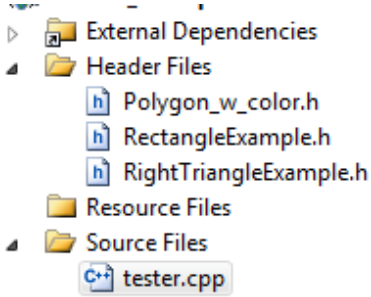
C/C++ Programming: C++ Projects




```
#include "RectangleExample.h"

// enumerator: finite set of choices for the colors
// blank and color_stop "bookend" the allowed colors
enum polygonColorOptions{blank=0, white, red, orange, yellow, green,
    light_blue, dark_blue, purple, color_stop};

class polygonColor{
private:
    polygonColorOptions color;
public:
    polygonColor(){color=blank;}
    void setColor(polygonColorOptions inp_color){ ... }
    polygonColorOptions getColor(){return color;}

    void inputColorFromKeyboard(){ ... }
    void printColorInfo(){ ... }

    void inputRandomColor(){ ... }
};

class rectangleWcolor{
public:
    rectangle the_rectangle;
    polygonColor the_color;
public:
    void inputFromKeyboard(){ ... }
    void printInfo(){ ... }
    void inputRandomValues(double max_val=100){ ... }
};
```

Course **EE3093** – Lecturer: Dr F. Verdicchio

Any question?

Relevant topics to ask questions:

- C++ Objects:
  - constructors;
  - composite objects;
  - arrays of objects.

Project

(Organized) Collection of:
Content you produce:

Content you supply (produced by others):

Instructions to the compiler:

Project

(Organized) Collection of:

Content you produce:

- Declaration (.h files) and definition (.h/.cpp files) of Objects
- Source (cpp) files implementing functions that use these objects
- One source file with a function called “main”

Content you supply (produced by others):

Instructions to the compiler:

Project

(Organized) Collection of:
Content you produce:
•Declaration (.h files) and definition (.h/.cpp files) of Objects
•Source (cpp) files implementing functions that use these objects
•One source file with a function called “main”

Content you supply (produced by others):
•Headers of system-specific or third party libraries (e.g. stdio)
•External libraries (.lib files)
•Pre-compiled headers

Instructions to the compiler:

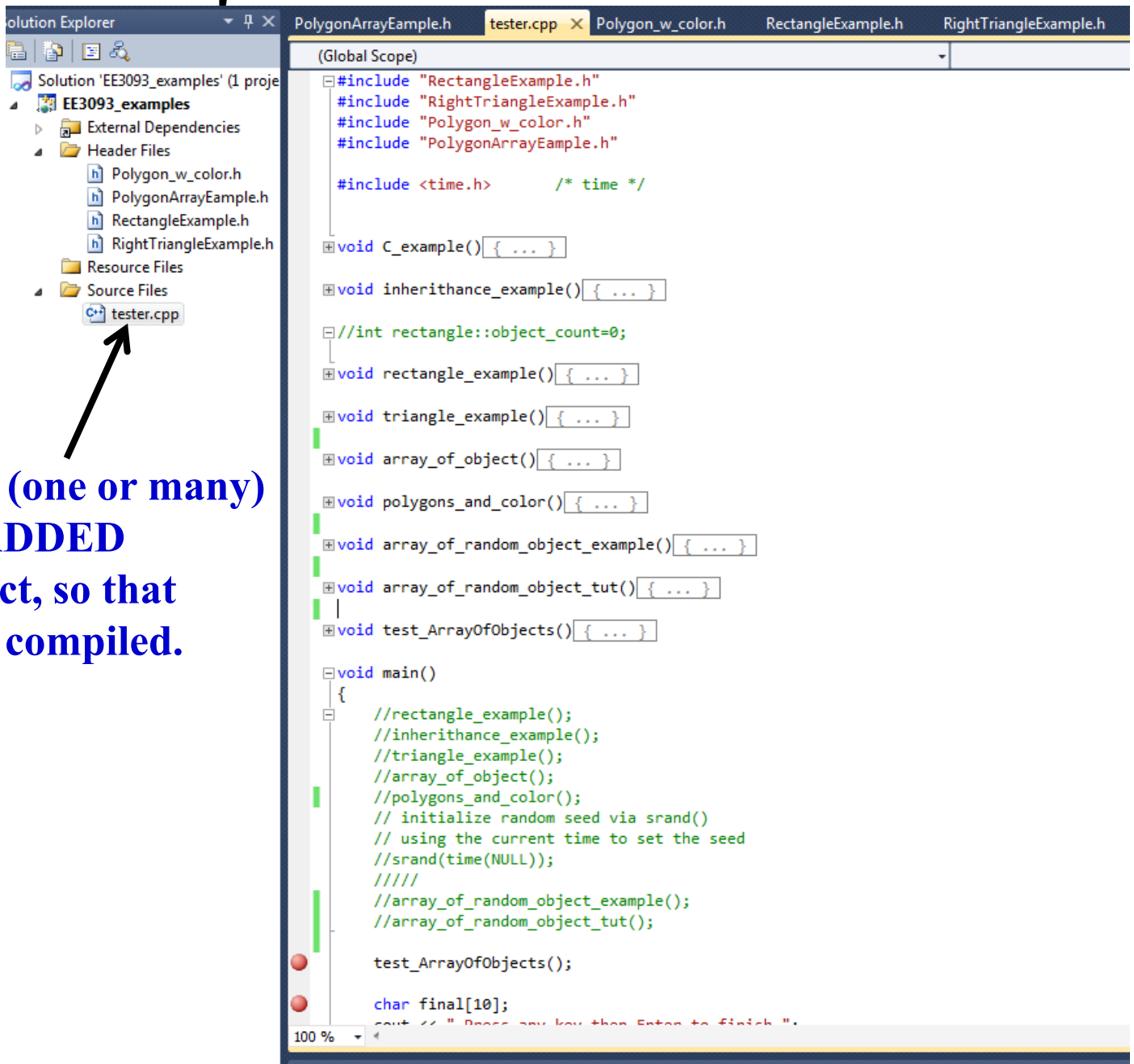
Project

(Organized) Collection of:
Content you produce:
•Declaration (.h files) and definition (.h/.cpp files) of Objects
•Source (cpp) files implementing functions that use these objects
•One source file with a function called “main”

Content you supply (produced by others):
•Headers of system-specific or third party libraries (e.g. stdio)
•External libraries (.lib files)
•Pre-compiled headers

