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

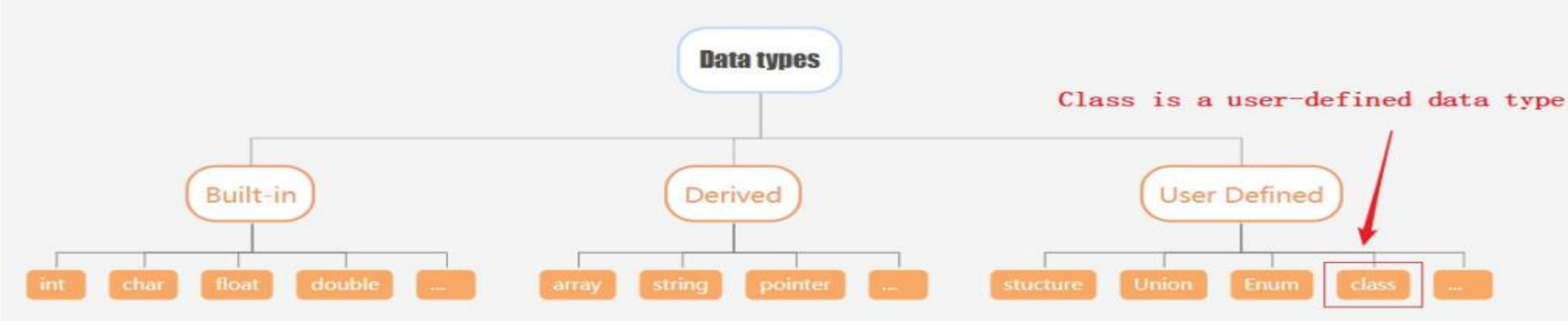
2.2 Access specifier: **private** and **public**

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
{
    Access specifier:
        Data members;
        Member Functions();
};
```

private/public/protected

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.

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

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

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

};

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;
}
```

Define a class named Box

The member variables of a class

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

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

```h
#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

```cpp
Box::Box() {
    length = 3;
    breadth = 4;
    height = 5;
}
Box::Box(int length, in
    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;
}
```

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

```cpp
#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
```

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

```
#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);
};
```

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

```
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.

```cpp
#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;
}
```

Invoke default constructor

Invoke parameterized constructor

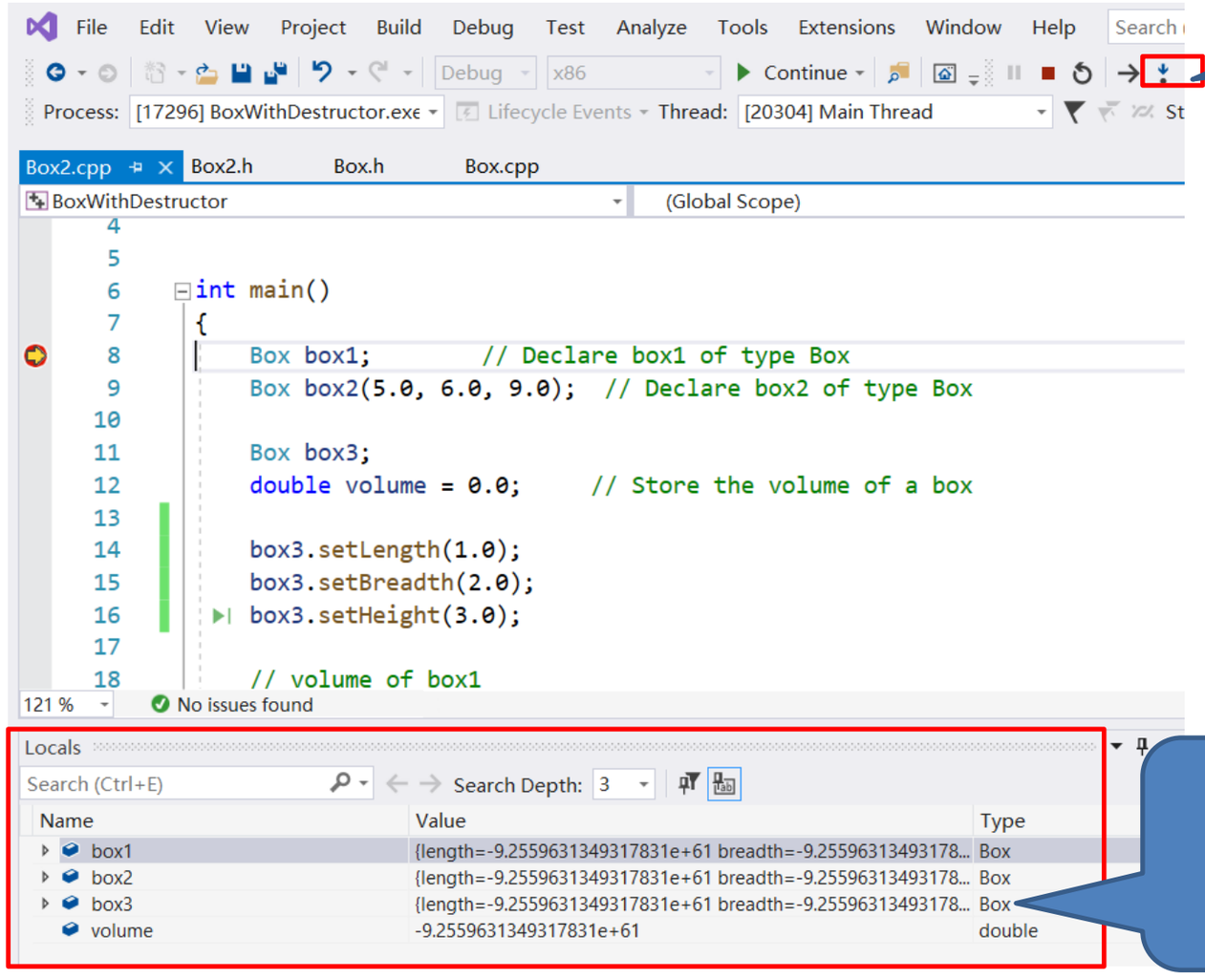
Invoke default constructor

Update the data field by setter

Output:

