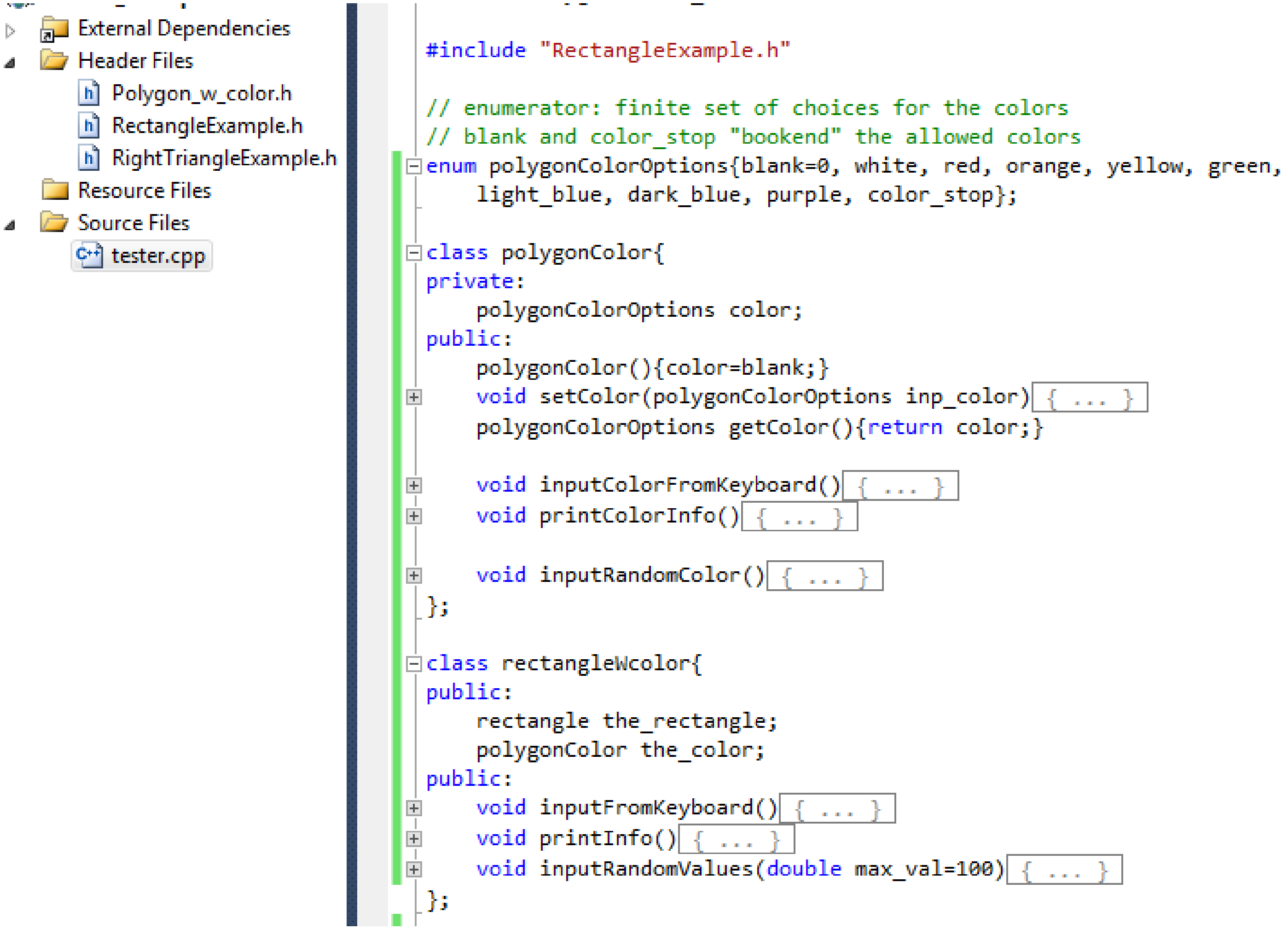
# C/C++ Programming: Further Examples

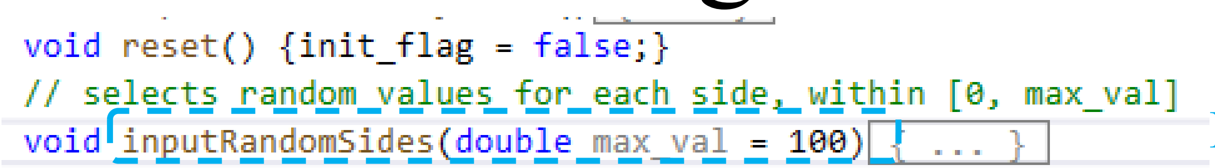


```
#include "RectangleExample.h"

// 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

# Overloading a member function

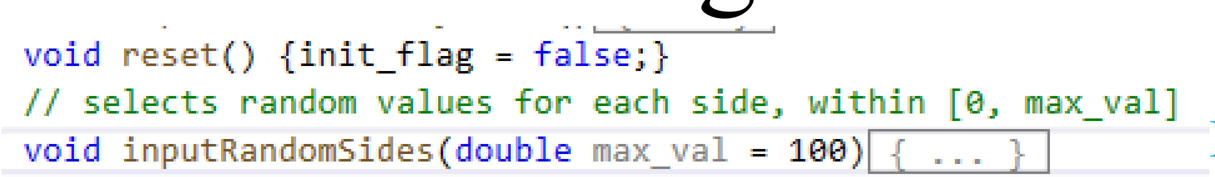
From file: RectangleExample.h



```
void reset() {init_flag = false;}
// selects random values for each side, within [0, max_val]
void inputRandomSides(double max_val = 100) { ... }
```

First version: one input value (or no input to use default)

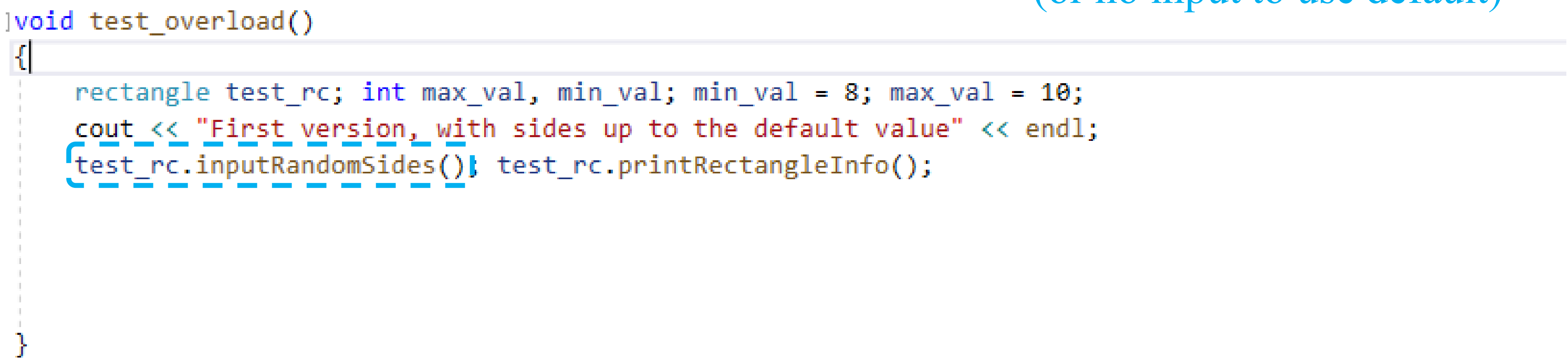
# Overloading a member function



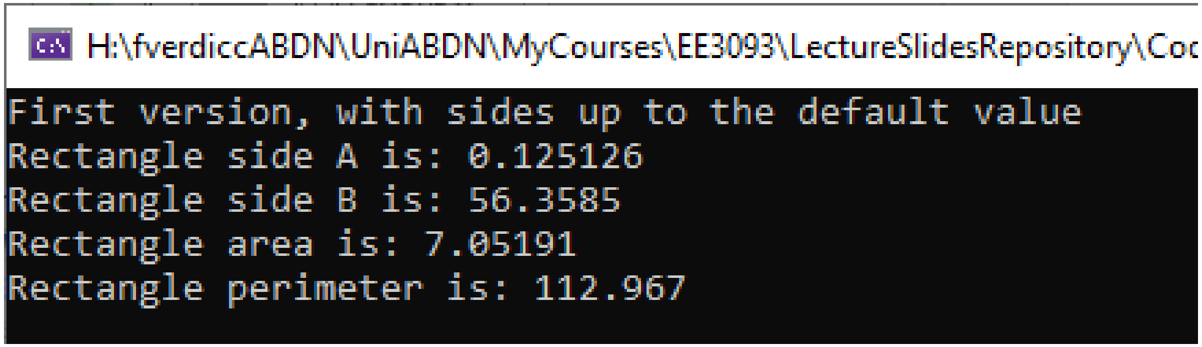
From file: RectangleExample.h

```
void reset() {init_flag = false;}
// selects random values for each side, within [0, max_val]
void inputRandomSides(double max_val = 100) { ... }
```

First version: one input value (or no input to use default)



```
void test_overload()
{
    rectangle test_rc; int max_val, min_val; min_val = 8; max_val = 10;
    cout << "First version, with sides up to the default value" << endl;
    test_rc.inputRandomSides(); test_rc.printRectangleInfo();




}
```

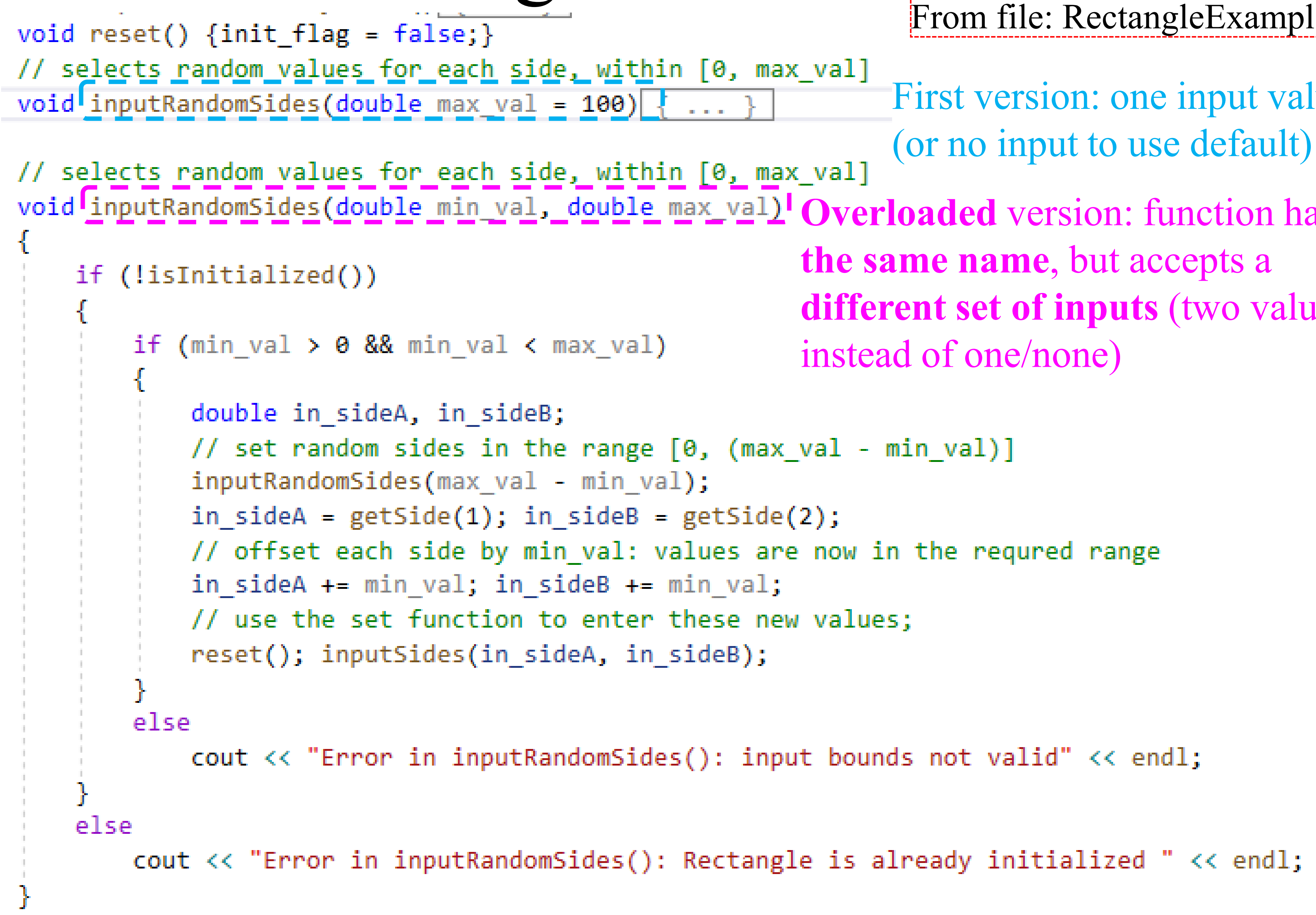


H:\fverdiccABDN\UniABDN\MyCourses\EE3093\LectureSlidesRepository\Coc

