National University of Computer & Emerging Sciences, Karachi Computer Science Department

Spring 2023, Lab Manual - 04

Course Code: CL-1004	Course : Object Oriented Programming Lab

Contents

1. Types of Classes
2. Constructor
3. Constructor Overloading
4. Initializer List
5. Lab Task

Class:
A class is a user-defined data type. It holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

Types of Classes:

- Concrete Classes
- Generalized classes
- Specialized classes

Concrete Classes:

A concrete class is a class that has an implementation for all of its methods. They cannot have any unimplemented methods.

Example:

```
class Person {
private:
   std::string name;
   int age;

public:
   Person(std::string n, int a) {
      name = n;
      age = a;
   }
```

```
    std::string getName() const {
        return name;
    }

    int getAge() const {
        return age;
    }

    void setName(std::string n) {
        name = n;
    }

    void setAge(int a) {
        age = a;
    }
};
```

Generalized Classes:

A class which tells the main features but not the specific details. The classes situated at the top of the inheritance hierarchy can be said as General.

Example:

"Car" can be considered generalized class.

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

class Car {
    public:
        int price;
        int year;
        string make;
        string model;

        Car(int p, int y, string m, string mo) {
            price = p;
            year = y;
            make = m;
            model = mo;
        }

        void displayInformation() {
            cout << "Price: " << price << endl;
            cout << "Year: " << year << endl;
            cout << "Make: " << make << endl;
            cout << "Model: " << model << endl;
        }
};
```

Specialized Classes:
A class which is very particular and states the specific details. The classes situated at the bottom of the inheritance hierarchy can be said as Specific.

Example:
In the code provided, the class "ToyotaCars" is an example of a specialized class. This class represents a specific type of car, namely Toyota cars, and provides specific information about them, such as the model, year, and color. The class is defined with private variables to store this information, and public methods to display it.

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

class ToyotaCars {
   private:
      string model;
      int year;
      string color;
   public:
      ToyotaCars(string model, int year, string color)
      {
         this->model = model;
         this->year = year;
         this->color = color;
      }
      void displayInfo()
      {
         cout<<"Model: "<<model<<endl;
         cout<<"Year: "<<year<<endl;
         cout<<"Color: "<<color<<endl;
      }
};
```

Introduction to Constructor

- **Constructor** is the special type of member function in C++ classes. It is automatically invoked when an object is being created. It is special because its name is same as the class name.
- **To initialize data member of class:** In the constructor member function (which the programmer will declare), we can initialize the default vales to the data members and they can be used further for processing.
- **To allocate memory for data member:** Constructor is also used to declare run time memory (dynamic memory for the data members).
- Constructor has the same name as the class name. It is case sensitive.
- Constructor does not have return type.
- We can overload constructor; it means we can create more than one constructor of class.
- It must be public type.

Types of Constructors

- **Default Constructors:** Default constructor is the constructor, which does not take any argument. It has no parameters.
- **Null constructors:** Null constructors in C++ are a special type of constructor that does nothing. The compiler knows that there is no code to execute, so it will not generate any executable code for the constructor.
- **Parameterized Constructors:** It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.
- **Copy Constructor:** A copy constructor is a member function, which initializes an object using another object of the same class. The copy constructor in C++ is used to copy data of one object to another.

Default Constructor Example:

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

class construct
{
public:
    int a, b;


    construct()
    {
        a = 10;
        b = 20;
    }
};

int main()
{

    construct c;
    cout << "a: " << c.a << endl
        << "b: " << c.b;
    return 1;
}
```

Parameterized Constructor Example:

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

class Point
{
private:
    int x, y;

public:

    Point(int x1, int y1)
    {
        x = x1;
        y = y1;
    }

    int getX()
    {
        return x;
    }
    int getY()
    {
        return y;
    }
};

int main()
{

    Point p1(10, 15);

    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();

    return 0;
}
```

<u>Copy Constructor Example:</u>

```
#include<iostream>
#include<conio.h>

using namespace std;

class Example {

    int a, b;
public:

    Example(int x, int y) {

        a = x;
        b = y;
        cout << "\nIm Constructor";
    }
Example(const Example& obj) {

        a = obj.a;
        b = obj.b;
        cout << "\nIm Copy Constructor";
    }

    void Display() {
        cout << "\nValues :" << a << "\t" << b;
    }
};
int main() {


    Example Object(10, 20);


    Example Object2(Object);


    Example Object3 = Object;

    Object.Display();
    Object2.Display();
    Object3.Display();
    return 0;
}
```

Constructor Overloading:

In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments. This concept is known as Constructor Overloading. Overloaded constructors have the same name (name of the class) but the different number of arguments. Depending upon the number and type of arguments passed, the corresponding constructor is called

Example:

