LECTURE-11

Constructor

A constructor is a special member function whose task is to initialize the objects of its class. It is special because its name is the same as the class name. The constructor is invoked whenever an object of its associated class is created. It is called constructor because it constructs the values of data members of the class. A constructor is declared and defined as follows:

```
class integer
{
        int m,n;
        public:
        integer (void);  //constructor declared
        ------------
        ------------
};
integer :: integer(void)
{
        m=0;
        n=0;
}
```

When a class contains a constructor like the one defined above it is guaranteed that an object created by the class will be initialized automatically.

For example:

```
integer int1;   //object int1 created
```

This declaration not only creates the object int1 of type integer but also initializes its data members m and n to zero. A constructor that accepts no parameter is called the default constructor. The default constructor for class A is A :: A( ). If no such constructor is defined, then the compiler supplies a default constructor. Therefore, a statement such as:

```
A  a;
```

Invokes the default constructor of the compiler to create the object "a";

Invokes the default constructor of the compiler to create the object a. The constructor functions have some characteristics:

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They don't have return types, not even void and therefore they cannot return values.
- They cannot be inherited, though a derived class can call the base class constructor.
- Like other C++ function, they can have default arguments,
- Constructor can't be virtual.
- An object with a constructor can't be used as a member of union.

Example of default constructor

```
#include<iostream>
using namespace std;
class A
{
	private:
		char nm[];
	public:
		A ()
		{
			cout<<"enter your name:";
			cin>>nm;
		}
		void display()
		{
			cout<<nm;
		}
};
main()
{
	A d;
	d.display();
}
```

Parameterized Constructor:

The constructors that can take arguments are called parameterized constructors. Using parameterized constructor, we can initialize the various data elements of different objects with different values when they are created. **Example:**

```
class integer
{
        int m,n;
        public:
                integer(int x, int y);
                --------
                ---------
};
integer:: integer (int x, int y)
{
m=x;n=y;
}
```

the argument can be passed to the constructor by calling the constructor implicitly.

integer int1 = integer (0,100); // **explicit call**

integer int1(0,100); //**implicite call**

```
#include <iostream>
using namespace std;
class MyClass {
private:
    int value1;
public:
    // Parameterized constructor
    MyClass(int v1) {
        value1 = v1;
    }

    void displayValues() {
        cout << "Value 1: " << value1 << endl;
    }
};
main() {
    MyClass obj(10);
    obj.displayValues();
}
```

Example

```
#include <iostream>
using namespace std;
class Rectangle {
private:
    int length;
    int width;
public:
    // Parameterized constructor
    Rectangle(int len, int wid) {
        length = len;
        width = wid;
    }
    int calculateArea() {
        return length * width;
    }
    void displayDimensions() {
        cout << "Length: " << length << endl;
        cout << "Width: " << width << endl;
    }
};
main() {
    Rectangle rect1(5, 10);
    rect1.displayDimensions();
    cout << "Area: " << rect1.calculateArea() << endl;
}
```

Example

```
#include<iostream>
using namespace std;
class integer
{
              int m,n;
       public:
              integer(int,int);
              void display(void)
              {cout<<"m=:"<<m ;cout<<"n="<<n;}};
integer :: integer( int x,int y) // constructor defined
       {m=x;  n=y;}
main()
{
       integer int1(0, 100); // implicit call
       int1.display( );
}
```

Example of a parameterized constructor in C++, demonstrating a class representing a simple bank account:

```
#include <iostream>
using namespace std;
class BankAccount {
private:
    string accountNumber;
    double balance;
public:
    BankAccount(string accNum, double initialBalance) {
        accountNumber = accNum;
        balance = initialBalance;
    }

    void deposit(double amount) {
        balance += amount;
        cout << "Deposited: " << amount << " into account " <<
accountNumber << endl;
    }

    void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
            cout << "Withdrawn: " << amount << " from account " <<
accountNumber << endl;
        } else {
            cout << "Insufficient balance in account " <<
accountNumber << " for withdrawal." << endl;
        }
    }

    void displayBalance() {
        cout << "Account Number: " << accountNumber << endl;
        cout << "Current Balance: " << balance << endl;
    }
};

int main() {
    BankAccount myAccount("123456789", 1000.0);
    myAccount.displayBalance();
    myAccount.deposit(500.0);
    myAccount.withdraw(200.0);
    myAccount.withdraw(1500.0);
    myAccount.displayBalance();
}
```