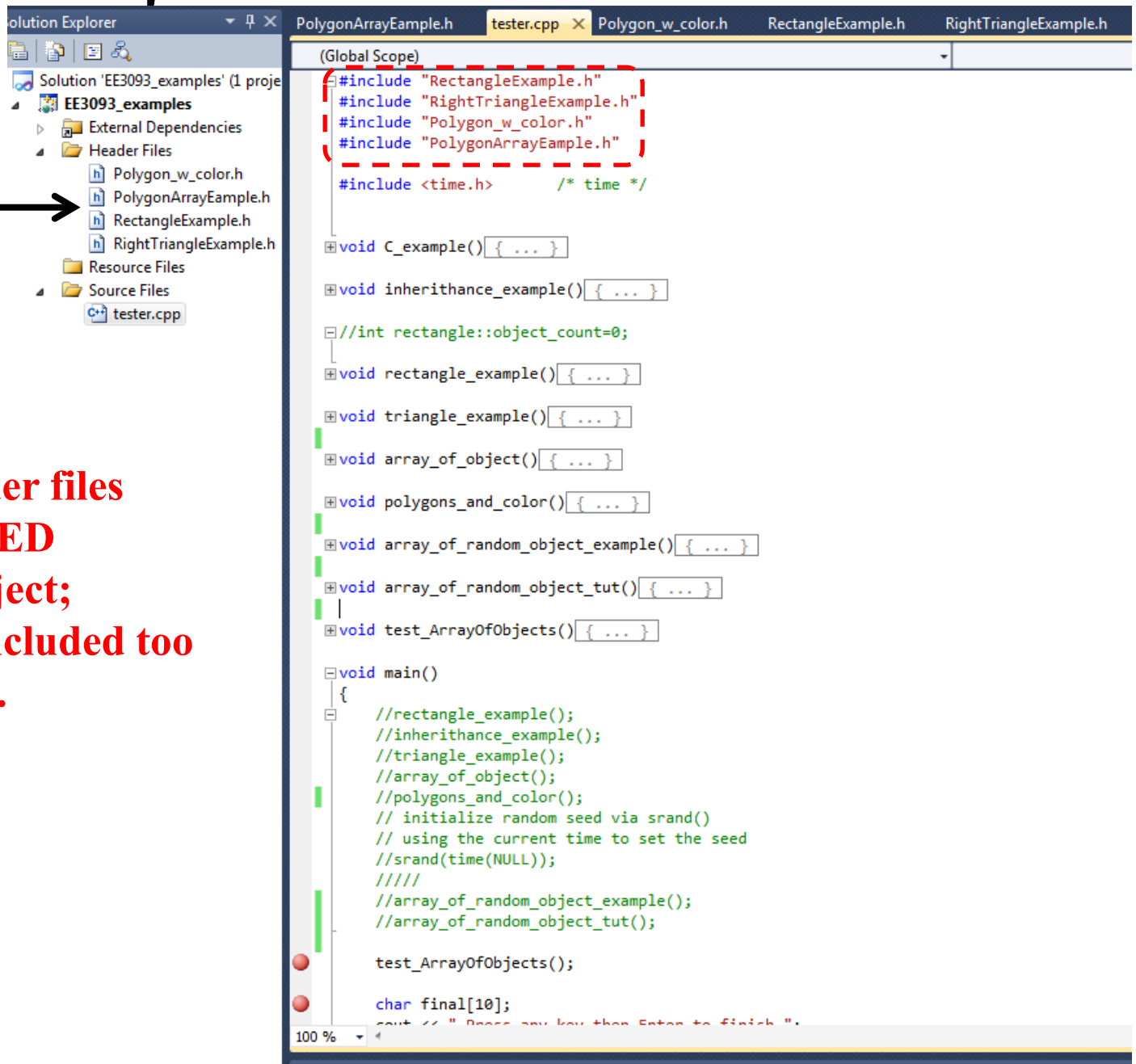
Instructions to the compiler:
•Source/ header files (or their named)
•Include directories (where files can be found)
•Compile options/setting (e.g. enable debug info, and much more)

Project: add source files



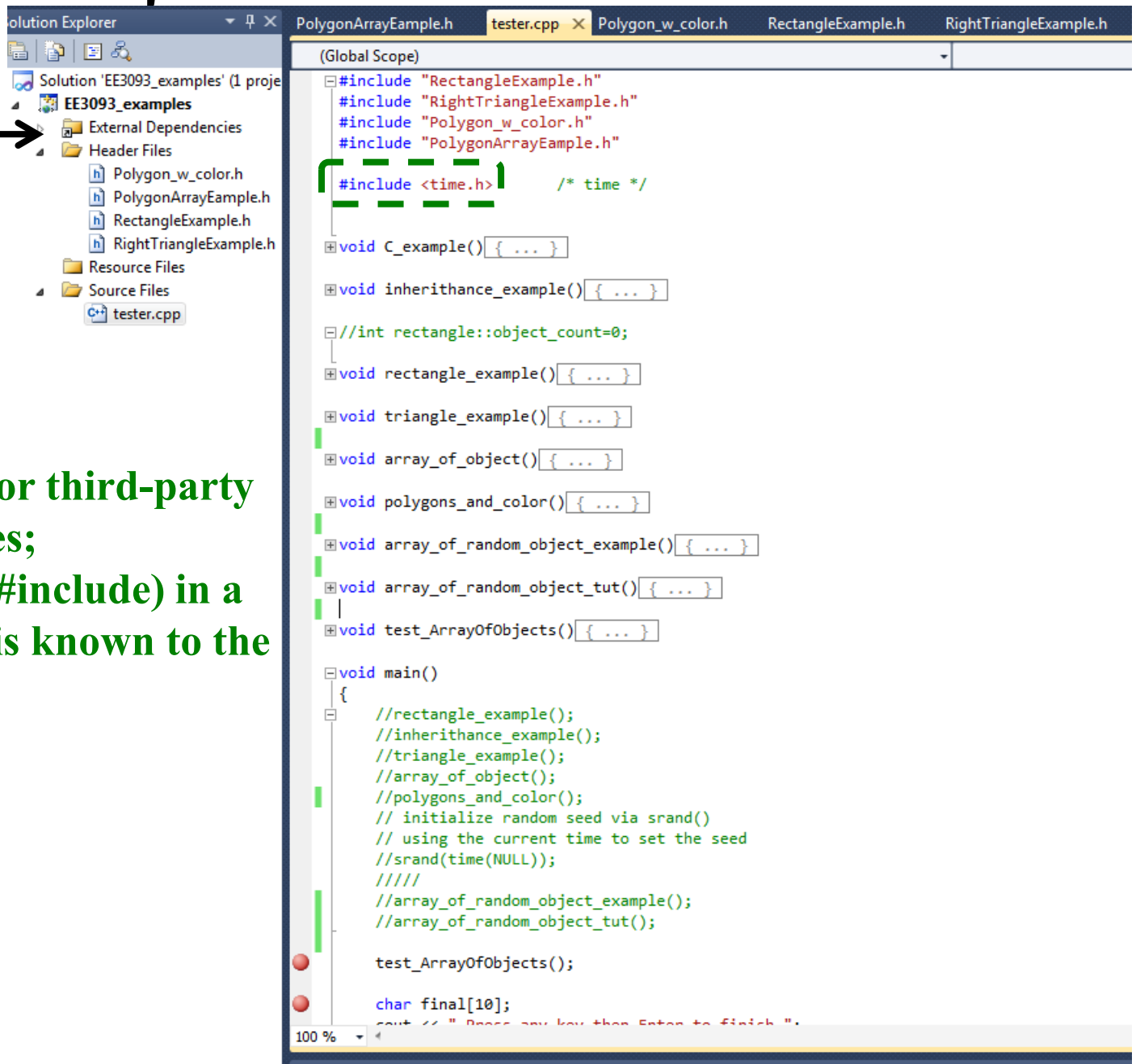
Source files (one or many)
These are ADDED
to the project, so that
they can be compiled.

