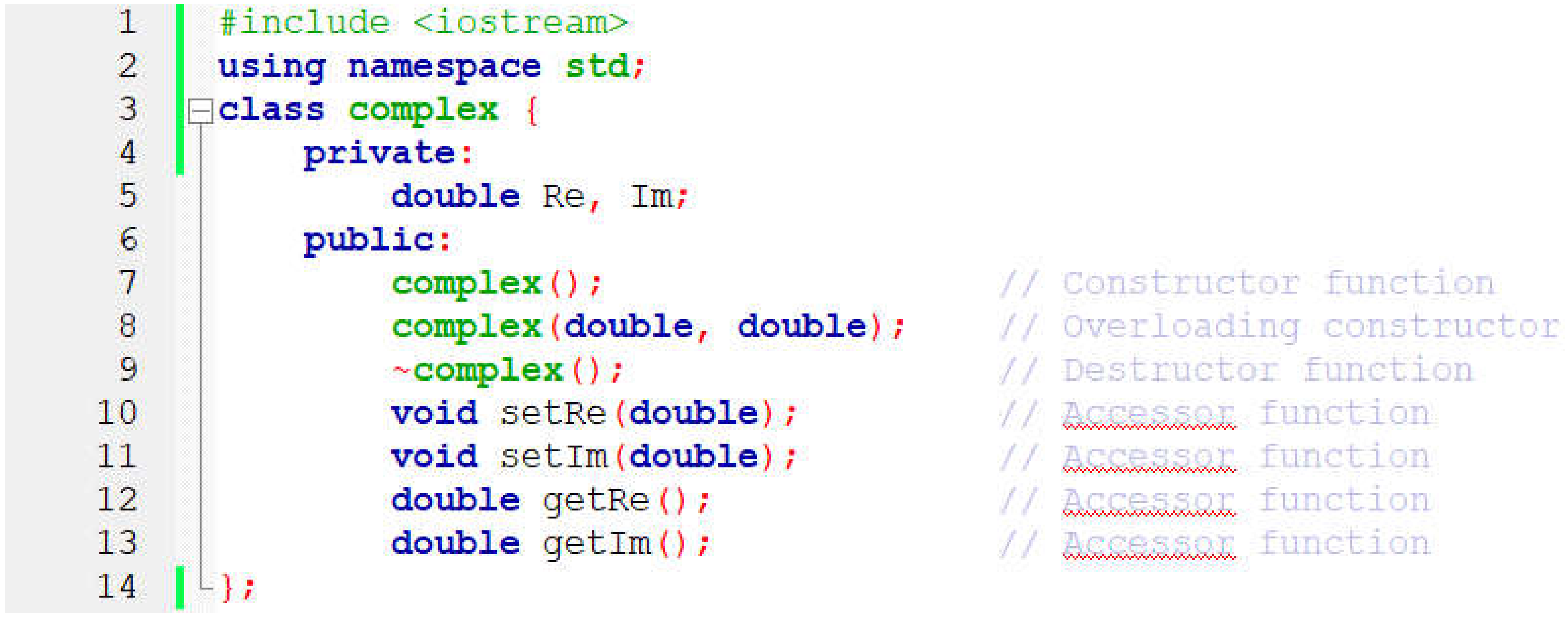
Lecture-23, 24

Class: project structure

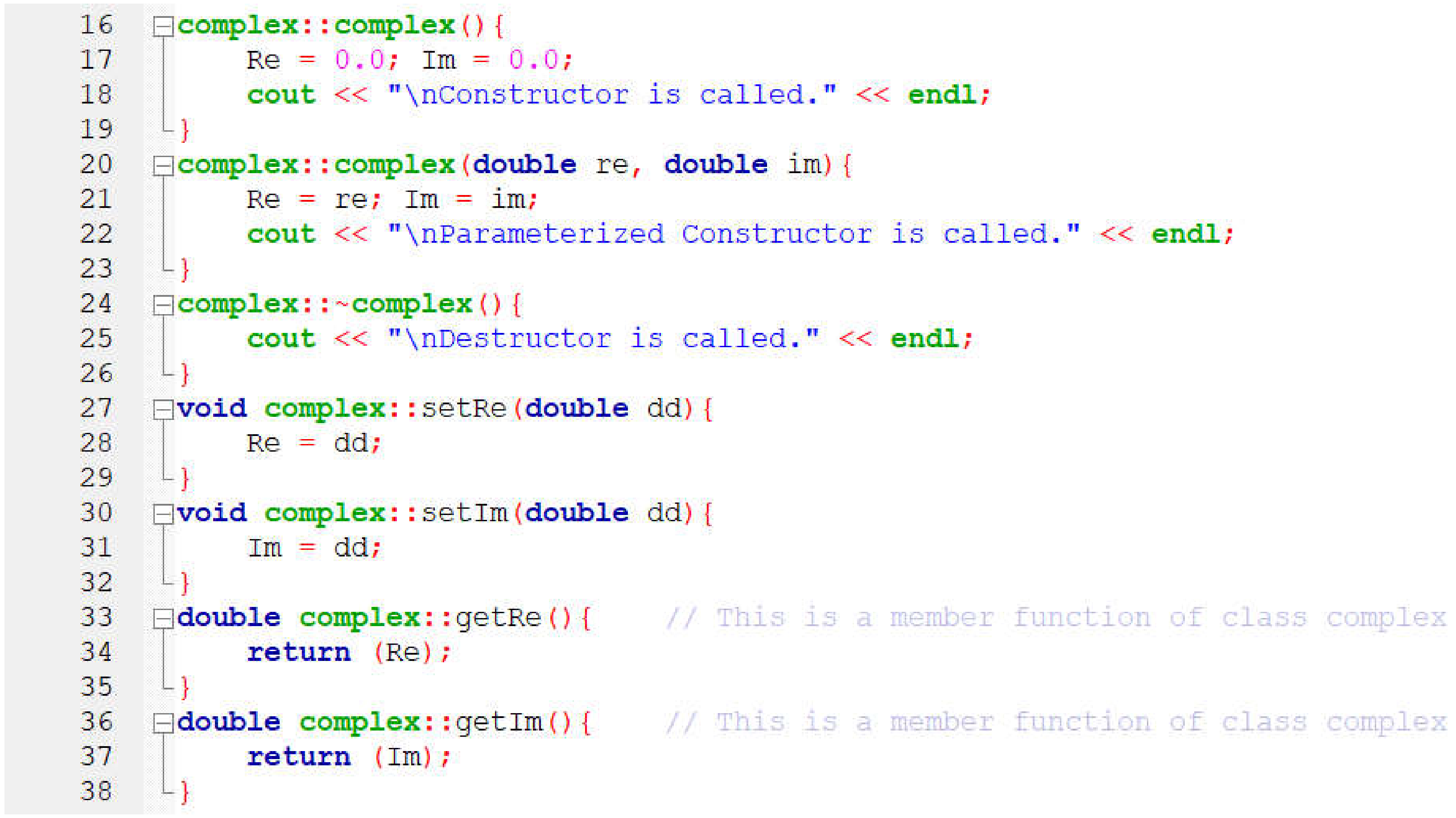
class datatype



```
#include <iostream>
using namespace std;
class complex {
    private:
        double Re, Im;
    public:
        complex();                     // Constructor function
        complex(double, double);       // Overloading constructor
        ~complex();                    // Destructor function
        void setRe(double);            // Accessor function
        void setIm(double);            // Accessor function
        double getRe();                // Accessor function
        double getIm();                // Accessor function
};
```