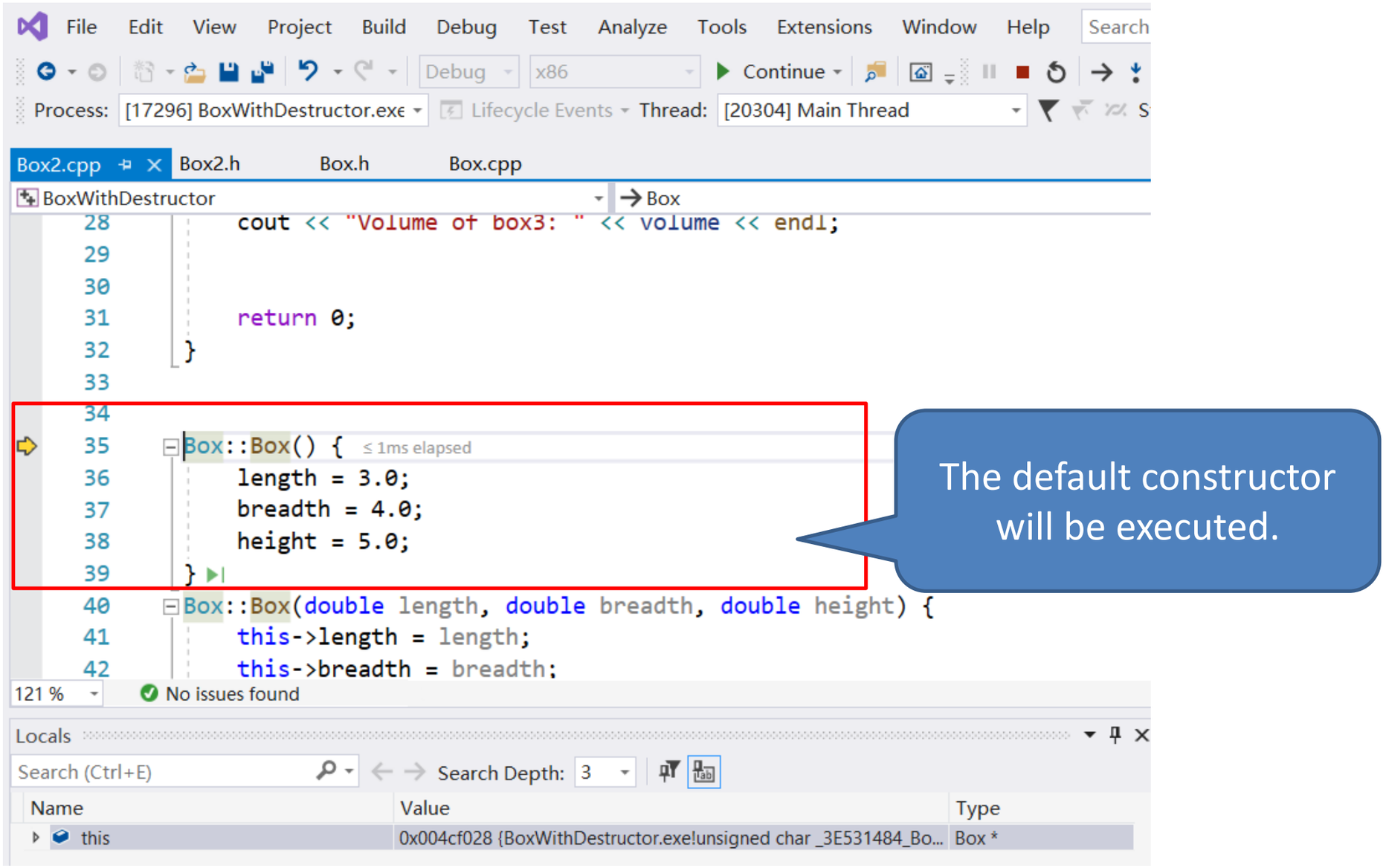
```
Volume of box1: 60
Volume of box2: 270
Volume of box3: 6
```