```
First version, with sides up to the default value
Rectangle side A is: 0.125126
Rectangle side B is: 56.3585
Rectangle area is: 7.05191
Rectangle perimeter is: 112.967
```

# Overloading a member function

From file: RectangleExample.h

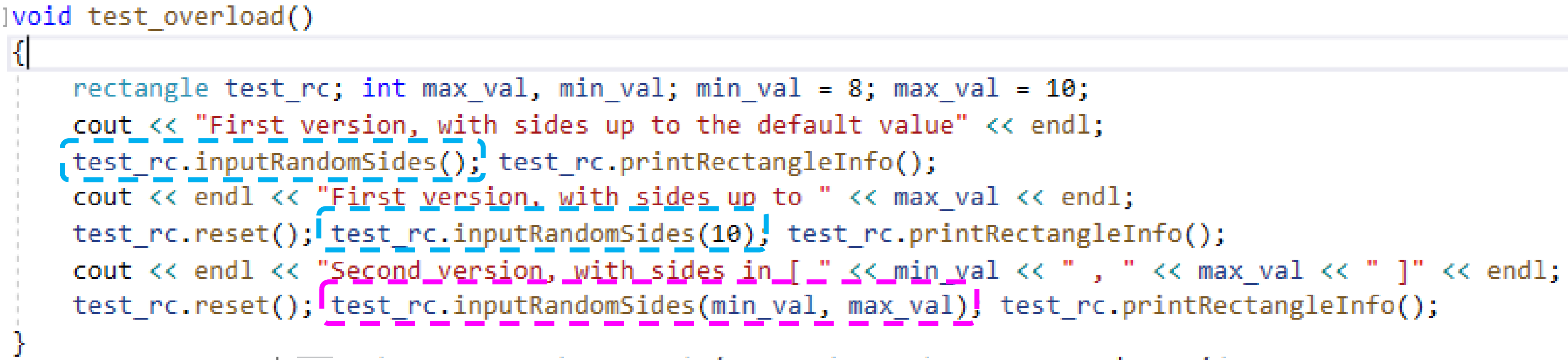


```
void reset() {init_flag = false;}
// selects random values for each side, within [0, max_val]
void inputRandomSides(double max_val = 100) { ... }

// selects random values for each side, within [0, max_val]
void inputRandomSides(double min_val, double max_val)
{
    if (!isInitialized())
    {
        if (min_val > 0 && min_val < max_val)
        {
            double in_sideA, in_sideB;
            // set random sides in the range [0, (max_val - min_val)]
            inputRandomSides(max_val - min_val);
            in_sideA = getSide(1); in_sideB = getSide(2);
            // offset each side by min_val: values are now in the requred range
            in_sideA += min_val; in_sideB += min_val;
            // use the set function to enter these new values;
            reset(); inputSides(in_sideA, in_sideB);
        }
        else
            cout << "Error in inputRandomSides(): input bounds not valid" << endl;
    }
    else
        cout << "Error in inputRandomSides(): Rectangle is already initialized " << endl;
}
```

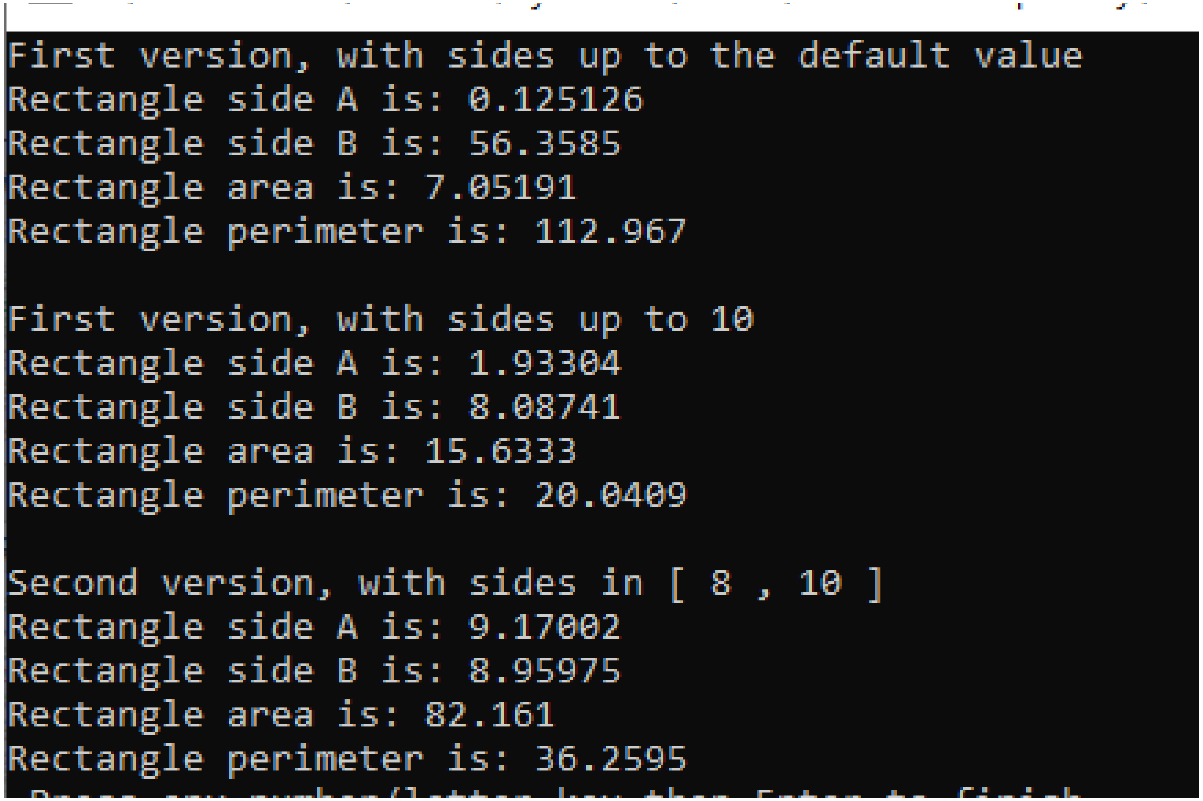
First version: one input value (or no input to use default)

**Overloaded** version: function has **the same name**, but accepts a **different set of inputs** (two values instead of one/none)

# Overloading a member function

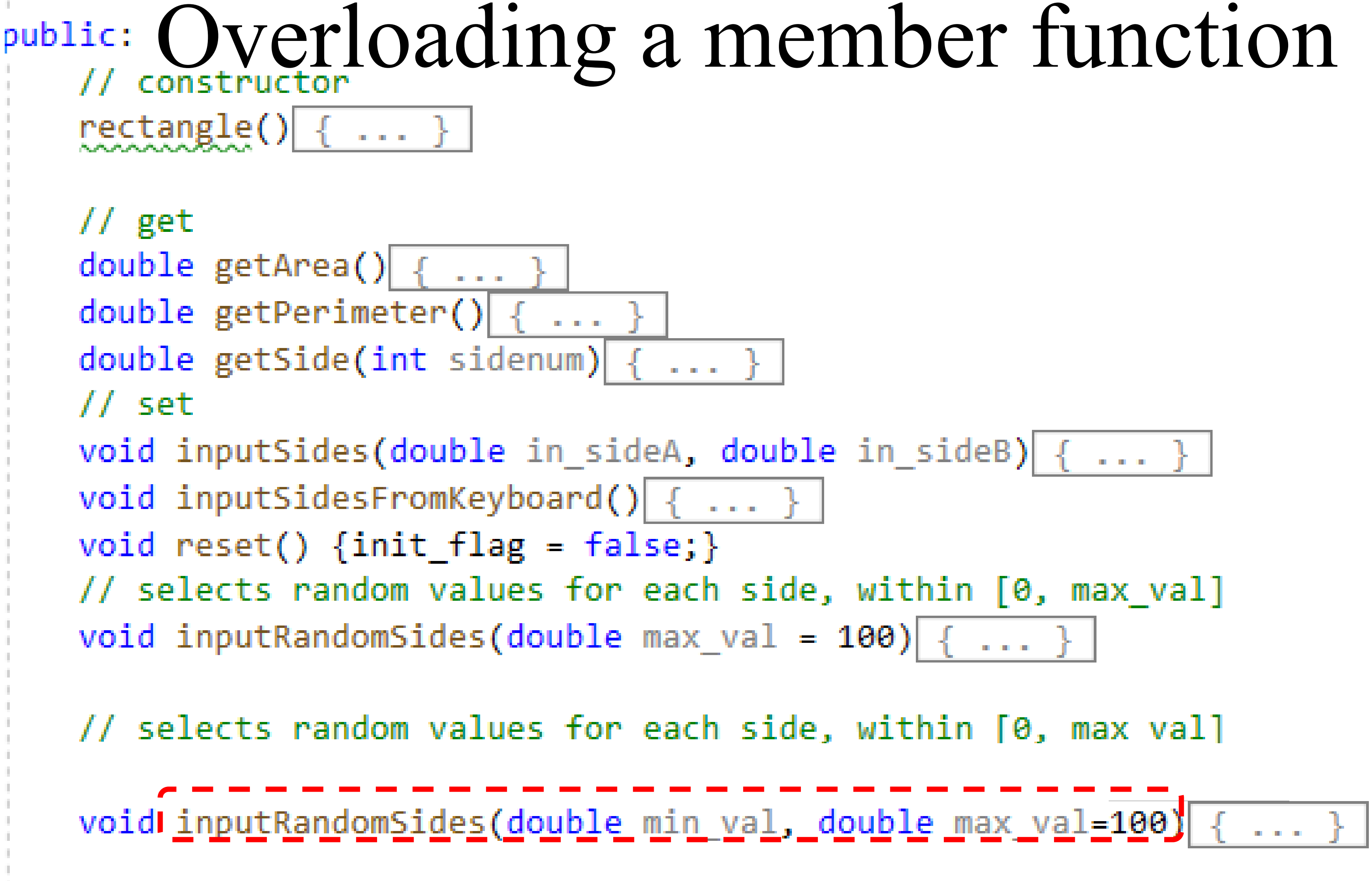


```
void test_overload()
{
    rectangle test_rc; int max_val, min_val; min_val = 8; max_val = 10;
    cout << "First version, with sides up to the default value" << endl;
    test_rc.inputRandomSides(); test_rc.printRectangleInfo();
    cout << endl << "First version, with sides up to " << max_val << endl;
    test_rc.reset(); test_rc.inputRandomSides(10); test_rc.printRectangleInfo();
    cout << endl << "Second version, with sides in [ " << min_val << " , " << max_val << " ]" << endl;
    test_rc.reset(); test_rc.inputRandomSides(min_val, max_val); test_rc.printRectangleInfo();
}
```



```
First version, with sides up to the default value
Rectangle side A is: 0.125126
Rectangle side B is: 56.3585
Rectangle area is: 7.05191
Rectangle perimeter is: 112.967

First version, with sides up to 10
Rectangle side A is: 1.93304
Rectangle side B is: 8.08741
Rectangle area is: 15.6333
Rectangle perimeter is: 20.0409

Second version, with sides in [ 8 , 10 ]
Rectangle side A is: 9.17002
Rectangle side B is: 8.95975
Rectangle area is: 82.161
Rectangle perimeter is: 36.2595
```

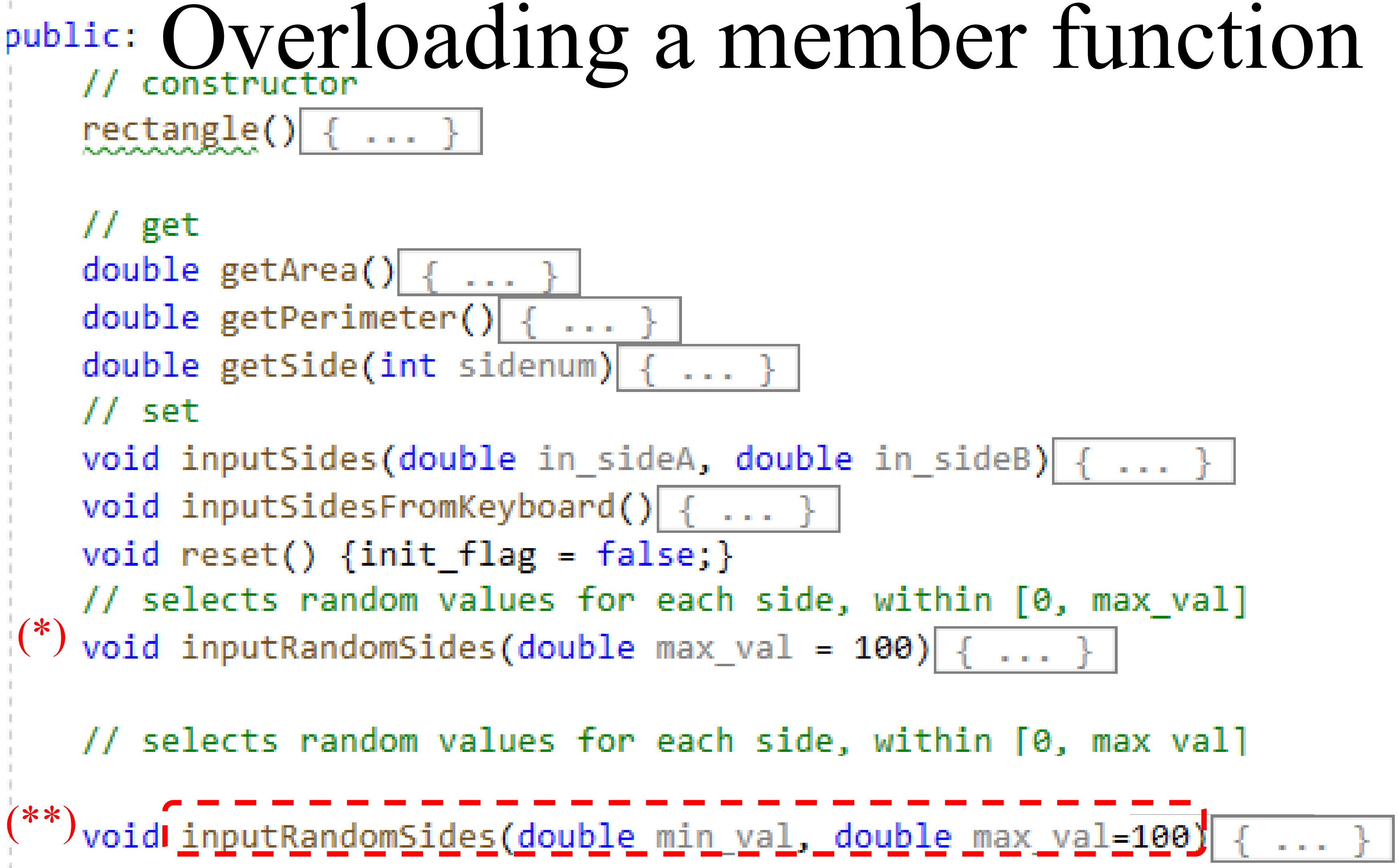
# Overloading a member function