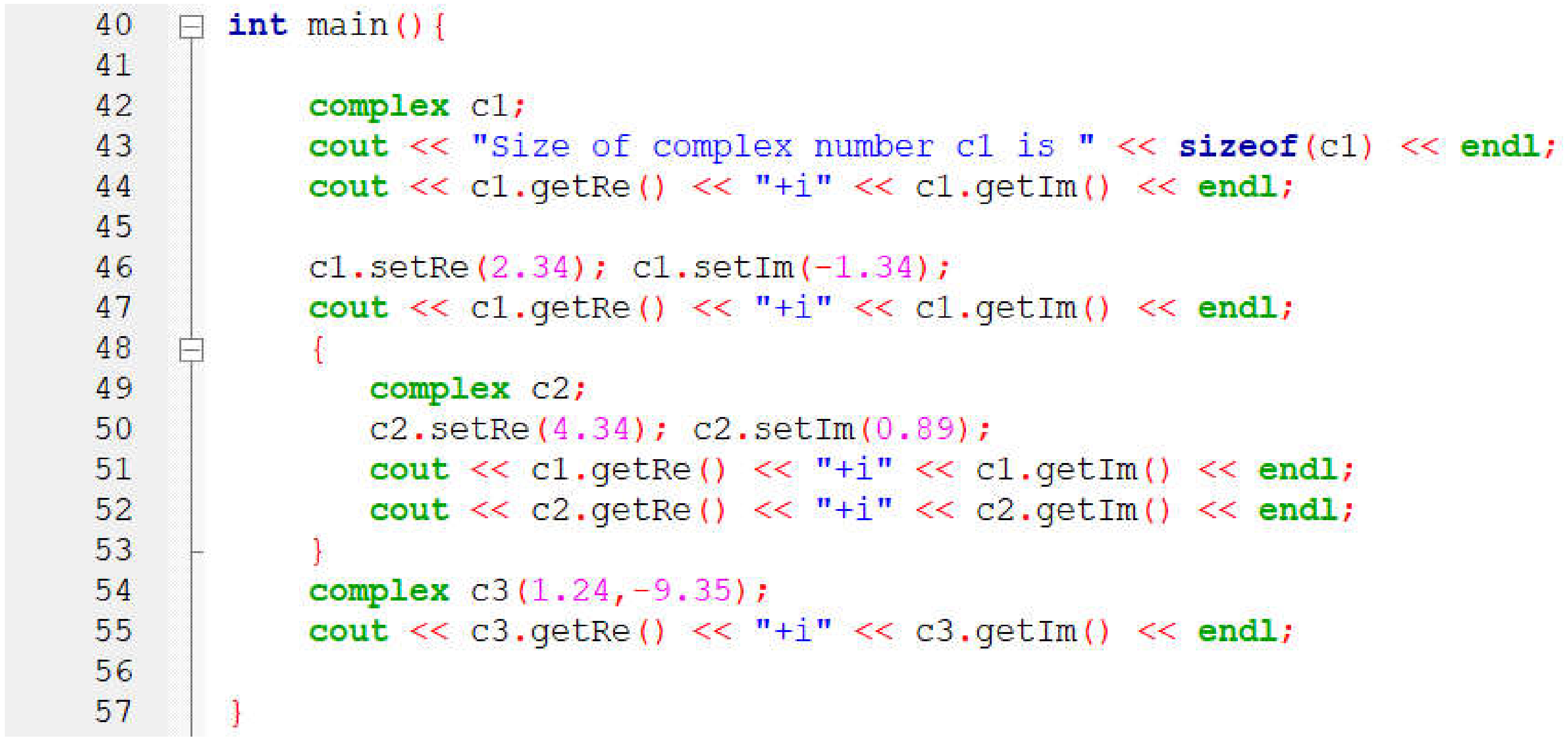
Unlike *struct* and *union*, the members of *class* are **private** by default.

