

# C/C++ Program Design

LAB 10

# CONTENTS

- Learn how to define and implement a **class**
- Learn how to create and use class **objects**
- Class **constructors** and **destructors**
- Master the difference between **private** and **public**
- Learn how to use **this** pointer
- Learn to use an **array of objects**

## 2 Knowledge Points

2.1 **Class** and its definition

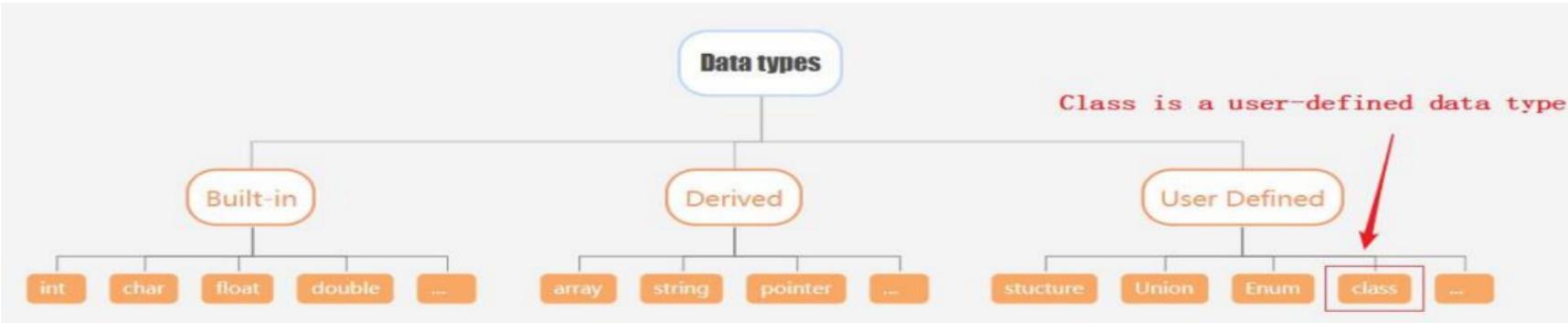
2.2 Access specifier: **private** and **public**

2.3 Class **constructors** and **destructors**

2.4 ***this*** pointer

2.5 An Array of Objects

# 2.1 Class



The general syntax for the class definition:

```
keyword      class name  
class ClassName private/public/protected  
{  
    Access specifier:  
    Data members;  
    Member Functions();  
};
```

member variable declaration  
member function declaration  
end of class with a semicolon

The syntax to declare object:

```
ClassName objectName;
```

A class is a blueprint for the object!

# Example: Define a class named Box and get the volume.

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

Define a class named Box

```
class Box
{
public:
    double length;
    double breadth;
    double height;

public:
    // Member functions declaration
    double getVolume()
    {
        return length * breadth * height;
    }
};
```

The member variables of a class

The member function of a class. It is an inline function.

```
int main()
{
    Box box;
    box.length = 5.0;
    box.breadth = 6.0;
    box.height = 7.0;

    cout << "The volume is: " << box.getVolume() << endl;

    return 0;
}
```

Create an object of Box

Accessing data member by . operator

Accessing member function by . operator

Defining a class, creating an object and accessing the data member of an object, these operations are as the same as we use in structure. The only difference is that we call class **instance** as object. You must use an object to access the attributes(data member) and member function of a class if their access specifier are **public**.

The definition of a class is in a .h file

```
#pragma once

class Box
{
private:
    double length;      //Length of a box
    double breadth;     // Breadth of a box
    double height;      // Height of a box

public:
    // Default Constructor
    Box();
    // Parameterized Constructor
    Box(double, double, double);

    // Member functions declaration
    double getVolume(void);
    void setLength(double len);
    void setBreadth(double bre);
    void setHeight(double hei);
};
```