```
public:
    // constructor
    rectangle() { ... }

    // get
    double getArea() { ... }
    double getPerimeter() { ... }
    double getSide(int sidenum) { ... }
    // set
    void inputSides(double in_sideA, double in_sideB) { ... }
    void inputSidesFromKeyboard() { ... }
    void reset() {init_flag = false;}
    // selects random values for each side, within [0, max_val]
    void inputRandomSides(double max_val = 100) { ... }

    // selects random values for each side, within [0, max_val]

    void inputRandomSides(double min_val, double max_val=100) { ... }
```

But this is **not allowed**:

# Overloading a member function



```
public:
    // constructor
    rectangle() { ... }

    // get
    double getArea() { ... }
    double getPerimeter() { ... }
    double getSide(int sidenum) { ... }
    // set
    void inputSides(double in_sideA, double in_sideB) { ... }
    void inputSidesFromKeyboard() { ... }
    void reset() {init_flag = false;}
    // selects random values for each side, within [0, max_val]
(*) void inputRandomSides(double max_val = 100) { ... }

    // selects random values for each side, within [0, max_val]

(**)void inputRandomSides(double min_val, double max_val=100) { ... }
```

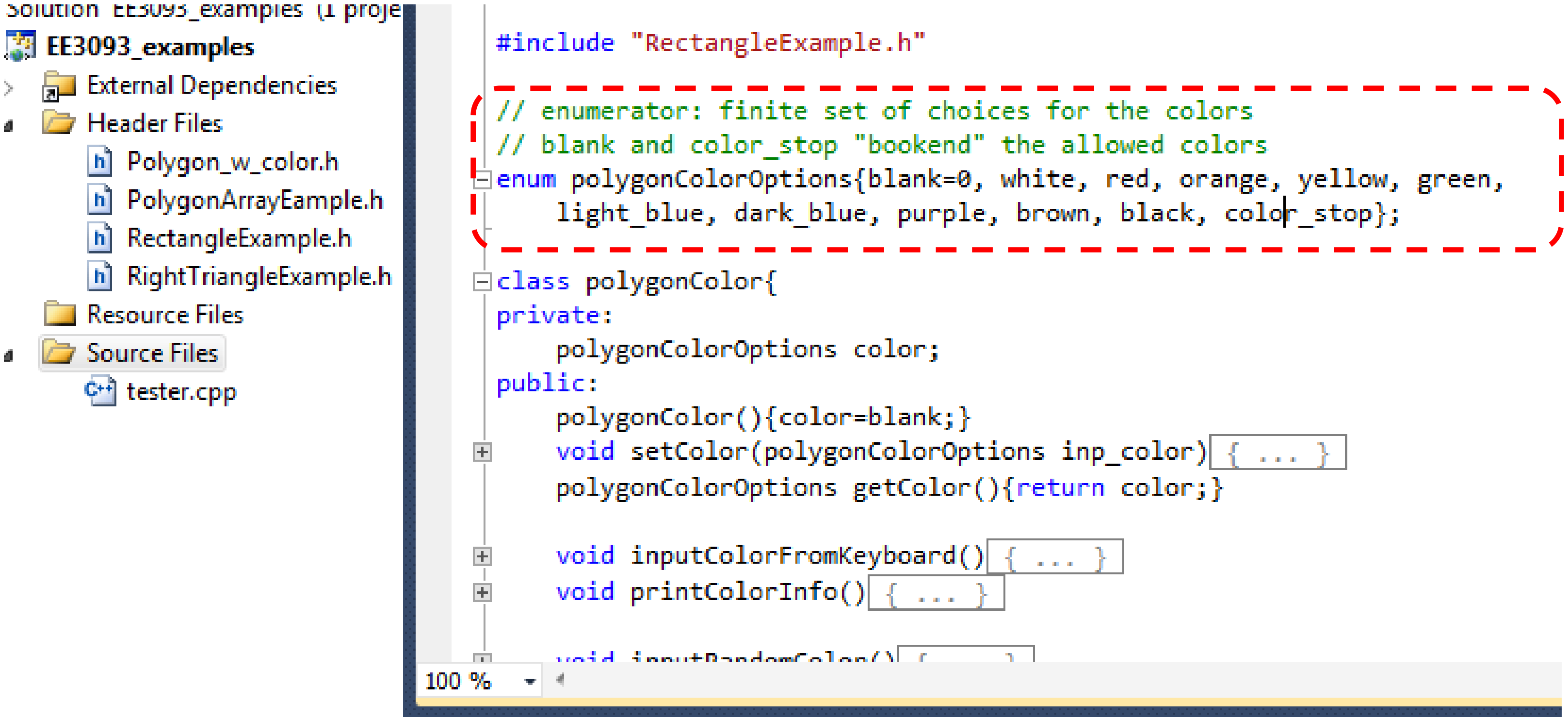
But this is **not allowed**: if this additional overloaded version has two inputs, and one (the last) has a default value, then a function call such as:

```
test_rc.inputRandomSides(10);
```

can be interpreted as a call to either version (*) or (**) with a single parameter.

# Any question?

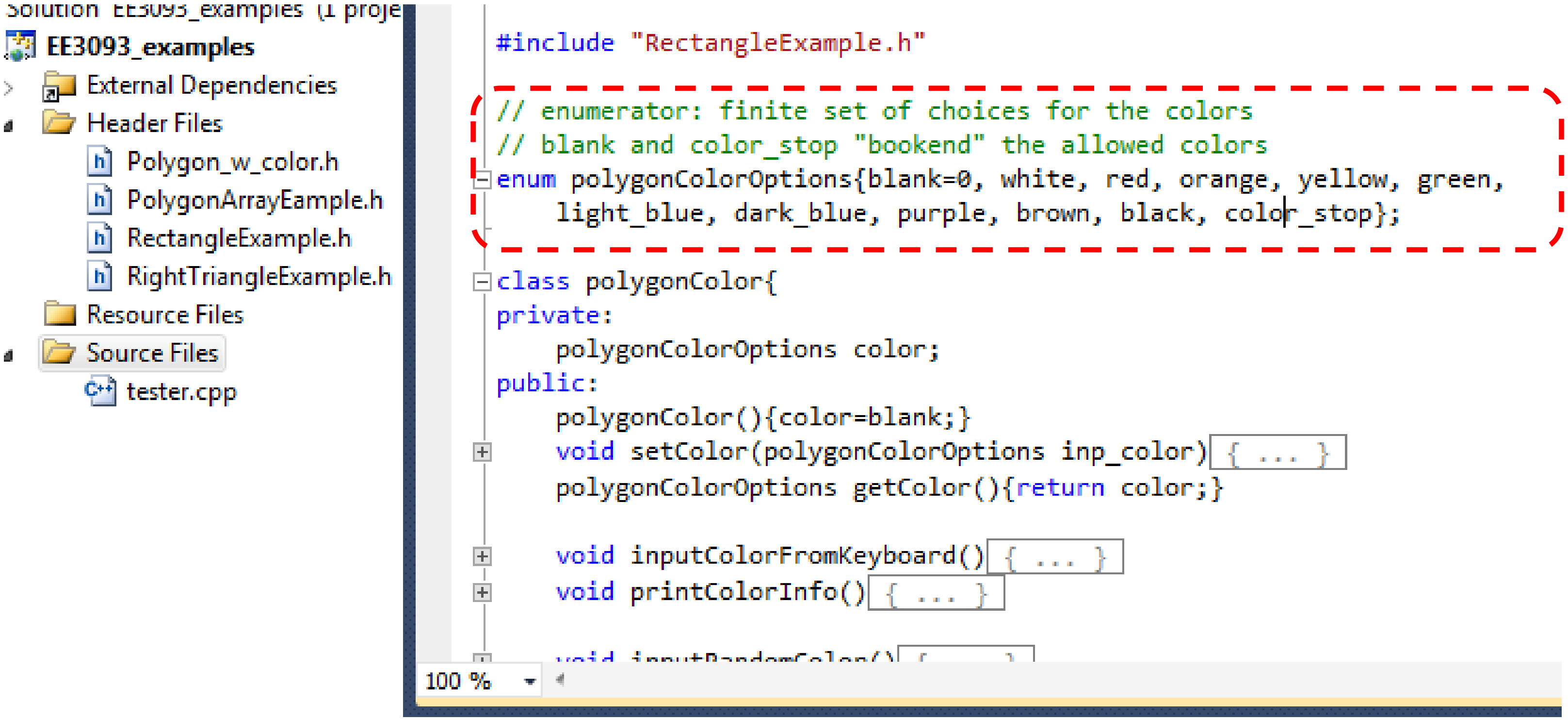
# Scoping examples



```
#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 *enum* polygonColorOptions are defined (C-stile) with global scope;
- “Symbolic equivalents” such as “white”, “orange”, “brown”, “black” used in this *enum* are now reserved and cannot be used within another enumerator;

# Scoping examples

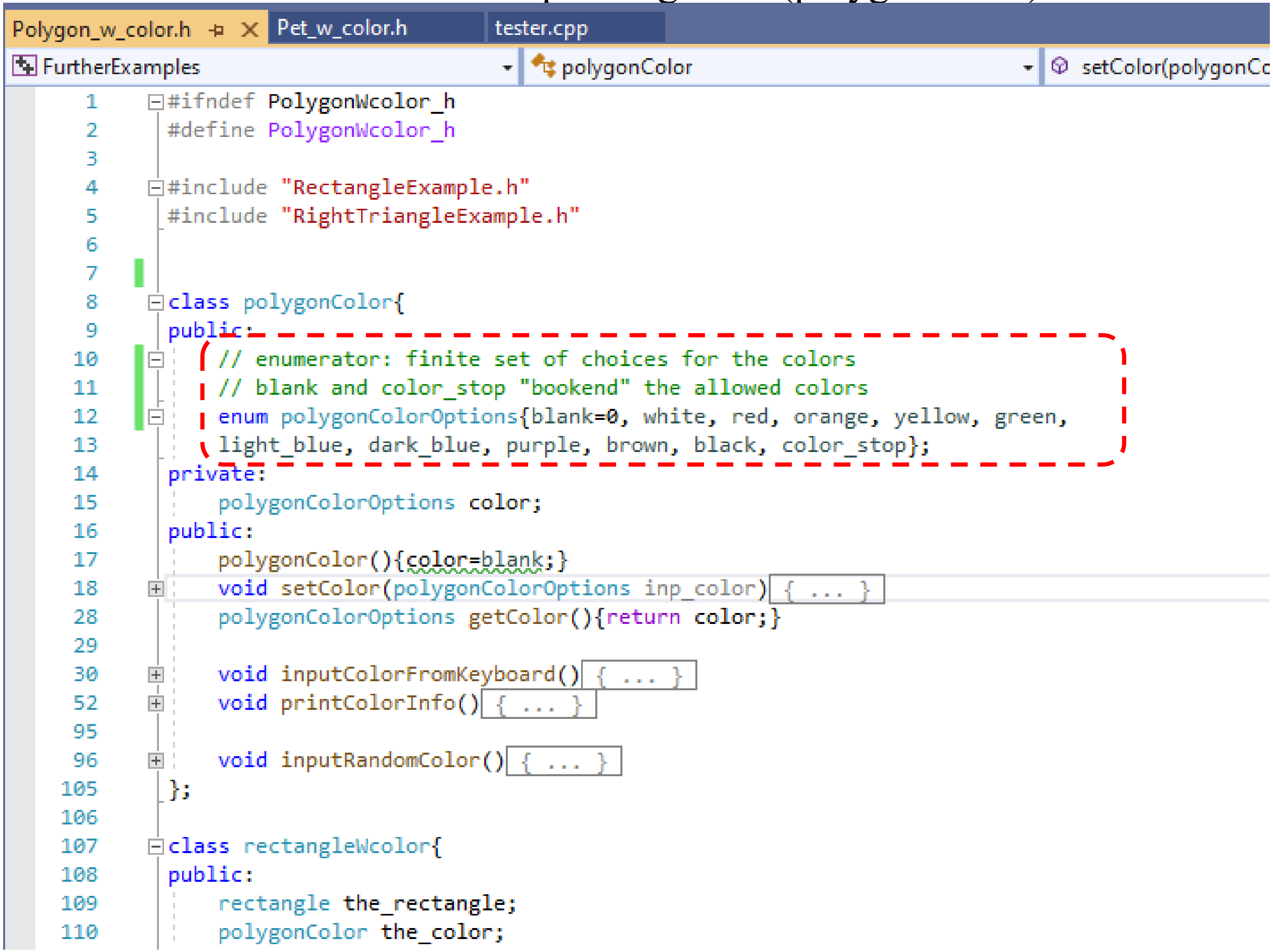


```
#include "RectangleExample.h"

// 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 *enum* polygonColorOptions are defined (C-stile) with global scope;
- “Symbolic equivalents” such as “white”, “orange”, “brown”, “black” used in this *enum* are now reserved and cannot be used within another enumerator;
- Say we want to create a class petColor where an *enum* petColorOptions indicates fur colour of pets: we can’t use symbols “white”, “orange”, “brown”, “black” without causing confusion… unless we define the *enum* inside the corresponding classes

# Scoping examples

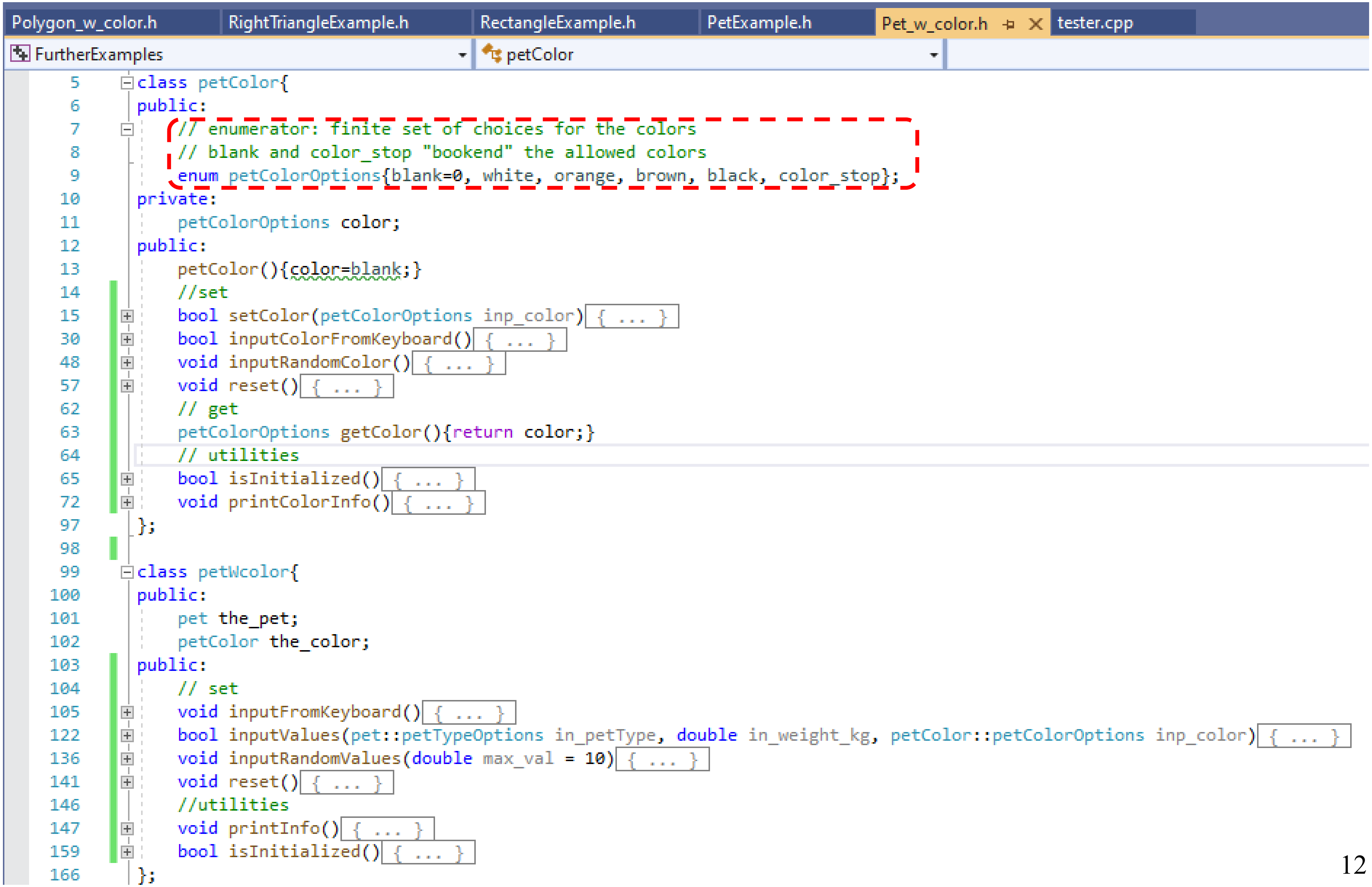
- Define the *enum* **within** the corresponding class (polygonColor)



