Chapter 4:
Object Construction and Destruction

Department of Computer Science and Engineering
Kathmandu University

Rajani Chulyadyo, PhD

Contents

Constructor

- A special kind of class member function that is automatically called when an object of that class is instantiated.
- Typically used to initialize member variables of the class to appropriate default or user-provided values, or to do any setup steps necessary for the class to be used (e.g. open a file or database).
- Unlike normal member functions, constructors have specific rules for how they must be named:
  - Constructors must have the same name as the class (with the same capitalization)
  - Constructors have no return type (not even void)

Constructor

Recall the `Point` class

Constructor (declaration)

```
// Point.h
#pragma once
class Point {
   private:
        double x, y;
   public:
        Point();
        void print() const {
            std::cout << "x = " << x << ", y = " << y << std::endl;
        }
};
```

Constructor

Constructor can be defined in either of the following ways:

```cpp
// Point.cpp
#include "Point.h"
Point::Point() {
    x = 0;
    y = 0;
}
```

```cpp
// Point.cpp
#include "Point.h"
Point::Point() : x(0), y(0)
{
}
```

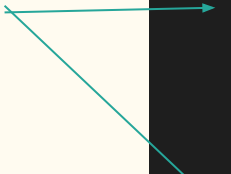
```cpp
// Point.h
#pragma once

class Point {
    private:
        double x, y;
    public:
        Point() {
            // inline defn
            x = y = 0;
        }
};
```

Here the constructor sets x and y to 0.

Constructor

Now, when a Point object is created, its x and y will be initialized to 0.

Constructor will be called.



```
#include <iostream>
#include "Point.h"

int main() {
    Point p1;
    p1.print(); // x = 0, y = 0

    Point p2{};
}
```

Constructors

Types:

- Default constructor
- Parameterized constructor
- Copy constructor

Default constructor

- The one that takes no parameters (or has parameters that all have default values)

Default constructor (no arguments)

```
// Point.h
#pragma once
class Point {
   private:
        double x, y;
   public:
        Point();
        void print() const {
            std::cout << "x = " << x << ", y = " << y << std::endl;
        }
};
```

Default constructor

- The one that takes no parameters (or has parameters that all have default values)

Default constructor (with default arguments)

```
// Point.h
#pragma once
class Point {
   private:
       double x, y;
   public:
       Point(double x = 0, double y = 0);
       void print() const {
           std::cout << "x = " << x << ", y = " << y << std::endl;
       }
};
```

Parameterized constructor

The one that takes parameters (without default values)

Parameterized constructor

```
// Point.h
#pragma once
class Point {
   private:
       double x, y;
   public:
       Point(double x, double y) { this->x = x; this->y = y; }
       void print() const {
           std::cout << "x = " << x << ", y = " << y << std::endl;
       }
};
```

Parameterized constructor

The one that takes parameters

```
// Point.h
#pragma once
class Point {
   private:
        double x, y;
   public:
        Point(double x, double y) : x(x), y(y){ }
        void print() const {
            std::cout << "x = " << x << ", y = " << y << std::endl;
        }
};
```

Can also be defined in this way

Parameterized constructor

The one that takes parameters

```
// Point.h
#pragma once
class Point {
   private:
        double x, y;
   public:
        Point(double x, double y = 0) : x(x), y(y){ }
        void print() const {
            std::cout << "x = " << x << ", y = " << y << std::endl;
        }
};
```

Note that all default parameters must follow any non-default parameters

Here the default value of y is 0.

```
Point p{10, 2};
Point p{10};
```

Implicit constructor

Recall that the `Point` class in Chapter 3 had no constructor.

If a class has no constructors, C++ will **automatically generate a public default constructor** that allows to create an object without the arguments.

This is sometimes called an **implicit constructor** (or implicitly generated constructor).

```
class Point {
   public:
       double x, y;
       // No constructors provided, so C++ creates
       // a public default constructor for us
};
int main() {
   Point p1{};  // Calls implicit constructor
}
```

Implicit constructor

If your class has any other constructors, the implicitly generated constructor will not be provided.

```
class Point {
   public:
       double x, y;
       Point(double x, double y) : x(x), y(y) { }
};
int main() {
   Point p1{10, 20};  // OK
   Point p1{};  // Error: no default constructor exists and the compiler won't generate one
}
```

Implicit constructor

If a class has members of type class, the constructors of those members will be called automatically.

```
#include <iostream>

class Point {
   private:
       double x, y;
   public:
       Point() { std::cout << "Point constructor " << std::endl; }
};
```

```
#include "Point.h"

class Rectangle {
   private:
       Point point1;
       Point point2;
};

int main() {
   Rectangle rect;
}
```

Copy constructor

Recall variable initialization

```
int y(1);  // Direct initialize an integer

int z{1}; // Uniform initialization of an integer (Only since C++11)

int x = 1; // Copy initialize an integer
```

The same applies to objects.

Copy constructor

Initializing a `Point` object

```
Point p1(10, 20);  // Direct initialization


Point p2{0, 20}; // Uniform initialization (Only since C++11)

Point p3 = Point(0, 10); // Copy initialization
Point p3(p2); // Copy initialization
```

During copy initialization, the copy constructor is called.

