

C/C++ Programming: C++ Projects



The screenshot shows a C++ IDE with a project structure on the left and a source file on the right. The project structure includes:

- External Dependencies
- Header Files
 - Polygon_w_color.h
 - RectangleExample.h
 - RightTriangleExample.h
- Resource Files
- Source Files
 - tester.cpp

The source file `tester.cpp` contains the following code:

```
#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){ ... }
};
```

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”

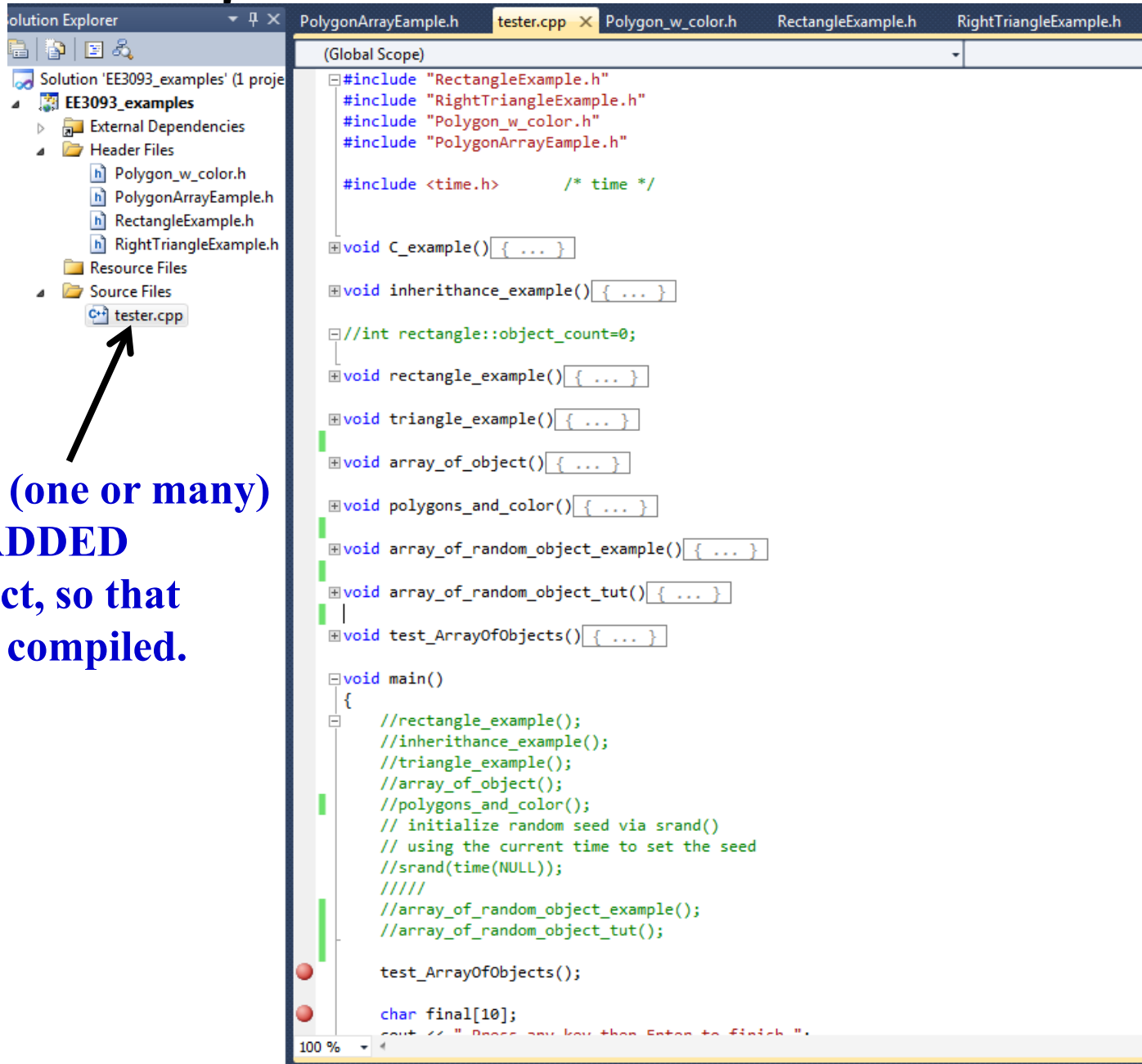
Content you supply (produced by others):

- Headers of system-specific or third party libraries (e.g. stdio)
- External libraries (.lib files)
- Pre-compiled headers

Instructions to the compiler:

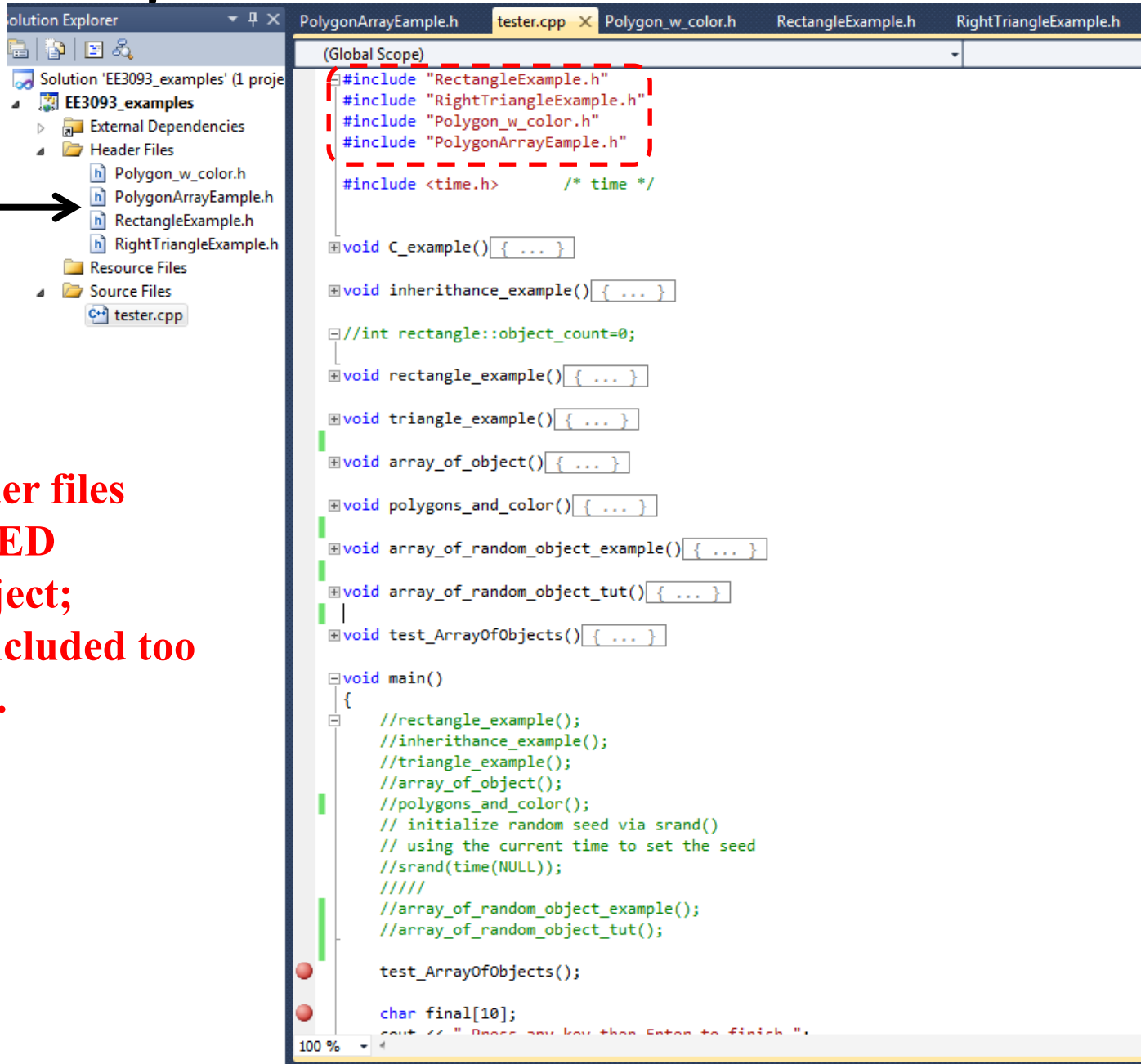
- Source/ header files (or their named)
- Include directories (where files can be found)
- Compile options/setting (e.g. enable debug info, and much more)