```
#ifndef PolygonWcolor_h
#define PolygonWcolor_h

#include "RectangleExample.h"
#include "RightTriangleExample.h"


class polygonColor{
public:
    // 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};
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;
```

# Scoping examples

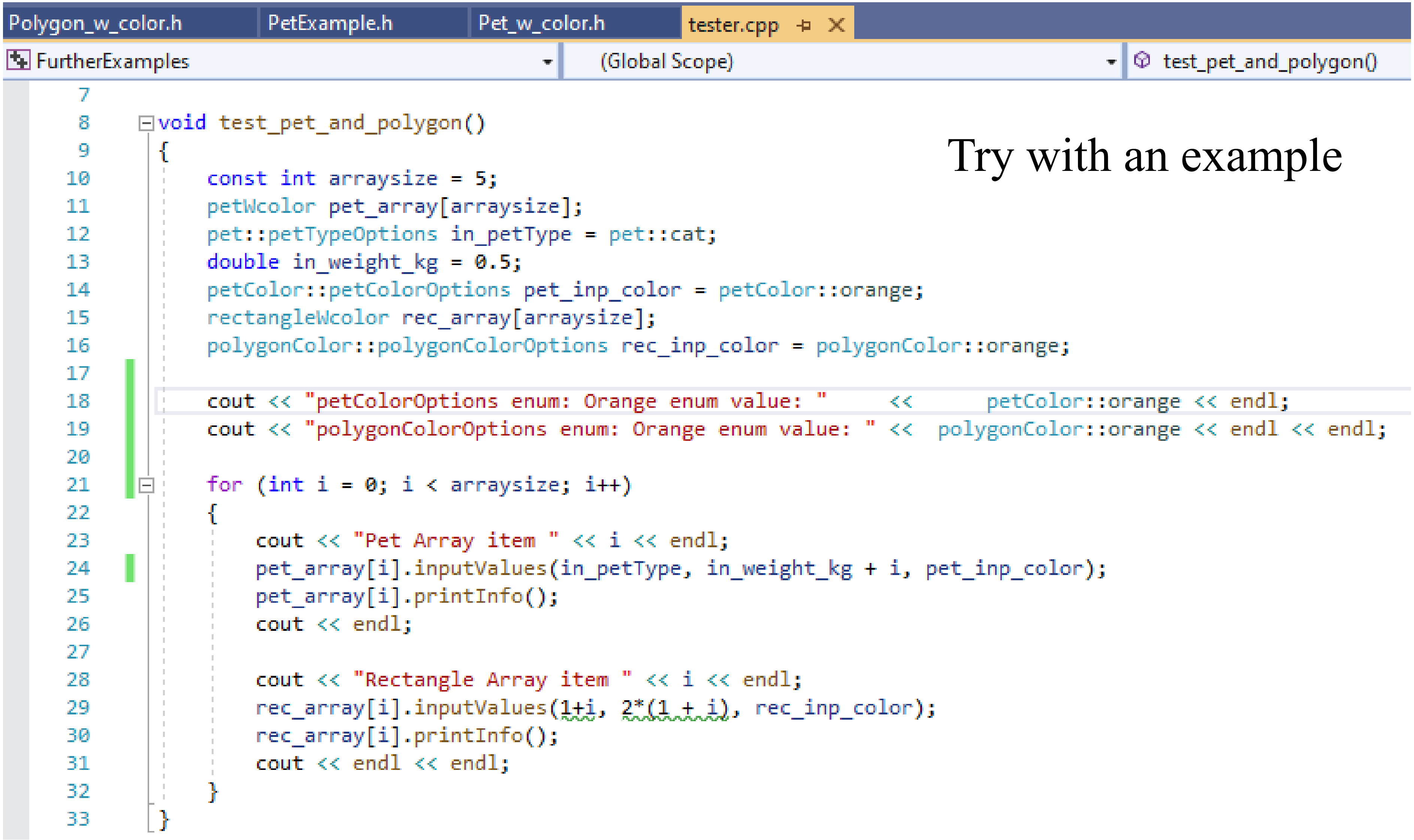
- Define the *enum* **within** the corresponding class (petColor)



```
class petColor{
public:
    // enumerator: finite set of choices for the colors
    // blank and color_stop "bookend" the allowed colors
    enum petColorOptions{blank=0, white, orange, brown, black, color_stop};
private:
    petColorOptions color;
public:
    petColor(){color=blank;}
    //set
    bool setColor(petColorOptions inp_color) { ... }
    bool inputColorFromKeyboard() { ... }
    void inputRandomColor() { ... }
    void reset() { ... }
    // get
    petColorOptions getColor(){return color;}
    // utilities
    bool isInitialized() { ... }
    void printColorInfo() { ... }
};

class petWcolor{
public:
    pet the_pet;
    petColor the_color;
public:
    // set
    void inputFromKeyboard() { ... }
    bool inputValues(pet::petTypeOptions in_petType, double in_weight_kg, petColor::petColorOptions inp_color) { ... }
    void inputRandomValues(double max_val = 10) { ... }
    void reset() { ... }
    //utilities
    void printInfo() { ... }
    bool isInitialized() { ... }
};
```

# Scoping examples

- Outside class member functions: specify class name with the scoping operator (::)

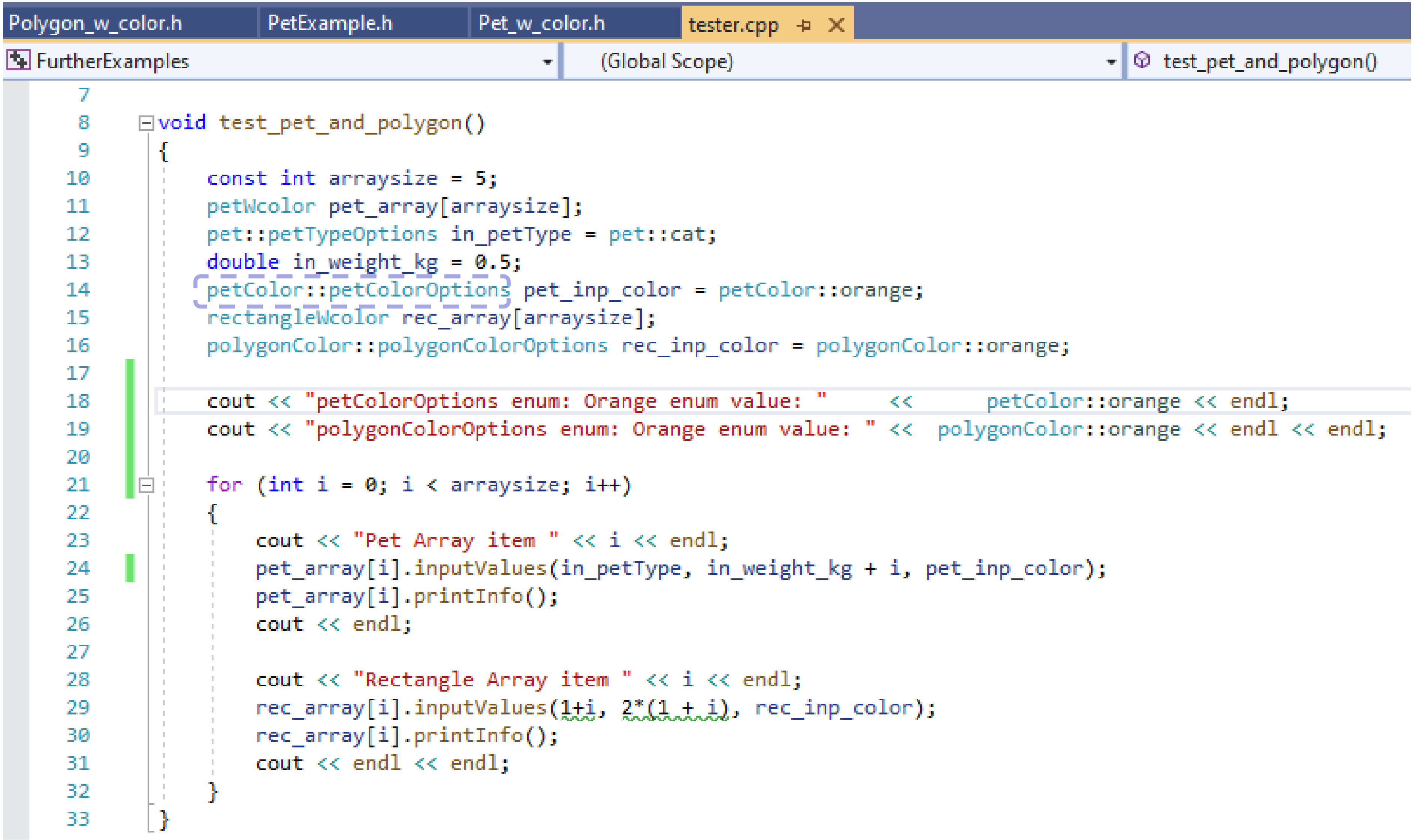


Try with an example