Run the program step by step

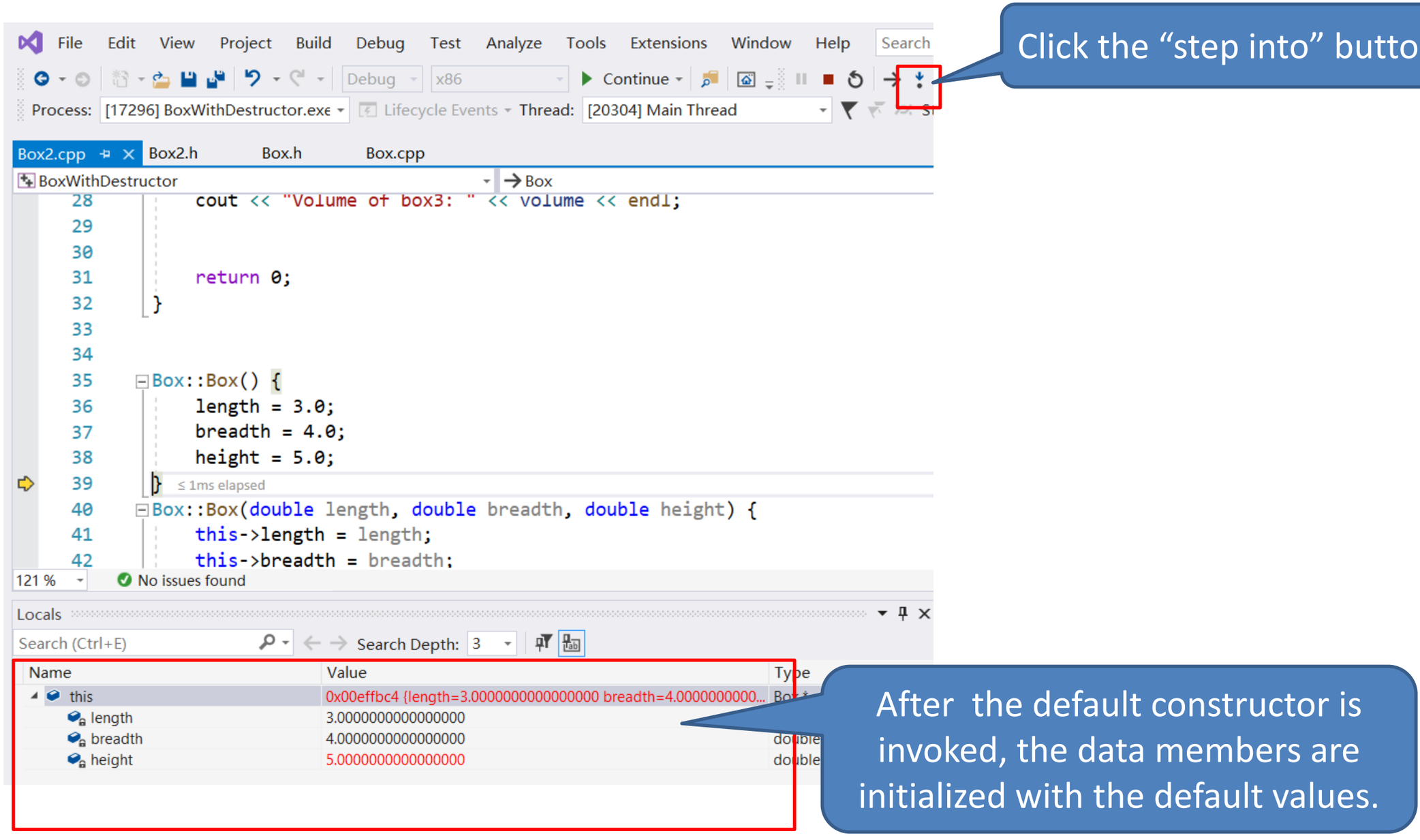


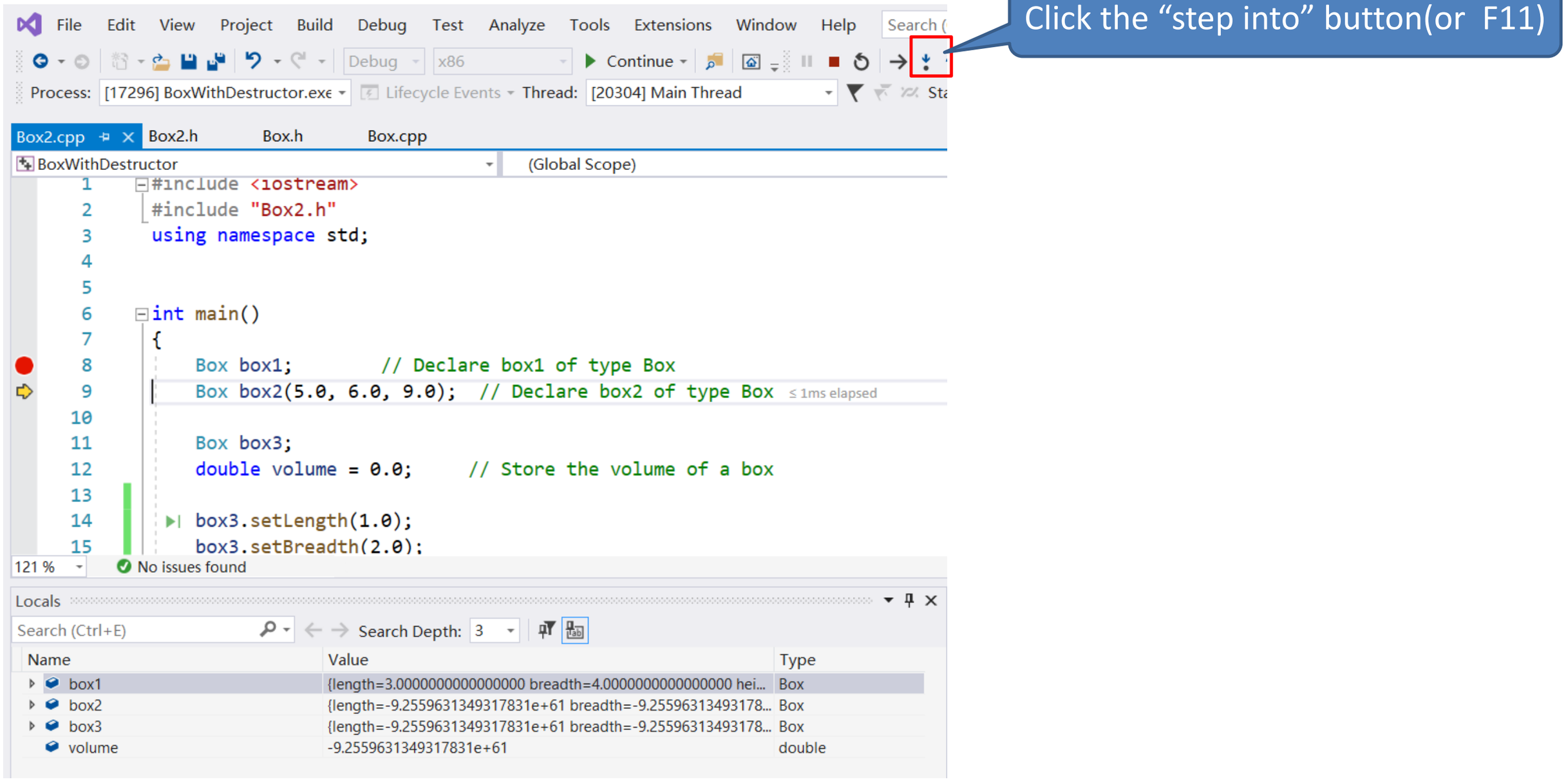
Click the “step into” button(or F11)