class datatype



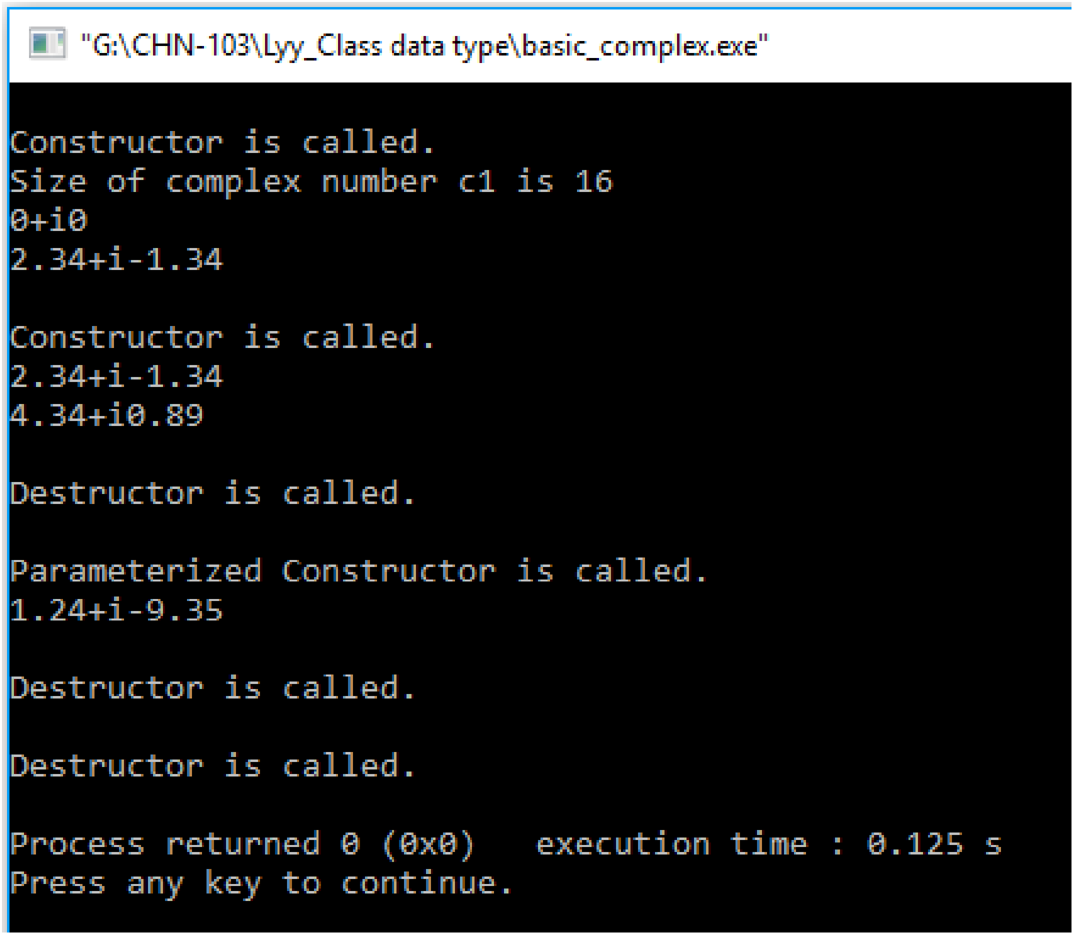
```cpp
complex::complex(){
    Re = 0.0; Im = 0.0;
    cout << "\nConstructor is called." << endl;
}
complex::complex(double re, double im){
    Re = re; Im = im;
    cout << "\nParameterized Constructor is called." << endl;
}
complex::~complex(){
    cout << "\nDestructor is called." << endl;
}
void complex::setRe(double dd){
    Re = dd;
}
void complex::setIm(double dd){
    Im = dd;
}
double complex::getRe(){    // This is a member function of class complex
    return (Re);
}
double complex::getIm(){    // This is a member function of class complex
    return (Im);
}
```

class datatype



```cpp
int main(){

    complex c1;
    cout << "Size of complex number c1 is " << sizeof(c1) << endl;
    cout << c1.getRe() << "+i" << c1.getIm() << endl;

    c1.setRe(2.34); c1.setIm(-1.34);
    cout << c1.getRe() << "+i" << c1.getIm() << endl;
    {
        complex c2;
        c2.setRe(4.34); c2.setIm(0.89);
        cout << c1.getRe() << "+i" << c1.getIm() << endl;
        cout << c2.getRe() << "+i" << c2.getIm() << endl;
    }
    complex c3(1.24,-9.35);
    cout << c3.getRe() << "+i" << c3.getIm() << endl;

}
```