The implementation of functions is in a .cpp file

Implementation of functions using the **scope resolution operator(::)**

```
Box::Box() {
    length = 3;
    breadth = 4;
    height = 5;
}

Box::Box(int length, int breadth, int height) {
    this->length = length;
    this->breadth = breadth;
    this->height = height;
}

Box::Box(const Box& b) {
    length = b.length;
    breadth = b.breadth;
    height = b.height;
}

Box::~Box() {
    std::cout << "Object is being deleted." << std::endl;
}

// Member functions definitions
double Box::getVolume() {
    return length * breadth * height;
}

void Box::setLength(double len) {
    length = len;
}

void Box::setBreadth(double bre) {
    breadth = bre;
}

void Box::setHeight(double hei) {
    height = hei;
}
```

```
#include <iostream>
#include "Box.h"
using namespace std;

int main()
{
    Box box1;          // Declare box1 of type Box
    Box box2(5.0, 6.0, 9.0); // Declare box2 of type Box

    Box box3;
    double volume = 0.0;      // Store the volume of a box

    box3.setLength(1.0);
    box3.setBreadth(2.0);
    box3.setHeight(3.0);

    // volume of box1
    volume = box1.getVolume();
    cout << "Volume of box1: " << volume << endl;

    // volume of box2
    volume = box2.getVolume();
    cout << "Volume of box2: " << volume << endl;

    // volume of box3
    volume = box3.getVolume();
    cout << "Volume of box3: " << volume << endl;

    return 0;
}
```

Accessing the members is in another .cpp file

Call member function of the  
object by . operator

Output:

```
Volume of box1: 60
Volume of box2: 270
Volume of box3: 6
```

将私有成员放在 class 中 private, structure 在 public.

## 2.2 Access specifier: **private** and **public**

The **private** keyword makes members private. Private members **can be accessed only inside the same class**. We usually make **data private** to prevent them from being modified outside the class. This is known as **data encapsulation (data hiding)**.

The **public** keyword makes members public. Public members **can be accessed out of the class**. We usually make **functions public** for accessing outside the class.

```
#pragma once

class Box
{
private:
    double length;           //Length of a box
    double breadth;          // Breadth of a box
    double height;           // Height of a box

public:
    // Default Constructor
    Box();
    // Parameterized Constructor
    Box(double, double, double);

    // Member functions declaration
    double getVolume(void);
    void setLength(double len);
    void setBreadth(double bre);
    void setHeight(double hei);
};
```

private members

public members

```
int main()
{
    Box box1;           // Declare box1 of type Box
    Box box2(5.0, 6.0, 9.0); // Declare box2 of type Box

    Box box3;
    double volume = 0.0; // Store the volume of a box

    // box3.setLength(1.0);
    // box3.setBreadth(2.0);
    // box3.setHeight(3.0);

    box3.length = 1.0;
    box3.breadth = 2.0;
    box3.height = 3.0;
}
```

Private members cannot be  
accessed outside the class

Note: **Private is the default access specifier** for a class in C++. This means that if no access specifier is specified for the members in a class, then it is considered private.

A **private** data field cannot be accessed by an object from outside the class. To make a private data field accessible, provide a **getter** method to return its value. To enable a private data field to be updated, provide a **setter** method to set a new value.

```
#pragma once

class Box
{
private:
    double length;      //Length of a box
    double breadth;     // Breadth of a box
    double height;      // Height of a box

public:
    // Default Constructor
    Box();
    // Parameterized Constructor
    Box(double, double, double);

    // Member functions declaration
    double getVolume(void);
    void setLength(double len);
    void setBreadth(double bre);
    void setHeight(double hei);
    double getLength();
    double getBreadth();
    double getHeight();
};

};
```

```
void Box::setLength(double len) {
    length = len;
}

void Box::setBreadth(double bre) {
    breadth = bre;
}

void Box::setHeight(double hei) {
    height = hei;
}

double Box::getLength() {
    return this->length;
}

double Box::getBreadth() {
    return this->breadth;
}

double Box::getHeight() {
    return this->height;
}
```

## 2.3 Class Constructors and Destructors

A class constructor is a special member function:

1. Has exact same name as the class
2. No return value
3. It is a public member function of the class
4. Invoked whenever you create objects of that class

```
#pragma once

class Box
{
private:
    double length;      //Length of a box
    double breadth;     // Breadth of a box
    double height;      // Height of a box

public:
    // Default Constructor
    Box();
    // Parameterized Constructor
    Box(double, double, double);

    // Member functions declaration
    double getVolume(void);
    void setLength(double len);
    void setBreadth(double bre);
    void setHeight(double hei);
};
```

```
Box::Box() {
    length = 3.0;
    breadth = 4.0;
    height = 5.0;
}

Box::Box(double length, double breadth, double height) {
    this->length = length;
    this->breadth = breadth;
    this->height = height;
}
```

“this” is a pointer points to the object itself

Constructors can be very useful for setting initial values for certain member variables.