Project: include header files



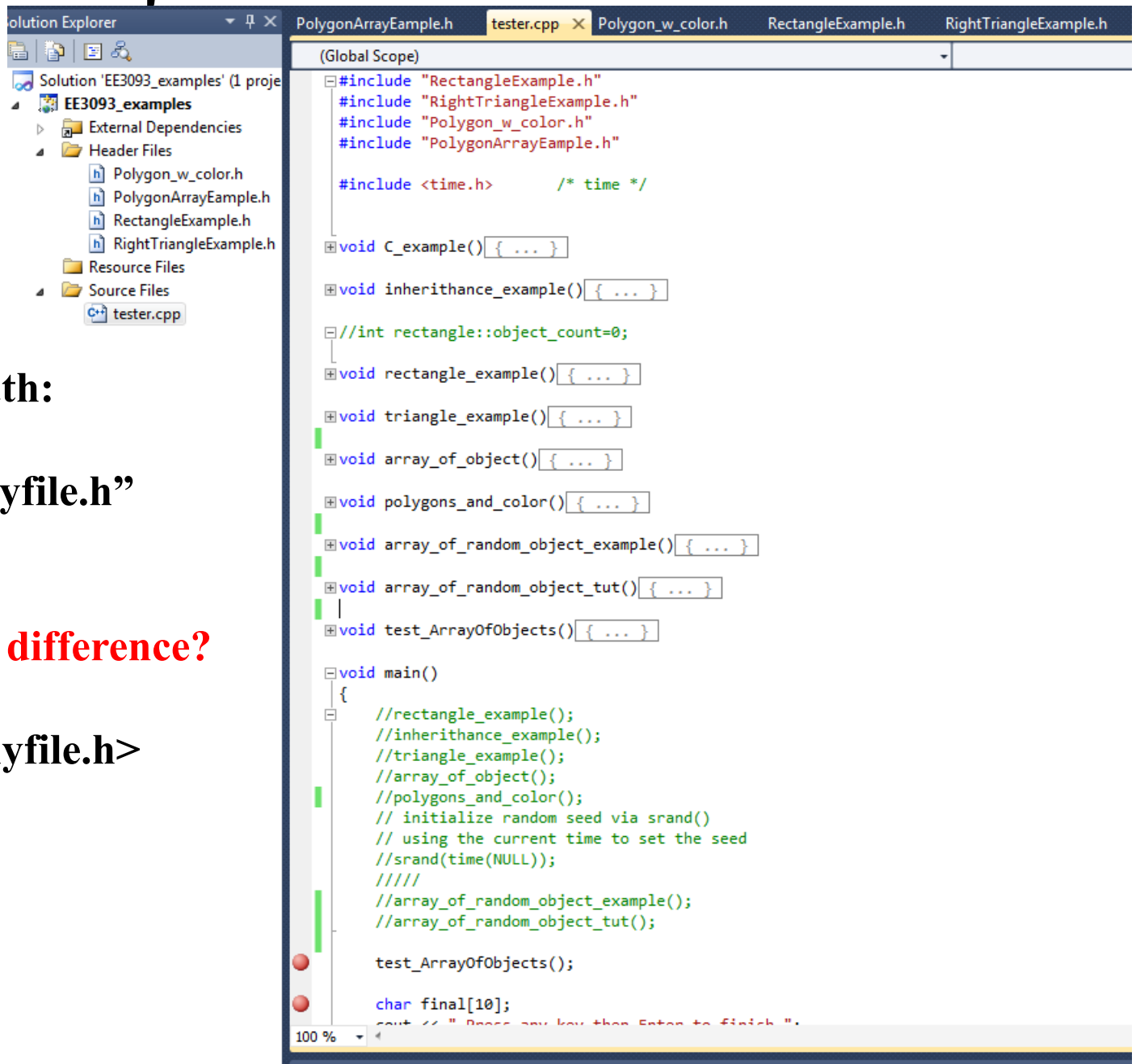
Your header files Also ADDED to the project; must be included too (#include).

Project: include header files



Standard or third-party header files; included (#include) in a path that is known to the compiler.

