Lecture-26

Class – static and friend members

class – *static* member

```
1   #ifndef COMPLEX_H
2   #define COMPLEX_H
3
4   class complex {
5       private:
6           double Re, Im;
7           static int counter;
8       public:
9           complex();                      // Constructor function
10          complex(double, double);        // Overloading constructor
11          ~complex();                     // Destructor function
12
13          static int getCounterValue();
14  };
15
16  #endif // COMPLEX_H
17
```

A *static* data member is shared by all the objects. The *static* member is not part of any specific object but exist as soon as the *class* is defined.

A member function accessing the *static* data member must also be *static* and exists as soon as *class* is defined.

class – *static* member

```
1  #include <iostream>
2  #include "complex.h"
3  using namespace std;
4
5  int complex::counter = 0;
6
7  int complex::getCounterValue(){
8      return counter;
9  }
10
11 complex::complex()
12     : Re(0.0), Im(0.0)
13 {
14     ++counter;
15     cout << "\nDefault Constructor is called for ("
16          << Re << "," << Im << ") # "
17          << counter << endl;
18 }
19 complex::complex(double re, double im)
20     : Re(re), Im(im)
21 {
22     ++counter;
23     cout << "\nParameterized Constructor is called for ("
24          << Re << "," << Im << ") # "
25          << counter << endl;
26 }
27 complex::~complex(){
28     cout << "\nDestructor is called for ("
29          << Re << "," << Im << ") # "
30          << counter << endl;
31     --counter;
32 }
33
```

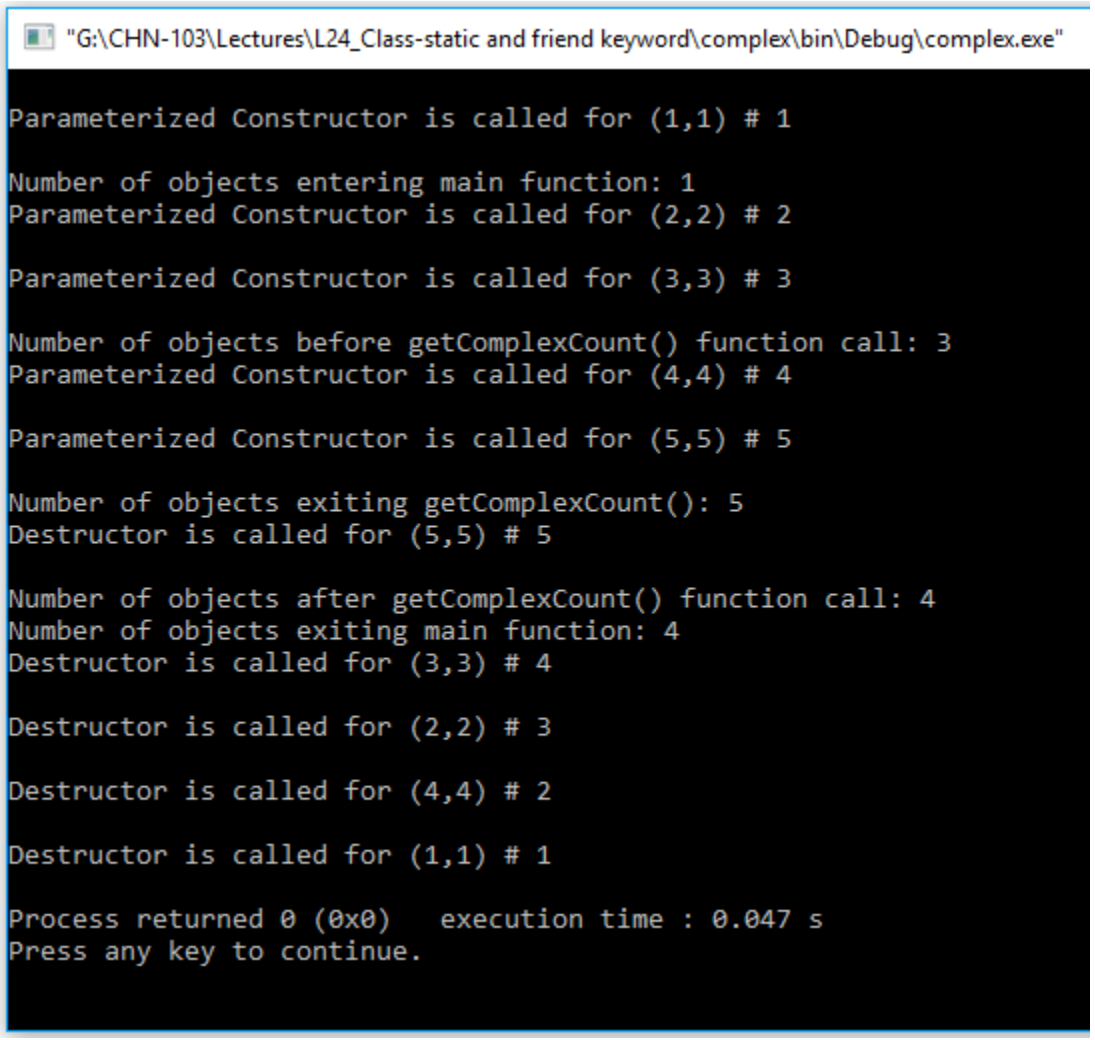
The static member must be initialized at global namespace scope without keyword *static*.