```
void test_pet_and_polygon()
{
    const int arraysize = 5;
    petWcolor pet_array[arraysize];
    pet::petTypeOptions in_petType = pet::cat;
    double in_weight_kg = 0.5;
    petColor::petColorOptions pet_inp_color = petColor::orange;
    rectangleWcolor rec_array[arraysize];
    polygonColor::polygonColorOptions rec_inp_color = polygonColor::orange;

    cout << "petColorOptions enum: Orange enum value: "     <<      petColor::orange << endl;
    cout << "polygonColorOptions enum: Orange enum value: " <<  polygonColor::orange << endl << endl;

    for (int i = 0; i < arraysize; i++)
    {
        cout << "Pet Array item " << i << endl;
        pet_array[i].inputValues(in_petType, in_weight_kg + i, pet_inp_color);
        pet_array[i].printInfo();
        cout << endl;

        cout << "Rectangle Array item " << i << endl;
        rec_array[i].inputValues(1+i, 2*(1 + i), rec_inp_color);
        rec_array[i].printInfo();
        cout << endl << endl;
    }
}
```

# Scoping examples

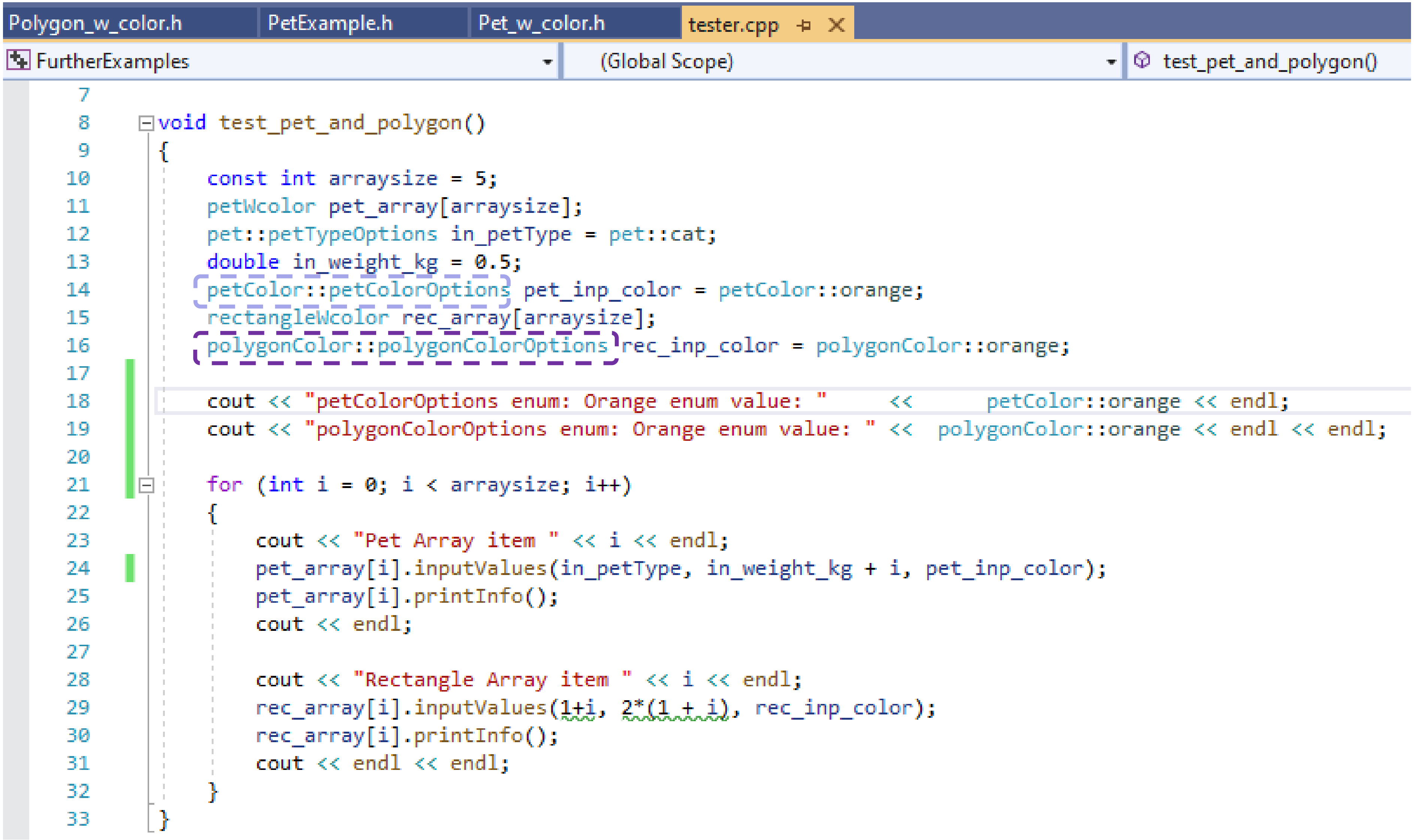
- Outside class member functions: specify class name with the scoping operator (::)



```cpp

void test_pet_and_polygon()
{
    const int arraysize = 5;
    petWcolor pet_array[arraysize];
    pet::petTypeOptions in_petType = pet::cat;
    double in_weight_kg = 0.5;
    petColor::petColorOptions pet_inp_color = petColor::orange;
    rectangleWcolor rec_array[arraysize];
    polygonColor::polygonColorOptions rec_inp_color = polygonColor::orange;

    cout << "petColorOptions enum: Orange enum value: "     <<      petColor::orange << endl;
    cout << "polygonColorOptions enum: Orange enum value: " <<  polygonColor::orange << endl << endl;

    for (int i = 0; i < arraysize; i++)
    {
        cout << "Pet Array item " << i << endl;
        pet_array[i].inputValues(in_petType, in_weight_kg + i, pet_inp_color);
        pet_array[i].printInfo();
        cout << endl;

        cout << "Rectangle Array item " << i << endl;
        rec_array[i].inputValues(1+i, 2*(1 + i), rec_inp_color);
        rec_array[i].printInfo();
        cout << endl << endl;
    }
}
```

# Scoping examples

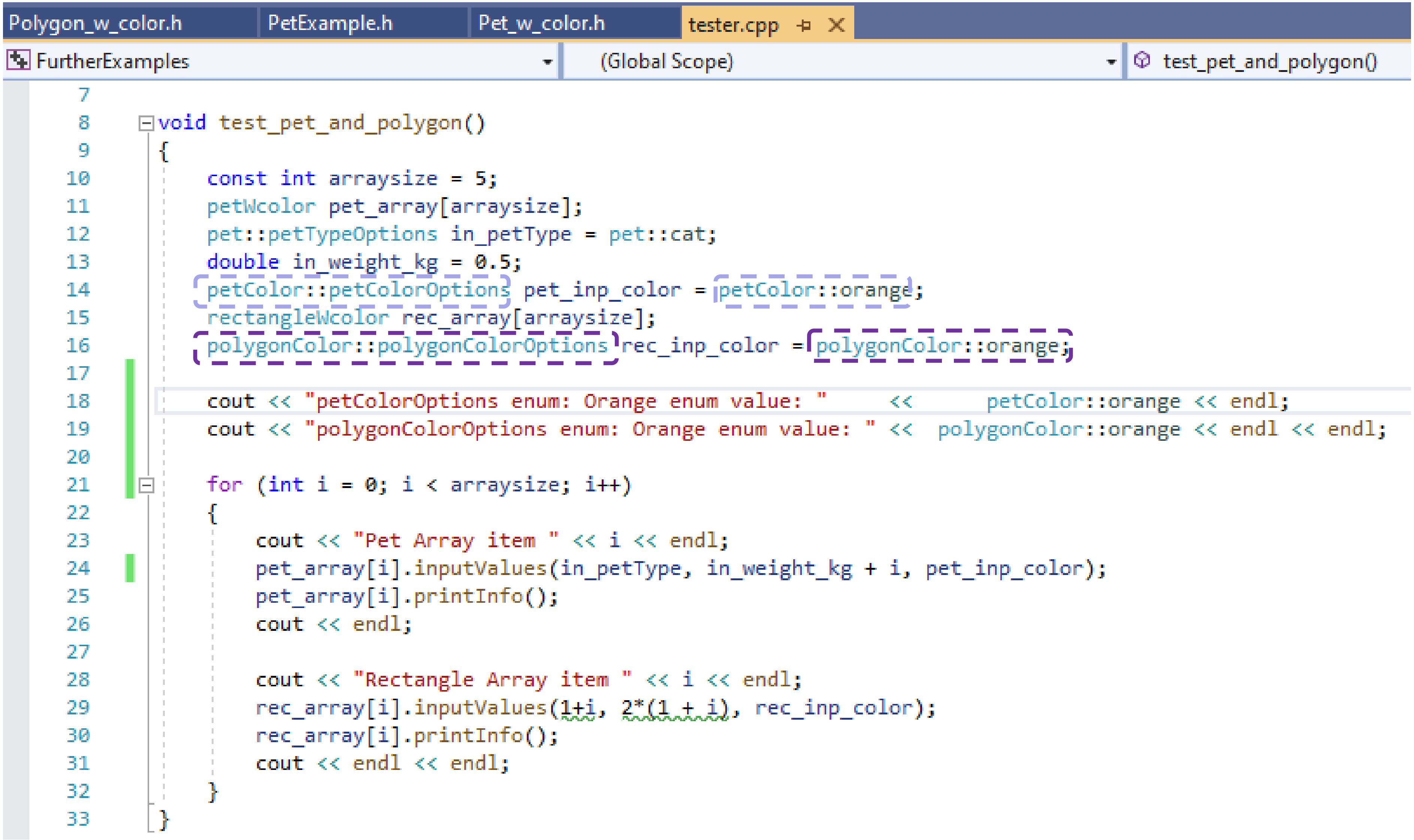
- Outside class member functions: specify class name with the scoping operator (::)



```

void test_pet_and_polygon()
{
    const int arraysize = 5;
    petWcolor pet_array[arraysize];
    pet::petTypeOptions in_petType = pet::cat;
    double in_weight_kg = 0.5;
    petColor::petColorOptions pet_inp_color = petColor::orange;
    rectangleWcolor rec_array[arraysize];
    polygonColor::polygonColorOptions rec_inp_color = polygonColor::orange;

    cout << "petColorOptions enum: Orange enum value: "     <<      petColor::orange << endl;
    cout << "polygonColorOptions enum: Orange enum value: " <<  polygonColor::orange << endl << endl;

    for (int i = 0; i < arraysize; i++)
    {
        cout << "Pet Array item " << i << endl;
        pet_array[i].inputValues(in_petType, in_weight_kg + i, pet_inp_color);
        pet_array[i].printInfo();
        cout << endl;

        cout << "Rectangle Array item " << i << endl;
        rec_array[i].inputValues(1+i, 2*(1 + i), rec_inp_color);
        rec_array[i].printInfo();
        cout << endl << endl;
    }
}
```

# Scoping examples

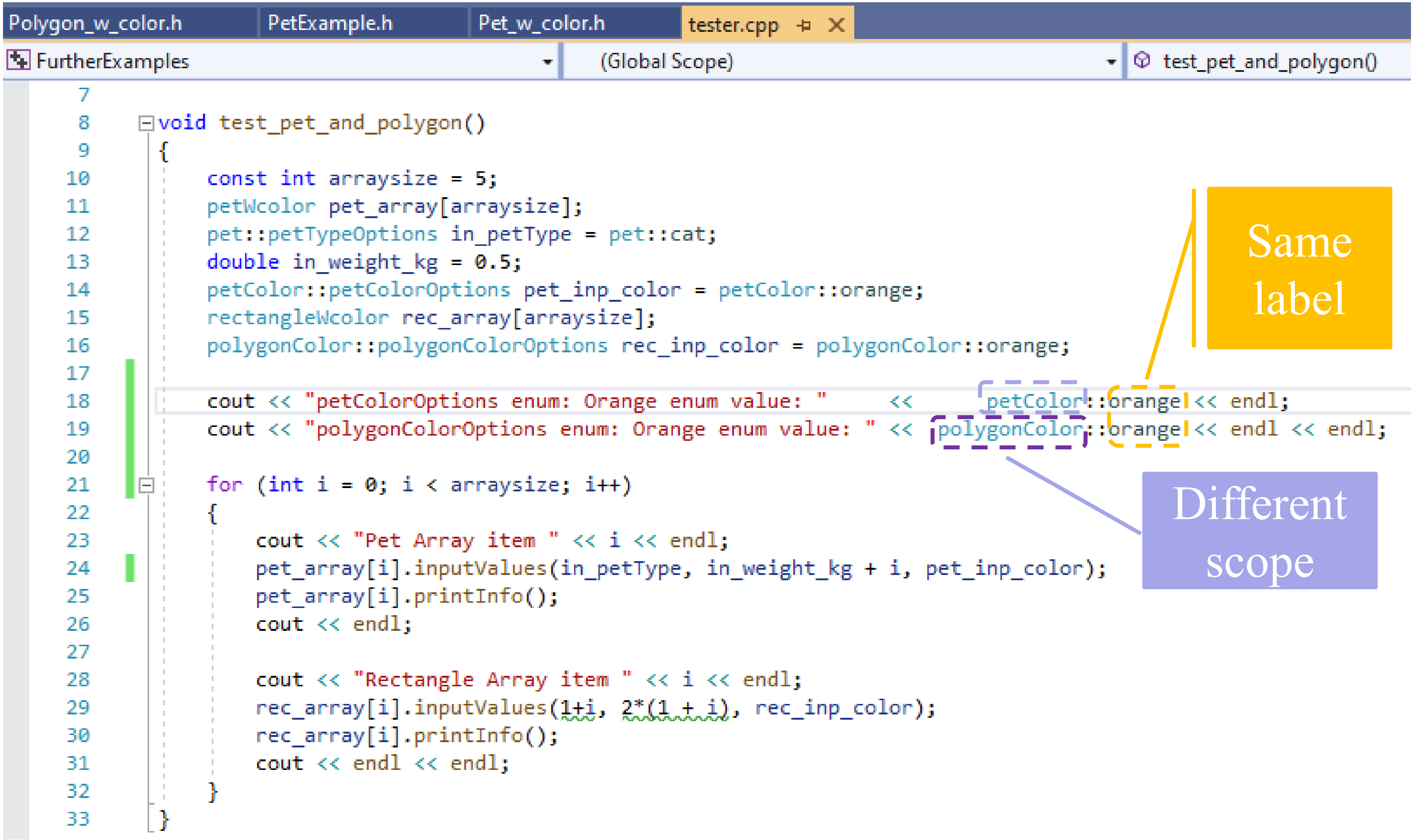
- Outside class member functions: specify class name with the scoping operator (::)