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



The screenshot shows the Visual Studio IDE. On the left, the Solution Explorer displays a project named 'EE3093_examples'. Under the 'Header Files' folder, four header files are listed: 'Polygon_w_color.h', 'PolygonArrayExample.h', 'RectangleExample.h', and 'RightTriangleExample.h'. A black arrow points from the text 'Your header files Also ADDED to the project; must be included too (#include).' to the 'Header Files' folder. The main editor window shows the 'tester.cpp' file. The code includes four header files: 'RectangleExample.h', 'RightTriangleExample.h', 'Polygon_w_color.h', and 'PolygonArrayExample.h'. These include statements are grouped together and enclosed in a red dashed box. The code also includes '<time.h>' for timing. The 'main()' function calls several example functions: 'rectangle_example()', 'inherithance_example()', 'triangle_example()', 'array_of_object()', 'polygons_and_color()', 'array_of_random_object_example()', 'array_of_random_object_tut()', and 'test_ArrayOfObjects()'. The code is formatted with syntax highlighting and line numbers.

```
(Global Scope)
#include "RectangleExample.h"
#include "RightTriangleExample.h"
#include "Polygon_w_color.h"
#include "PolygonArrayExample.h"

#include <time.h> /* time */

void C_example() { ... }

void inherithance_example() { ... }

//int rectangle::object_count=0;
void rectangle_example() { ... }

void triangle_example() { ... }

void array_of_object() { ... }

void polygons_and_color() { ... }

void array_of_random_object_example() { ... }

void array_of_random_object_tut() { ... }

void test_ArrayOfObjects() { ... }

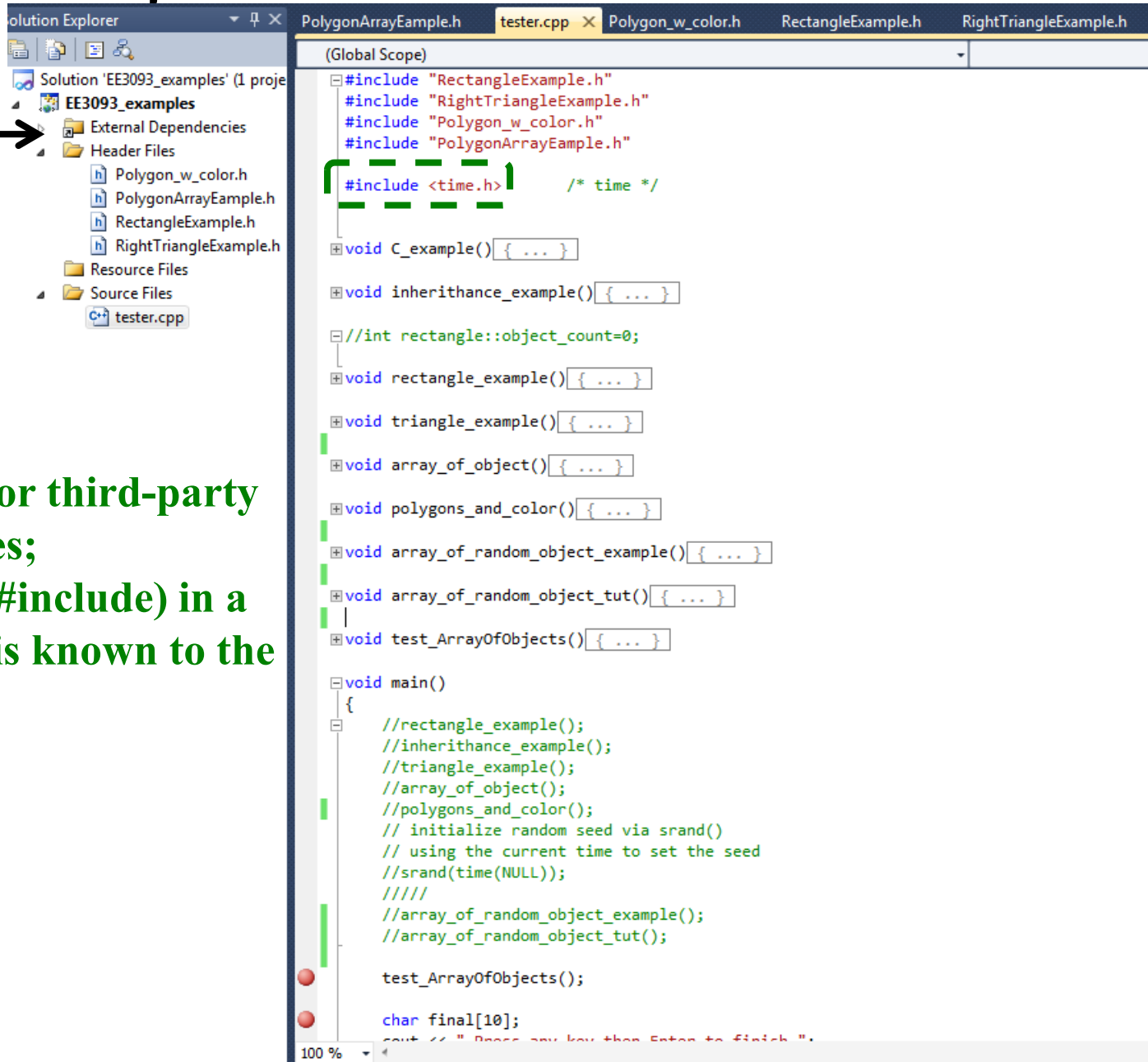
void main()
{
    //rectangle_example();
    //inherithance_example();
    //triangle_example();
    //array_of_object();
    //polygons_and_color();
    // initialize random seed via srand()
    // using the current time to set the seed
    //srand(time(NULL));
    //
    //array_of_random_object_example();
    //array_of_random_object_tut();

    test_ArrayOfObjects();

    char final[10];
    cout << "Press any key then Enter to finish ";
```