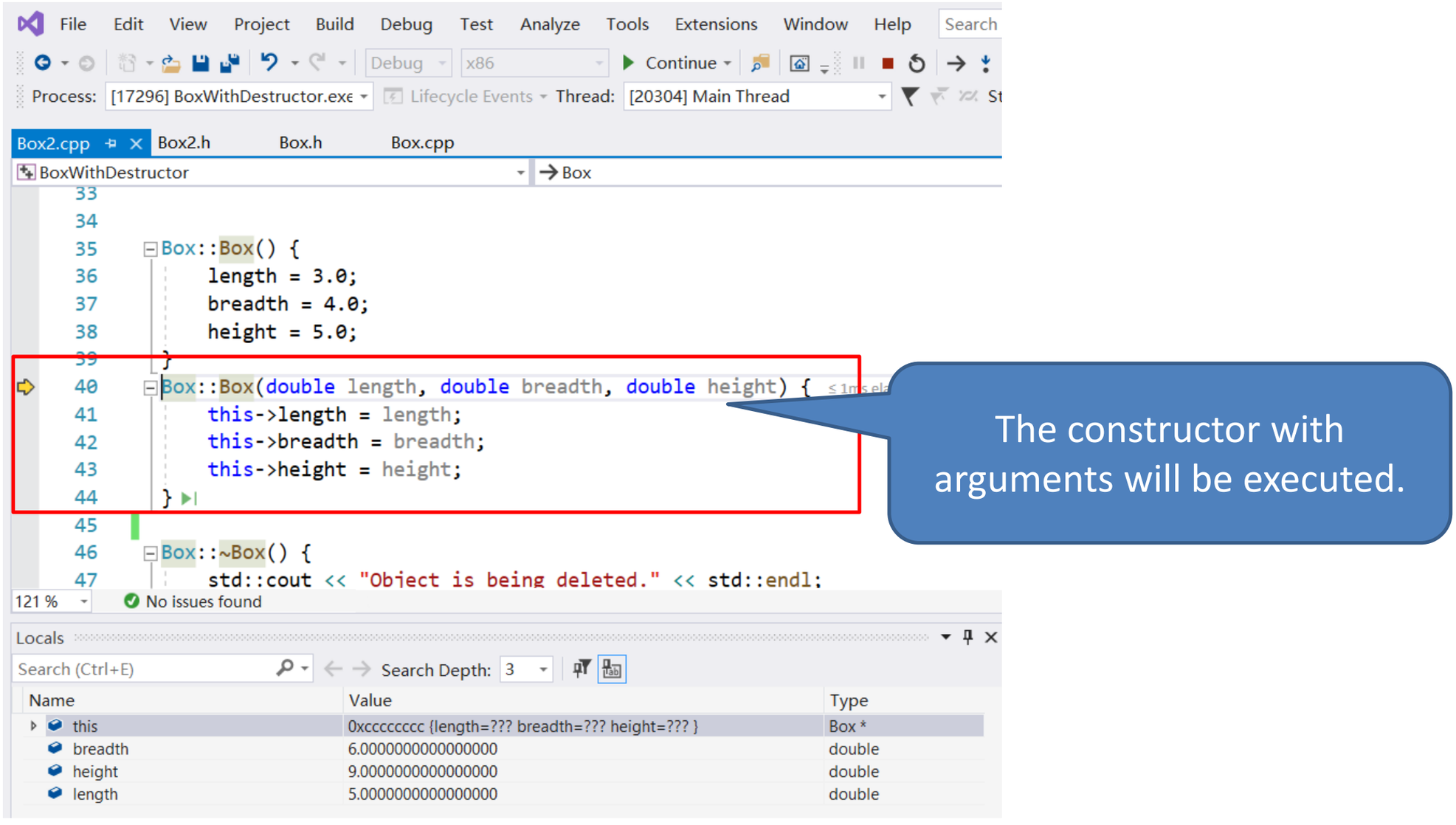
From beginning, the object is not created.



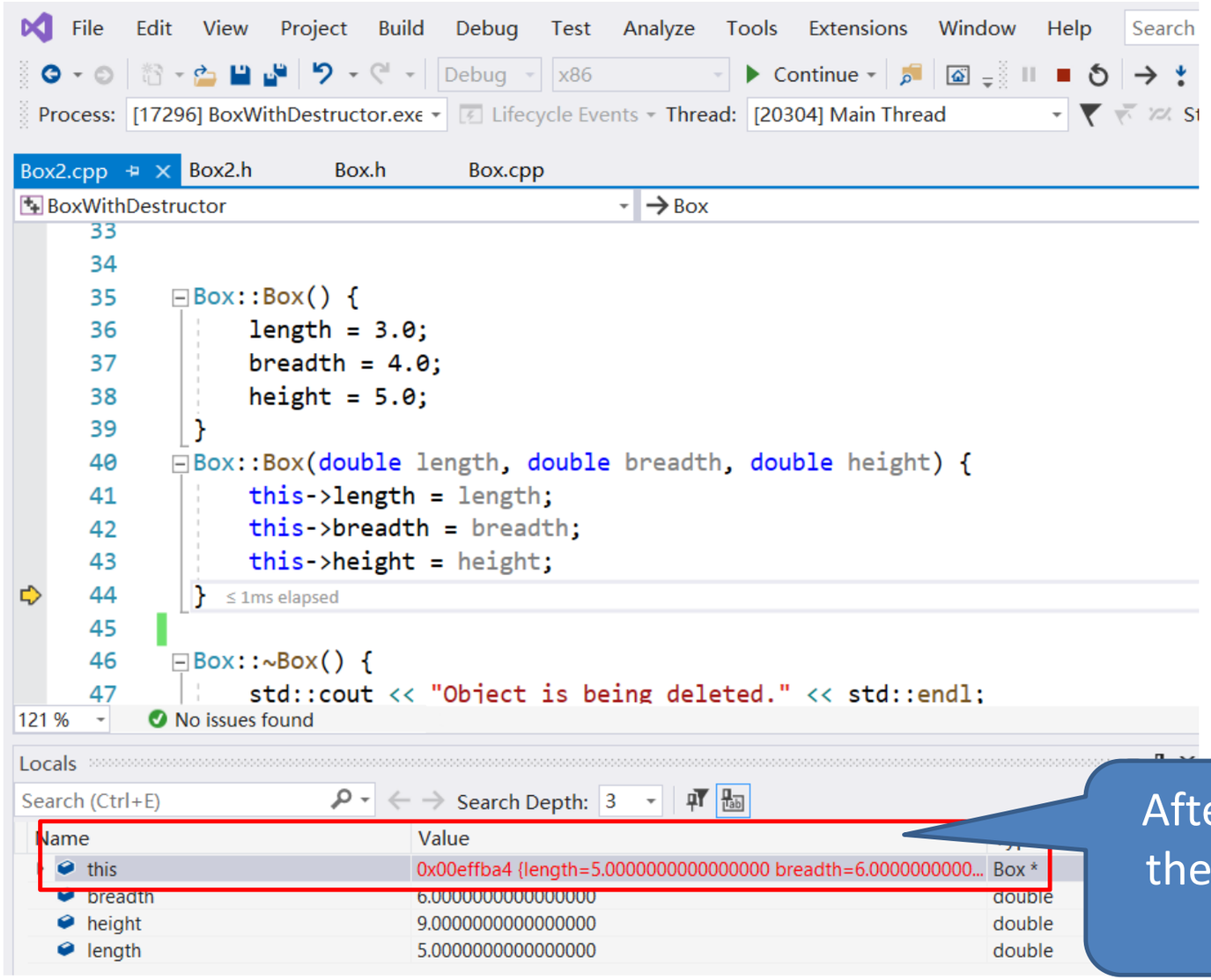
The default constructor will be executed.