The constructors can access and update a unique copy of the static data member

class – *static* member

```
1   #include <iostream>
2   #include "complex.h"
3   using namespace std;
4
5   complex c1(1,1);
6
7   void getComplexCount(){
8       static complex c4(4,4);
9       complex c5(5,5);
10      cout << "\nNumber of objects exiting getComplexCount(): "
11           << c1.getCounterValue();
12  }
13
14  int main(){
15
16      cout << "\nNumber of objects entering main function: "
17           << c1.getCounterValue();
18      complex c2(2,2);
19      complex c3(3,3);
20      cout << "\nNumber of objects before getComplexCount() function call: "
21           << c1.getCounterValue();
22      getComplexCount();
23      cout << "\nNumber of objects after getComplexCount() function call: "
24           << c1.getCounterValue();
25      cout << "\nNumber of objects exiting main function: "
26           << c1.getCounterValue();
27
28  }
29
```

class – *static* member



Sum two complex numbers

```
34  complex sum (complex c1, complex c2){
35      complex result;
36      result.Re = c1.Re + c2.Re;
37      result.Im = c1.Im + c2.Im;
38      return result;
39  }
```

Make a global function like this

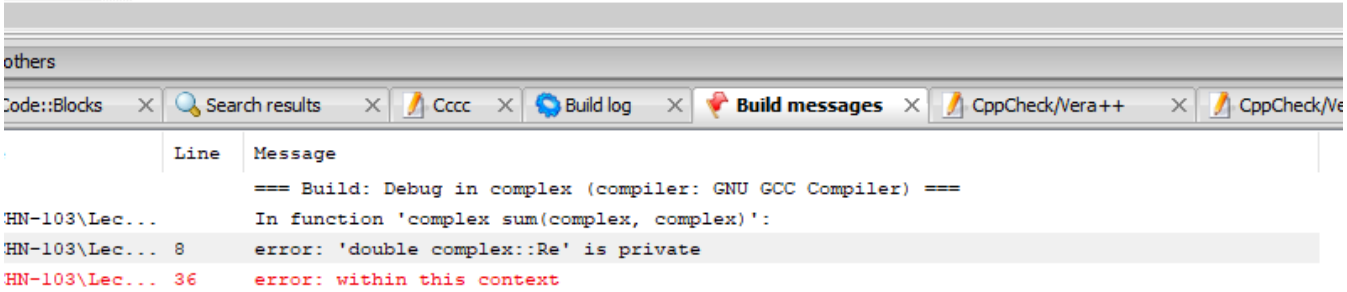
```
1   #include <iostream>
2   #include "complex.h"
3   using namespace std;
4
5   int main(){
6
7       complex c1(1.2, 3.2);
8       complex c2 (-2.1,4.5);
9       complex c3;
10      c3 = sum(c1,c2);
11  }
12
```

Write a main function that calls this function sum on two complex numbers.

Sum two complex numbers

```
1   #ifndef COMPLEX_H
2   #define COMPLEX_H
3   #include <iostream>
4   using namespace std;
5
6   class complex {
7       private:
8           double Re, Im;
9           static int counter;
10      public:
11          complex();                  // Constructor function
12          complex(double, double);    // Overloading constructor
13          ~complex();                 // Destructor function
14
15          static int getCounterValue();
16  };
17
18  #endif // COMPLEX_H
19
```

We get error because global function *sum()* does not have access to private data member *Re* and *Im*.



friend function

```
1   #ifndef COMPLEX_H
2   #define COMPLEX_H
3   #include <iostream>
4   using namespace std;
5
6   class complex {
7       private:
8           double Re, Im;
9           static int counter;
10      public:
11          complex();                      // Constructor function
12          complex(double, double);        // Overloading constructor
13          ~complex();                     // Destructor function
14
15          static int getCounterValue();
16          friend complex sum (complex c1, complex c2);
17  };
18
19  #endif // COMPLEX_H
20
```

To enable access the global function is declared inside the class scope with keyword *friend*.

