Tutorial Link https://codequotient.com/tutorials/class Constructors/5b38c902c6a1d0259e728e9b

TUTORIAL

class Constructors

Chapter

1. class Constructors

Topics

Constructors and destructors are special member functions of a class. These decide how the objects are created, initialized, copied, and destroyed. When an object is created, the constructor is executed and member variables are initialized. However, the programmer can also pass values to the constructor to initialize the member variables with different values. The destructor destroys the object. The destructor is executed at the end of the function when objects are of no use. Both Constructors and Destructors have the same name as the class they belong to. The only difference is that destructor is preceded by ~ (tilde) operator.

Constructors

Constructors are methods that are used to initialize an object at its definition time. A constructor is executed when an object is declared. It

has same name as that of the class it belongs. A simple example of a constructor showing its syntax is as follows:

```
class Number
{
        int m, n;
        public:
                Number (void); // Declaration of Constructor
                ……..
};
```

```
Number : : Number ( void ) // Definition of Constructor
{
        m = 0;
        n = 0;
}
```

As other methods, the constructors can take arguments and be overloaded. These have neither return value nor void. The constructors with arguments are called **parameterized constructors**. The constructor without arguments is called as **default constructor**. When a reference of an object is passed as an argument to the constructor then it is called a **copy constructor**.

Let us take an example of class Circle showing all these types of constructors.

C++

```
1
2 #include<iostream>
3 #include<cstdio>
4 #include<cmath>
```

```
// Include headers as needed

using namespace std;

class Circle
{
  private:
  double radius;
  public:
  Circle ( ) // Default Constructor
  {
    radius=0;
  }
  Circle (double r) // Parameterized Constructor
  {
    radius = r;
  }
  Circle (Circle &c2) // Copy Constructor
  {
    radius = c2.radius;
  }
  double showArea( ) // Member function
  {
    return radius*radius*3.1416;
  }
};

int main( )
{
  Circle c1, c2(3.5);
  Circle c3(c2);
  Circle c4 = c2;

  cout<< "Area = " << c2.showArea( ) <<endl;
  cout<< "Area = " << c3.showArea( ) <<endl;
  cout<< "Area = " << c4.showArea( ) <<endl;
  return 0;
```

```
42  }
43
44
```

OUTPUT:

```
area = 38.4846
area = 38.4846
area = 38.4846
```

In the above program, class Circle is declared with one member variable radius and three constructors. In the function main ( ), the object c1 is created, the default constructor is invoked. When object c2 is created, 3.5 is passed to the constructor and hence constructor with one argument i.e. parameterized constructor is invoked. When object c3 and c4 are created, the object c2 is passed and copy constructor is invoked.

Some special characteristics of constructors are summarized as follows:

- Constructor has the same name as that of the class it belongs to.
- The constructor is invoked automatically when the object is declared.
- Constructors have neither return type nor void.
- They should be declared in the public section of the class.
- Constructors cannot be virtual.
- They cannot be inherited. However, a derived class constructor can call the base class constructor.
- They can have default arguments like other functions.
- The constructors can also be defined as inline functions.
- The constructors can be overloaded.
- The constructor without arguments is called as default constructor.
- A reference of an object can be passed to the constructor called a copy constructor.

Default Constructor

As earlier discussed, a constructor that accepts no arguments is called a default constructor. The following program prints the sum of two integers with the help of the default constructor.

C++

```
#include<iostream>
#include<cstdio>
#include<cmath>
// Include headers as needed

using namespace std;

class IntegerSum
{
  int a, b;
  public:
  IntegerSum(void); // Declaration of Default Constructor
  int display( void);
};

IntegerSum :: IntegerSum( ) // Definition of Default Constructor
{
  a = 5;
  b = 10;
}

int IntegerSum :: display ( )
{
  return a + b;
}

int main( )
{
  IntegerSum Sum1;
```

```
31    cout<<"Sum = "<< Sum1.display();
32    return 0;
33  }
34
```

OUTPUT:

```
Sum = 15
```

In the above program, a default constructor is defined which initializes the object Sum1 by initializes the two integers. The member function display ( ) returns the sum of these two integers.

Parameterized Constructor

In practice, it is required to initialize the various data elements of different objects with different values when they are created. It is achieved by passing arguments to the constructors when the objects are created. The constructors that can accept arguments are called parameterized constructors. In the case of constructor with arguments, we need to pass arguments as per the definition of constructor. They can take any number of arguments. The constructors can take default arguments also. The arguments can be passed to the constructor functions in two ways:

i. By calling the constructor implicitly.

ii. By calling the constructor explicitly.

