FAST NATIONAL UNIVERSTIY OF COMPUTER AND EMERGING SCIENCES, PESHAWAR

DEPARTMENT OF COMPUTER SCIENCE

CL217 – OBJECT ORIENTED PROGRAMMING LAB

CLASSES AND OBJECTS IN C++

LAB MANUAL # 05

More about Classes and Objects

Instructor: Fariba Laiq

SEMESTER SPRING 2023

Table of Contents

Constructor in C++

- Special method that is implicitly invoked.
- Used to create an object (an instance of the class) and initialize it.
- Every time an object is created, at least one constructor is called.
- It is special member function having same name as class name and is used to initialize object.
- It is invoked/called at the time of object creation.
- It constructs value i.e. provide data for the object that is why it called constructor.
- Can have parameter list or argument list. Can never return any value (no even void).
- Normally declared as public.
- At the time of calling constructor, memory for the object is allocated in the memory.
- It calls a default constructor if there is no constructor available in the class.

Note: It is called constructor because it constructs the values at the time of object creation.

There can be two types of constructors in C++.

1. Default constructor (no-argument constructor)
2. Parameterized constructor.

Default constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- It is also called non-parameterized constructor.
- A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.
- Note: If we have not defined a constructor in our class, then the C++ compiler will automatically create a default constructor with an empty code and no parameters.
- Let's see the simple example of C++ default Constructor.

Example Code:

```
#include<iostream>
using namespace std;
class Rectangle
{
        int length;
        int width;
        public:
        Rectangle() // default constructor
        {
                length=0;
                width=0;
        }
        public:
        int getLength()
        {
                return length;
        }
        int getWidth()
        {
                return width;
        }
};
int main()
{
        Rectangle r;  // default constructor implicitly called.
        cout<<"\nLength: "<<r.getLength();
        cout<<"\nWidth: "<<r.getWidth();
}
```

Output:

```
Length: 0
Width: 0
```

Parameterized Constructor

Just like normal methods, constructor can also accept arguments. We pass arguments to a constructor during the object creation inside the parenthesis ().

Example Code:

```
#include<iostream>
using namespace std;
class Rectangle
{
        int length;
        int width;
        public:
        Rectangle(int l, int w) // Parameterized constructor
        {
                length=l;
                width=w;
        }
        public:
        int getLength()
        {
                return length;
        }
        int getWidth()
        {
                return width;
        }
};
int main()
{
        Rectangle r(2, 3); // Parameterized constructor automatically
called.
        cout<<"\nLength: "<<r.getLength();
        cout<<"\nWidth: "<<r.getWidth();
}
```

Output:

```
Length: 2
Width: 3
```

Constructor Overloading

Similar to methods, constructor can also be overloaded based on the number, type, and order of arguments.

```
#include<iostream>
using namespace std;
class Rectangle
{
        int length;
        int width;
        public:
        Rectangle() // Default constructor
        {
                length=0;
                width=0;
        }
        Rectangle(int l, int w) // Parameterized constructor
        {
                length=l;
                width=w;
        }
        public:
        int getLength()
        {
                return length;
        }
        int getWidth()
        {
                return width;
        }
};
int main()
{
        Rectangle r; // Non-Parameterized constructor automatically called.
        cout<<"\nR1 Length: "<<r.getLength();
        cout<<"\nR1 Width: "<<r.getWidth();
        Rectangle r2(2, 3); // Parameterized constructor called.
        cout<<"\n\mR2 Length: "<<r2.getLength();
        cout<<"\nR2 Width: "<<r2.getWidth();
}
```

Output:

```
Length: 0
Width: 0
Length: 2
Width: 3
```

Destructor:

A destructor is a member function that is invoked automatically when the object goes out of scope or is explicitly destroyed by a call to **`delete`**. A destructor has the same name as the class, preceded by a tilde (~).

If no user-defined destructor exists for a class and one is needed, the compiler implicitly declares a destructor. This implicitly declared destructor is an inline public member of its class.

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

class Test
{
public:
        Test()
        {
                cout<<"\n Constructor executed";
        }

        ~Test()
        {
                cout<<"\n Destructor executed";
        }
};
main()
{
        Test t;

        return 0;
}
```

Const Getter method:

This is a a constant function, meaning a function that will not alter any member variables of the class it belongs to.