Project: include header files



Inclusion path:

#include "myfile.h"

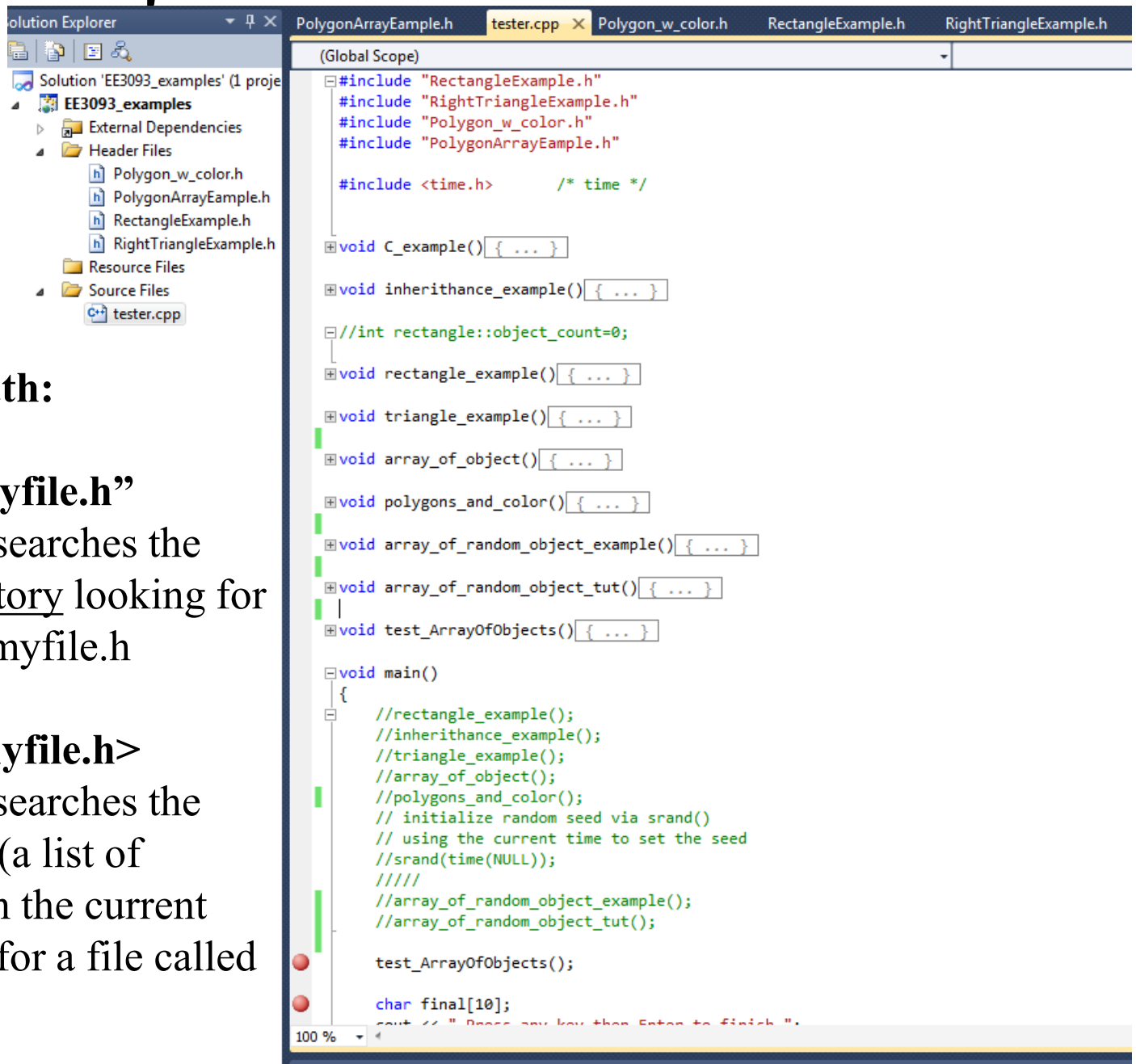
↑

What is the difference?

↓

#include <myfile.h>

Project: include header files



Inclusion path:

#include "myfile.h"
→ compiler searches the <u>current directory</u> looking for a file called myfile.h

#include <myfile.h>
→ compiler searches the <u>include path</u> (a list of directories on the current PC) looking for a file called myfile.h

Project: include header files

Solution Explorer

Solution 'EE3093_examples' (1 proje
- EE3093_examples
  - External Dependencies
  - Header Files
    - Polygon_w_color.h
    - PolygonArrayEample.h
    - RectangleExample.h
    - RightTriangleExample.h
  - Resource Files
  - Source Files
    - tester.cpp

PolygonArrayEample.h | tester.cpp | Polygon_w_color.h | RectangleExample.h

polygonColor

```
#ifndef PolygonWcolor_h
#define PolygonWcolor_h

#include "RectangleExample.h"

// enumerator: finite set of choices for the colors
// blank and color_stop "bookend" the allowed colors
enum polygonColorOptions{blank=0, white, red, orange, yellow, green,
    light_blue, dark_blue, purple, brown, black, color_stop};

class polygonColor{
private:
    polygonColorOptions color;
public:
```

Solution Explorer

Solution 'EE3093_examples' (1 proje
- EE3093_examples
  - External Dependencies
  - Header Files
    - Polygon_w_color.h
    - PolygonArrayEample.h
    - RectangleExample.h
    - RightTriangleExample.h
  - Resource Files
  - Source Files
    - tester.cpp

PolygonArrayEample.h | tester.cpp | Polygon_w_color.h | RectangleExample.h

rectangle

```
#ifndef RectangleExample_h
#define RectangleExample_h

#include <iostream>
using namespace std;

// rand, srand
#include <stdlib.h>

class rectangle{
protected:
    // static member variable used for count
    //static int object_count;
    //static int initialized_object_count;
```

Project: include header files

Solution Explorer

- Solution 'EE3093_examples' (1 proje
- EE3093_examples
  - External Dependencies
  - Header Files
    - Polygon_w_color.h
    - PolygonArrayEample.h
    - RectangleExample.h
    - RightTriangleExample.h
  - Resource Files
  - Source Files
    - tester.cpp

PolygonArrayEample.h | tester.cpp | Polygon_w_color.h | RectangleExample.h

polygonColor

```
#ifndef PolygonWcolor_h
#define PolygonWcolor_h

#include "RectangleExample.h"

// enumerator: finite set of choices for the colors
// blank and color_stop "bookend" the allowed colors
enum polygonColorOptions{blank=0, white, red, orange, yellow, green,
    light_blue, dark_blue, purple, brown, black, color_stop};

class polygonColor{
private:
    polygonColorOptions color;
public:
```

Solution Explorer

- Solution 'EE3093_examples' (1 proje
- EE3093_examples
  - External Dependencies
  - Header Files
    - Polygon_w_color.h
    - PolygonArrayEample.h
    - RectangleExample.h
    - RightTriangleExample.h
  - Resource Files
  - Source Files
    - tester.cpp

PolygonArrayEample.h | tester.cpp | Polygon_w_color.h | RectangleExample.h

rectangle

```
#ifndef RectangleExample_h
#define RectangleExample_h

#include <iostream>
using namespace std;

// rand, srand
#include <stdlib.h>

class rectangle{
protected:
    // static member variable used for count
    //static int object_count;
    //static int initialized_object_count;
```