Copy constructor

A copy constructor is a special type of constructor used **to create a new object as a copy of an existing object.**

```
// Point.h
#pragma once
class Point {
    private:
        double x, y;
    public:
        Point(const Point& other);
        // Point(const Point other);
};
```

Takes a reference to an object of type Point

Copies the contents of `other` to `this` object.

`const` because the constructor does not modify `other`.

This is invalid.

Copy constructor

A copy constructor is a special type of constructor used **to create a new object as a copy of an existing object.**

```
// Point.h
#pragma once
class Point {
   private:
       double x, y;
   public:
       Point(const Point& other);
       // Point(const Point other);
};
```

Copies the contents of `other` to `this` object.

```
// Point.cpp
#include "Point.h"

Point::Point(const Point& other) {
   x = other.x;
   y = other.y;
}
```

`const` because the constructor does not modify `other`.

Copy constructor

If the class does not have a copy constructor, C++ will create a copy constructor, which uses memberwise initialization.

Memberwise initialization simply means that each member of the copy is initialized directly from the member of the class being copied.

Constructor overloading

Multiple constructors can be declared in the same class.

```
class Point {
   private:
       double x, y, z;
   public:
       Point() {  }                                  // Point p{}; will call this
       Point(int x, int y) : x(x), y(y) { }          // Point p{10, 20}; will call this
       Point(double x, double y) : x(x), y(y) { }    // Point p{10., 20.}; will call this
       Point(const Point& other);                    // Point p1(p); will call this
};
Point::Point(int x, int y) : x(x), y(y) { }
Point::Point(double x, double y) : x(x), y(y) { }
Point::Point(const Point& other) : x(other.x), y(other.y) { }
```

Rule of 3

The rule of three is a rule of thumb in C++ (prior to C++11) that claims that if a class defines any of the following then it should probably explicitly define all three:

1. Destructor
2. Copy constructor
3. Copy assignment operator

Destructor

- A special kind of class member function that is executed when an object of that class is destroyed.
- When an object goes out of scope normally, or a dynamically allocated object is explicitly deleted using the delete keyword, the class destructor is automatically called (if it exists) to do any necessary clean up before the object is removed from memory.
- The destructor must have the same name as the class, preceded by a tilde (~).
- The destructor cannot take arguments, and has no return type.

Destructor

- Destructors are designed to help clean up.
  - For simple classes, a destructor is not needed because C++ will automatically clean up the memory for you.
  - If your class object is holding any resources (e.g. dynamic memory, or a file or database handle), or if you need to do any kind of maintenance before the object is destroyed, the destructor is the perfect place to do so, as it is typically the last thing to happen before the object is destroyed.

Destructor

Example

```
// Point.h
#pragma once
class Point {
   public:
       double x, y;
       Point() { }
       Point(double x, double y) :
           x(x), y(y) { }
};
```

```
// Rectangle.h
#pragma once
#include "Point.h"

class Rectangle {
   private:
       Point* point1;
       Point* point2;
   public:
       Rectangle();
       Rectangle(const Point& p1, const Point& p2);
       ~Rectangle();
       void displayAllPoints();
};
```

Destructor

Example

```cpp
// Rectangle.cpp
#include "Rectangle.h"

Rectangle::Rectangle() {
    point1 = nullptr;
    point2 = nullptr;
}

Rectangle::Rectangle(const Point& p1,
                     const Point& p2) {
    point1 = new Point(p1);
    point2 = new Point(p2);
}
```

```cpp
Rectangle::~Rectangle() { // Destructor
    std::cout << "Deleting points" << std::endl;
    delete point1;
    delete point2;
}
void Rectangle::displayAllPoints() {
    std::cout << point1->x << ", " << point1->y << std::endl;
    std::cout << point2->x << ", " << point1->y << std::endl;
    std::cout << point2->x << ", " << point2->y << std::endl;
    std::cout << point1->x << ", " << point2->y << std::endl;
}
```

Destructor

Example

```cpp
// main.cpp

#include <iostream>
#include "Rectangle.h"

int main() {
   Point p1(10, 20);
   Point p2(3, 25);

   Rectangle rect(p1, p2);
   rect.displayAllPoints();
   return 0;
}
```

Try it

```
g++ -std=c++11 main.cpp Rectangle.cpp
```

Lab 3 exercise

Question 1:

Define a class, Vector, which represents either a column vector or a row vector. A column (row) vector is a matrix consisting of a single column (row) of m elements.

Let m = 3 in this program. (Use three member variables instead of an array.)

Use appropriate constructors and destructor.

Implement the following method:

```
Vector add(Vector)   // Adds the input vector with the calling vector
```

Resources

1. https://en.cppreference.com/w/cpp/
2. http://cplusplus.com/doc/tutorial/
3. https://www.learncpp.com/
4. https://www.edureka.co/blog/namespace-in-cpp/

Assignment

Assignment # 2

1. Which drawbacks of structured programming are addressed by object-oriented programming?
2. What is data encapsulation?
3. What do you understand by access specifiers? Explain, with examples, different types of access specifiers in C++.
4. Explain the `this` pointer.
5. Differentiate between a constructor and a destructor.
6. Explain different types of constructors.