Example

```
#include <iostream>
#include <string>
using namespace std;
class Employee {
private:
    string name;
    int employeeID;
    double salary;

public:
    // Parameterized constructor
    Employee(string empName, int empID, double empSalary) {
        name = empName;
        employeeID = empID;
        salary = empSalary;
    }

    void displayDetails() {
        cout << "Employee Name: " << name << endl;
        cout << "Employee ID: " << employeeID << endl;
        cout << "Salary: $" << salary << endl;
    }

    void raiseSalary(double raiseAmount) {
        salary += raiseAmount;
        cout << "Salary raised by $" << raiseAmount << " for employee
" << name << endl;
    }
};

int main() {
    // Creating an Employee object with initial details
    Employee emp1("John Doe", 1001, 50000.0);

    // Displaying initial employee details
    emp1.displayDetails();

    // Giving a raise to the employee
    emp1.raiseSalary(5000.0);
    // Displaying updated employee details
    emp1.displayDetails();
}
```

Example

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

class Car {
private:
    string make;
    string model;
    int year;

public:
    // Parameterized constructor
    Car(string carMake, string carModel, int carYear) {
        make = carMake;
        model = carModel;
        year = carYear;
    }

    // Method to display car information
    void displayInfo() {
        cout << "Make: " << make << endl;
        cout << "Model: " << model << endl;
        cout << "Year: " << year << endl;
    }
};

int main() {
    // Creating a Car object using the parameterized constructor
    Car myCar("Toyota", "Corolla", 2022);

    // Displaying car information using the displayInfo() method
    myCar.displayInfo();
}
```

Overloaded Constructor

In C++, an overloaded constructor refers to having multiple constructors within a class with different parameter lists. This allows objects to be initialized in various ways, providing flexibility in object creation and initialization. An example demonstrating overloaded constructors in C++:

```
#include <iostream>
using namespace std;
class Box {
private:
    int length;
    int width;
    int height;

public:
    // Default constructor
    Box() {
        length = 0;
        width = 0;
        height = 0;
    }

    // Constructor with one parameter
    Box(int side) {
        length = side;
        width = side;
        height = side;
    }

    // Constructor with three parameters
    Box(int len, int wid, int hei) {
        length = len;
        width = wid;
        height = hei;
    }

    // Method to calculate volume of the box
    int calculateVolume() {
        return length * width * height;
    }
};
```

An example of how you might use these constructors:

```
main() {
    // Creating objects using different constructors
    Box box1; // Default constructor
    Box box2(5); // Constructor with one parameter
    Box box3(3, 4, 6); // Constructor with three parameters

    // Calculating volumes of the boxes
    cout << "Volume of box1: " << box1.calculateVolume() << endl;
    cout << "Volume of box2: " << box2.calculateVolume() << endl;
    cout << "Volume of box3: " << box3.calculateVolume() << endl;
}
```

Example illustrating the concept of constructor overloading in C++ using a class to represent a simple student:

```
#include <iostream>
#include <string>
using namespace std;
class Student {
private:
    string name;
    int age;
    string grade;
public:
    // Default constructor
    Student() {
        name = "Undefined";
        age = 0;
        grade = "N/A";
    }
    // Constructor with two parameters
    Student(string studentName, int studentAge) {
        name = studentName;
        age = studentAge;
        grade = "N/A"; // Default grade
    }
    // Constructor with three parameters
    Student(string studentName, int studentAge, string studentGrade) {
        name = studentName;
        age = studentAge;
        grade = studentGrade;
    }
```

```
    // Method to display student information
    void displayInfo() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
        cout << "Grade: " << grade << endl;
    }
};
main() {
    // Creating objects using different constructors
    Student student1; // Default constructor
    Student student2("Alice", 20); // Constructor with two parameters
    Student student3("Bob", 22, "A"); // Constructor with three
parameters
    // Displaying student information
    cout << "Student 1 Info:" << endl;
    student1.displayInfo();
    cout << "\nStudent 2 Info:" << endl;
    student2.displayInfo();
    cout << "\nStudent 3 Info:" << endl;
    student3.displayInfo();
}
```

Example