Project: use of namespaces

iostream is a library file (the extension .h has been dropped for some C++ libraries):
Many useful objects are declared and created within this library, for example *cout, cin*.
These objects and relative functions have global scope (can be accessed from a function/file different form the one where these are instantiated)

Project: use of namespaces

iostream is a library file (the extension .h has been dropped for some C++ libraries): Many useful objects are declared and created within this library, for example *cout, cin*. These objects and relative functions have global **scope** (can be accessed from a function/file different form the one where these are instantiated)

```
int globalscope_variable;

void C_example()
{
    double test_variable;
    test_variable = 10.2;
    cout << "test_variable value is " << test_variable << endl;
    globalscope_variable=10;
}

void main()
{
    double test_variable;
    cout << "test_variable value is competely different here" << test_variable << endl;
    cout << "globalscope_variable" << globalscope_variable << endl;
}
```

This has global scope

The scope of test_variable is limited to within function C_example

The scope of test_variable is limited to within function main

Project: use of namespaces

iostream is a library file (the extension .h has been dropped for some C++ libraries): Many useful objects are declared and created within this library, for example *cout, cin*. These objects and relative functions have global **scope** (can be accessed from a function/file different form the one where these are instantiated), but are defined within a "namespace" called "std"; to access them, we should use their complete identifier (name), including **namespace** and then the **name of the variable**; for example **std::cout.**

Project: use of namespaces

iostream is a library file (the extension .h has been dropped for some C++ libraries): Many useful objects are declared and created within this library, for example *cout, cin*. These objects and relative functions have global scope (can be accessed from a function/file different form the one where these are instantiated), but are defined within a "namespace" called "std"; to access them, we should use their complete identifier (name), including **namespace** and then the **name of the variable**; for example **std::cout.**

This allows the user to define its own variable called "cout" and still be able to use **std::cout** without confusion.

Where there is no risk of confusion, one can adopt a namespace (using namespace std) and refer to the variables in the namespace simply by their name, e.g. **cout**.

Any question?

Project: define/declare your objects

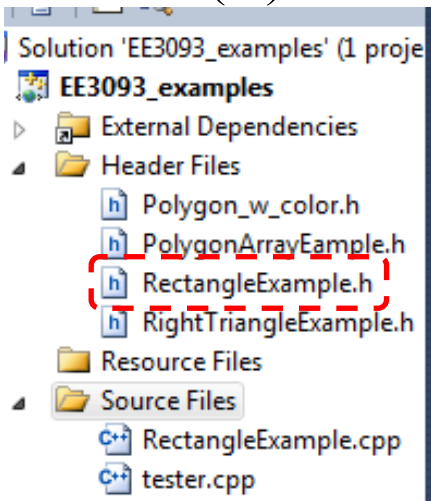
You contribute a class to a project by including:

- A header (.h) file with the **declaration** of the class and its functions:
  - (Name and) **type** of all public/protected/private member variables (example: *double sideA, sideB, area, perimeter*); Knowing this allows the compiler to calculate the size (memory space) for an instantiation of the class.
  - (Name and) **type** of all input/output parameters for public/protected/private member functions (example: *double getSide(int sidenum)*); Knowing this allows the compiler to verify that function calls for an instantiation of the class follow the expected syntax (example: *myobject.getSide(2, a)* is incorrect).
- A source file (cpp) with the **definition** of class functions:
  - Implementation of each public/protected/private member function.
  - Instantiation of any static member variable of the class.
  - Note: remember to include the header file; you can compile the source file to check all is OK.

Project: define/declare your objects

You contribute a class to a project by including:

- A header (.h) file with the **declaration** of the class and its functions:

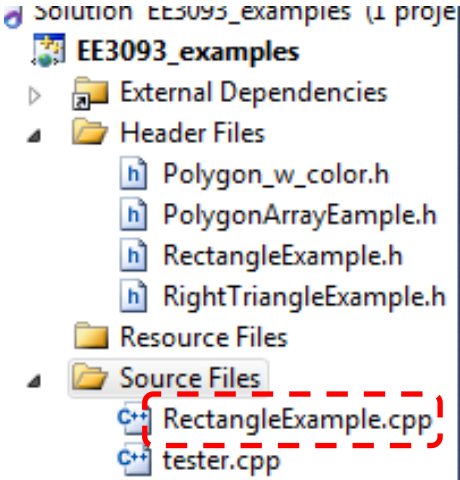



```
class rectangle{
protected:
    // static member variable used for count
    //static int object_count;
    //static int initialized_object_count;
    // variables
    double sideA, sideB;
    double area;
    double perimeter;
    // initialization flag
    bool init_flag;
    // functions
    void computeArea() { ... }
    void computePerimeter() { ... }
    void set_init_flag(bool setval) { ... }
public:
    // constructor
    rectangle();
    // copy constructor
    rectangle(const rectangle& source);
    // destructor
    ~rectangle();
    double getArea();
    double getPerimeter();
    bool isInitialized();
    double getSide(int sidenum);
    void inputSides(double in_sideA, double in_sideB);
    void inputSidesFromKeyboard();
    void printRectangleInfo();
    void inputRandomSides(double max_val=100);
    void resetRectangle();
```

Project: define/declare your objects

You contribute a class to a project by including:

- A source file (cpp) with the **definition** of class functions :

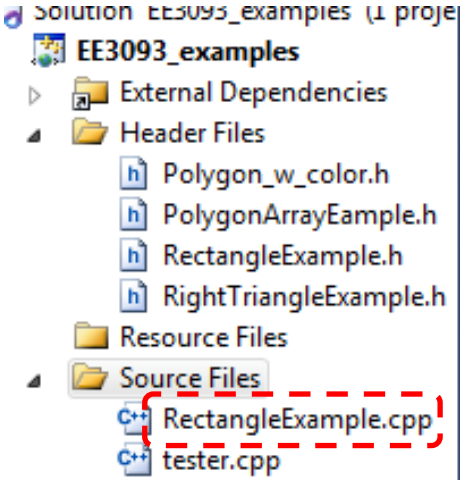



```cpp
#include "RectangleExample.h"

double rectangle::getArea()
{
    if(isInitialized())
        return area;
    else
        cout << "Error in getArea(): Rectangle is not initialized " << endl;
    return 0;
}

double rectangle::getPerimeter()
{
    if(isInitialized())
        return perimeter;
    else
        cout << "Error in getPerimeter(): Rectangle is not initialized " << endl;
    return 0;
}

double rectangle::getPerimeter()
{
    if(isInitialized())
        return perimeter;
    else
        cout << "Error in getPerimeter(): Rectangle is not initialized " << endl;
    return 0;
}
bool rectangle::isInitialized(){return init_flag;}
```