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

Project: include header files



The screenshot shows the Visual Studio IDE. On the left, the Solution Explorer displays a project named 'EE3093_examples' with a folder 'Header Files' containing several .h files: Polygon_w_color.h, PolygonArrayExample.h, RectangleExample.h, and RightTriangleExample.h. A black arrow points from the text on the left to this 'Header Files' folder. The main editor window shows the code for 'tester.cpp'. The code includes several headers: 'RectangleExample.h', 'RightTriangleExample.h', 'Polygon_w_color.h', and 'PolygonArrayExample.h'. A line of code is highlighted with a green dashed box: `#include <time.h> /* time */`. The code also defines several functions: `C_example()`, `inheritance_example()`, `rectangle_example()`, `triangle_example()`, `array_of_object()`, `polygons_and_color()`, `array_of_random_object_example()`, `array_of_random_object_tut()`, `test_ArrayOfObjects()`, and `main()`. The `main()` function calls several of these functions and uses `srand(time(NULL))` to initialize a random seed.

```
(Global Scope)
#include "RectangleExample.h"
#include "RightTriangleExample.h"
#include "Polygon_w_color.h"
#include "PolygonArrayExample.h"
#include <time.h> /* time */

void C_example() { ... }

void inheritance_example() { ... }

//int rectangle::object_count=0;
void rectangle_example() { ... }

void triangle_example() { ... }

void array_of_object() { ... }

void polygons_and_color() { ... }

void array_of_random_object_example() { ... }

void array_of_random_object_tut() { ... }

void test_ArrayOfObjects() { ... }

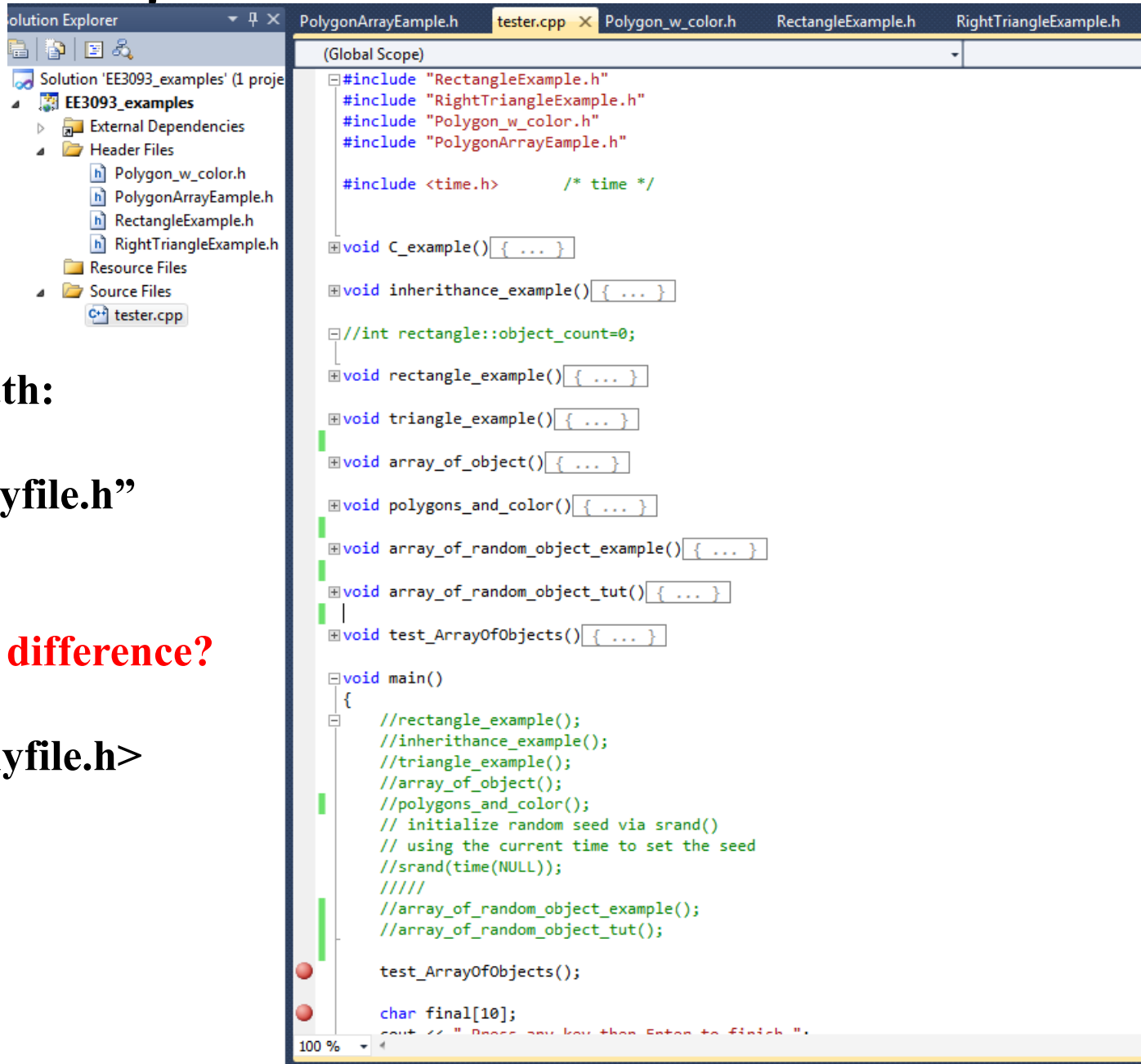
void main()
{
    //rectangle_example();
    //inheritance_example();
    //triangle_example();
    //array_of_object();
    //polygons_and_color();
    // initialize random seed via srand()
    // using the current time to set the seed
    //srand(time(NULL));
    ////
    //array_of_random_object_example();
    //array_of_random_object_tut();

    test_ArrayOfObjects();

    char final[10];
    cout << "Press any key then Enter to finish ";
```

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

Project: include header files



```
(Global Scope)
#include "RectangleExample.h"
#include "RightTriangleExample.h"
#include "Polygon_w_color.h"
#include "PolygonArrayEample.h"

#include <time.h> /* time */

void C_example() { ... }

void inherithance_example() { ... }

//int rectangle::object_count=0;
void rectangle_example() { ... }

void triangle_example() { ... }

void array_of_object() { ... }

void polygons_and_color() { ... }

void array_of_random_object_example() { ... }

void array_of_random_object_tut() { ... }

void test_ArrayOfObjects() { ... }

void main()
{
    //rectangle_example();
    //inherithance_example();
    //triangle_example();
    //array_of_object();
    //polygons_and_color();
    // initialize random seed via srand()
    // using the current time to set the seed
    //srand(time(NULL));
    ////
    //array_of_random_object_example();
    //array_of_random_object_tut();

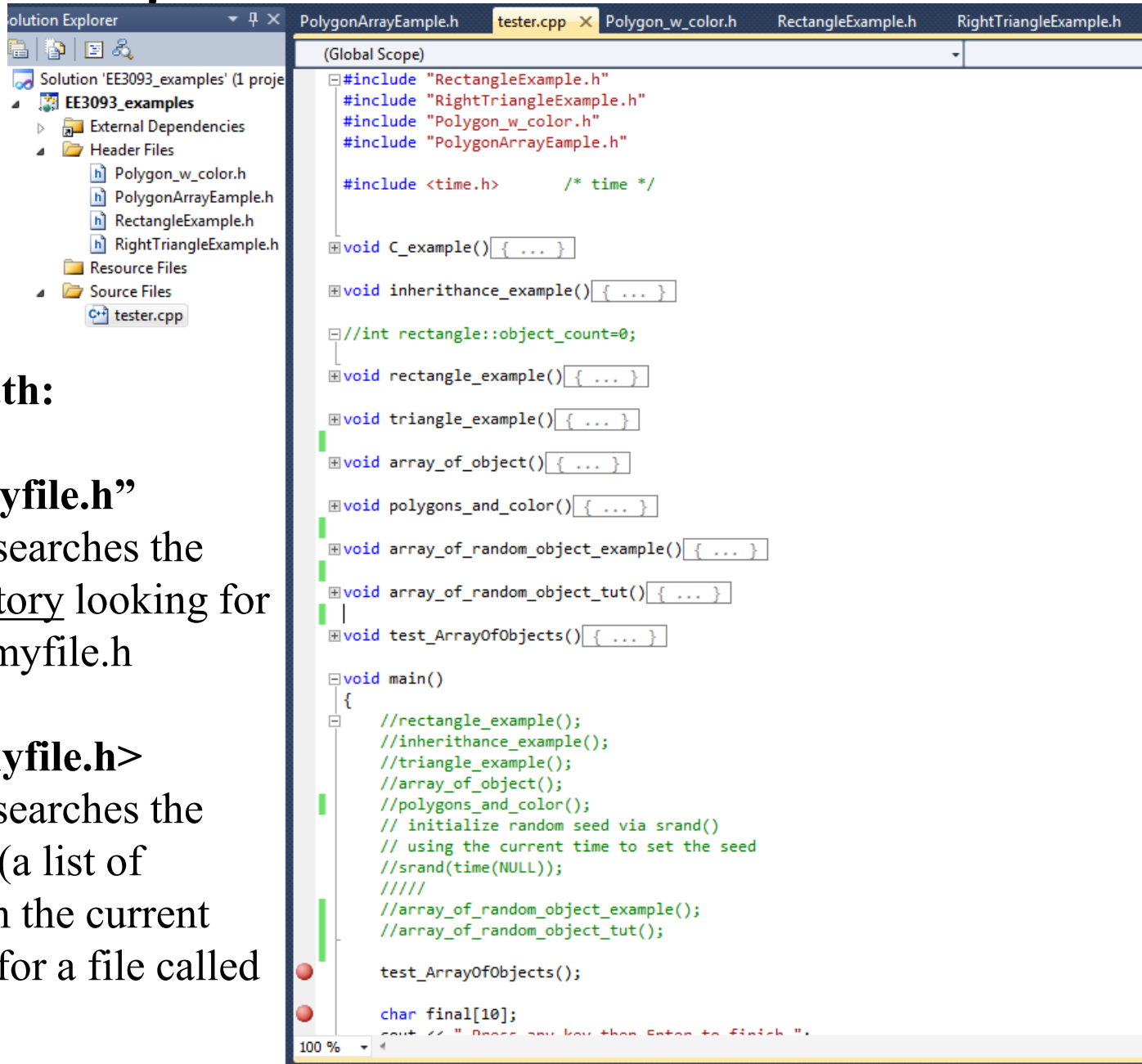
    test_ArrayOfObjects();

    char final[10];
    cout << "Press any key then Enter to finish ";
```