If you do not provide a constructor, C++ compiler **generates** a **default constructor** (has no parameters and an empty body) for you.

```
#include <iostream>
#include "Box.h"
using namespace std;

int main()
{
    Box box1;           // Declare box1 of type Box
    Box box2(5.0, 6.0, 9.0); // Declare box2 of type Box
                           // Invoke parameterized constructor
    Box box3; ← Invoke default constructor
    double volume = 0.0; // Store the volume of a box

    box3.setLength(1.0); ← Update the data field by setter
    box3.setBreadth(2.0);
    box3.setHeight(3.0);

    // volume of box1
    volume = box1.getVolume();
    cout << "Volume of box1: " << volume << endl;

    // volume of box2
    volume = box2.getVolume();
    cout << "Volume of box2: " << volume << endl;

    // volume of box3
    volume = box3.getVolume();
    cout << "Volume of box3: " << volume << endl;

    return 0;
}
```

Output:

```
Volume of box1: 60
Volume of box2: 270
Volume of box3: 6
```

# Run the program step by step

```
4
5
6 int main()
7 {
8     Box box1;          // Declare box1 of type Box
9     Box box2(5.0, 6.0, 9.0); // Declare box2 of type Box
10
11    Box box3;
12    double volume = 0.0;      // Store the volume of a box
13
14    box3.setLength(1.0);
15    box3.setBreadth(2.0);
16    ▶| box3.setHeight(3.0);
17
18    // volume of box1
19 }
```

Click the “step into” button(or F11)

Name	Type
box1	Box
box2	Box
box3	Box
volume	double

From beginning, the  
object is not created.

The screenshot shows the Microsoft Visual Studio IDE interface. The menu bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search. The toolbar has icons for file operations like Open, Save, and Build. The status bar indicates the process is [17296] BoxWithDestructor.exe, Lifecycle Events, and Thread: [20304] Main Thread.

The code editor window displays the file Box2.cpp. The code is as follows:

```
Box2.cpp
Box.h
Box.cpp

BoxWithDestructor
28     cout << "Volume of box3: " << volume << endl;
29
30
31     return 0;
32 }
33
34
35 Box::Box() { // 1ms elapsed
36     length = 3.0;
37     breadth = 4.0;
38     height = 5.0;
39 }
40 Box::Box(double length, double breadth, double height) {
41     this->length = length;
42     this->breadth = breadth;
```

The constructor call at line 35 is highlighted with a red box. A blue callout bubble points to this line with the text: "The default constructor will be executed."

The Locals window at the bottom shows the variable 'this' with a value of 0x004cf028 and type Box \*.

The default constructor  
will be executed.

The screenshot shows the CLion IDE interface. The top bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search field. A red box highlights the "step into" button (F11) in the toolbar. The status bar indicates the process is [17296] BoxWithDestructor.exe, Lifecycle Events, and Thread: [20304] Main Thread.

Code Editor:

```
Box2.cpp  X Box2.h    Box.h    Box.cpp
BoxWithDestructor
28     cout << "Volume of box3: " << volume << endl;
29
30
31     return 0;
32 }
33
34
35     Box::Box() {
36         length = 3.0;
37         breadth = 4.0;
38         height = 5.0;
39     }
40     Box::Box(double length, double breadth, double height) {
41         this->length = length;
42         this->breadth = breadth;
```