Project: define/declare your objects

You contribute a class to a project by including:

- A source file (cpp) with the **definition** of class functions :




```
#include "RectangleExample.h"

double rectangle::getArea()
{
    if(isInitialized())
        return area;
    else
        cout << "Error in getArea(): Rectangle is not initialized " << endl;
    return 0;
}

double rectangle::getPerimeter()
{
    if(isInitialized())
        return perimeter;
    else
        cout << "Error in getPerimeter(): Rectangle is not initialized " << endl;
    return 0;
}

double rectangle::getPerimeter()
{
    if(isInitialized())
        return perimeter;
    else
        cout << "Error in getPerimeter(): Rectangle is not initialized " << endl;
    return 0;
}
bool rectangle::isInitialized(){return init_flag;}
```

Scope Resolution, i.e. operator :: (required for member function definition)

Project: define/declare your objects

You contribute a class to a project by including:

- A header (.h) file with the **declaration** of the class and its functions:
  - (Name and) **type** of all public/protected/private member variables (example: *double sideA, sideB, area, perimeter*); Knowing this allows the compiler to calculate the size (memory space) for an instantiation of the class.
  - (Name and) **type** of all input/output parameters for public/protected/private member functions (example: *double getSide(int sidenum)*); Knowing this allows the compiler to verify that function calls for an instantiation of the class follow the expected syntax (example: *myobject.getSide(2, a)* is incorrect).
- A source file (cpp) with the **definition** of class functions:
  - Implementation of each public/protected/private member function.
  - Instantiation of any static member variable of the class.
  - Note: remember to include the header file; you can compile the source file to check all is OK.

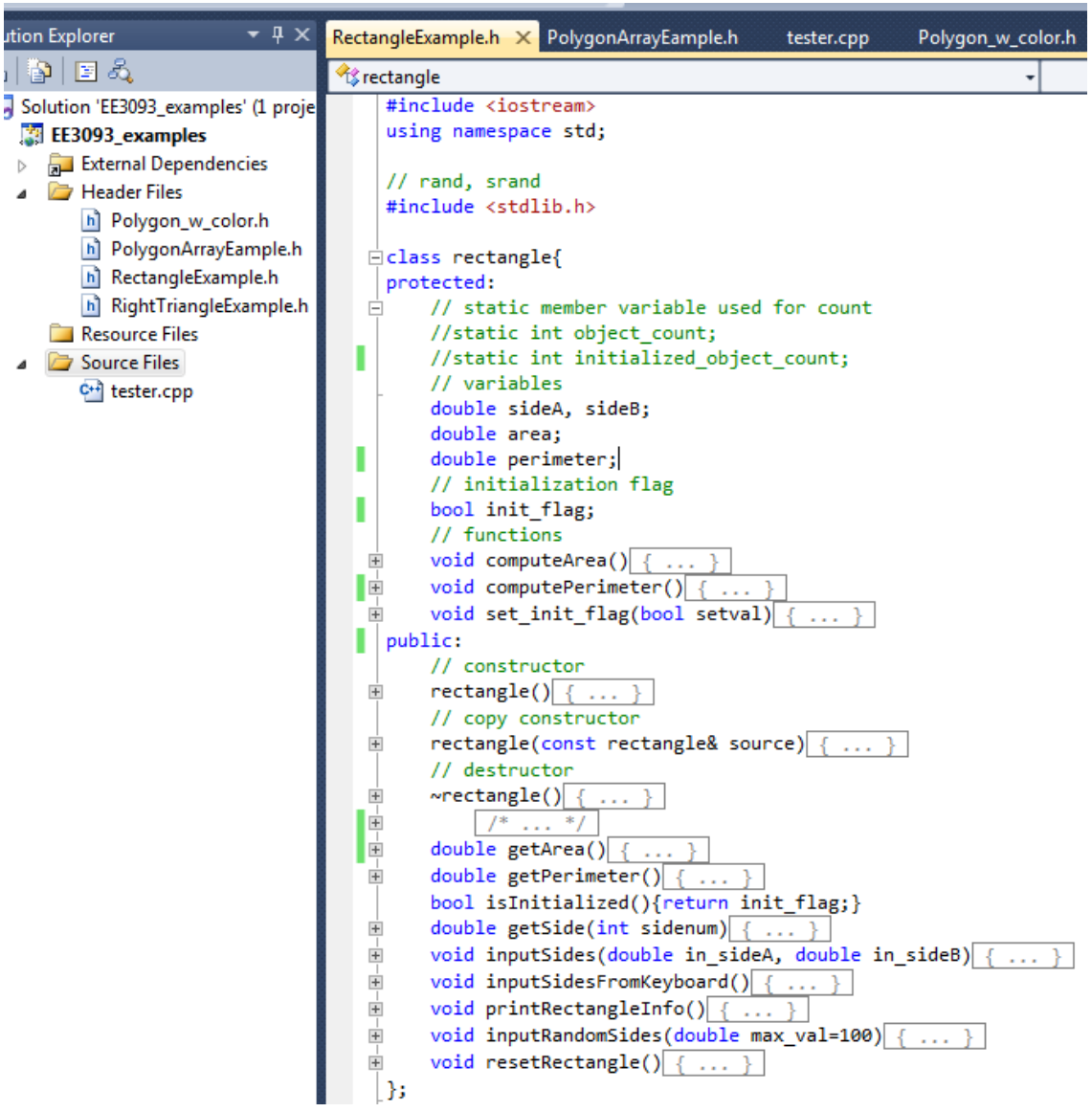
Alternatively:

- A single header file where the class is both declared and defined.
  - Note: most compilers do not attempt to compile a header file on its own: the header file must be included by a source file (e.g. main.cpp; possibly declaring an instance of the class).