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search
Debug x86 Continue
Process: [17296] BoxWithDestructor.exe Lifecycle Events Thread: [20304] Main Thread
Box2.cpp Box2.h Box.h Box.cpp
BoxWithDestructor Box
cout << "Volume of box3: " << volume << endl;
return 0;
}
Box::Box() {
length = 3.0;
breadth = 4.0;
height = 5.0;
} ≤ 1ms elapsed
Box::Box(double length, double breadth, double height) {
this->length = length;
this->breadth = breadth;
121 %
No issues found
Locals
Search (Ctrl+E)
Search Depth: 3
Name
Value
Type
this
0x00effbc4 {length=3.0000000000000000 breadth=4.0000000000...
length
3.0000000000000000
breadth
4.0000000000000000
height
5.0000000000000000
Click the “step into” button(or F11)
After the default constructor is invoked, the data members are initialized with the default values.

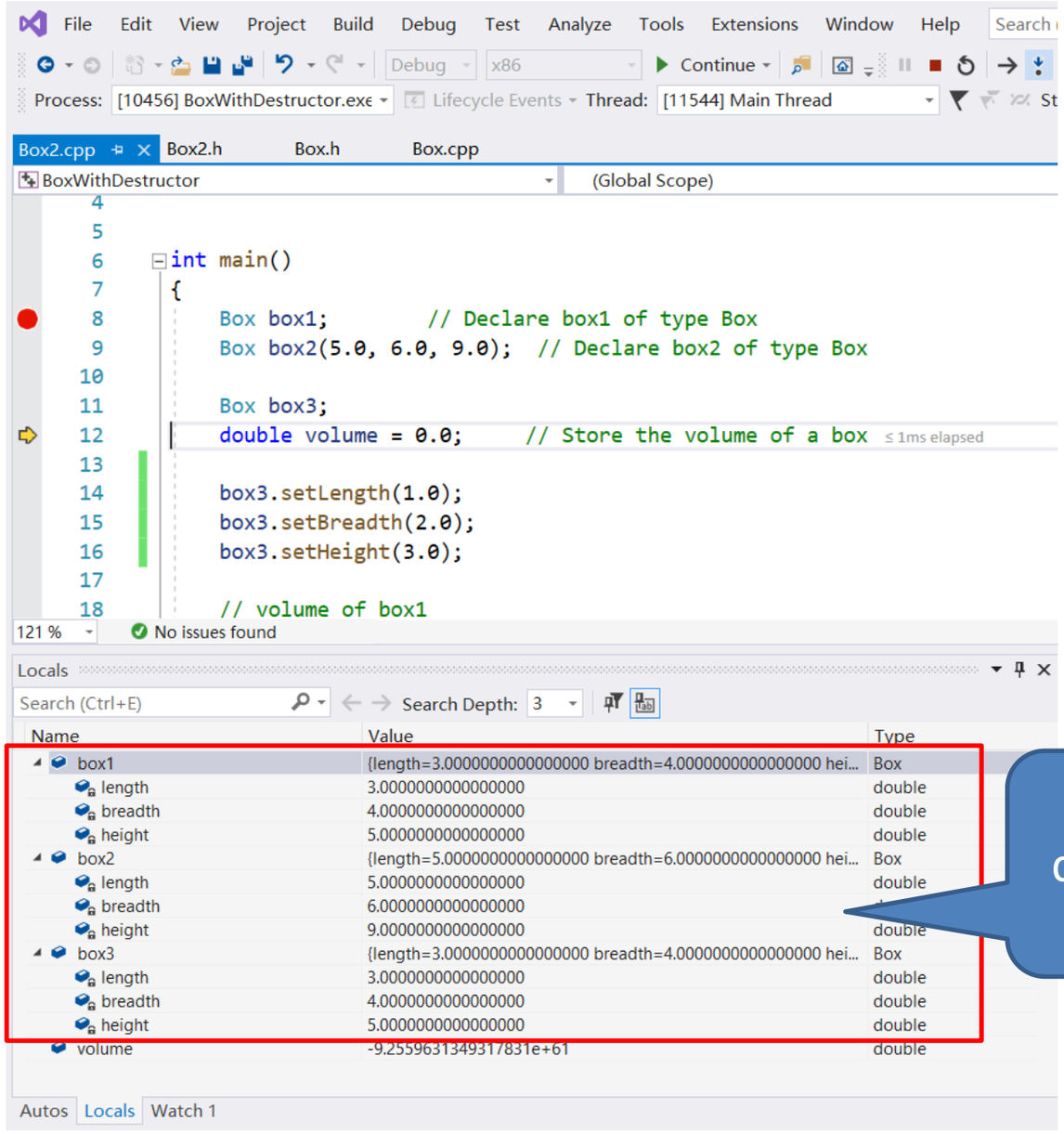

Click the “step into” button(or F11)
File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Debug x86 Continue
Process: [17296] BoxWithDestructor.exe Lifecycle Events Thread: [20304] Main Thread
Box2.cpp Box2.h Box.h Box.cpp
BoxWithDestructor (Global Scope)
#include <iostream>
#include "Box2.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 ≤ 1ms elapsed
Box box3;
double volume = 0.0; // Store the volume of a box
box3.setLength(1.0);
box3.setBreadth(2.0);
121 %
No issues found
Locals
Search (Ctrl+E)
Search Depth: 3
Name Value Type
box1 {length=3.0000000000000000 breadth=4.0000000000000000 hei... Box
box2 {length=-9.2559631349317831e+61 breadth=-9.25596313493178... Box
box3 {length=-9.2559631349317831e+61 breadth=-9.25596313493178... Box
volume -9.2559631349317831e+61 double



The constructor with arguments will be executed.