Inclusion path:
`#include "myfile.h"`

↑
What is the difference?
↓
`#include <myfile.h>`

Project: include header files



Inclusion path:

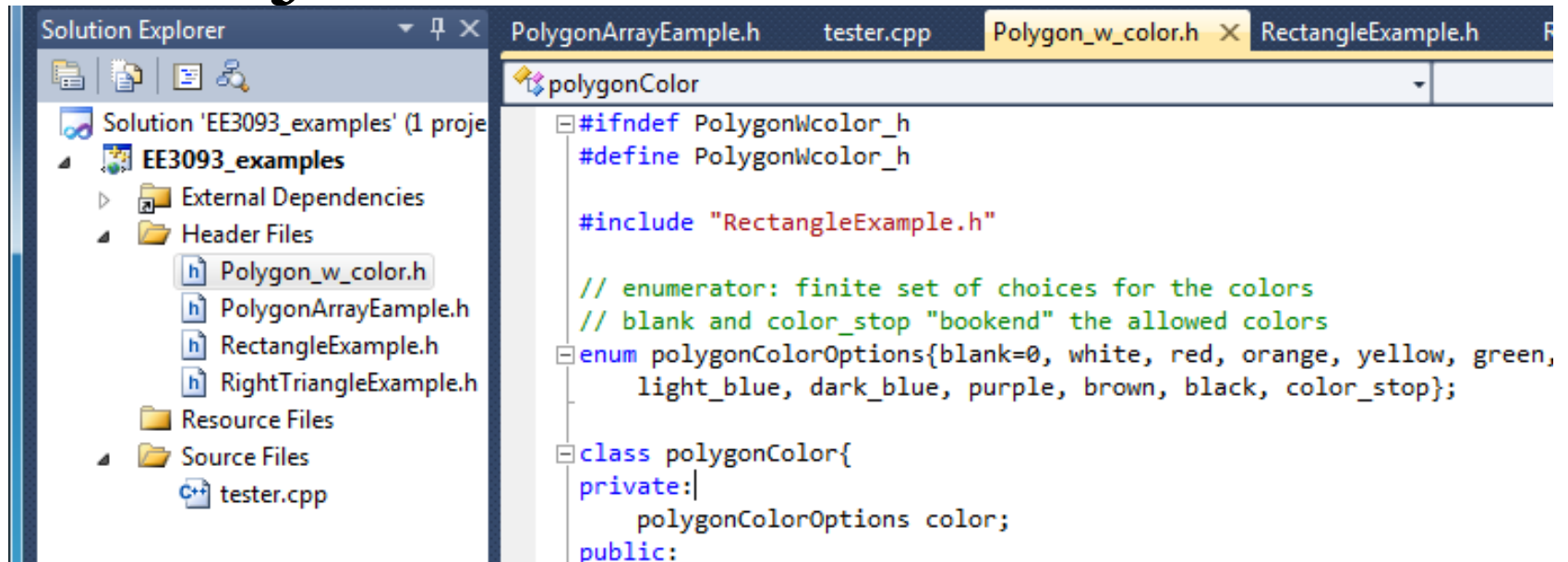
#include "myfile.h"

→ compiler searches the current directory looking for a file called myfile.h

#include <myfile.h>

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

Project: include header files



This screenshot shows the Visual Studio IDE with the 'EE3093_examples' project open. The Solution Explorer on the left shows the project structure, including 'Header Files' and 'Source Files'. The 'Header Files' folder is expanded, showing 'Polygon_w_color.h' selected. The main editor window displays the contents of 'Polygon_w_color.h'. The code defines a 'polygonColor' class with a private 'polygonColorOptions' member and a public interface. It includes 'RectangleExample.h' and defines an enumeration for color options.

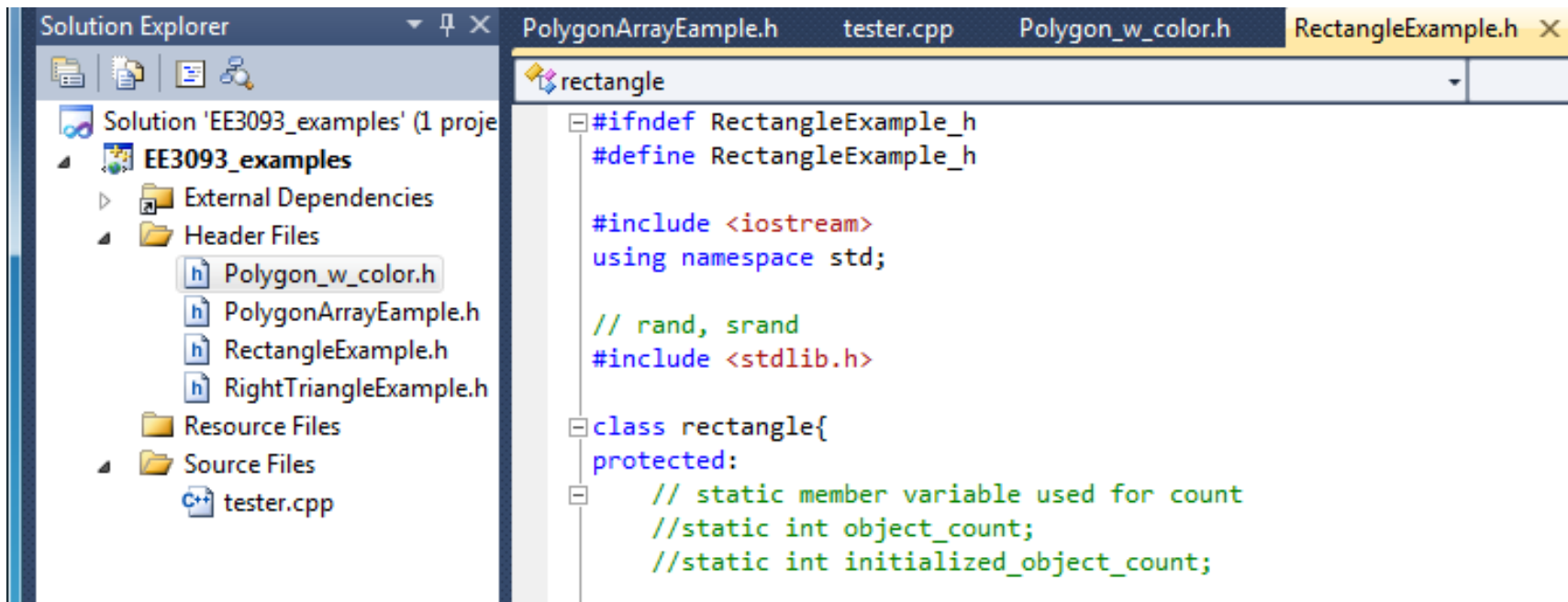
```
Solution Explorer
Solution 'EE3093_examples' (1 proje
  EE3093_examples
    External Dependencies
    Header Files
      Polygon_w_color.h
      PolygonArrayEample.h
      RectangleExample.h
      RightTriangleExample.h
    Resource Files
    Source Files
      tester.cpp

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:
```