Project: define/declare your objects

You contribute a class to a project by including:

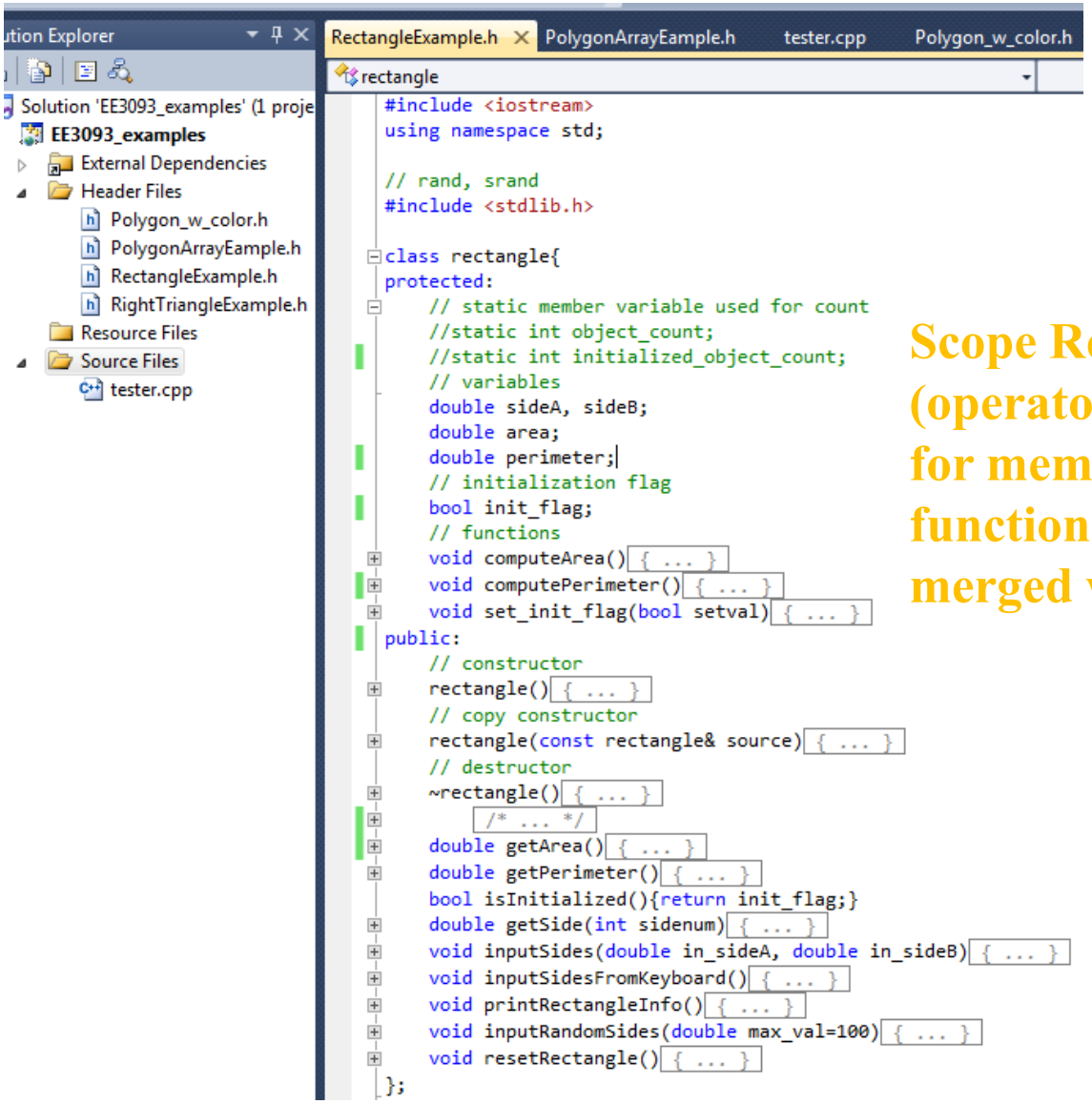
- A single header file where the class is both declared and defined.



Project: define/declare your objects

You contribute a class to a project by including:

- A single header file where the class is both declared and defined.

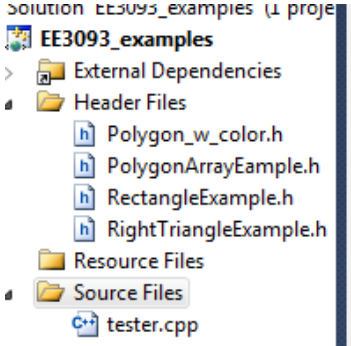


Scope Resolution, (operator ::) not required for member function declaration (here merged with definition)

Project: define/declare your objects

Avoid **problems** (redefinition) with **multiple inclusions** of the same **header** file:

- One header file may be included by multiple (source or header) files,




```
#include "RectangleExample.h"

// enumerator: finite set of choices for the colors
// blank and color_stop "bookend" the allowed colors
enum polygonColorOptions{blank=0, white, red, orange, yellow, green,
    light_blue, dark_blue, purple, brown, black, color_stop};

class polygonColor{
private:
    polygonColorOptions color;
public:
    polygonColor(){color=blank;}
    void setColor(polygonColorOptions inp_color){ ... }
    polygonColorOptions getColor(){return color;}

    void inputColorFromKeyboard(){ ... }
    void printColorInfo(){ ... }
```