```cpp

void test_pet_and_polygon()
{
    const int arraysize = 5;
    petWcolor pet_array[arraysize];
    pet::petTypeOptions in_petType = pet::cat;
    double in_weight_kg = 0.5;
    petColor::petColorOptions pet_inp_color = petColor::orange;
    rectangleWcolor rec_array[arraysize];
    polygonColor::polygonColorOptions rec_inp_color = polygonColor::orange;

    cout << "petColorOptions enum: Orange enum value: "     <<    petColor::orange << endl;
    cout << "polygonColorOptions enum: Orange enum value: " <<  polygonColor::orange << endl << endl;

    for (int i = 0; i < arraysize; i++)
    {
        cout << "Pet Array item " << i << endl;
        pet_array[i].inputValues(in_petType, in_weight_kg + i, pet_inp_color);
        pet_array[i].printInfo();
        cout << endl;

        cout << "Rectangle Array item " << i << endl;
        rec_array[i].inputValues(1+i, 2*(1 + i), rec_inp_color);
        rec_array[i].printInfo();
        cout << endl << endl;
    }
}
```

# Scoping examples

- Outside class member functions: specify class name with the scoping operator (::)



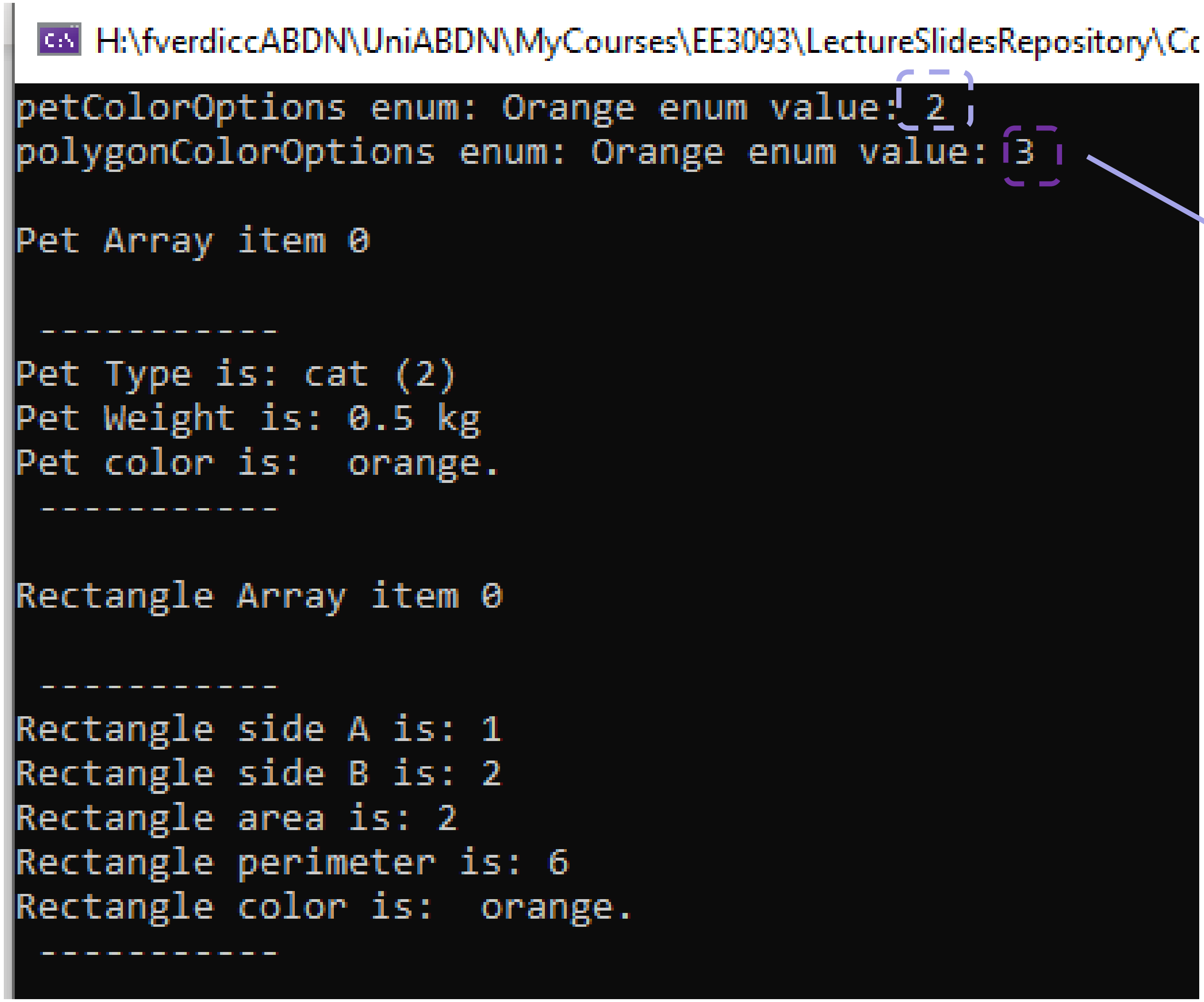
```
void test_pet_and_polygon()
{
    const int arraysize = 5;
    petWcolor pet_array[arraysize];
    pet::petTypeOptions in_petType = pet::cat;
    double in_weight_kg = 0.5;
    petColor::petColorOptions pet_inp_color = petColor::orange;
    rectangleWcolor rec_array[arraysize];
    polygonColor::polygonColorOptions rec_inp_color = polygonColor::orange;

    cout << "petColorOptions enum: Orange enum value: "     <<       petColor::orange << endl;
    cout << "polygonColorOptions enum: Orange enum value: " << polygonColor::orange << endl << endl;

    for (int i = 0; i < arraysize; i++)
    {
        cout << "Pet Array item " << i << endl;
        pet_array[i].inputValues(in_petType, in_weight_kg + i, pet_inp_color);
        pet_array[i].printInfo();
        cout << endl;

        cout << "Rectangle Array item " << i << endl;
        rec_array[i].inputValues(1+i, 2*(1 + i), rec_inp_color);
        rec_array[i].printInfo();
        cout << endl << endl;
    }
}
```

Same label

Different scope

# Scoping examples

- Outside class member functions: specify class name with the scoping operator (::)



H:\fverdiccABDN\UniABDN\MyCourses\EE3093\LectureSlidesRepository\Cc

```
petColorOptions enum: Orange enum value: 2
polygonColorOptions enum: Orange enum value: 3

Pet Array item 0

 -----------
Pet Type is: cat (2)
Pet Weight is: 0.5 kg
Pet color is:  orange.
 -----------

Rectangle Array item 0

 -----------
Rectangle side A is: 1
Rectangle side B is: 2
Rectangle area is: 2
Rectangle perimeter is: 6
Rectangle color is:  orange.
 -----------
```

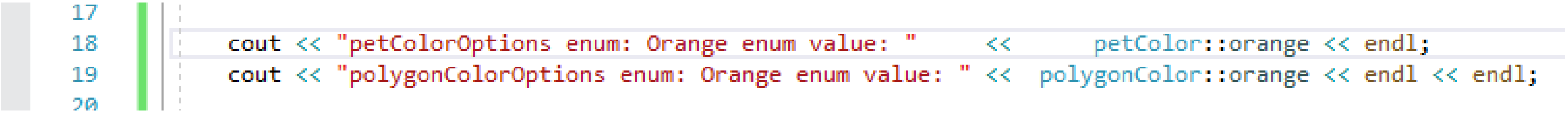
Different (numerical) values

# Any question?

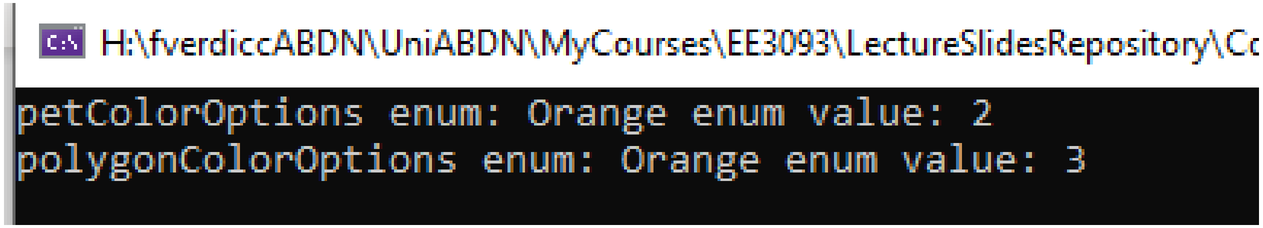
Otherwise, let’s go on

# Printing variable to screen

- Printing variable of type *enum* using *cout* can only show the numerical value



```
    cout << "petColorOptions enum: Orange enum value: "    <<     petColor::orange << endl;
    cout << "polygonColorOptions enum: Orange enum value: " << polygonColor::orange << endl << endl;
```

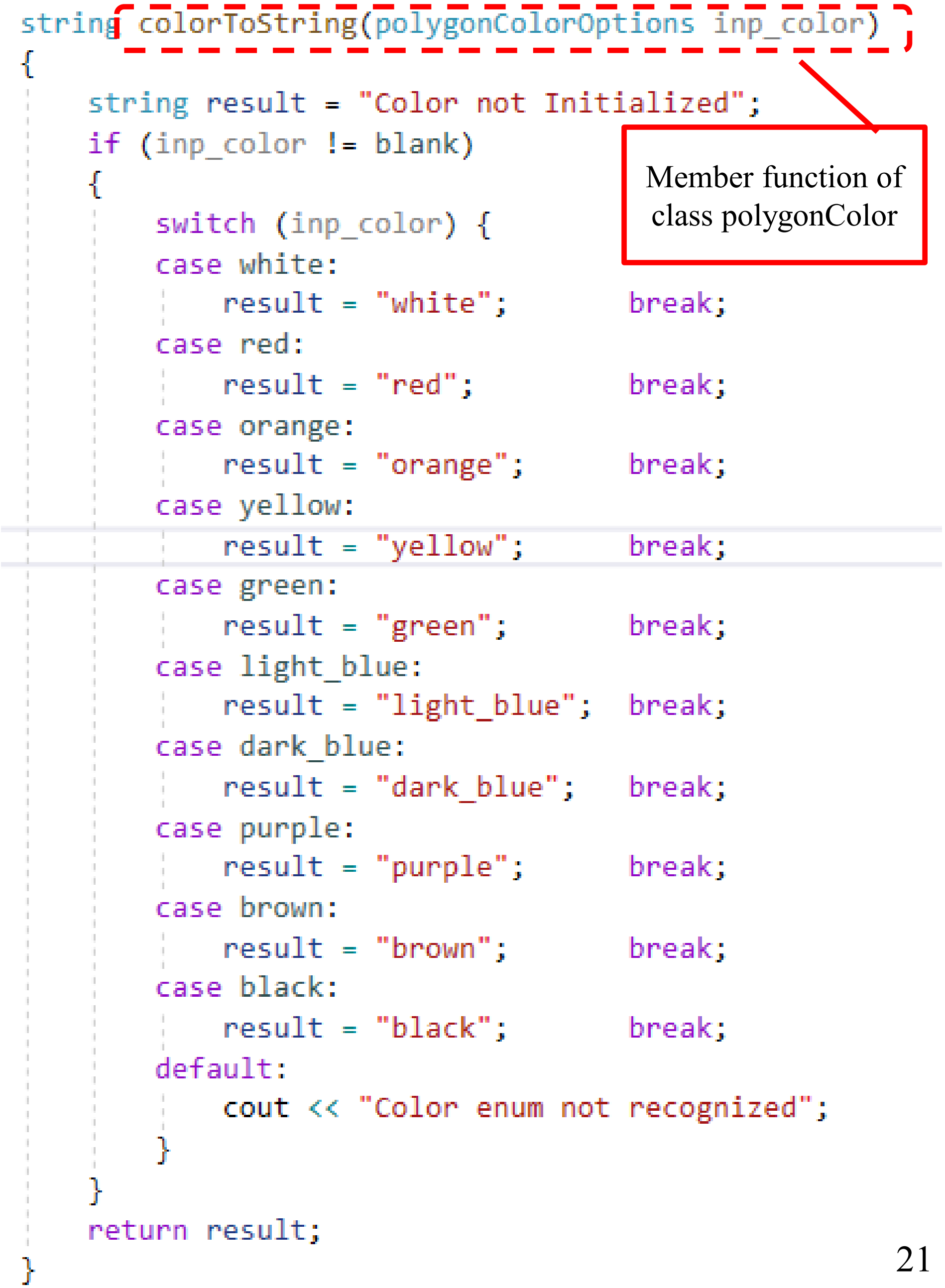


H:\fverdiccABDN\UniABDN\MyCourses\EE3093\LectureSlidesRepository\Cc