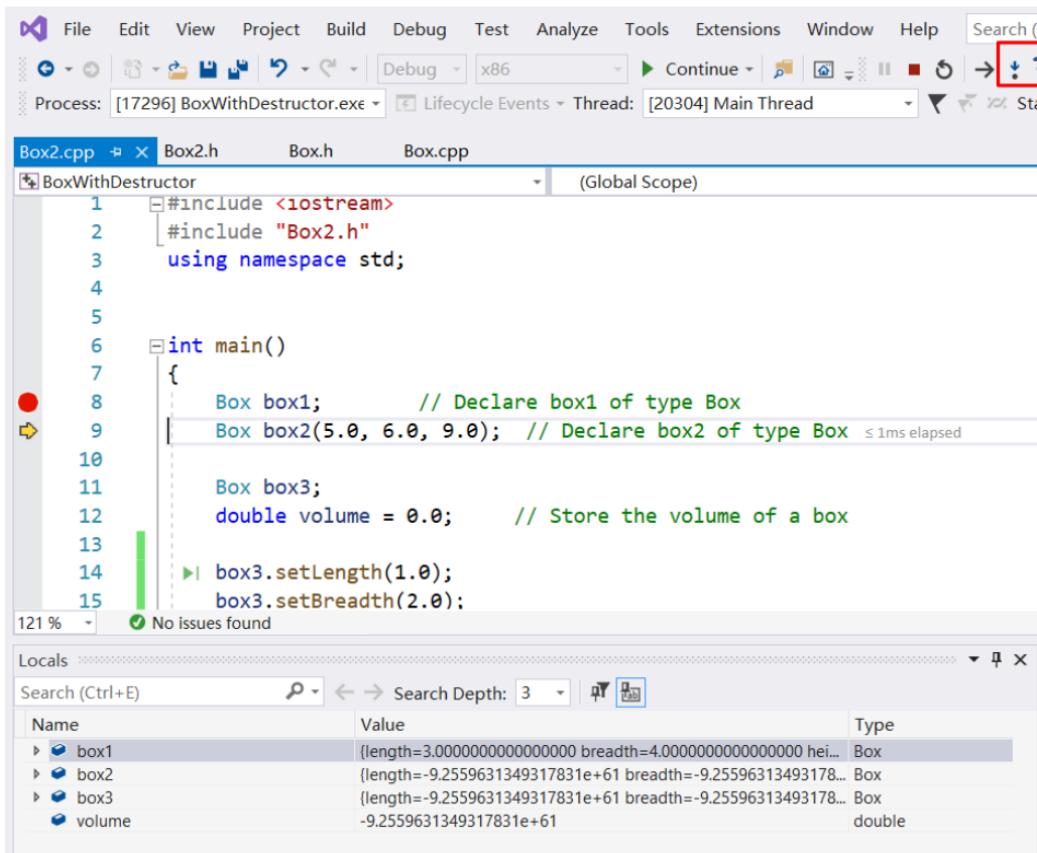
Locals Tool Window:

Name	Type
this	Box
length	double
breadth	double
height	double

The Locals window shows the variable 'this' pointing to the memory location 0x00efffb4. The data members 'length', 'breadth', and 'height' are initialized to 3.000000000000000, 4.000000000000000, and 5.000000000000000 respectively.

Click the “step into” button(or F11)

After the default constructor is invoked, the data members are initialized with the default values.



Click the “step into” button(or F11)

The constructor with arguments will be executed.

```
Box2.cpp  Box2.h  Box.h  Box.cpp
BoxWithDestructor  Box
```

```
33
34
35     Box::Box() {
36         length = 3.0;
37         breadth = 4.0;
38         height = 5.0;
39     }
40     Box::Box(double length, double breadth, double height) { // simple
41         this->length = length;
42         this->breadth = breadth;
43         this->height = height;
44     }
45
46     Box::~Box() {
47         std::cout << "Object is being deleted." << std::endl;
}
```

Locals

Name	Type
this	Box *
breadth	double
height	double
length	double

The screenshot shows a debugger interface with the following details:

- File Menu:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search.
- Process:** [17296] BoxWithDestructor.exe.
- Thread:** [20304] Main Thread.
- Code Editor:** Box2.cpp (selected), Box2.h, Box.h, Box.cpp. The code defines a Box class with three constructors and a destructor. The destructor prints a message to the console.
- Locals Window:** Shows the current state of variables after the constructor has been invoked. The variable 'this' is highlighted with a red border and has a tooltip showing its value: 0x0effba4 (length=5.000000000000000 breadth=6.000000000000000 height=9.000000000000000 length=5.000000000000000).

After the constructor is invoked,  
the data members are initialized  
with the assinged values.