Let us revise the above program of the sum of two integers by using parameterized constructors.

C++

```
1  #include<iostream>
2  #include<cstdio>
3  #include<cmath>
4  // Include headers as needed
5
```

```
using namespace std;

class IntegerSum
{
  int a, b;
  public:
  IntegerSum( int x, int y=0) // Parameterized Constructor
  {
    a= x;
    b = y;
  }
  int display ( )
  {
    return a + b;
  }
};

int main ( )
{
  IntegerSum Sum1(6); // Constructor called Implicitly
  IntegerSum Sum2 = IntegerSum( 35, 25 ); // Constructor called Explicitly
  cout<<"Sum = "<< Sum1.display( )<<endl;
  cout<<"Sum = "<< Sum2.display( )<<endl;
  return 0;
}

```

OUTPUT:

```
Sum = 6
Sum = 60
```

In the above program, the constructor IntegerSum has two arguments i.e. x and y in which the default value of argument y is zero. It is also called the default argument constructor. The statement

```
IntegerSum Sum1 ( 6 );
```

assigns the value 6 to x variable and 0 to y variable(by default). However, the statement

```
IntegerSum Sum2 = IntegerSum ( 35, 25 );
```

assigns the value 35 to x variable and 25 to y variable. The actual parameter overrides the default value here.

Copy Constructor

A copy constructor is used to initialize an object by using another object of the same class. It is possible to declare and initialize one object using the reference of another object. The argument by value cannot be passed to the copy constructor. Let us consider the example of class IntegerSum again.

C++

```
#include<iostream>
#include<cstdio>
#include<cmath>
// Include headers as needed

using namespace std;

class IntegerSum
{
  int a, b;
  public:
  IntegerSum( int x, int y) // Parameterized Constructor
  {
    a= x;
    b = y;
  }
  IntegerSum( IntegerSum &I) // Copy Constructor
```

```
  {
    a = I.a;
    b = I.b;
  }
  int display ( )
  {
    return a + b;
  }
};

int main ( )
{
  IntegerSum Sum1(10, 20);
  IntegerSum Sum2(Sum1);
  IntegerSum Sum3 = Sum1;
  cout<<"Sum = "<< Sum1.display( )<<endl;
  cout<<"Sum = "<< Sum2.display( )<<endl;
  cout<<"Sum = "<< Sum3.display( )<<endl;
  return 0;
}

```

OUTPUT:

```
Sum = 30
Sum = 30
Sum = 30
```

In the above program, class IntegerSum is declared with two member variables a and b and with two constructors. In the function main ( ), when object Sum1 is created, 10 and 20 are passed to the constructor and hence constructor with two arguments is invoked. When object Sum2 and Sum3 are created, the object Sum1 is passed and copy constructor is invoked. The statements

```
IntegerSum Sum2 (Sum1);
IntegerSum Sum3 = Sum1;
```

have same meanings.

'this' pointer

Every object in C++ has access to its own address through an important pointer called **this pointer**. The this pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

Friend functions do not have a this pointer, because friends are not members of a class. Only member functions have a this pointer.

The 'this' pointer is passed as a hidden argument to all non static member function calls and is available as a local variable within the body of all non static functions.

this pointer can be used in the following situations:

1. **When local variable's name is same as member's name**
2. **To return reference to the calling object**

this pointer can be used like,

C++

```
#include<iostream>
#include<cstdio>
#include<cmath>
using namespace std;
// Using this for accessing class variables
class Point{
    int x,y,z;
    public:
        Point(int x,int y,int z){
            this->x = x;
            this->y = y;
```

```
            this->z = z;
        }
        void setX(int x){
            this->x = x;
        }
        void setY(int y){
            this->y = y;
        }
        void setZ(int z){
            this->z = z;
        }
        int getX(){
            return this->x;
        }
        int getY(){
            return this->y;
        }
        int getZ(){
            return this->z;
        }
};

int main(){
    Point point(10,15,20);
    cout<<point.getX()<<" "<<point.getY()<<" "<<point.getZ()<<endl;
    point.setX(50);
    cout<<point.getX()<<" "<<point.getY()<<" "<<point.getZ()<<endl;
    point.setY(25);
    cout<<point.getX()<<" "<<point.getY()<<" "<<point.getZ()<<endl;
    point.setZ(100);
    cout<<point.getX()<<" "<<point.getY()<<" "<<point.getZ()<<endl;
    return 0;
}
```