100 %

RightTriangleExample.h

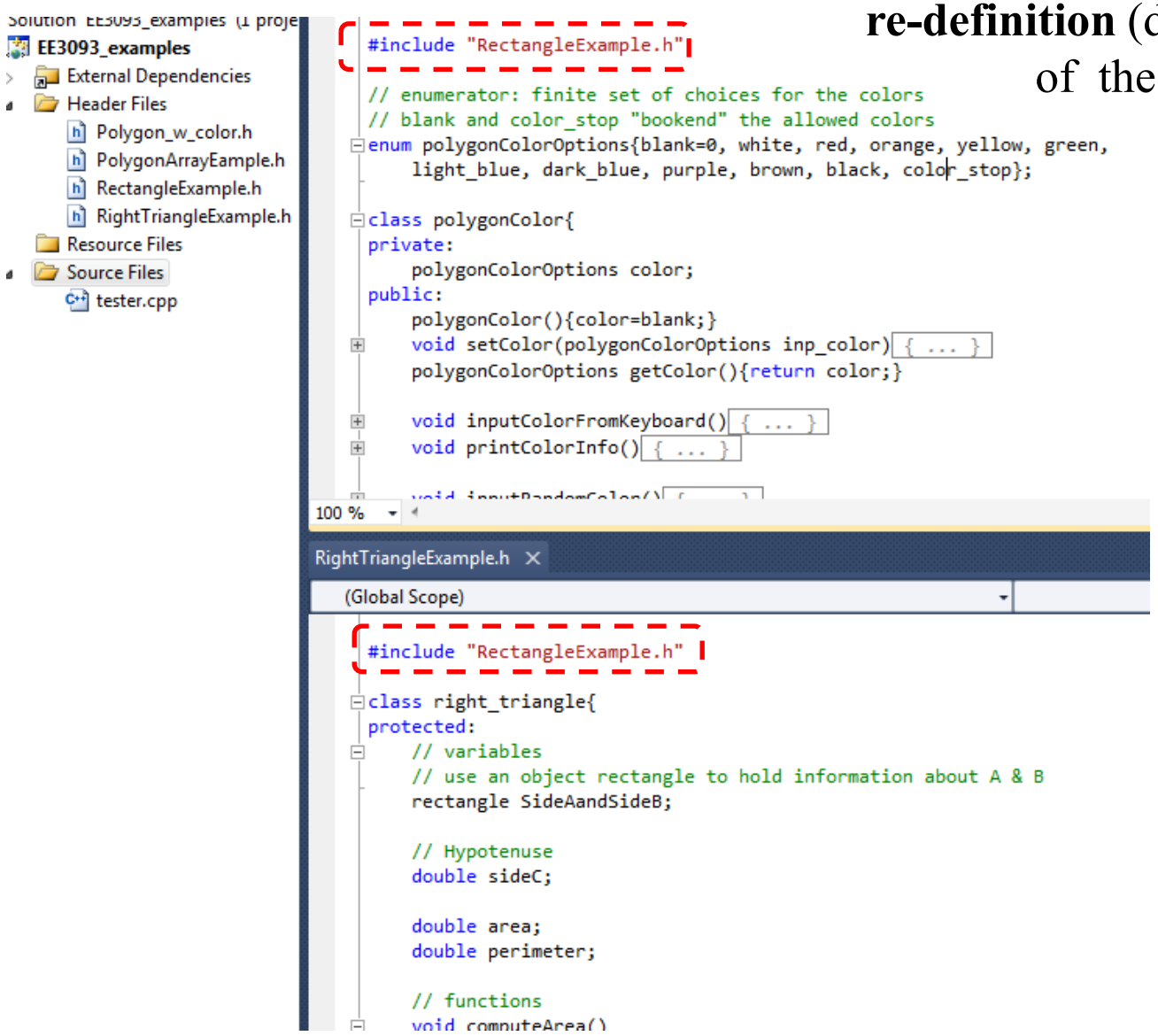
(Global Scope)

```
#include "RectangleExample.h"

class right_triangle{
protected:
    // variables
    // use an object rectangle to hold information about A & B
    rectangle SideAandSideB;

    // Hypotenuse
    double sideC;

    double area;
    double perimeter;

    // functions
    void computeArea()
```

Project: define/declare your objects

Avoid **problems** (redefinition) with **multiple inclusions** of the same **header** file:

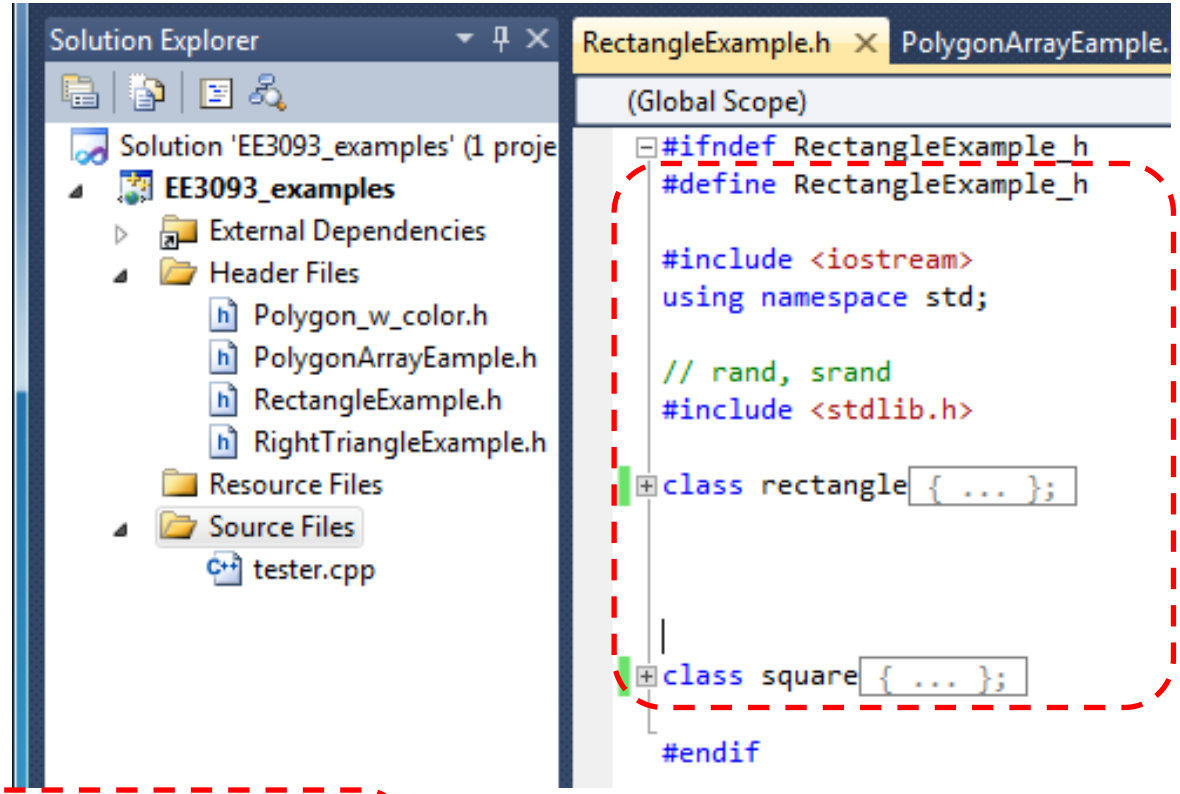
- One header file may be included by multiple (source or header) files, causing **re-definition** (duplicate definition) of the **header content**.



Project: define/declare your objects

Avoid **problems** (redefinition) with **multiple inclusions** of the same **header** file:

- *#define, #ifdef ,#ifndef ,#endif* : pre-processor macros performed before compilation.



- The file content is bookended by the ***#ifndef symbol*** and ***#endif*** macros; ***ifndef*** stands for "if not defined".
  The compiler only reads past the line with *#ifndef symbol* if *symbol* is not defined. Define *symbol* on the following line (and nowhere else) and the file is read by the compiler **only once** (at the first instance of inclusion, ignoring following ones).

Any question?