After created three objects, the data field have been assigned the proper values by constructors

```
Box2.cpp  Box2.h  Box.h  Box.cpp
BoxWithDestructor
4
5
6 int main()
{
7
8     Box box1;          // Declare box1 of type Box
9     Box box2(5.0, 6.0, 9.0); // Declare box2 of type Box
10
11    Box box3;
12    double volume = 0.0;    // Store the volume of a box  1ms elapsed
13
14    box3.setLength(1.0);
15    box3.setBreadth(2.0);
16    box3.setHeight(3.0);
17
18 // volume of box1
121 % - No issues found
```

Name	Type	Value
box1	Box	(length=3.0000000000000000 breadth=4.0000000000000000 height=5.0000000000000000)
length	double	3.0000000000000000
breadth	double	4.0000000000000000
height	double	5.0000000000000000
box2	Box	(length=5.0000000000000000 breadth=6.0000000000000000 height=9.0000000000000000)
length	double	5.0000000000000000
breadth	double	6.0000000000000000
height	double	9.0000000000000000
box3	Box	(length=3.0000000000000000 breadth=4.0000000000000000 height=5.0000000000000000)
length	double	3.0000000000000000
breadth	double	4.0000000000000000
height	double	5.0000000000000000
volume	double	-9.2559631349317831e+61

## Default constructor

A default constructor is a constructor that is used to create an object when you don't provide explicit initialization values. If you don't provide any constructors, C++ will automatically supply a default constructor. It's an implicit version of a default constructor, and it does nothing. There is only one default constructor in a class.

If you define any constructor for a class, the compiler will not provide default constructor.

**You must define your own default constructor that takes no arguments.**

You can define a default constructor **two ways**. **One** is to provide default values for all the arguments to the existing constructor:

声明 `Stock(const string & co = "Error", int n = 0; double pr = 0.0);`

定义 `Stock (const ..., int ..., double ...)`

The **second** is to use function overloading to define a second constructor that has no arguments: 且其一

`Stock();`

```
#ifndef STOCK10_H_
#define STOCK10_H_

#include <string>

class Stock
{
private:
    std::string company;
    long shares;
    double share_val;
    double total_val;
    void set_tot() { total_val = shares * share_val; }

public:
    //two constructors
    Stock();      //default constructor
    Stock(const std::string & co, long n = 0, double pr = 0.0);

    ~Stock();     //noisy destructor

    void buy(long num, double price);
    void sell(long num, double price);
    void update(double price);
    void show();
};

#endif
```

```
Stock::Stock() // default constructor
{
    std::cout << "Default constructor called\n";
    company = "no name";
    shares = 0;
    share_val = 0.0;
    total_val = 0.0;
}

Stock::Stock(const std::string & co, long n, double pr)
{
    std::cout << "Constructor using " << co << " called\n";
    company = co;

    if(n < 0)
    {
        std::cout << "Number of shares can't be negative;" 
                  << company << " shares set to 0.\n";
        shares = 0;
    }
    else
        shares = n;

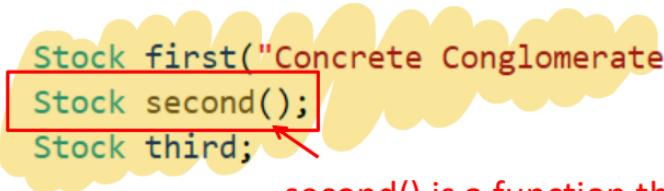
    share_val = pr;
    set_tot();
}
```

## Using constructor

After you define default constructor, you can declare object variables without initializing them explicitly.

```
Stock first;           // call the default constructor implicitly
Stock second = Stock(); // call it explicitly
Stock* prelief = new Stock; // call it implicitly
```

You shouldn't be misled by implicit form of the nondefault constructor.



```
Stock first("Concrete Conglomerate"); // call the nondefault constructor
Stock second();                      // declare a function
Stock third;                         // call the default constructor
```

second() is a function that returns a Stock object.

When you implicitly call the default constructor, do not use parentheses.