```
class Rectangle {
private:
   int width;
   int height;
public:
int getWidth() const {
      return width;
   }
   int getHeight() const {
      return height;
   }
};
```

In main code

```
const Rectangle rect(10, 20);
int width = rect.getWidth();
int height = rect.getHeight();
```

Array of Objects

```
#include <iostream>
using namespace std;
class Student
{
        string name;
        float gpa;
        int marks;
        public:
        string getStudentName()
        {
                return name;
        }
        float getGPA()
        {
                return gpa;
        }
        int getMarks()
        {
                return marks;
        }
        void setStudentName(string name)
        {
                this->name=name;
        }
        void setStudentGPA(float gpa)
        {
                this->gpa=gpa;
        }
        void setStudentMarks(int marks)
        {
                this->marks=marks;
        }
};
void displayAllStudents(Student s[])
{
        for(int i=0; i<3; i++)
        {
                cout<<"name: "<<endl;
                cout<<s[i].getStudentName()<<endl;
                cout<<"gpa: "<<endl;
                cout<<s[i].getGPA()<<endl;;
                cout<<"marks: "<<endl;
                cout<<s[i].getMarks()<<endl;

                cout<<"\n\n";

        }
```

```
int main()
{
        Student s[3];
        int marks;
        string name;
        float gpa;
        for(int i=0; i<3; i++)
        {
                cout<<"Enter name: "<<endl;
                cin>>name;
                s[i].setStudentName(name);
                cout<<"Enter marks: "<<endl;
                cin>>marks;
                s[i].setStudentMarks(marks);
                cout<<"Enter gpa: "<<endl;
                cin>>gpa;
                s[i].setStudentGPA(gpa);
                cout<<"\n\n";
        }
        displayAllStudents(s);

}
```

Separating Header and Implementation Files:

C++ classes (and often function prototypes) are normally split up into two files. The header file has the extension of .h and contains class definitions and functions. The implementation of the class goes into the .cpp file. By doing this, if your class implementation doesn't change then it won't need to be recompiled. Most IDE's will do this for you – they will only recompile the classes that have changed. This is possible when they are split up this way, but it isn't possible if everything is in one file (or if the implementation is all part of the header file).

Make a project a keep all the separated files in it.

Below are the contents of Rectangle.h file

```
class Rectangle
{
  private:
  int length;
  int width;
          public:
  Rectangle();
  Rectangle(int l, int w);
  int getLength();
  int getWidth();
  void setLength(int l);
  void setWidth(int w);
  int calc_Area();
};
```

Below are the contents of Rectangle.cpp file

```
#include "Rectangle.h"
   Rectangle::Rectangle()
  {
                        length=0;
                        width=0;
  }

   Rectangle::Rectangle(int l, int w)
  {
                        length=l;
                        width=w;
  }

 int Rectangle::getLength()
  {
                        return length;
  }

  int Rectangle:: getWidth()
  {
                        return width;
  }

  void Rectangle::setLength(int l)
  {
                        length=l;
  }

  void Rectangle::setWidth(int w)
  {
                        width=w;
  }

int Rectangle::calc_Area()
  {
           int area=length*width;
           return area;
  }
```

Below are the contents of main.cpp file

```
#include<iostream>

#include "Rectangle.h"

using namespace std;

int main () {

            int length, width;

            cout<<"Enter length: "<<endl;

            cin>>length;

            cout<<"Enter width: "<<endl;

            cin>>width;

            Rectangle r;

            r.setLength(length);

            r.setWidth(width);

            cout<<"\nLength: "<<r.getLength();

            cout<<"\nWidth: "<<r.getWidth();

            cout<<"\nArea: "<<r.calc_Area();

             return 0;

}
```

Now to run the project, it can be done in a single click from your IDE. Or you can also use the following step by step commands:

Compile your implementation file(where the function definition is provided)

g++ -c implementation.cpp

Then compile your main.cpp file

g++ -c main.cpp

After this step, two object files will be created. Now we will call the linker to link these two object files into a single executable file.

```
g++ implementation.o main.o -o myprogram
```

An exe file named myprogram will be created. Run it.