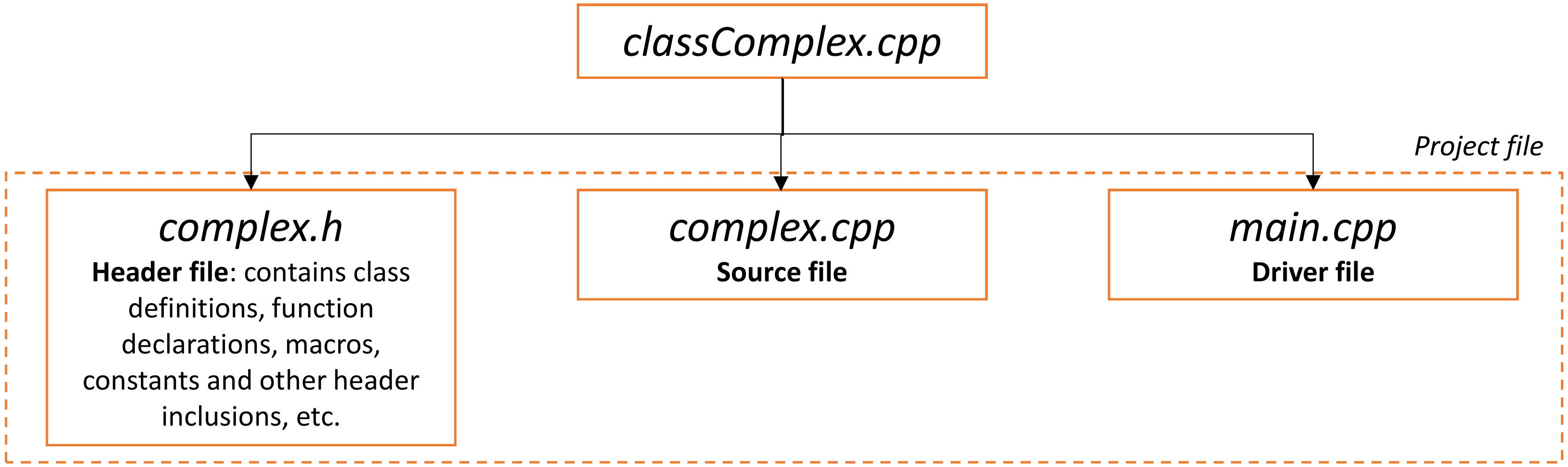
class datatype



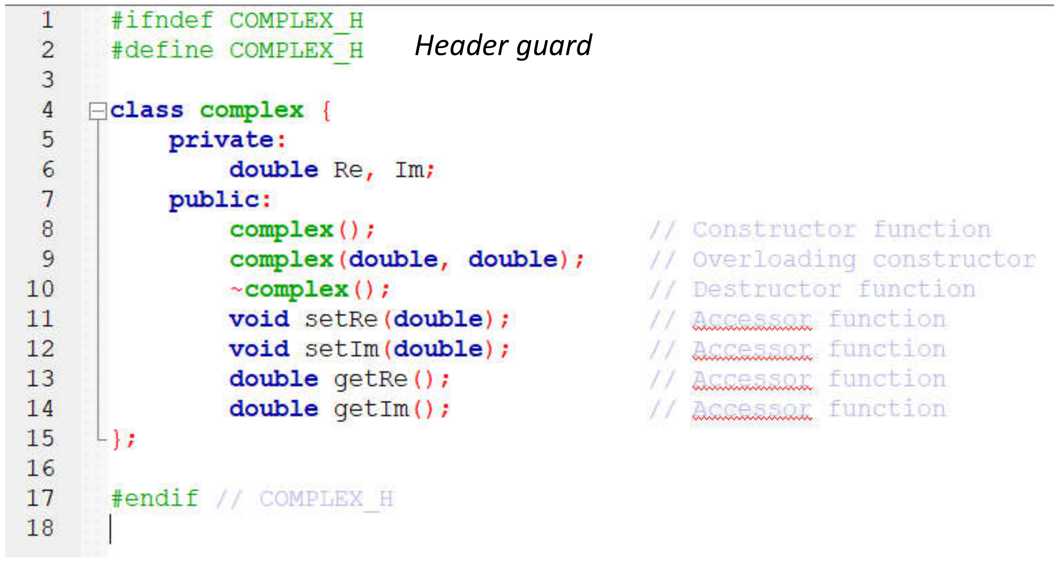
"G:\CHN-103\Lyy_Class data type\basic_complex.exe"

```
Constructor is called.
Size of complex number c1 is 16
0+i0
2.34+i-1.34

Constructor is called.
2.34+i-1.34
4.34+i0.89

Destructor is called.

Parameterized Constructor is called.
1.24+i-9.35

Destructor is called.

Destructor is called.

Process returned 0 (0x0)   execution time : 0.125 s
Press any key to continue.
```

Creating project structure



Header file: contains class definitions, function declarations, macros, constants and other header inclusions, etc.

