CL-1004 Object Oriented Programming

LAB - 04

Working with classes and Constructors, setters and getters

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES Spring 2024

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 Concrete {
private:
   string info;
public:
   Concrete(string s) : info(s) { }
   void printContent() {
      cout << "Concrete Object Information\n" << info << endl;
   }};
```

```
int main()
{
string s;
s = "Object Creation";
   Concrete c(s);
   c.printContent();
}
```

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)
      {
         model = model;
         year = year;
         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;
}
```

Copy Constructor Example:

```
#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;
}
```

Lab Tasks

Question # 01:

Create a class for a Book. A book has a name, author, ISBN number, number of pages, etc. Create an additional variable that stores the number of pages read.

For this task:

- Create default and parameterized constructors for the Book class. Leave the default one empty, but parameterized one should initialize all the values. Values can be hardcoded from the main method.
- Create a method that updates the number of pages read. If a person has read all the pages of the book, it should display "You have finished the book" instead. Call this method from your main to update the number of pages read.

Question # 02:

For the above question, remove the parameterized constructor, and update the default constructor such that it sets the number of pages to 1000, and number of pages read to 0, through a member initialization list.

Generate setters for the remaining three attributes, and update the values of the member variables using those setters in the main function.

Also create a method called "**showBookInfo**" that displays all the information about the variables. Call this function in your main method as well.

Question # 03:

Create a class called **WeekDays** with the following private data members:

- **Days** – String array of size 7
- **CurrentDay** – integer variable

Create the following constructors and member functions for the class:

- Default Constructor – Initializes the **Days** array with values from Sunday to Saturday.
- Parameterized Constructor – Accepts an integer value and assigns it to the **CurrentDay** variable. If the value is greater than 6, perform the mod operation on it and then use it. (For example: 8%7 = 1 —– you'll store 1 in the **CurrentDay** variable). It also initializes the **Days** array with values from Sunday to Saturday.
- getCurrentDay – Returns the value of the current day in string format. (Days[CurrentDay])
- getNextDay – Returns the value of the next day in string format.

- getPreviousDay – Returns the value of the previous day in string format.
- getNthDayFromToday – Returns the value of the day N days from now. (If today is Monday, what day will it be in N=20 days?)

Question # 04:

Create a class called **Office** which holds some information about things inside a real-world office.

- Location – of the office (string) – default value = “”
- SeatingCapacity – of the office (int) – default value = 0
- Furniture – in the office. Make a string array for this. Assume size to be 3. – default value = {“”, “”, “”}

For this task, create a parameterized constructor. Assign some default values to all of the arguments. The constructor body should have nothing in it, and all values should be initialized via member initialization list.

Try creating multiple objects by using:

- Zero arguments
- One argument
- Two arguments
- Three arguments

No need for user input for this question.

Question # 05:

For the above question, remove the furniture array, and instead make it a pointer.

Change the constructor such that it accepts a number instead of an array. This number should be the size of the array. Within your constructor ask dynamically allocate an array of the given size.

Create a destructor for you class, which deallocates the memory allocated for your furniture pointer.

In your main function, dynamically allocate an object of type Office by passing required values in the constructor arguments. Then delete the dynamically allocated object to verify if the destructor is being called properly.