Computer Programming

Dr. Deepak B Phatak
Dr. Supratik Chakraborty
Department of Computer Science and Engineering
IIT Bombay

Session: Default and Copy Constructors

Quick Recap of Relevant Topics

- Object-oriented programming with structures and classes
- Accessing members and controlling access to members
- Constructor and destructor functions
- Closer look at constructors
  - Explicit invocation
  - Default parameters
  - Initialization lists

Overview of This Lecture

IIT Bombay

- Continuing study of constructors
  - Default constructors
  - Copy constructors

Acknowledgment

- Much of this lecture is motivated by the treatment in

  An Introduction to Programming Through C++

  by Abhiram G. Ranade

  McGraw Hill Education 2014

- Examples taken from this book are indicated in slides by the citation **AGRBook**

Recap: Constructor and Destructor Functions

IIT Bombay

- **Constructor:** Invoked **automatically** when an object of the class is allocated
  - Object is allocated first, then constructor is invoked on object
  - Convenient way to initialize data members
- **Destructor:** Invoked **automatically** when an object of the class is de-allocated
  - Destructor is invoked on object first, then object is de-allocated
  - Convenient way to do book-keeping/cleaning-up before de-allocating object

Default Constructor

- A constructor that doesn’t take any arguments is called a “default constructor”

```
class V3 {
   private:  double x, y, z;
   public:
     V3(double vx, double vy, double vz) {
           x = vx; y = vy; z = vz; return;
     }
     V3() {x = y = z = 0.0; return;}
    … Destructor and other member functions …
};
```

Non-default constructor of V3

Default constructor of V3

Arrays and Default Constructors

Suppose we want to define an array of V3 objects

```
V3 myArray[100];
```

- 100 objects of class V3 must be allocated
- **Which V3 constructor should be invoked on each of them?**

  Default constructor (one without any arguments)

- **What if we had not defined a default constructor for V3?**

  Could be by oversight or even by design

Arrays and Default Constructors

- If no constructor is defined for a class, C++ compiler will provide a bare-bones default constructor
  - No parameters and does nothing in its body
  - Allows array of objects to be defined
  - Similar default destructor also provided by C++ compiler
- **If a non-default constructor is defined, but not a default constructor, C++ compiler will NOT provide a bare-bones default constructor**
  - **Arrays of such objects cannot be defined !!!**

Best practice: Define default constructors

Copy Constructor

- Suppose a new object is created by making a copy of another object of the same class

```
V3 myFunc(V3 a) {
          V3 v;
           v = a.scale(2.0);
           return v;
 }
  int main() {
       V3 a(0.0, 1.0, 2.0);
       V3 a1 = a, a2;
        a2 = myFunc(a);  retu
}
```

Case 1: Initalization in declaration

Case 2: Parameter passing by value

Case 3: Function returning object (May be optimized away by compiler)

Copy Constructor

- Regular assignment statements **do not need** copy constructor since they do not create a new object

```
V3 myFunc(V3 a) {
            V3 v;
             v = a.scale(2.0);
             return v;
 }
  int main() {
        V3 a(0.0, 1.0, 2.0);
        V3 a1 = a;
         a1 = myFunc(a);  return 0;
}
```

Regular assignment: No need for copy constructor

Copy Constructor

- A copy constructor must be specified separately from an ordinary constructor

```
class V3 {
   private:  double x, y, z;
   public:
      V3(double vx, double vy, double vz) {
             x = vx; y = vy; z = vz; return;
      }
      V3() {x = y = z = 0.0; return;}
      V3(const V3 &src) {x = src.x; y = src.y; z = src.z; }
     … Destructor and other member functions …
};
```

Ordinary constructors

Copy Constructor

- A copy constructor must be specified separately from an ordinary constructor

```
class V3 {
   private:  double x, y, z;
   public:
     V3(double vx, double vy, double vz) {
            x = vx; y = vy; z = vz; return;
     }
     V3() {x = y = z = 0.0; return;}
     V3(const V3 &src) {x = src.x; y = src.y; z = src.z; }
    … Destructor and other member functions …
};
```

(Uninteresting)
Copy constructor

Note difference
in parameter
passing

Default Copy Constructor

- If you need a copy constructor in your program, but have not defined it yourself, the C++ compiler will create a default copy constructor
  - Copies values of all data members of source object to corresponding members of receiver object
  - Same as usual assignment
- Sometimes default copy constructors are not good enough
  - More interesting user-defined copy constructors needed

Another Copy Constructor [Ref AGRBook]

```
class myString {
  public:
     char *cArray;  int length;
     myString(const char initString[]) { … }   //  ordinary constructor
     ~myString() {delete [] cArray; return;}
     myString(const myString &source) : length(source.length) { // copy constructor
        cArray = new char[length+1];
        if (cArray == NULL) { … Handle error appropriately … }
        else { for (int i = 0; i <= length; i++) { cArray[i] = (source.cArray)[i]; } return; }
     }
     … Other member functions …
};
```

Summary

- Default constructors
  - Importance in defining arrays
- Copy constructors
  - Importance in creating a new object by copying an existing object

An Interesting Copy Constructor [Ref AGRBook]

```
class  Queue{
     private: int front, nWaiting, elements[100];
     public:
        Queue() {front = nWaiting = 0; }     // ordinary constructor
        Queue (const Queue &source) :     // copy constructor
             front(source.front), nWaiting(source.nWaiting)  {
             for (int i = front, j = 0; j < nWaiting; j++) {
                elements[i] = source.elements[i];
                 i = (i + 1) % 100;
             }
        }
     ... Other member functions ...
};
```