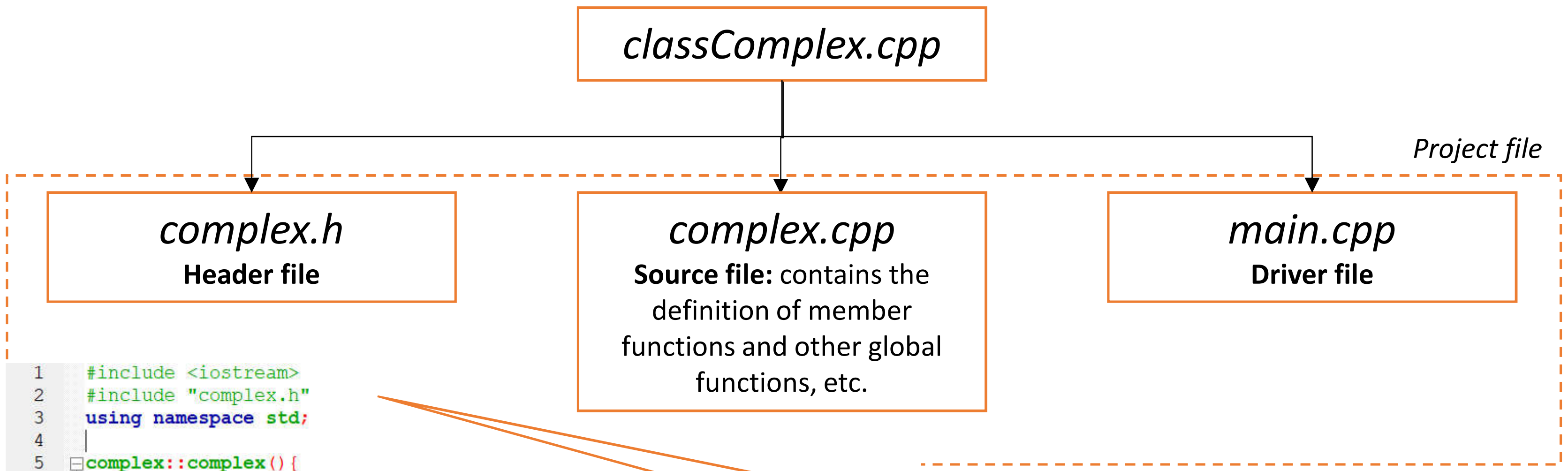


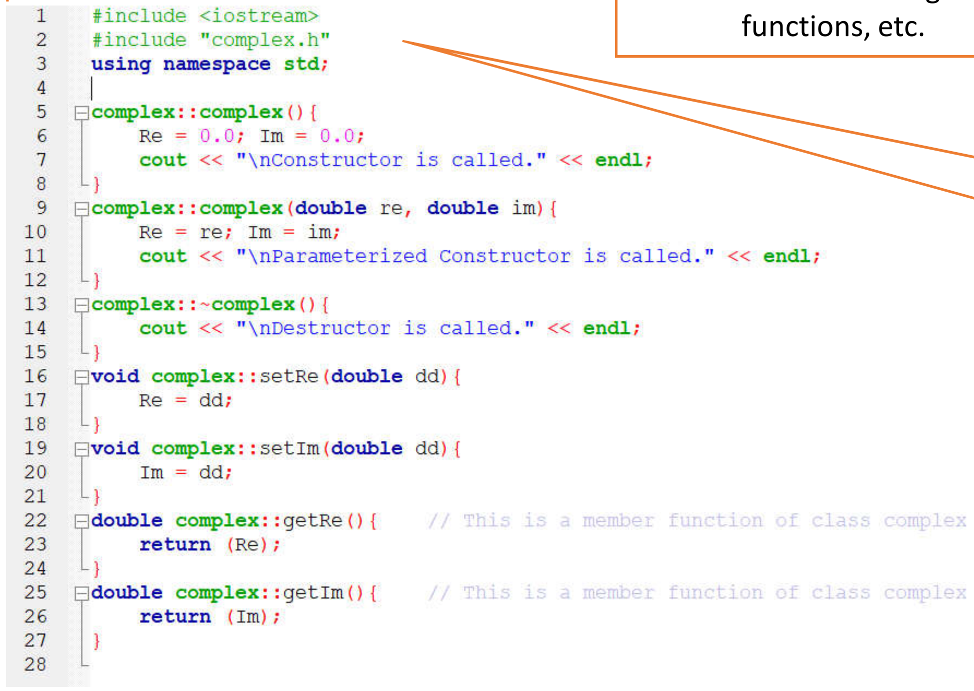
```
#ifndef COMPLEX_H
#define COMPLEX_H

class complex {
    private:
        double Re, Im;
    public:
        complex();                  // Constructor function
        complex(double, double);    // Overloading constructor
        ~complex();                 // Destructor function
        void setRe(double);         // Accessor function
        void setIm(double);         // Accessor function
        double getRe();             // Accessor function
        double getIm();             // Accessor function
};

#endif // COMPLEX_H