# Destructors

```
#pragma once

class Box
{
private:
    double length;      //Length of a box
    double breadth;     // Breadth of a box
    double height;      // Height of a box

public:
    // Default Constructor
    Box();
    // Parameterized Constructor
    Box(double, double, double);

    // Destructor
    ~Box();

    // Member functions declaration
    double getVolume(void);
    void setLength(double len);
    void setBreadth(double bre);
    void setHeight(double hei);
    double getLength();
    double getBreadth();
    double getHeight();
};
```

A class destructor is also a special member function:

1. A destructor name is the same as the classname but begins with tilde(~) sign.
2. Destructor has no return value.
3. A destructor has no arguments.
4. There can be only one destructor in a class.
5. The compiler always creates a default destructor if you fail to provide one for a class.
6. Invoke when an object goes out of scope or the delete is applied to a pointer to the object.

```
Box::~Box() {
    std::cout << "Object is being deleted." << std::endl;
}
```

Destructor can be very useful for **releasing resource** before coming out of the program like closing files, releasing memories etc.

```
#pragma once
```

```
class Box
{
private:
    double length;      //Length of a box
    double breadth;     // Breadth of a box
    double height;      // Height of a box

public:
    // Default Constructor
    Box();
    // Parameterized Constructor
    Box(double, double, double);

    // Destructor
    ~Box();

    // Member functions declaration
    double getVolume(void);
    void setLength(double len);
    void setBreadth(double bre);
    void setHeight(double hei);
    double getLength();
    double getBreadth();
    double getHeight();
};
```

```
Box::Box() {
    length = 3.0;
    breadth = 4.0;
    height = 5.0;
}

Box::Box(double length, double breadth, double height) {
    this->length = length;
    this->breadth = breadth;
    this->height = height;
}

Box::~Box() {
    std::cout << "Object is being deleted." << std::endl;
}

// Member functions definitions
double Box::getVolume() {
    return length * breadth * height;
}

void Box::setLength(double len) {
    length = len;
}

void Box::setBreadth(double bre) {
    breadth = bre;
}

void Box::setHeight(double hei) {
    height = hei;
}
```

## Creating three objects

```
int main()
{
    Box box1;          // Declare box1 of type Box
    Box box2(5.0, 6.0, 9.0); // Declare box2 of type Box

    Box box3;
    double volume = 0.0; // Store the volume of a box

    box3.setLength(1.0);
    box3.setBreadth(2.0);
    box3.setHeight(3.0);

    // volume of box1
    volume = box1.getVolume();
    cout << "Volume of box1: " << volume << endl;

    // volume of box2
    volume = box2.getVolume();
    cout << "Volume of box2: " << volume << endl;

    // volume of box3
    volume = box3.getVolume();
    cout << "Volume of box3: " << volume << endl;

    return 0;
}
```

```
Volume of box1: 60
Volume of box2: 270
Volume of box3: 6
Object is being deleted.
Object is being deleted.
Object is being deleted.
```

The screenshot shows the Microsoft Visual Studio IDE interface during a debugging session. The top menu bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search. The toolbar contains various icons for file operations and debugging. A status bar at the bottom indicates the process is [17296] BoxWithDestructor.exe and the thread is [20304] Main Thread.

The code editor window displays a C++ file named Box2.cpp. The cursor is positioned at line 31, which contains the statement `return 0;`. A callout bubble with a red border points to the "step into" button in the toolbar, with the text "Click the ‘step into’ button(or F11)".

The Locals window shows the current state of variables:

Name	Type
<code>std::operator&lt;&lt; std::char_traits&lt;char&gt;(...)</code>	<code>std::basic_ostream&lt;...&gt;</code>
<code>box1</code>	<code>Box</code>
<code>box2</code>	<code>Box</code>
<code>box3</code>	<code>Box</code>
<code>volume</code>	<code>double</code>

A second callout bubble points to the line `return 0;` in the code editor, with the text "When the last statement will be executed, click ‘step into’ button".

The status bar at the bottom shows "121 % No issues found".

The destructor will be executed.

```
37     breadth = 4.0;
38     height = 5.0;
39 }
40 Box::Box(double length, double breadth, double height) {
41     this->length = length;
42     this->breadth = breadth;
43     this->height = height;
44 }
45
46 Box::~Box() { // 1ms elapsed
47     std::cout << "Object is being deleted." << std::endl;
48 }
49 // Member functions definitions
50 Box::double getVolume() {
51     return length * breadth * height;
52 }
53
54 void Box::setLength(double len) {
55     length = len;
```

```
D:\csourcecode\2020FallC\lab09Class\BoxClass\Debug\Box...
Volume of box1: 60
Volume of box2: 270
Volume of box3: 6
Object is being deleted.
```

The destructor will be executed for three times, because three objects are created.