This screenshot shows the Visual Studio IDE with the 'EE3093_examples' project open. The Solution Explorer on the left shows the project structure, including 'Header Files' and 'Source Files'. The 'Header Files' folder is expanded, showing 'Polygon_w_color.h' selected. The main editor window displays the contents of 'RectangleExample.h'. The code defines a 'rectangle' class with a protected 'object_count' member and a public interface. It includes 'iostream' and 'stdlib.h' and defines a static member variable 'object_count'.

```
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

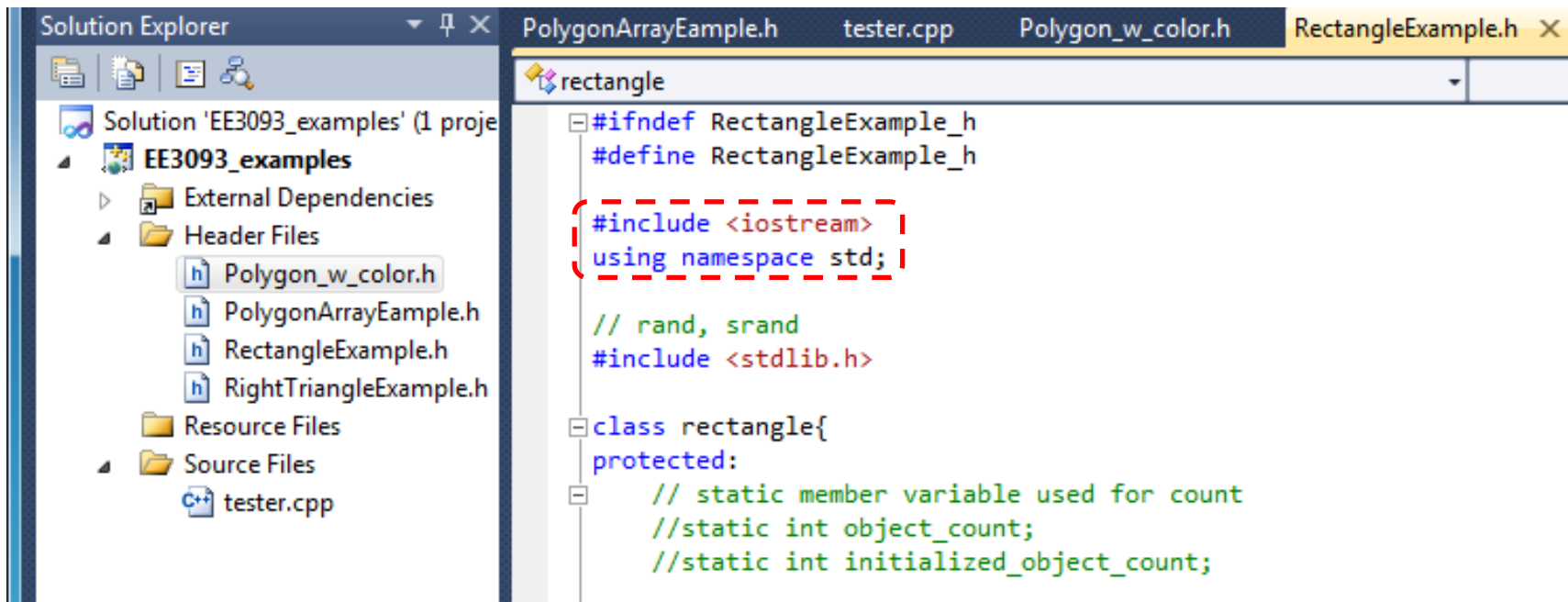
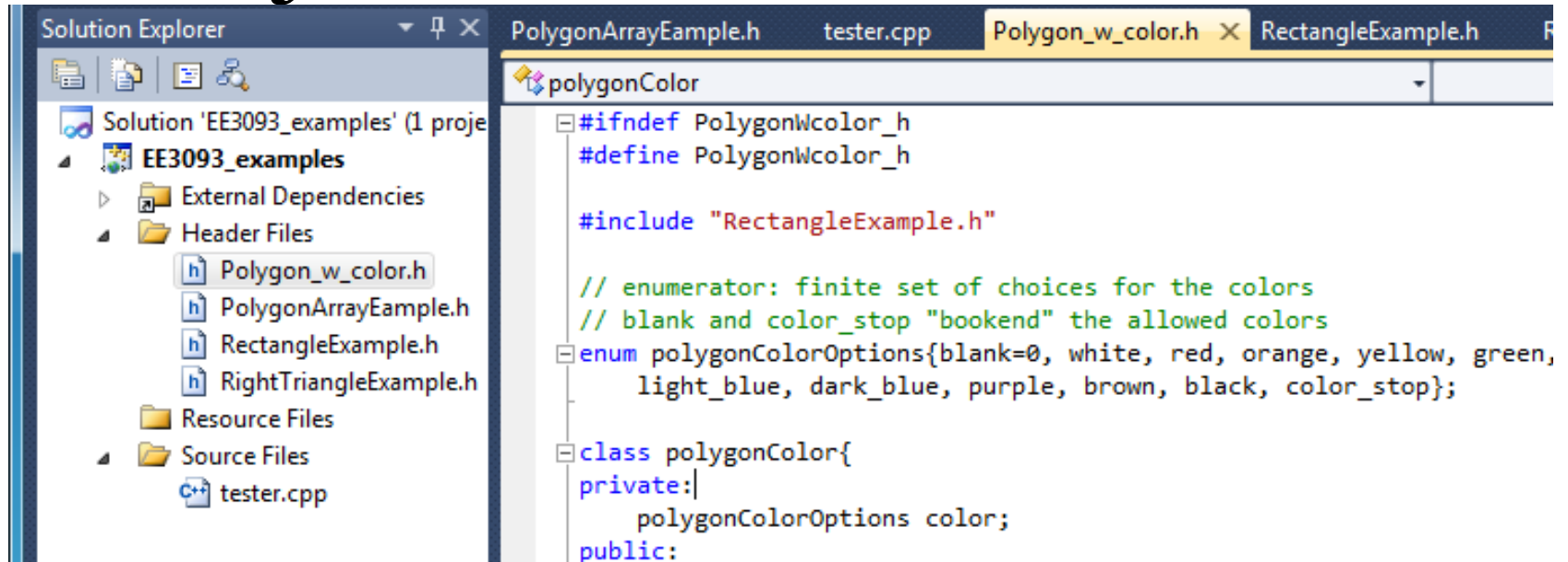
PolygonArrayEample.h  tester.cpp  Polygon_w_color.h  RectangleExample.h
rectangle
#ifndef RectangleExample_h
#define RectangleExample_h

#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





Project: use of namespaces

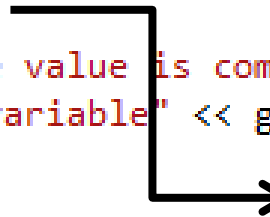
`iostream` is a library file (the extension `.h` has been dropped for some C++ libraries):
Many useful objects are declared and created within this library, for example *cout*, *cin*.
These objects and relative functions have global scope (can be accessed from a function/file different from the one where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