```

Header guard

Header guards makes sure that the contents of the file (between #ifndef and #endif) are included only once when preprocessed.

Creating project structure


classComplex.cpp
Project file
complex.h
Header file
complex.cpp
Source file: contains the definition of member functions and other global functions, etc.
main.cpp
Driver file

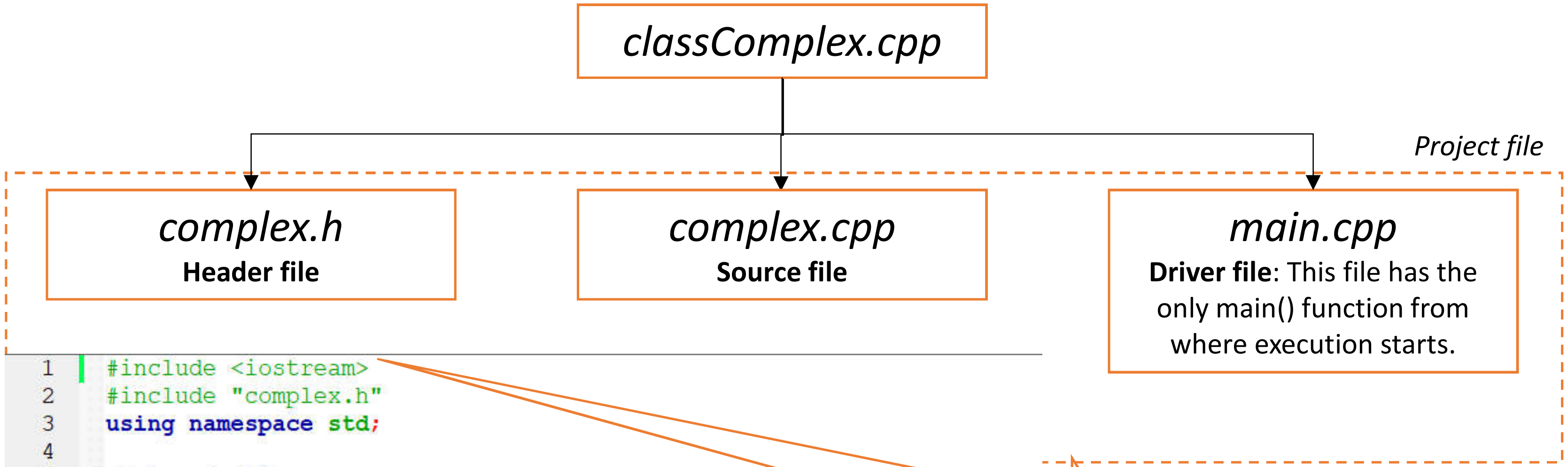


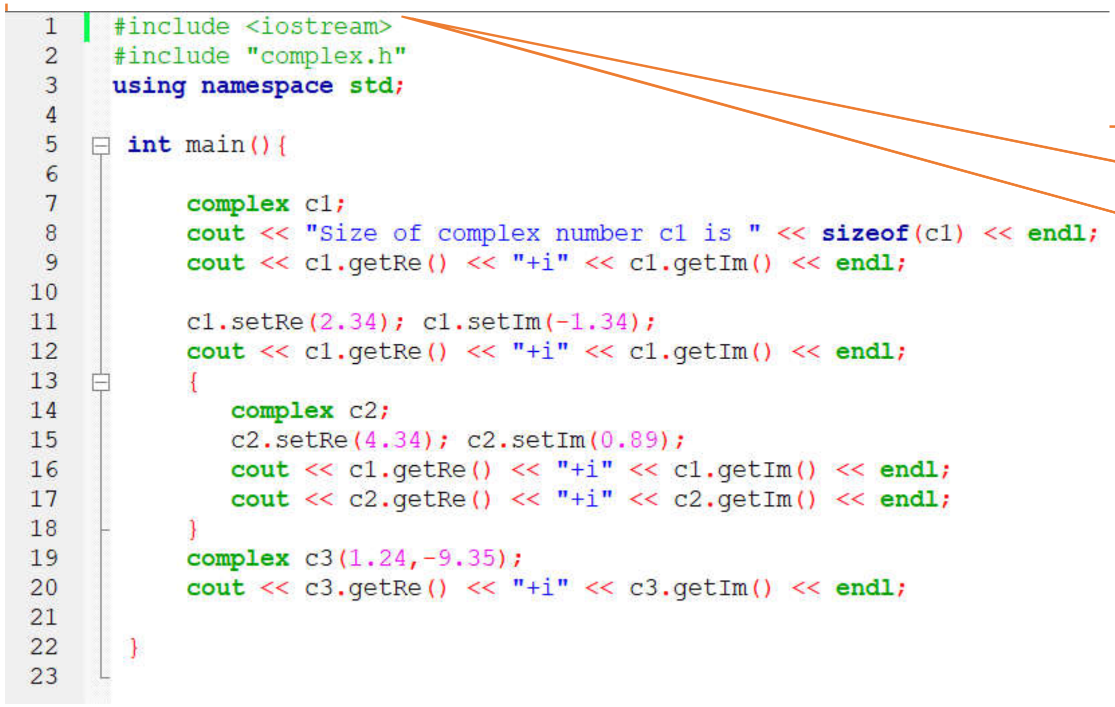
```
#include <iostream>
#include "complex.h"
using namespace std;

complex::complex(){
    Re = 0.0; Im = 0.0;
    cout << "\nConstructor is called." << endl;
}
complex::complex(double re, double im){
    Re = re; Im = im;
    cout << "\nParameterized Constructor is called." << endl;
}
complex::~complex(){
    cout << "\nDestructor is called." << endl;
}
void complex::setRe(double dd){
    Re = dd;
}
void complex::setIm(double dd){
    Im = dd;
}
double complex::getRe(){    // This is a member function of class complex
    return (Re);
}
double complex::getIm(){    // This is a member function of class complex
    return (Im);
}

```

It also has the directives for including other libraries and header files.

Here when the **#include "complex.h"** directive is executed by the preprocessor it defines COMPLEX_H macro for first time and therefore ***includes*** the header file.

Creating project structure





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

 int main(){

     complex c1;
     cout << "Size of complex number c1 is " << sizeof(c1) << endl;
     cout << c1.getRe() << "+i" << c1.getIm() << endl;

     c1.setRe(2.34); c1.setIm(-1.34);
     cout << c1.getRe() << "+i" << c1.getIm() << endl;
     {
        complex c2;
        c2.setRe(4.34); c2.setIm(0.89);
        cout << c1.getRe() << "+i" << c1.getIm() << endl;
        cout << c2.getRe() << "+i" << c2.getIm() << endl;
     }
     complex c3(1.24,-9.35);
     cout << c3.getRe() << "+i" << c3.getIm() << endl;

 }
```

It also has the directives for including other libraries and header files.

Here, when the **#include "complex.h"** directive is executed by the preprocessor it finds that COMPLEX_H macro is already defined and therefore ***ignores*** the header file.