```
#include <iostream>
using namespace std;
class Calculator {
private:
    int operand1;
    int operand2;
public:
    Calculator() {
        operand1 = 0;
        operand2 = 0;
    }
    Calculator(int op1, int op2) {
        operand1 = op1;
        operand2 = op2;
    }
    int add() {
        return operand1 + operand2;
    }
    int multiply() {
        return operand1 * operand2;
    }
};
```

```
int main() {
    Calculator calc1; // Default constructor
    Calculator calc2(5, 3); // Constructor with two parameters
    // Performing addition and multiplication
    cout << "Addition using calc1: " << calc1.add() << endl;
    cout << "Multiplication using calc1: " << calc1.multiply() <<
endl;
    cout << "Addition using calc2: " << calc2.add() << endl;
    cout << "Multiplication using calc2: " << calc2.multiply() <<
endl;
}
```

Copy Constructor:

A copy constructor is used to declare and initialize an object from another object.

Example: The statement

integer L2(11);

Would define the object L2 and at the same time initialize it to the values of L1.

Another form of this statement is: integer L2=11;

The process of initialization through a copy constructor is known as copy initialization. **Example:**

```
#include<iostream>
using namespace std;
class code
{
       int id;
       public:
              code ( ) { } //constructor
              code (int a) { id=a; } //constructor
              code(code &x)
              {
                     id=x.id;
              }
              void display( )
              {
                     cout<<id;
              }
};
```

```
main( )
{
        code A(100);
        code B(A);
        code C=A;
        code D;
        D=A;
        cout<<" \n id of A :"; A.display( );
        cout<<" \nid of B :"; B.display( );
        cout<<" \n id of C:"; C.display( );
        cout<<" \n id of D:"; D.display( );
}
```

A copy constructor in C++ is a special type of constructor that creates a new object as a copy of an existing object of the same class. It is used to initialize a new object with the values of an existing object. The copy constructor takes an object of the same class as a parameter and creates a new object with the same values.

```
#include <iostream>
using namespace std;
class MyClass {
private:
    int data;
public:
    // Constructor
    MyClass(int value) : data(value) {
        cout << "Regular Constructor called" << endl;
    }
    // Copy constructor
    MyClass(const MyClass &other) : data(other.data) {
        cout << "Copy Constructor called" << endl;
    }
    // Method to display data
    void displayData() {
        cout << "Data: " << data << endl;
    }
};
int main() {
    // Creating an object using the regular constructor
    MyClass obj1(10);

    // Creating another object using the copy constructor
    MyClass obj2 = obj1; // Copy constructor invoked here
```

```
    // Displaying data of both objects
    cout << "Object 1: ";
    obj1.displayData();
    cout << "Object 2: ";
    obj2.displayData();
}
```

Dynamic Constructor:

The constructors can also be used to allocate memory while creating objects. This will enable the system to allocate the right amount of memory for each object when the objects are not of the same size, thus resulting in the saving of memory. Allocate of memory to objects at the time of their construction is known as dynamic constructors of objects. The memory is allocated with the help of new operator.

In C++, a constructor is a special member function of a class that gets executed whenever an object of that class is created. A constructor is used to initialize the object's data members or perform any necessary setup.

A dynamic constructor refers to the concept of dynamically allocating memory or performing dynamic initialization within the constructor. However, in C++, constructors themselves aren't explicitly designated as "dynamic" – they can involve dynamic memory allocation or other dynamic operations within their body. An example demonstrating how you might perform dynamic memory allocation within a constructor:

Example

```
#include <iostream>
using namespace std;
class MyClass {
private:
    int* dynamicArray;
    int size;

public:
    // Constructor
    MyClass(int n) {
        size = n;
        dynamicArray = new int[size]; // Allocating memory dynamically
        for (int i = 0; i < size; ++i) {
            dynamicArray[i] = i * 2; // Assigning some values (just an
example)
        }
    }
    // Method to display elements
    void display() {
        for (int i = 0; i < size; ++i) {
            cout << dynamicArray[i] << " ";
        }
        cout << std::endl;
    }
};
main() {
    int size = 5;
    MyClass obj(size); // Creating an object with dynamically
allocated memory
    obj.display(); // Displaying elements
}
```