```
int globalscope_variable;  This has global scope

void C_example()
{
    double test_variable;  The scope of test_variable is limited to within
    test_variable = 10.2;      function C_example
    cout << "test_variable value is " << test_variable << endl;
    globalscope_variable=10;
}

void main()
{
    double test_variable;  The scope of test_variable is limited to within
    cout << "test_variable value is completely different here" << test_variable << endl;
    cout << "globalscope_variable" << globalscope_variable << endl;
}
```

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 from 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 from 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

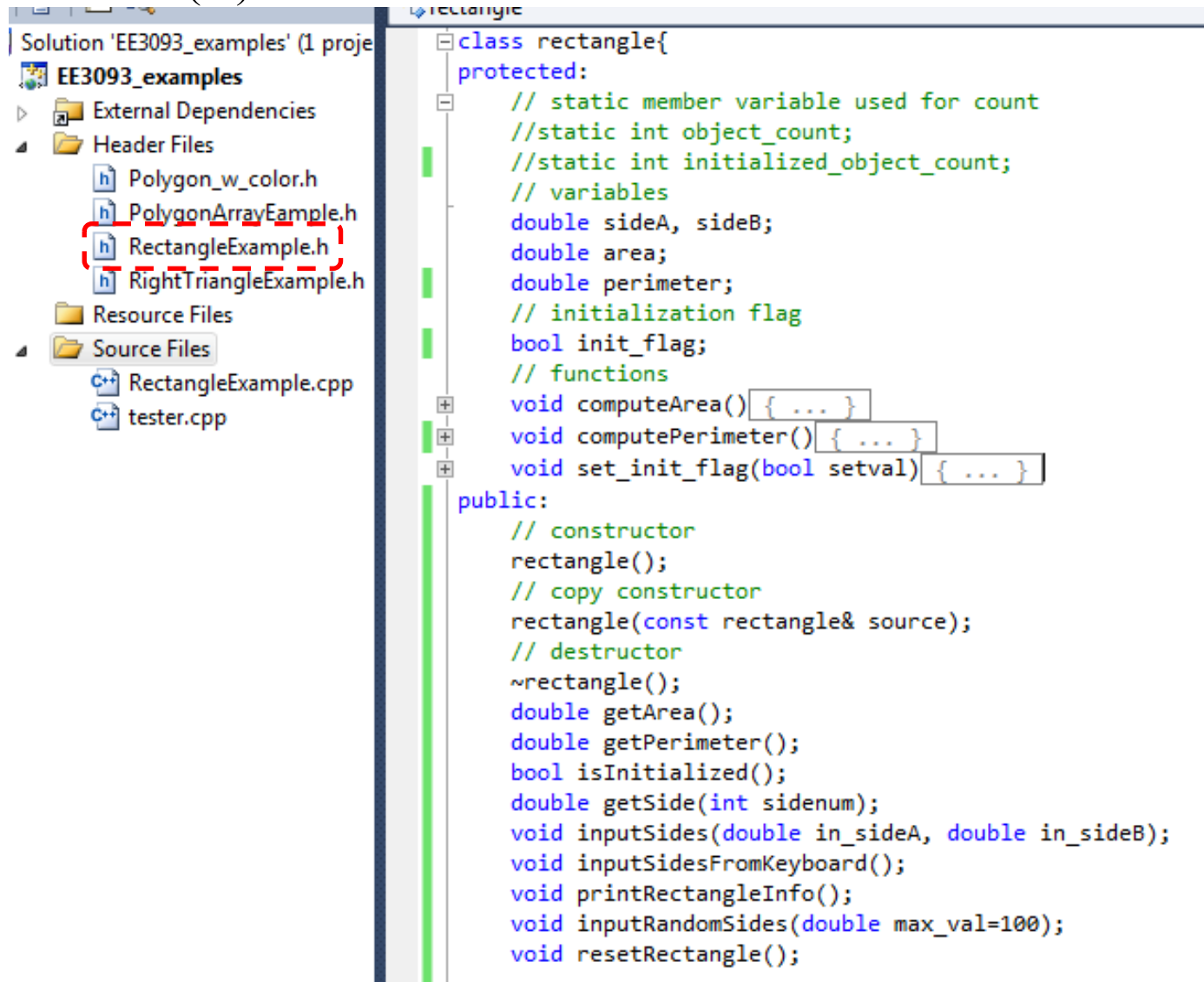
You contribute a class to a project by including:

- 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:



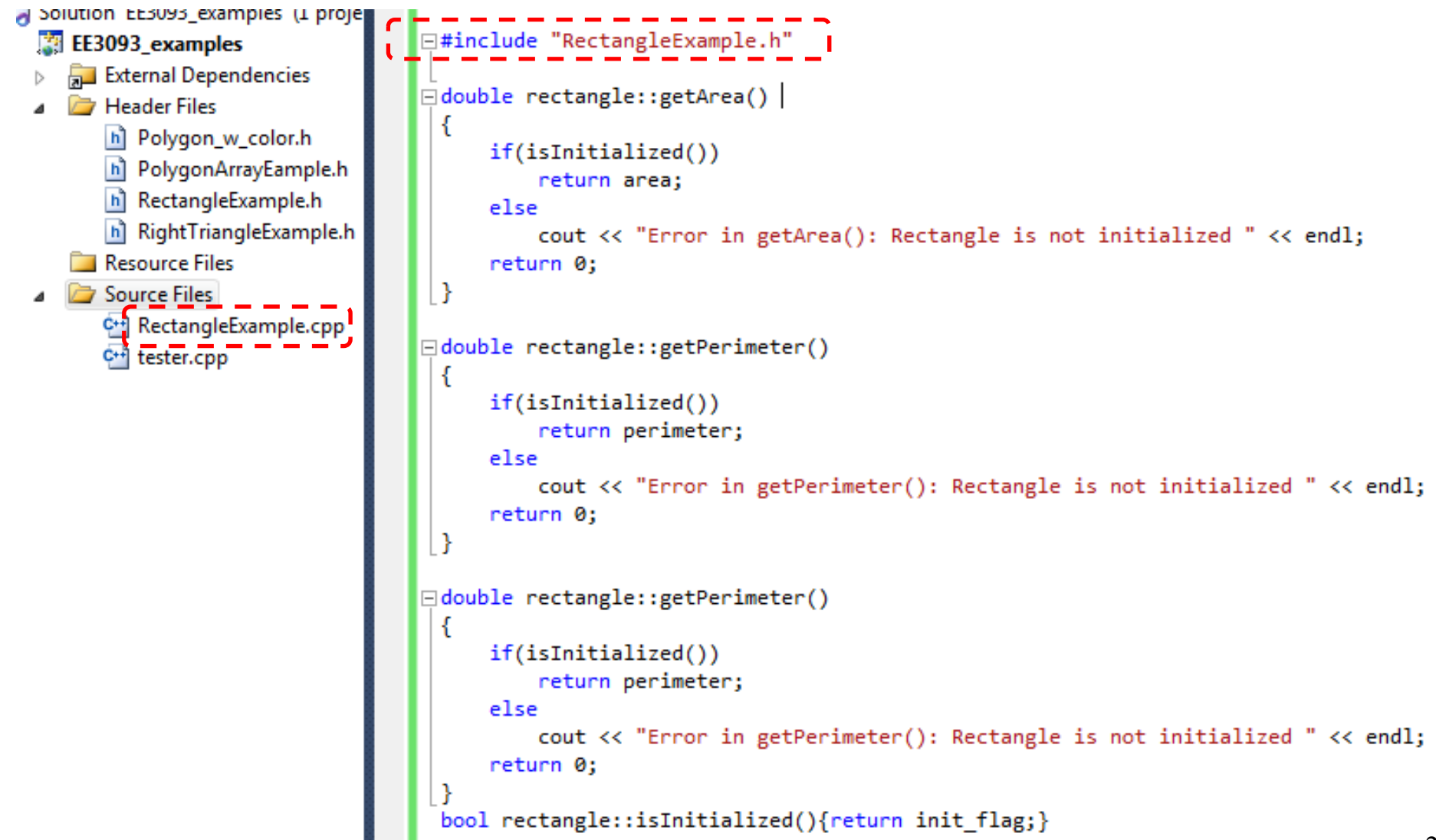
The screenshot shows a C++ IDE with a project named 'EE3093_examples'. The left sidebar displays the project structure, including 'Header Files' and 'Source Files'. In the 'Header Files' section, 'RectangleExample.h' is highlighted with a red dashed box. The main editor window shows the code for 'rectangle.h', which defines a 'rectangle' class. The code includes comments for static member variables, initialization flags, and various methods for computing area, perimeter, and handling input.