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

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 not provide any constructors, C++ will automatically supplies 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:

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

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

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 implictly
```

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.

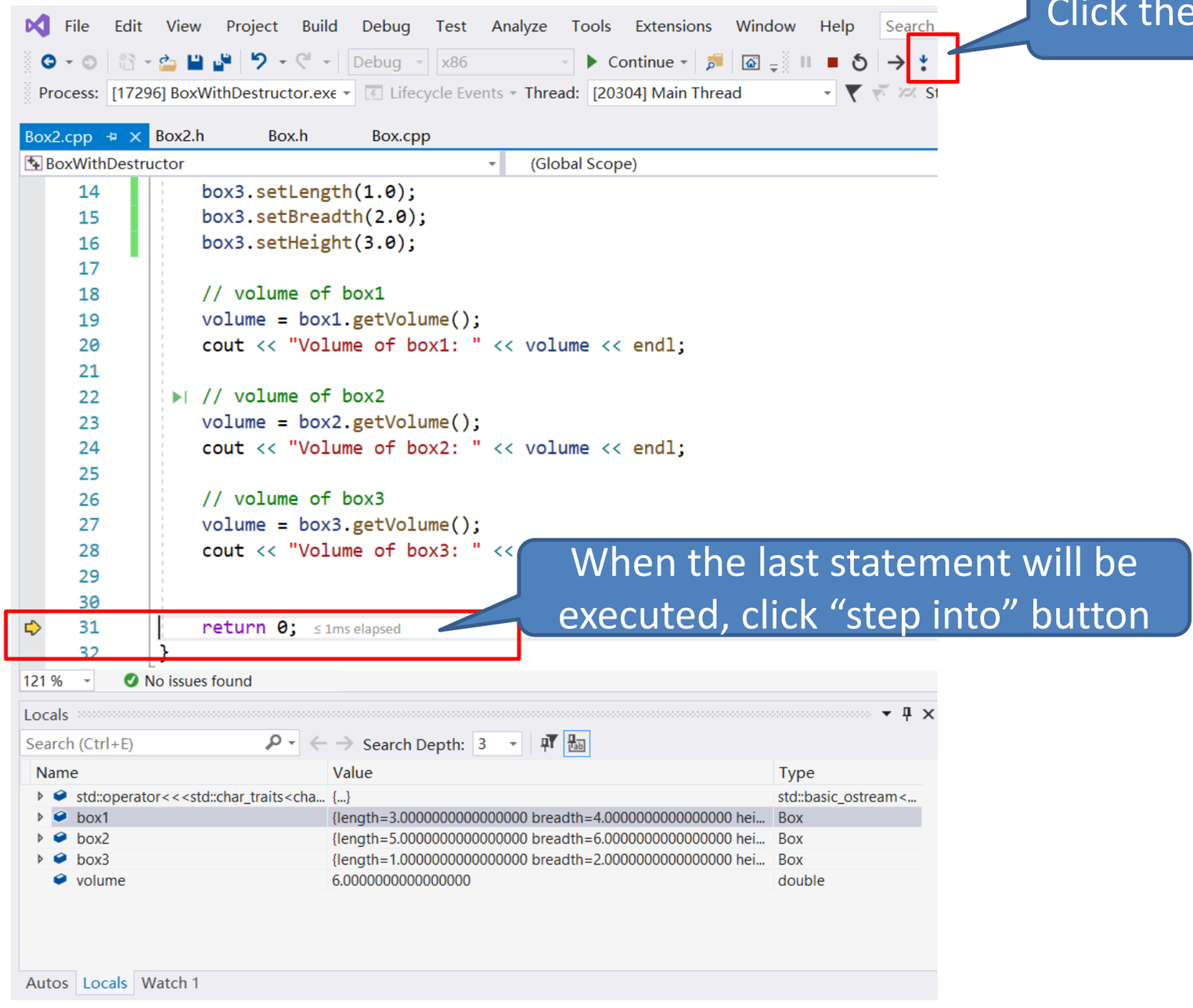
```cpp
#pragma once

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();
};
```

```cpp
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

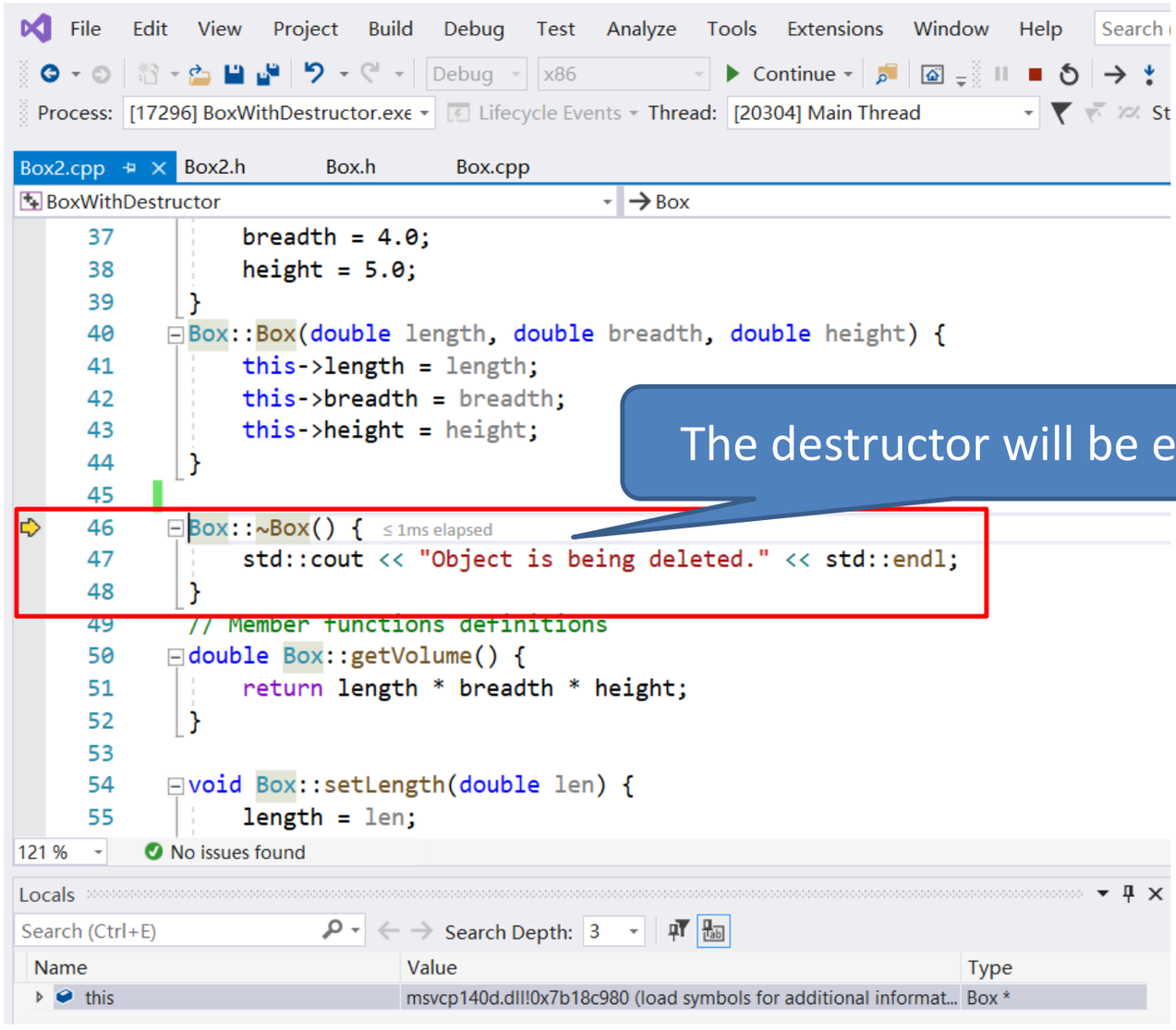
```
int main()
{
    Box box1;          // Declare box1 of type Box
    Box box2(5.0, 6.0, 9.0);  // Declare box2 of type Box

    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.
```