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

```
class Student
{
    private:
        string name;
        int age;
        string address;
        string department;

    public:
        // Default constructor
        Student()
        {
            name = "";
            age = 0;
            address = "";
            department = "";
        }

        // Overloaded constructor with name and age parameters
        Student(string studentName, int studentAge)
        {
            name = studentName;
            age = studentAge;
            address = "";
            department = "";
        }

        // Overloaded constructor with name, age, and address parameters
        Student(string studentName, int studentAge, string studentAddress)
        {
            name = studentName;
            age = studentAge;
            address = studentAddress;
            department = "";
        }

        // Overloaded constructor with all parameters
        Student(string studentName, int studentAge, string studentAddress, string studentDepartment)
        {
            name = studentName;
            age = studentAge;
            address = studentAddress;
            department = studentDepartment;
        }

        // Accessor functions to access private data members
        string getName()
        {
            return name;
        }
        int getAge()
        {
            return age;
        }
        string getAddress()
        {
            return address;
        }
        string getDepartment()
        {
            return department;
        }
};

int main()
{
    // Creating objects using different constructors
    Student student1;
    Student student2("Ali Hasan", 20);
    Student student3("Junaid Khan", 22, "Gulshan, Khi");
    Student student4("Ayesha Usman", 23, "Saddar, Khi", "Computer Science");

    // Printing details of each student
    cout << "Student 1 Details:" << endl;
    cout << "Name: " << student1.getName() << endl;
    cout << "Age: " << student1.getAge() << endl;
    cout << "Address: " << student1.getAddress() << endl;
    cout << "Department: " << student1.getDepartment() << endl;
    cout << endl;

    cout << "Student 2 Details:" << endl;
    cout << "Name: " << student2.getName() << endl;
    cout << "Age: " << student2.getAge() << endl;
    cout << "Address: " << student2.getAddress() << endl;
    cout << "Department: " << student2.getDepartment() << endl;
    cout << endl;
    cout << "Student 3 Details:" << endl;
    cout << "Name: " << student3.getName() << endl;
    cout << "Age: " << student3.getAge() << endl;
    cout << "Address: " << student3.getAddress() << endl;
    cout << "Department: " << student3.getDepartment() << endl;
    cout << endl;

    cout << "Student 4 Details:" << endl;

    cout << "Name: " << student4.getName() << endl;
    cout << "Age: " << student4.getAge() << endl;
    cout << "Address: " << student4.getAddress() << endl;
    cout << "Department: " << student4.getDepartment() << endl;
    cout << endl;
}
```

Initializer List:

Initializer List is used in initializing the data members of a class. The list of members to be initialized is indicated with constructor as a comma-separated list followed by a colon.

Syntax:

```
Constructorname(datatype value1, datatype value2) : datamember(value1), datamember(value2)
{
    ...
}
```

Example:

```
class Point {
        private:
                int x;
                int y;
        public:
                Point(int i = 0, int j = 0) :x(i), y(j) {}

                int getX() const { return x; }
                int getY() const { return y; }
        };

        int main() {
                Point t1(10, 15);
```

```
                    cout << "x = " << t1.getX() << ", ";
                    cout << "y = " << t1.getY();
return 0;
}
```

Task 1:

Your task is to create a trapezoid constructor that creates a trapezoid with two sides and height provided by arguments. The trapezoid constructed must have two getters getArea() (area=(a+b)h/2) and getPerimeter() with formula (P=a+b+c+d) which give both respective areas and perimeter.

Task 2:

Create an Account class to represent a bank's client bank accounts. Include a data member to reflect the balance of the account. Give each member three functions. The member function credit must be added to the current balance. Member function debit should make a withdrawal from the Account. The get Balance member function should return the current balance.

Task 3:

Create a "Box" class given with private members with float values of width and length and implement the constructors as per the following requirements:
a) A constructor that receives both height and width as parameter to create new Box object only if width and height are both positive values
b) A constructor that receives only height as parameter and takes width as input from the user
c) A constructor that receives no parameter and takes both width and height as user input

Task 4:
Create a class that contains a String variable named language.
1) Create a default constructor that sets the language to “C++”.
2) Create a parameterized constructor that sets this variable.
3) Create a function that contains a print statement that displays the languages.
4) In the main function create 2 objects. One with the default constructor and one with the parameterized constructor with the value “C”.
5) Display the result.

Task 5:

Write a class called Holiday. This class has three instance variables: name, Day & month.

a) Write a constructor for the class Holiday, which takes a String representing the name, an int representing the day, and a String representing the month as its arguments, and sets the class variables to these values.

b) Write a method inSameMonth, which compares two days of the class Holiday, and returns the Boolean value true if they have the same month, and false if they do not.

c) Create a Holiday instance with the name “Independence Day”, with the day “14”, and with the month “August”.

Task 6:

Create A class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance variables - a part number (type String), a part description (type String), a quantity of the item being purchased (type int) and a price per item (double). Your class should have a constructor that initialize the four instance variables. In addition, provide a method named getInvoiceAmount that calculates the invoice amount (i.e., multiples the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0.