Sum two complex numbers

Constructor for c1

Constructor for c2

Constructor for c3

Constructor for result

Destructor for result

Destructor for c1 copied into sum()

Destructor for c2 copied into sum()

Destructor for c3

Destructor for c2

Destructor for c1

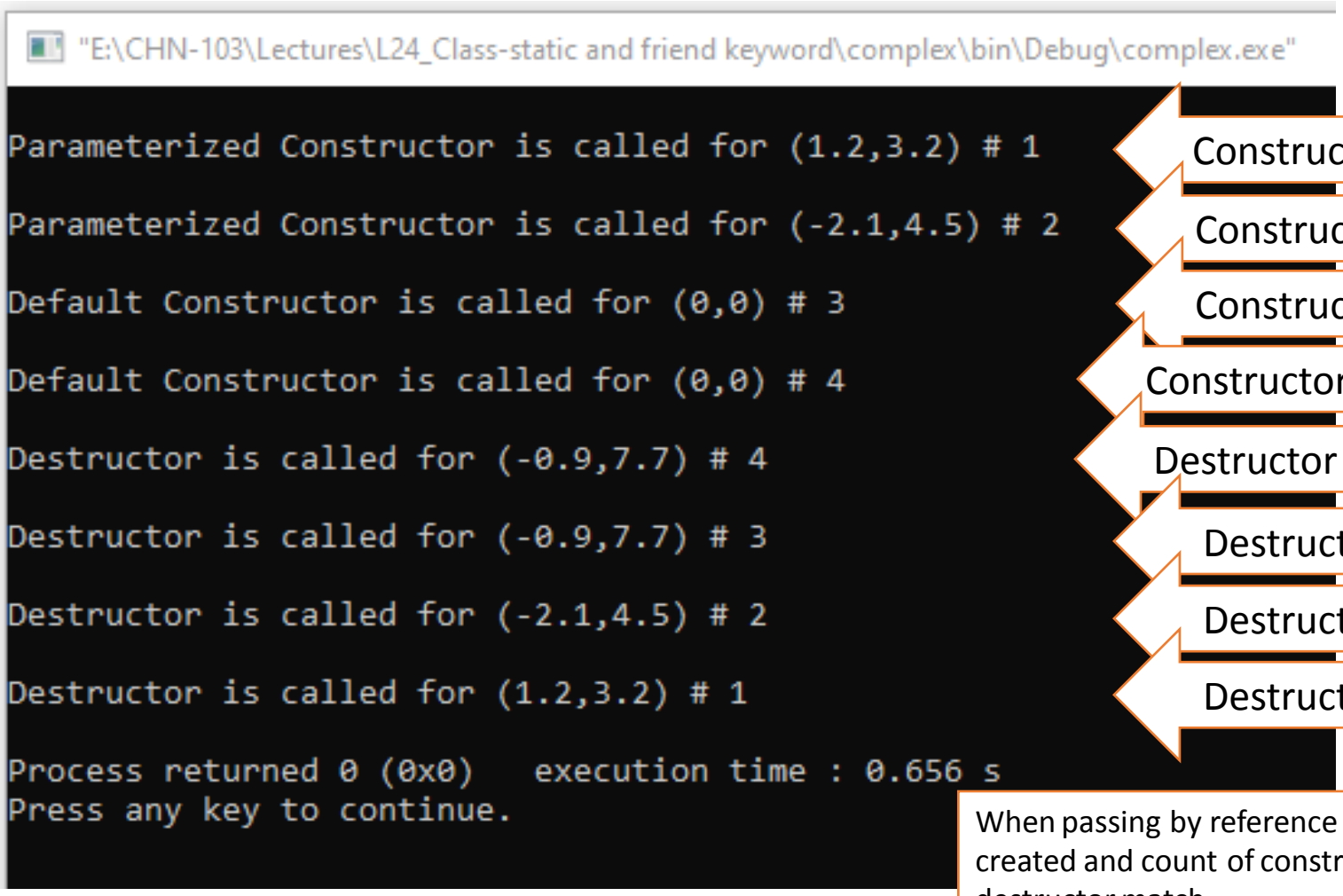
Copies of c1 and c2 were created by copy constructors implicitly provided by the compiler, hence not counted.

friend function

```
34  complex sum (complex& c1, complex& c2){
35      complex result;
36      result.Re = c1.Re + c2.Re;
37      result.Im = c1.Im + c2.Im;
38      return result;
39  }
```

```
1   #ifndef COMPLEX_H
2   #define COMPLEX_H
3   #include <iostream>
4   using namespace std;
5
6   class complex {
7       private:
8           double Re, Im;
9           static int counter;
10      public:
11          complex();                      // Constructor function
12          complex(double, double);        // Overloading constructor
13          ~complex();                     // Destructor function
14
15          static int getCounterValue();
16          friend complex sum (complex& c1, complex& c2);
17  };
18
19  #endif // COMPLEX_H
20
```

Sum two complex numbers



Constructor for c1

Constructor for c2

Constructor for c3

Constructor for result

Destructor for result

Destructor for c3

Destructor for c2

Destructor for c1

When passing by reference no copies are created and count of constructor and destructor match