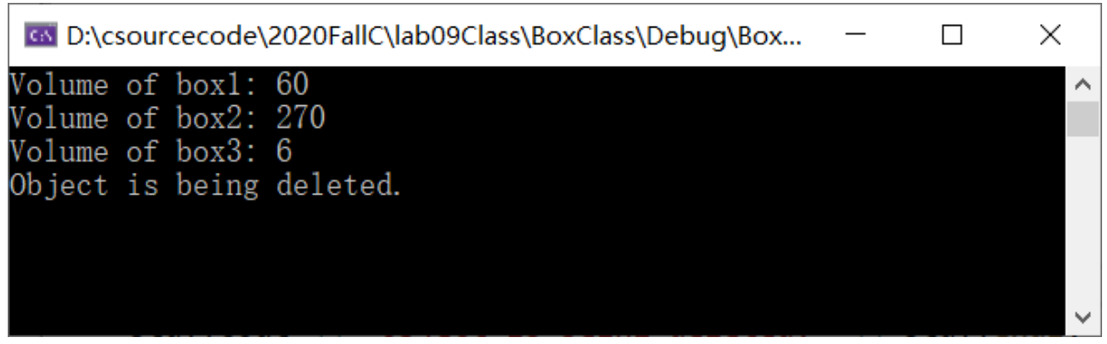


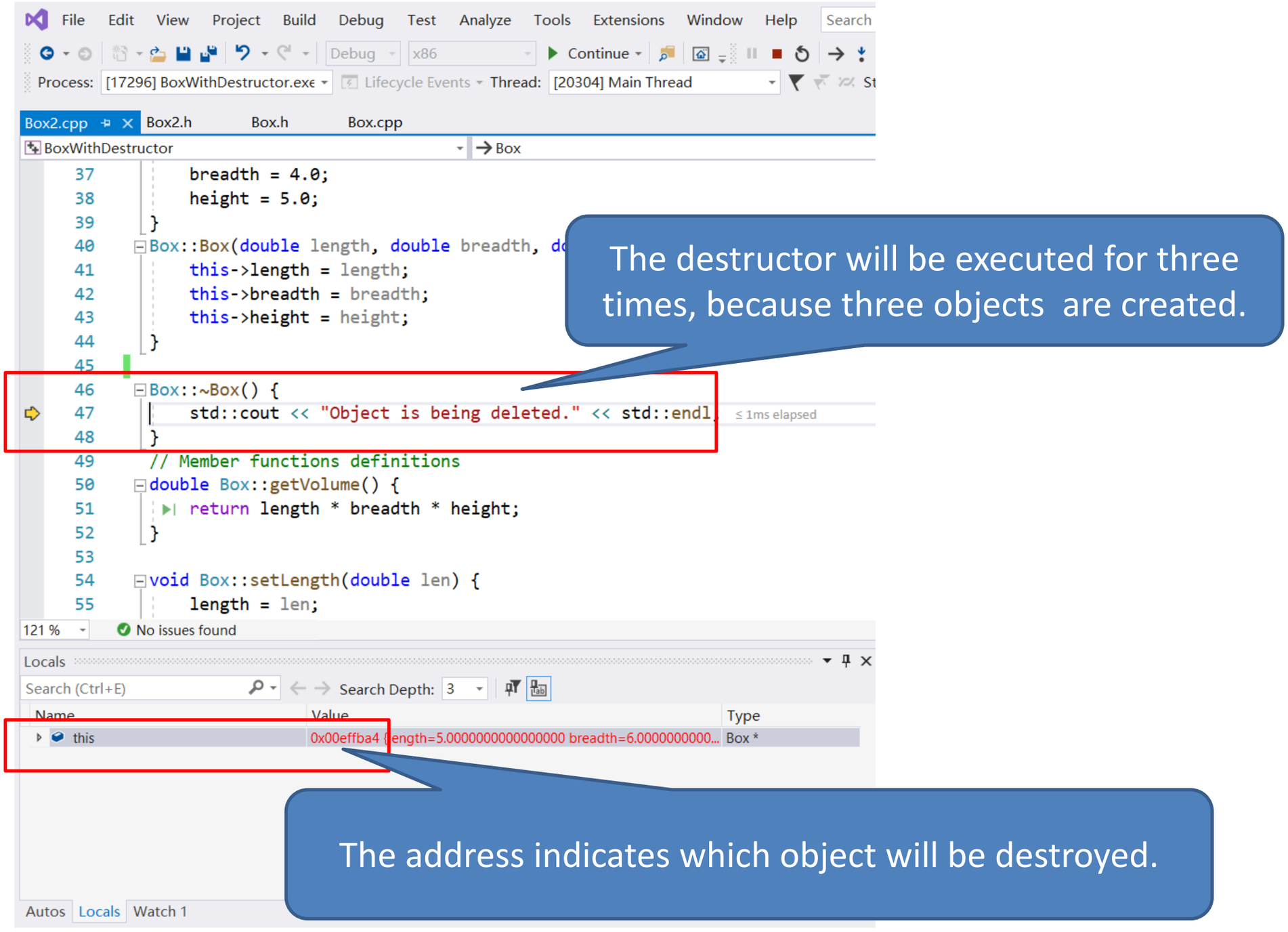
Click the “step into” button(or F11)