```
petColorOptions enum: Orange enum value: 2
polygonColorOptions enum: Orange enum value: 3
```

# Printing variable to screen

- Better still: a member function to output a *string* object with the “name” of an input color



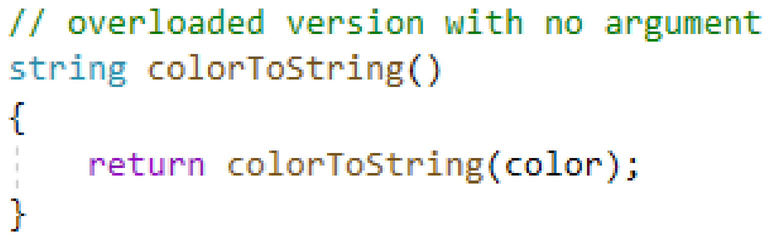
```
string colorToString(polygonColorOptions inp_color)
{
    string result = "Color not Initialized";
    if (inp_color != blank)
    {
        switch (inp_color) {
        case white:
            result = "white";       break;
        case red:
            result = "red";         break;
        case orange:
            result = "orange";      break;
        case yellow:
            result = "yellow";      break;
        case green:
            result = "green";       break;
        case light_blue:
            result = "light_blue";  break;
        case dark_blue:
            result = "dark_blue";   break;
        case purple:
            result = "purple";      break;
        case brown:
            result = "brown";       break;
        case black:
            result = "black";       break;
        default:
            cout << "Color enum not recognized";
        }
    }
    return result;
}
```

Member function of class polygonColor

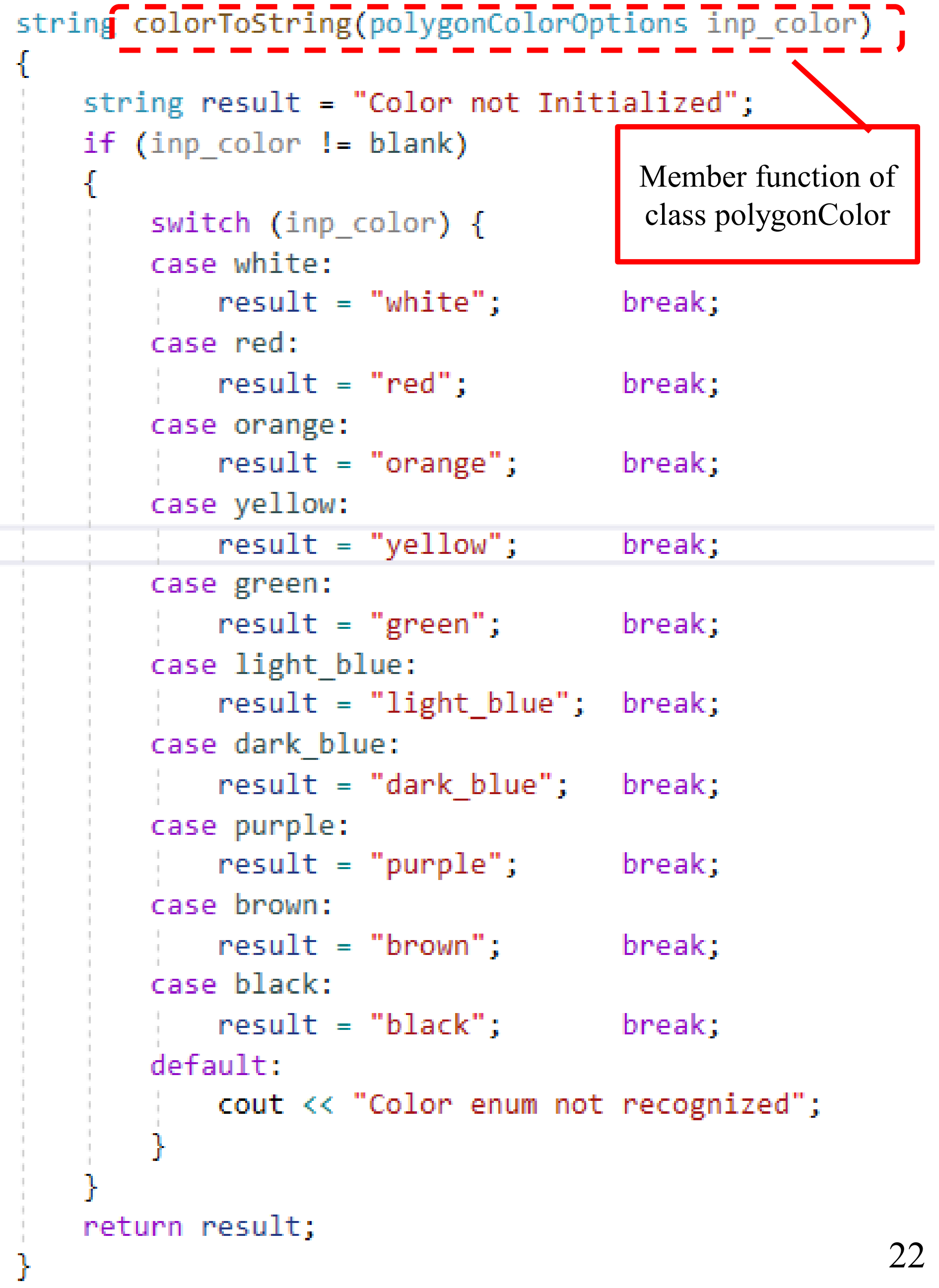
# Printing variable to screen

- Better still: a member function to output a *string* object with the “name” of an input color

We can overload it so that it returns the object’s current color (name)



```
// overloaded version with no argument
string colorToString()
{
    return colorToString(color);
}
```



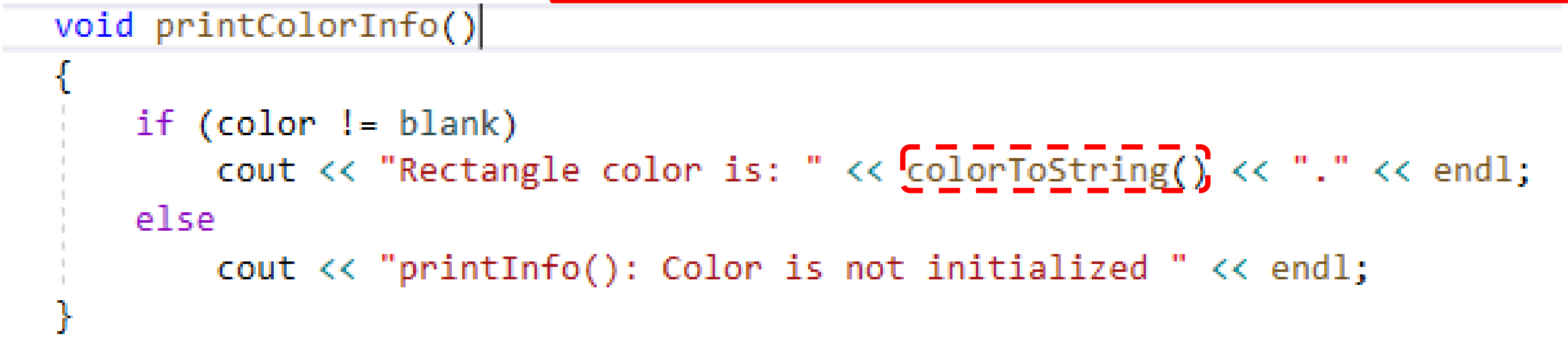
```
string colorToString(polygonColorOptions inp_color)
{
    string result = "Color not Initialized";
    if (inp_color != blank)
    {
        switch (inp_color) {
        case white:
            result = "white";       break;
        case red:
            result = "red";         break;
        case orange:
            result = "orange";      break;
        case yellow:
            result = "yellow";      break;
        case green:
            result = "green";       break;
        case light_blue:
            result = "light_blue";  break;
        case dark_blue:
            result = "dark_blue";   break;
        case purple:
            result = "purple";      break;
        case brown:
            result = "brown";       break;
        case black:
            result = "black";       break;
        default:
            cout << "Color enum not recognized";
        }
    }
    return result;
}
```

Member function of class polygonColor

# Printing variable to screen

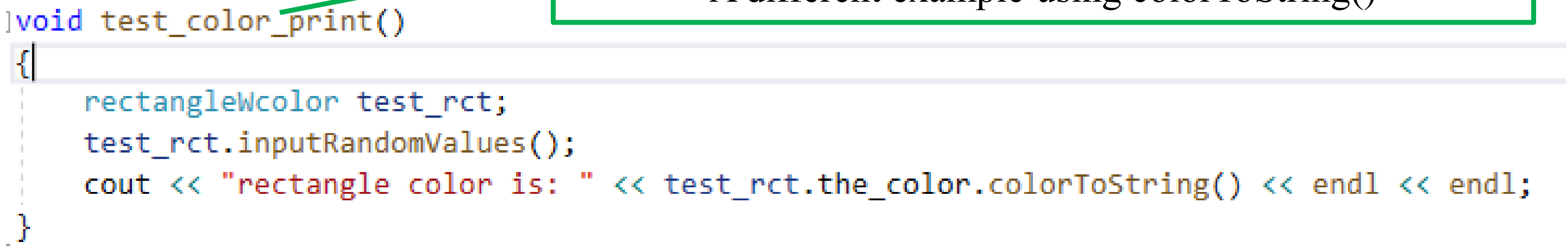
- A member function to output a *string* object with the  colour name

Alternative implementation of printColorInfo()  using colorToString()

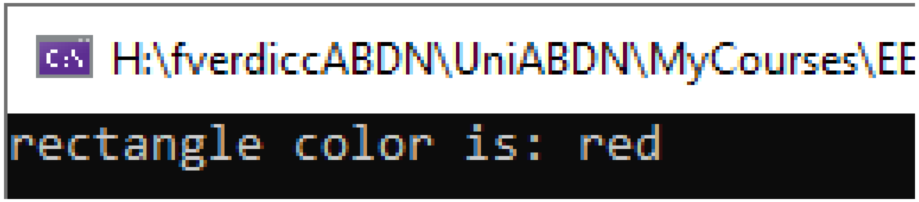


```
void printColorInfo()
{
    if (color != blank)
        cout << "Rectangle color is: " << colorToString() << "." << endl;
    else
        cout << "printInfo(): Color is not initialized " << endl;
}
```

A different example using colorToString()



```
void test_color_print()
{
    rectangleWcolor test_rct;
    test_rct.inputRandomValues();
    cout << "rectangle color is: " << test_rct.the_color.colorToString() << endl << endl;
}
```



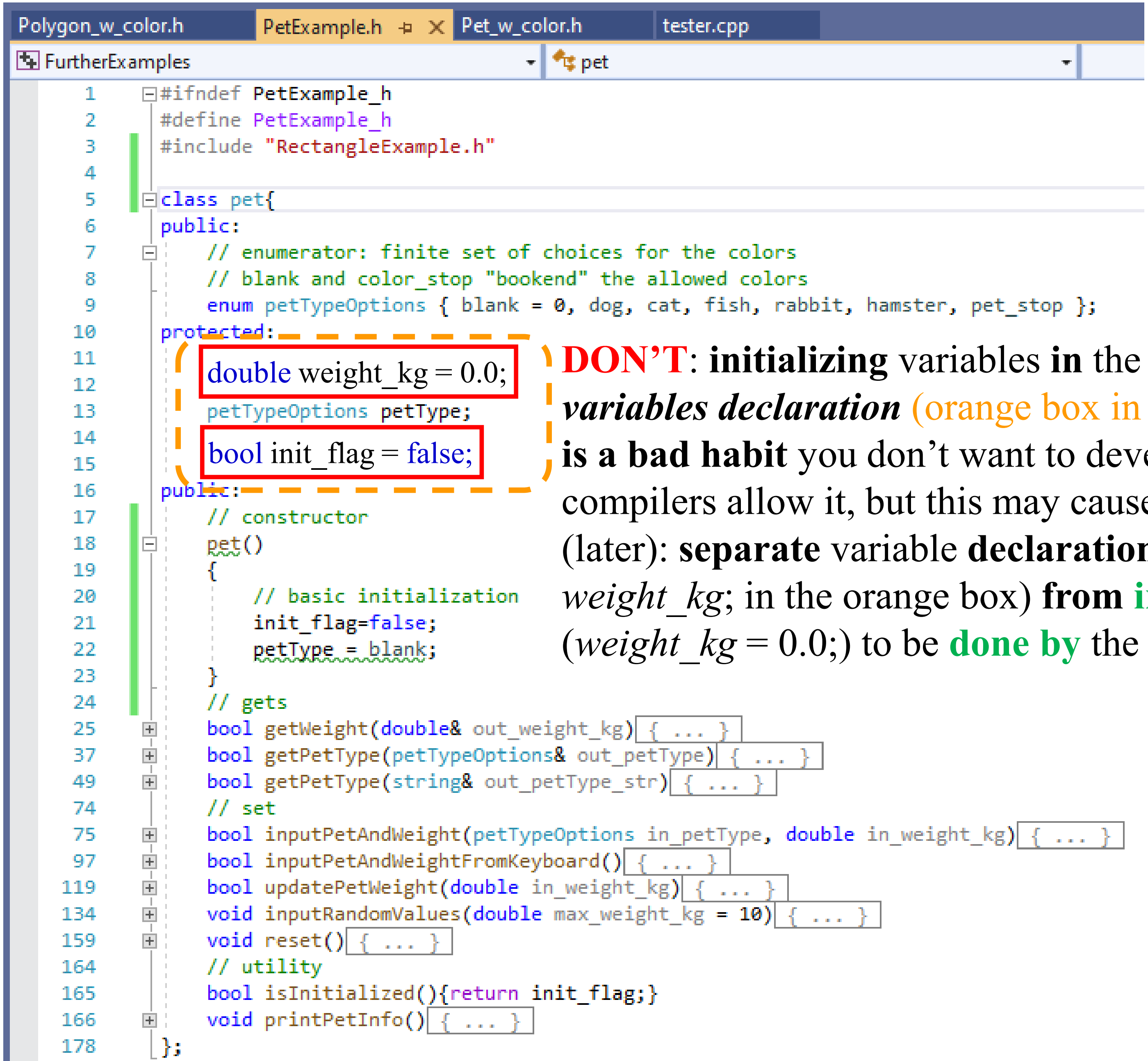
```
H:\fverdiccABDN\UniABDN\MyCourses\EE
rectangle color is: red
```

(This approach is completed by defining *typecast* operators for *strings*; we'll introduce those later in the course: "stay tuned!")

# Any question?

Otherwise, let’s go on

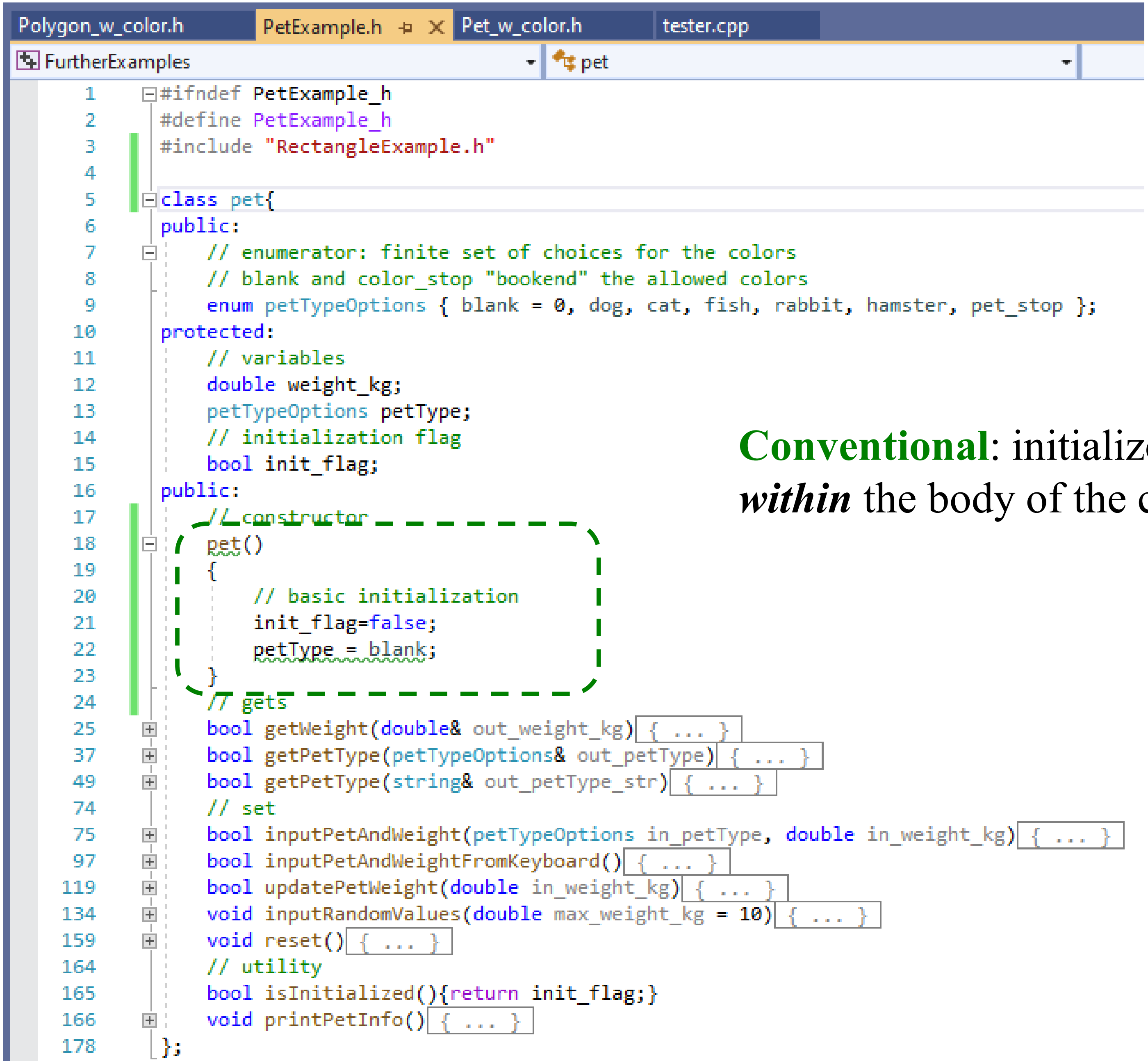
# Initializing values