```
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 :



```
#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;
}

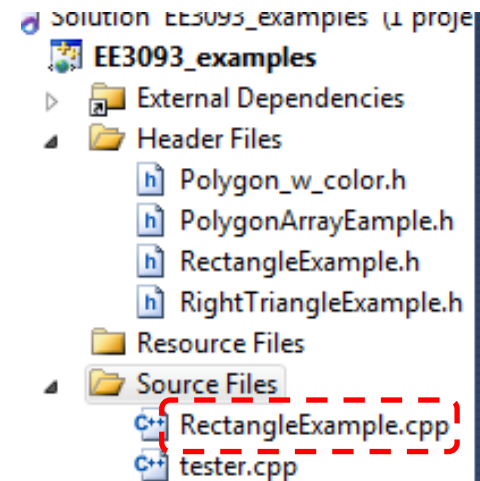
double rectangle::getPerimeter()
{
    if(isInitialized())
        return perimeter;
    else
        cout << "Error in getPerimeter(): Rectangle is not initialized " << endl;
    return 0;
}

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.

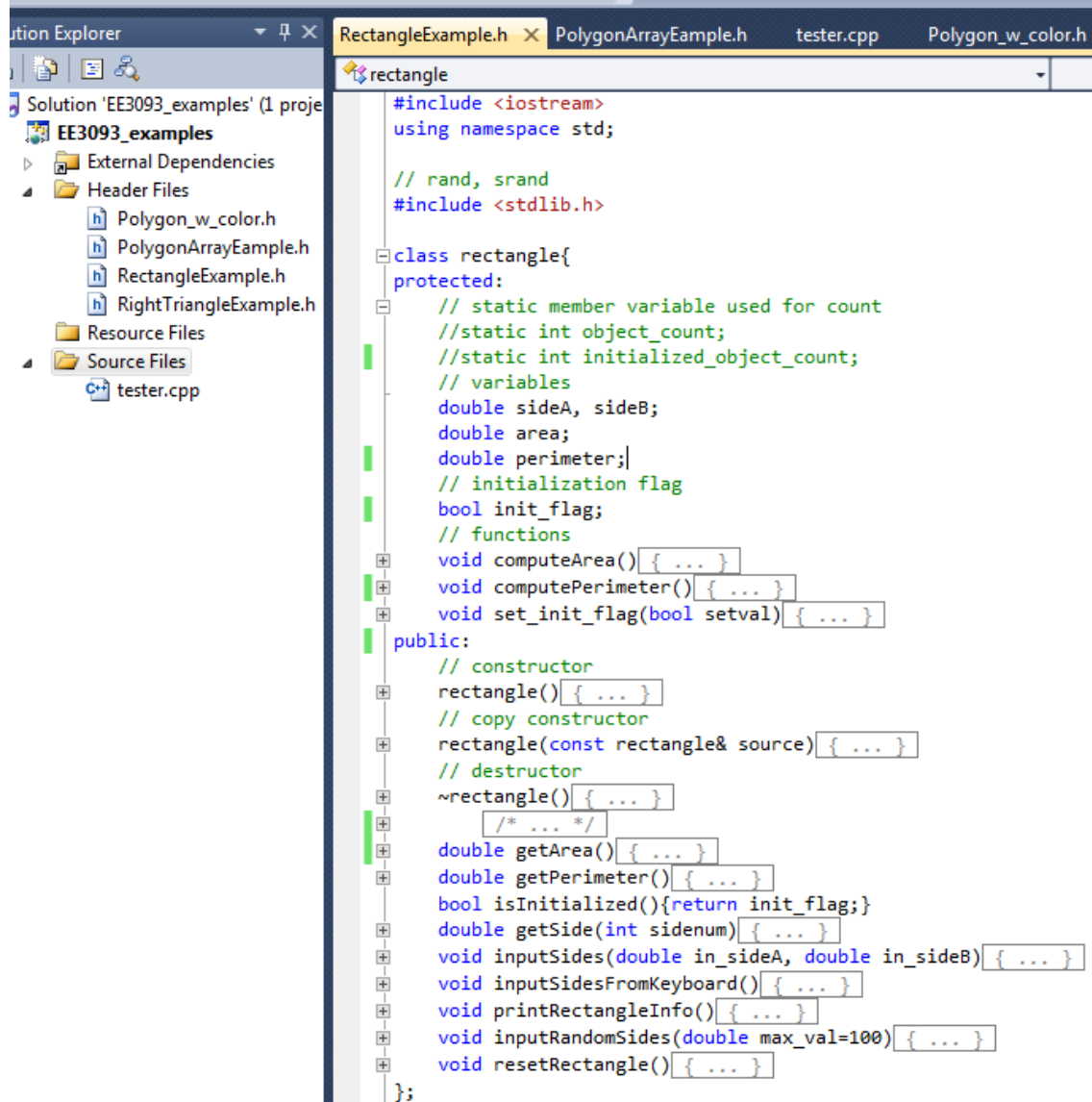
Alternatively:

- 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.



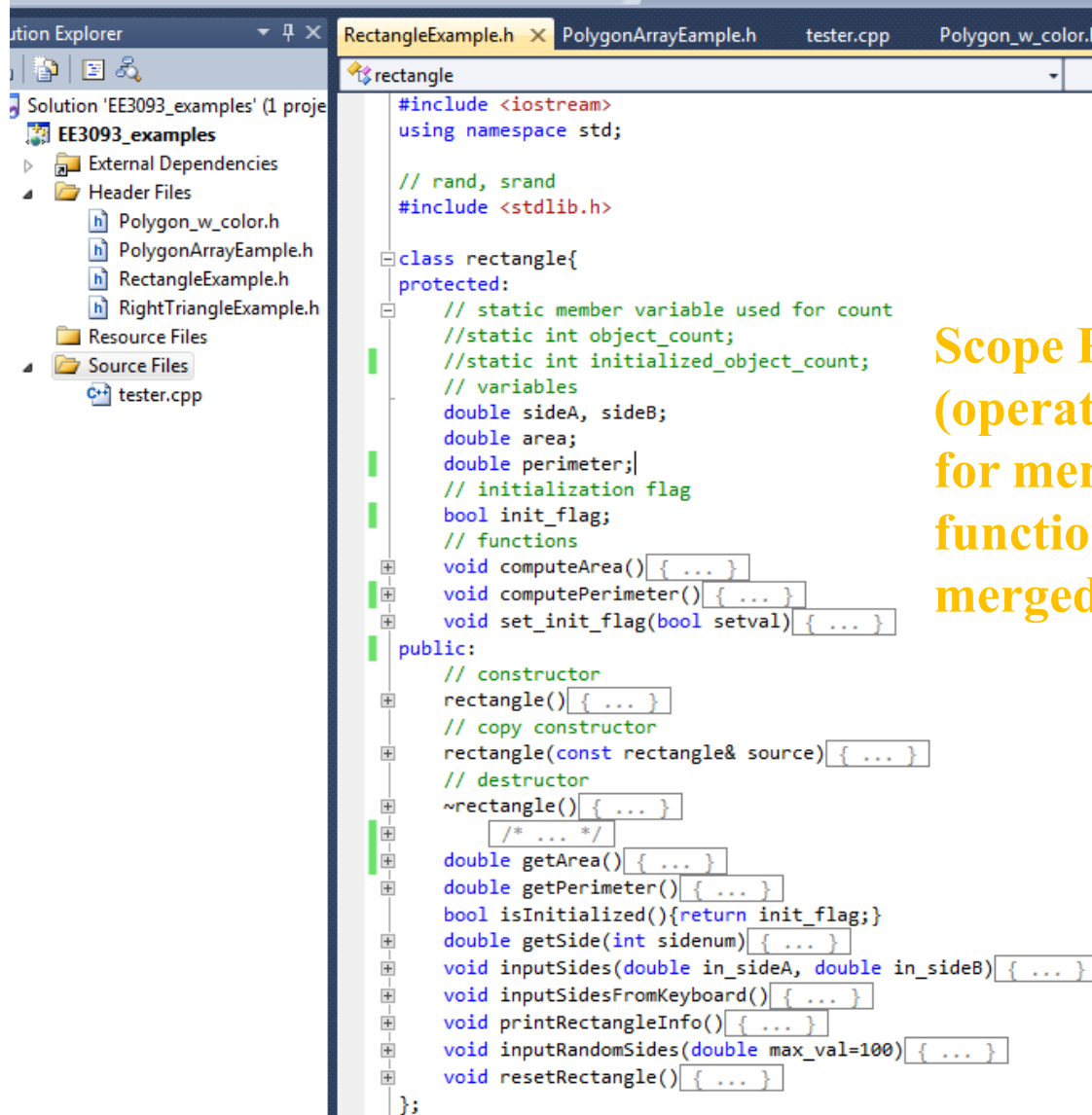
The screenshot shows a C++ IDE with a Solution Explorer on the left and a code editor on the right. The Solution Explorer displays a project named 'EE3093_examples' with a 'Source Files' folder containing 'tester.cpp'. The code editor shows the contents of 'RectangleExample.h', which defines a 'rectangle' class. The class includes static member variables for counting objects, instance variables for sides, area, and perimeter, an initialization flag, and several methods for computing area/perimeter, setting the flag, and inputting sides. The code is color-coded and includes line numbers on the left margin.