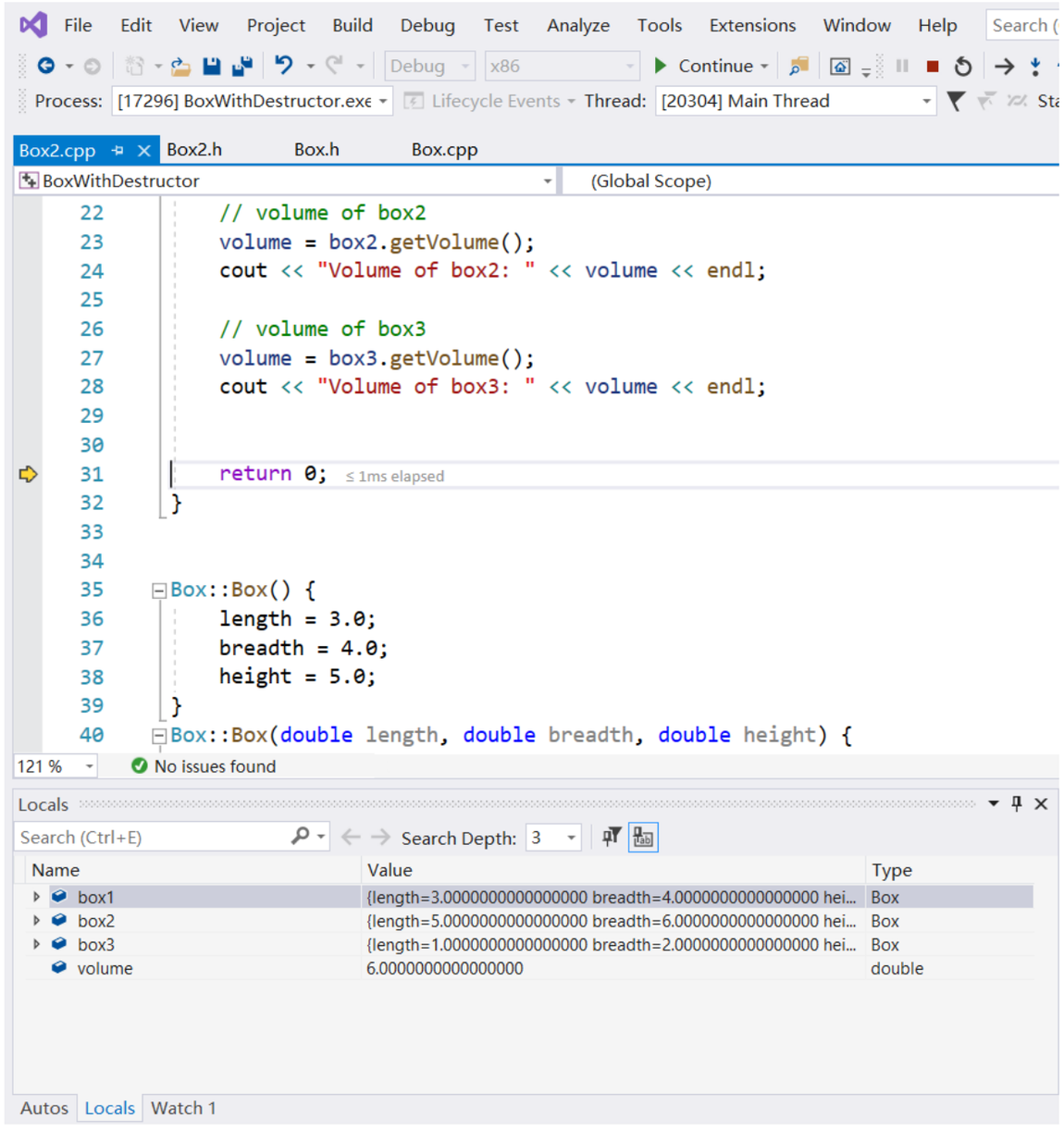
When the last statement will be executed, click “step into” button

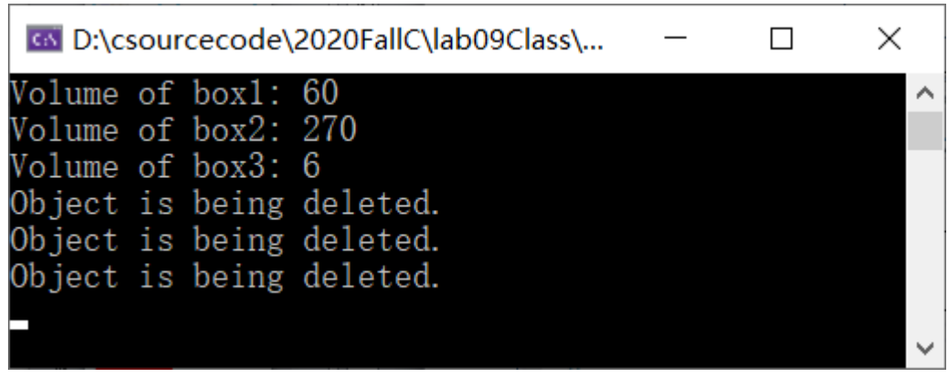


The destructor will be executed.




File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search
Debug x86 Continue
Process: [17296] BoxWithDestructor.exe Lifecycle Events Thread: [20304] Main Thread
Box2.cpp Box2.h Box.h Box.cpp
BoxWithDestructor Box
37 breadth = 4.0;
38 height = 5.0;
39 }
40 Box::Box(double length, double breadth, do
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;
≤ 1ms elapsed
121 %
No issues found
Locals
Search (Ctrl+E)
Search Depth: 3
Name
Value
Type
this
0x00effba4 {length=5.0000000000000000 breadth=6.0000000000...
Box *
Autos Locals Watch 1
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
Debug x86 Continue
Process: [17296] BoxWithDestructor.exe Lifecycle Events Thread: [20304] Main Thread
Box2.cpp Box2.h Box.h Box.cpp
BoxWithDestructor (Global Scope)
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 }
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) {
121 % No issues found
Locals
Search (Ctrl+E) Search Depth: 3
Name Value Type
box1 {length=3.0000000000000000 breadth=4.0000000000000000 hei... Box
box2 {length=5.0000000000000000 breadth=6.0000000000000000 hei... Box
box3 {length=1.0000000000000000 breadth=2.0000000000000000 hei... Box
volume 6.0000000000000000 double
Autos Locals Watch 1


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";
}
```

```cpp
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;

    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;
}
```

```
Using constructors to create new objects
Constructor using NanoSmart called
Company: NanoSmart Shares: 12
 Share Price: $20.000 Total Worth: $240.00

Constructor using Boffo Objects called
Company: Boffo Objects Shares: 2
 Share Price: $2.000 Total Worth: $4.00

Assigning stock1 to stock2:
Listing stock1 and stock2:
stock1:Company: NanoSmart Shares: 12
 Share Price: $20.000 Total Worth: $240.00

stock2:Company: NanoSmart Shares: 12
 Share Price: $20.000 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: $20.000 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