class type.
Virtual functions & Polymorphism: Concept of binding - early binding and late binding, virtual functions, pure virtual functions, abstract clasess, virtual destructors.

Section 1 : Constructors and Destructors

Constructors and destructors are fundamental concepts in object-oriented programming languages like C++. Constructors are special member functions of a class that initialize objects of that class. On the other hand, destructors are used to clean up resources allocated by an object when it goes out of scope or is explicitly destroyed.

1.1 Constructors

Constructors in C++ are special member functions of a class that are automatically called when an object is created. They are primarily used for initializing the objects and setting up any required resources.
Constructors have the same name as the class and can be overloaded to accept different sets of parameters. It is declared in public section of the class. It has no return type so it can't return any value. It can't be inherited though derived classes can call a base class constructor. It can't be virtual.

Need for Constructors

- Initialization: Constructors are used to initialize an object's data members when it is created.
- Default Values: They provide a way to set default values for data members.
- Resource Allocation: They can allocate resources such as memory or file handles when an object is created.

Types of Constructors

1. **Default Constructor:** Initializes an object without any parameters.
2. **Parameterized Constructor:** Initializes an object with given parameters.
3. **Copy Constructor:** Initializes an object using another object of the same class.
4. **Dynamic Constructor:** Allocates memory dynamically for an object.
5. **Explicit Constructor:** Prevents implicit conversions to the class type.

A. Default Constructor

A default constructor is a constructor that does not take any arguments. It is automatically called when an object is created without providing any initialization values. If no constructor is defined in the class, the compiler generates a default constructor with an empty code and no parameter.

Example:

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

class MyClass {
public:
    int value;

    // Default constructor
    MyClass() {
        value = 0;
        cout << "Default constructor called" << endl;
    }

    void display() {
        cout << "Value: " << value << endl;
    }
};

int main() {
    MyClass obj;     // Default constructor is called
    obj.display();   // Value: 0
    return 0;
}
```

B. Parameterized Constructor

A parameterized constructor is a constructor that takes one or more parameters or arguments. It allows the user to initialize objects with specific values at the time of creation.

Example:

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

class MyClass {
public:
    int value;

    // Parameterized constructor
    MyClass(int val) {
        value = val;
        cout << "Parameterized constructor called" << endl;
    }

    void display() {
        cout << "Value: " << value << endl;
    }
};

int main() {
    MyClass obj(42);  // Parameterized constructor is called
    obj.display();    // Value: 42
    return 0;
}
```

C. Copy Constructor

A copy constructor is a special type of constructor that is used to copy data from one object to another. It is called automatically when a new object is created from an existing object, typically using the assignment operator or passing an object by value to a function. The copy constructor performs a deep copy of the object's data members.

Example:

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

class MyClass {
public:
    int value;

    // Parameterized constructor
    MyClass(int val) {
        value = val;
    }

    // Copy constructor
    MyClass(const MyClass &obj) {
        value = obj.value;
        cout << "Copy constructor called" << endl;
    }

    void display() {
        cout << "Value: " << value << endl;
    }
};
```

```
int main() {
    MyClass obj1(42);      // Parameterized constructor is called
    MyClass obj2 = obj1;   // Copy constructor is called
    obj2.display();        // Value: 42
    return 0;
}
```

D. Dynamic Constructor

A dynamic constructor allocates memory dynamically using new or malloc. It is useful when the size of data members is not known at compile time and needs to be allocated at runtime.
Dynamic memory allocation in C++ is typically done using the new operator. While it's not a constructor itself, it's often used within constructors to allocate memory dynamically for data members of a class.

Example:

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

class MyClass {
private:
    int *ptr;
public:
    MyClass(int size) {
        ptr = new int[size];
        cout << "Dynamic constructor called" << endl;
    }

    ~MyClass() {
        delete[] ptr;
        cout << "Destructor called" << endl;
    }
};

int main() {
    MyClass obj(5); // Dynamic constructor is called
    return 0;
}
```

E. Explicit Constructor

An explicit constructor is used to prevent implicit conversions. It is declared with the explicit keyword, ensuring that the constructor cannot be used for implicit conversions and copy-initialization.