```
1  #include <iostream>
2  using namespace std;
3
4  // rand, srand
5  #include <stdlib.h>
6
7  class rectangle{
8  protected:
9      // static member variable used for count
10     //static int object_count;
11     //static int initialized_object_count;
12     // variables
13     double sideA, sideB;
14     double area;
15     double perimeter;
16     // initialization flag
17     bool init_flag;
18     // functions
19     void computeArea() { ... }
20     void computePerimeter() { ... }
21     void set_init_flag(bool setval) { ... }
22 public:
23     // constructor
24     rectangle() { ... }
25     // copy constructor
26     rectangle(const rectangle& source) { ... }
27     // destructor
28     ~rectangle() { ... }
29     /* ... */
30     double getArea() { ... }
31     double getPerimeter() { ... }
32     bool isInitialized(){return init_flag;}
33     double getSide(int sidenum) { ... }
34     void inputSides(double in_sideA, double in_sideB) { ... }
35     void inputSidesFromKeyboard() { ... }
36     void printRectangleInfo() { ... }
37     void inputRandomSides(double max_val=100) { ... }
38     void resetRectangle() { ... }
39 };
```


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.



The screenshot shows a C++ IDE with a Solution Explorer on the left and a code editor on the right. The Solution Explorer shows a project named 'EE3093_examples' with a 'Source Files' folder containing 'tester.cpp'. The code editor shows the 'RectangleExample.h' file, which defines a 'rectangle' class. The class has private static variables for counting objects, public member functions for area, perimeter, and initialization, and a constructor. The code is as follows:

```
#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;
    // 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() { return init_flag; }
    double getSide(int sidenum) { ... }
    void inputSides(double in_sideA, double in_sideB) { ... }
    void inputSidesFromKeyboard() { ... }
    void printRectangleInfo() { ... }
    void inputRandomSides(double max_val=100) { ... }
    void resetRectangle() { ... }
};
```

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,

The screenshot shows a C++ IDE with a project named 'EE3093_examples'. The 'Header Files' folder contains 'Polygon_w_color.h', 'PolygonArrayExample.h', 'RectangleExample.h', and 'RightTriangleExample.h'. The 'Source Files' folder contains 'tester.cpp'. The 'RectangleExample.h' file is open, showing the following code:

```
#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(){ ... }
    void inputRandomColor(){ ... }
```

The 'RightTriangleExample.h' file is also open, showing the following code:

```
#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.

The screenshot shows a C++ IDE with a project named 'EE3093_examples'. The project structure includes 'Header Files' (Polygon_w_color.h, PolygonArrayExample.h, RectangleExample.h, RightTriangleExample.h) and 'Source Files' (tester.cpp). The 'tester.cpp' file is open, showing the following code:

```
#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(){ ... }
    void inputRandomColor(){ ... }
}
```

The 'RightTriangleExample.h' file is also open, showing the following code:

```
#include "RectangleExample.h"

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

    // Hypotenuse
    double sideC;

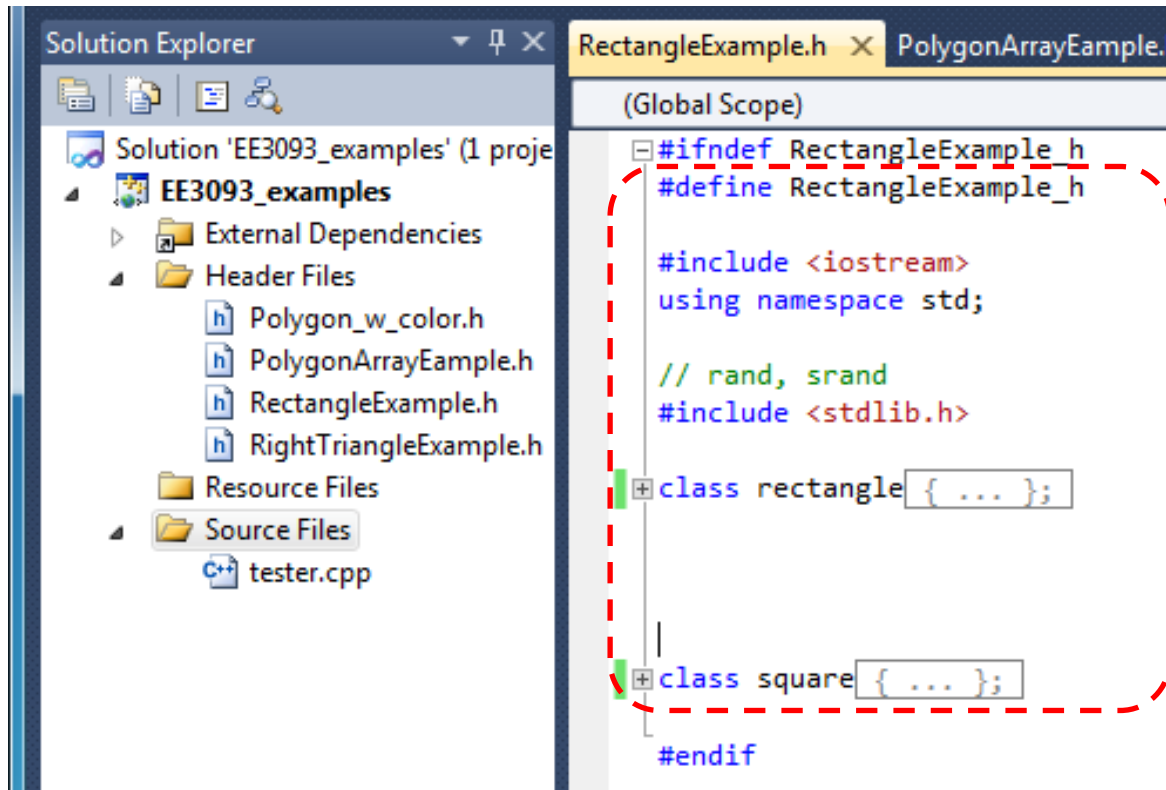
    double area;
    double perimeter;

    // functions
    void computeArea()
```

Project: define/declare your objects

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

- *#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?