The address indicates which object will be destroyed.

```
File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search
Process: [17296] BoxWithDestructor.exe Lifecycle Events Thread: [20304] Main Thread
Box2.cpp Box2.h Box.h Box.cpp
BoxWithDestructor
37     breadth = 4.0;
38     height = 5.0;
39 }
40 Box::Box(double length, double breadth, double height) {
41     this->length = length;
42     this->breadth = breadth;
43     this->height = height;
44 }
45
46 Box::~Box() {
47     std::cout << "Object is being deleted." << std::endl;
48 }
49 // Member functions definitions
50 double Box::getVolume() {
51     return length * breadth * height;
52 }
53
54 void Box::setLength(double len) {
55     length = len;
}
121% No issues found
```

Locals

Name	Type
this	0x00effba4 (length=5.000000000000000 breadth=6.000000000000000 Box*)

A screenshot of Microsoft Visual Studio showing a C++ project named "BoxWithDestructor".

The code in Box2.cpp includes:

```
22 // volume of box2
23 volume = box2.getVolume();
24 cout << "Volume of box2: " << volume << endl;
25
26 // volume of box3
27 volume = box3.getVolume();
28 cout << "Volume of box3: " << volume << endl;
29
30
31 return 0; < 1ms elapsed
32 }
```

The constructor definitions are:

```
35 Box::Box() {
36     length = 3.0;
37     breadth = 4.0;
38     height = 5.0;
39 }
40 Box::Box(double length, double breadth, double height) {
```

The Locals window shows:

Name	Type
box1	Box
box2	Box
box3	Box
volume	double

The Output window shows the console output:

```
Volume of box1: 60
Volume of box2: 270
Volume of box3: 6
Object is being deleted.
Object is being deleted.
Object is being deleted.
```

A screenshot of a terminal window showing the execution results of the program.

```
D:\csourcecode\2020FallC\lab09Class\... Volume of box1: 60 Volume of box2: 270 Volume of box3: 6 Object is being deleted. Object is being deleted. Object is being deleted.
```

```
Stock::Stock() // default constructor
{
    std::cout << "Default constructor called\n";
    company = "no name";
    shares = 0;
    share_val = 0.0;
    total_val = 0.0;
}

Stock::Stock(const std::string & co, long n, double pr)
{
    std::cout << "Constructor using " << co << " called\n";
    company = co;
    if(n < 0)
    {
        std::cout << "Number of shares can't be negative;" 
                  << company << " shares set to 0.\n";
        shares = 0;
    }
    else
        shares = n;

    share_val = pr;
    set_tot();
}

Stock::~Stock() //verbose class destructor
{
    std::cout << "Bye, " << company << "!\n";
}
```

```

int main()
{
    using std::cout;
    cout << "Using constructors to create new objects\n";
    Stock stock1("NanoSmart", 12, 20.0); //syntax 1
    stock1.show();
    cout << '\n';

    Stock stock2 = Stock("Boffo Objects", 2, 2.0); //syntax 2
    stock2.show();
    cout << '\n';

    cout << "Assigning stock1 to stock2:\n";
    stock2 = stock1; //array间不可用“=”操作
    cout << "Listing stock1 and stock2:\n";
    cout << "stock1:";
    stock1.show();
    cout << '\n';
    cout << "stock2:";
    stock2.show();
    cout << '\n';

    cout << "Using a constructor to reset an object\n";
    stock1 = Stock("Nifty Foods", 10, 50.0); // temp object
    cout << "Revised stock1:\n";
    stock1.show();
    cout << "Done\n";
    return 0;
}

```

**重新赋值 地址未变**

Using constructors to create new objects  
 Constructor using NanoSmart called  
 Company: NanoSmart Shares: 12  
 Share Price: \$20.00 Total Worth: \$240.00

Constructor using Boffo Objects called  
 Company: Boffo Objects Shares: 2  
 Share Price: \$2.00 Total Worth: \$4.00

Assigning stock1 to stock2:  
 Listing stock1 and stock2:  
 stock1:Company: NanoSmart Shares: 12  
 Share Price: \$20.00 Total Worth: \$240.00

stock2:Company: NanoSmart Shares: 12  
 Share Price: \$20.00 Total Worth: \$240.00

Using a constructor to reset an object  
 Constructor using Nifty Foods called  
 Bye, Nifty Foods!  
 Revised stock1:  
 Company: Nifty Foods Shares: 10  
 Share Price: \$50.000 Total Worth: \$500.00  
 Done  
 Bye, NanoSmart!  
 Bye, Nifty Foods!

The last object created is the first deleted.

## 2.4 *this* pointer

There's only one copy of each class's functionality, but there can be many objects of a class. Every object has access to its own address through a pointer called *this*. The *this* pointer is passed (by the compiler) as an implicit argument to each of the object's non-static member functions.

```
const Stock & Stock::topval(const Stock & s) const
{
    if (s.total_val > total_val)
        return s;                      // argument object
    else
        return ?????;                // invoking object
}
```

```
const Stock & Stock::topval(const Stock & s) const
{
    if (s.total_val > total_val)
        return s;                      // argument object
    else
        return *this;                 // invoking object
}
```

## Using the *this* pointer to avoid naming collisions

```
void Box::setLength(double length) {  
    this->length = length;  
}
```

```
void Box::setBreadth(double breadth) {  
    this->breadth = breadth;  
}
```

```
void Box::setHeight(double height) {  
    this->height = height;  
}
```

Implicitly and explicitly using the *this* pointer to access an object's data member:

```
double Box::getLength() {  
    return this->length;  
}
```

```
double Box::getBreadth() {  
    return this->breadth;  
}
```

```
double Box::getHeight() {  
    return this->height;  
}
```

## 2.5 An Array of Objects

You can declare an array of objects the same way you declare an array of any of the standard types.

```
Stock mystuff[4]; // creates an array of 4 Stock objects
```

You can use a constructor to initialize the array elements by calling the constructor for each individual element:

```
const int STKS = 4;
Stock stocks[STKS] = {
    Stock("NanoSmart", 12.5, 20),
    Stock("Boffo Objects", 200, 2.0),
    Stock("Monolithic Obelisks", 130, 3.25),
    Stock("Fleep Enterprises", 60, 6.5)
};
```

If the class has more than one constructor, you can use different constructors for different elements:

```
const int STKS = 10;  
Stock stocks[STKS] = {  
    Stock("NanoSmart", 12.5, 20),  
    Stock(),  
    Stock("Monolithic Obelisks", 130, 3.25),  
};
```

The remaining seven members are initialized using the default constructor.

```

const int STKS = 4;

int main()
{
    //create an array of initialized objects
    Stock stocks[STKS] = {
        Stock("NanoSmart", 12, 20.0),
        Stock("Boffo Objects", 200, 2.0),
        Stock("Monolithic Obelisks", 130, 3.25),
        Stock("Fleep Enterprises", 60, 6.5)
    };

    std::cout << "Stock holdings:\n";
    int st;
    for(st = 0; st < STKS; st++)
        stocks[st].show();

    //set pointer to first element
    const Stock * top = &stocks[0];
    for(st = 1; st < STKS; st++)
        top = &top->topval(stocks[st]);

    //now top points to the most valuable holding
    std::cout << "\nMost valuable holding:\n";
    top->show();

    return 0;
}

```

Constructor using NanoSmart called  
 Constructor using Boffo Objects called  
 Constructor using Monolithic Obelisks called  
 Constructor using Fleep Enterprises called

Stock holdings:

Company: NanoSmart Shares: 12  
 Share Price: \$20.00 Total Worth: \$240.00  
 Company: Boffo Objects Shares: 200  
 Share Price: \$2.000 Total Worth: \$400.00  
 Company: Monolithic Obelisks Shares: 130  
 Share Price: \$3.250 Total Worth: \$422.50  
 Company: Fleep Enterprises Shares: 60  
 Share Price: \$6.500 Total Worth: \$390.00

Most valuable holding:

Company: Monolithic Obelisks Shares: 130  
 Share Price: \$3.250 Total Worth: \$422.50

Bye, Fleep Enterprises!  
 Bye, Monolithic Obelisks!  
 Bye, Boffo Objects!  
 Bye, NanoSmart!

The destructor is called in the reverse order  
that the constructor is called