```
#ifndef PetExample_h
#define PetExample_h
#include "RectangleExample.h"

class pet{
public:
    // enumerator: finite set of choices for the colors
    // blank and color_stop "bookend" the allowed colors
    enum petTypeOptions { blank = 0, dog, cat, fish, rabbit, hamster, pet_stop };
protected:

    double weight_kg = 0.0;
    petTypeOptions petType;

    bool init_flag = false;
public:
    // constructor
    pet()
    {
        // basic initialization
        init_flag=false;
        petType = blank;
    }
    // gets
    bool getWeight(double& out_weight_kg){ ... }
    bool getPetType(petTypeOptions& out_petType){ ... }
    bool getPetType(string& out_petType_str){ ... }
    // set
    bool inputPetAndWeight(petTypeOptions in_petType, double in_weight_kg){ ... }
    bool inputPetAndWeightFromKeyboard(){ ... }
    bool updatePetWeight(double in_weight_kg){ ... }
    void inputRandomValues(double max_weight_kg = 10){ ... }
    void reset(){ ... }
    // utility
    bool isInitialized(){return init_flag;}
    void printPetInfo(){ ... }
};
```

**DON'T**: **initializing** variables **in** the ***member variables declaration*** (orange box in this example) **is a bad habit** you don't want to develop. Some compilers allow it, but this may cause problems (later): **separate** variable **declaration** (*double weight_kg*; in the orange box) **from initialization** (*weight_kg* = 0.0;) to be **done by** the **constructor**

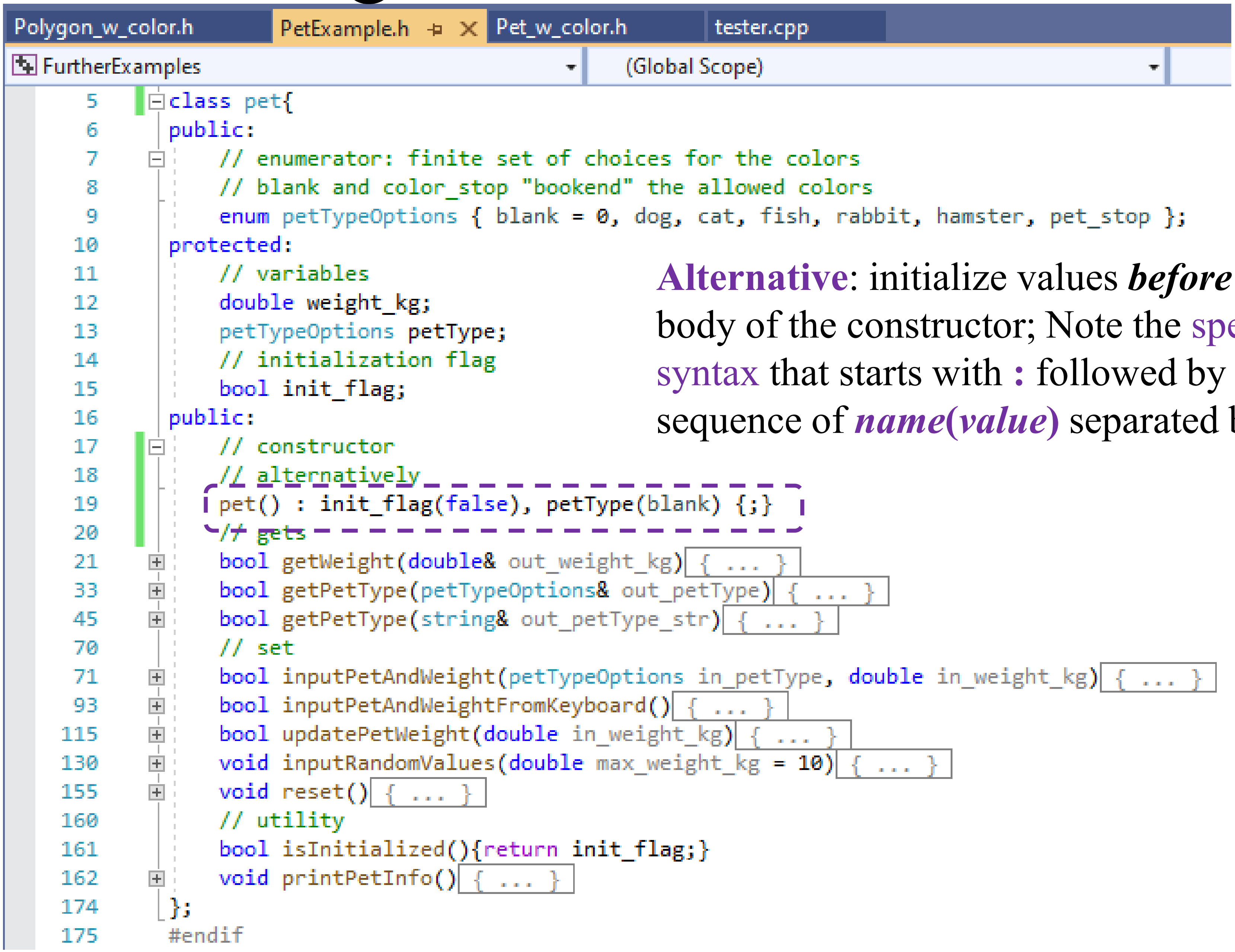
# Initializing values with constructors



```cpp
#ifndef PetExample_h
#define PetExample_h
#include "RectangleExample.h"

class pet{
public:
    // enumerator: finite set of choices for the colors
    // blank and color_stop "bookend" the allowed colors
    enum petTypeOptions { blank = 0, dog, cat, fish, rabbit, hamster, pet_stop };
protected:
    // variables
    double weight_kg;
    petTypeOptions petType;
    // initialization flag
    bool init_flag;
public:
    // constructor
    pet()
    {
        // basic initialization
        init_flag=false;
        petType = blank;
    }
    // gets
    bool getWeight(double& out_weight_kg){ ... }
    bool getPetType(petTypeOptions& out_petType){ ... }
    bool getPetType(string& out_petType_str){ ... }
    // set
    bool inputPetAndWeight(petTypeOptions in_petType, double in_weight_kg){ ... }
    bool inputPetAndWeightFromKeyboard(){ ... }
    bool updatePetWeight(double in_weight_kg){ ... }
    void inputRandomValues(double max_weight_kg = 10){ ... }
    void reset(){ ... }
    // utility
    bool isInitialized(){return init_flag;}
    void printPetInfo(){ ... }
};
```

**Conventional**: initialize values ***within*** the body of the constructor

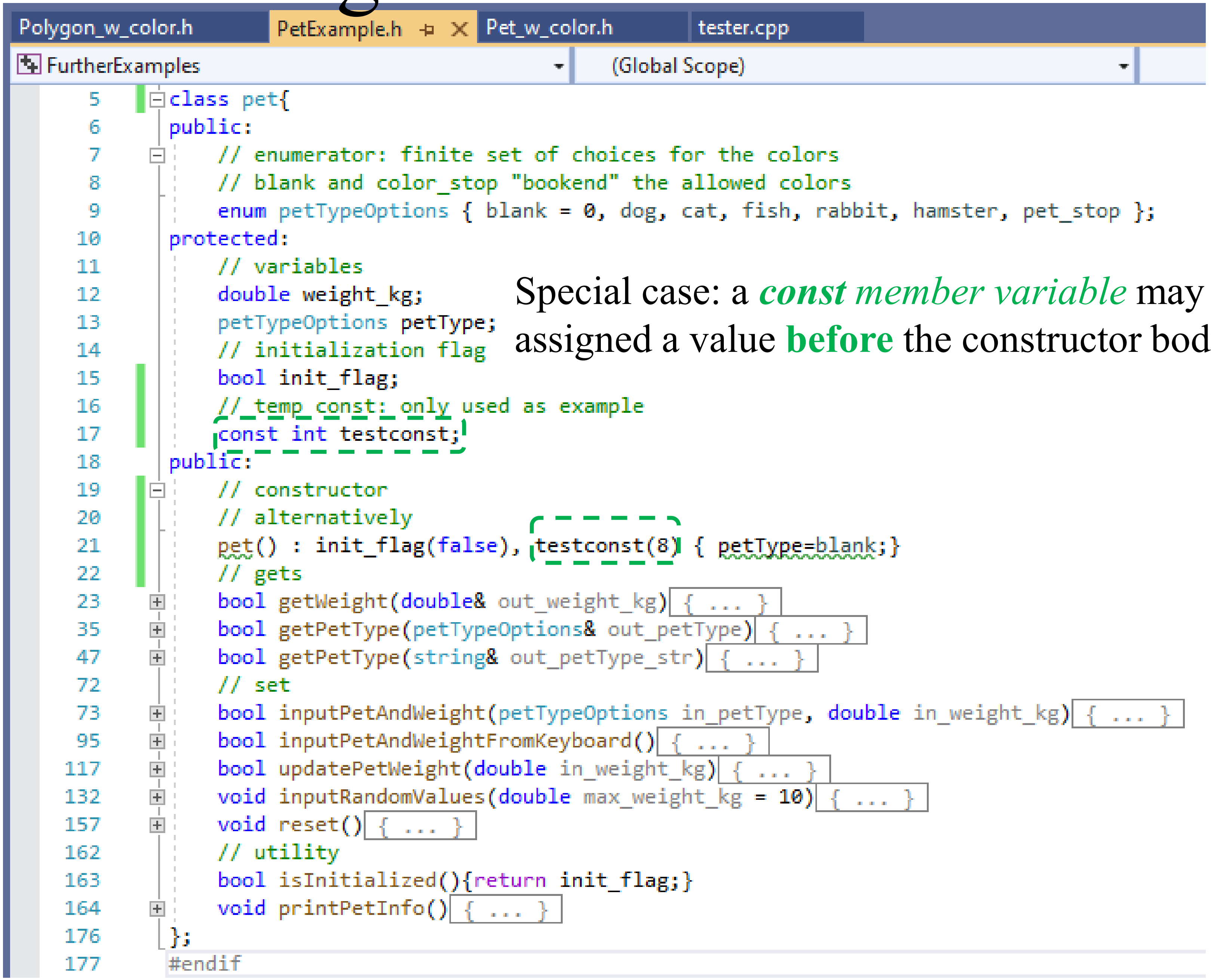
# Initializing values with constructors



```
class pet{
public:
    // enumerator: finite set of choices for the colors
    // blank and color_stop "bookend" the allowed colors
    enum petTypeOptions { blank = 0, dog, cat, fish, rabbit, hamster, pet_stop };
protected:
    // variables
    double weight_kg;
    petTypeOptions petType;
    // initialization flag
    bool init_flag;
public:
    // constructor
    // alternatively
    pet() : init_flag(false), petType(blank) {;}
    // gets
    bool getWeight(double& out_weight_kg) { ... }
    bool getPetType(petTypeOptions& out_petType) { ... }
    bool getPetType(string& out_petType_str) { ... }
    // set
    bool inputPetAndWeight(petTypeOptions in_petType, double in_weight_kg) { ... }
    bool inputPetAndWeightFromKeyboard() { ... }
    bool updatePetWeight(double in_weight_kg) { ... }
    void inputRandomValues(double max_weight_kg = 10) { ... }
    void reset() { ... }
    // utility
    bool isInitialized(){return init_flag;}
    void printPetInfo() { ... }
};
#endif
```

**Alternative**: initialize values ***before*** the body of the constructor; Note the special syntax that starts with **:** followed by a sequence of ***name*(*value*)** separated by **,**

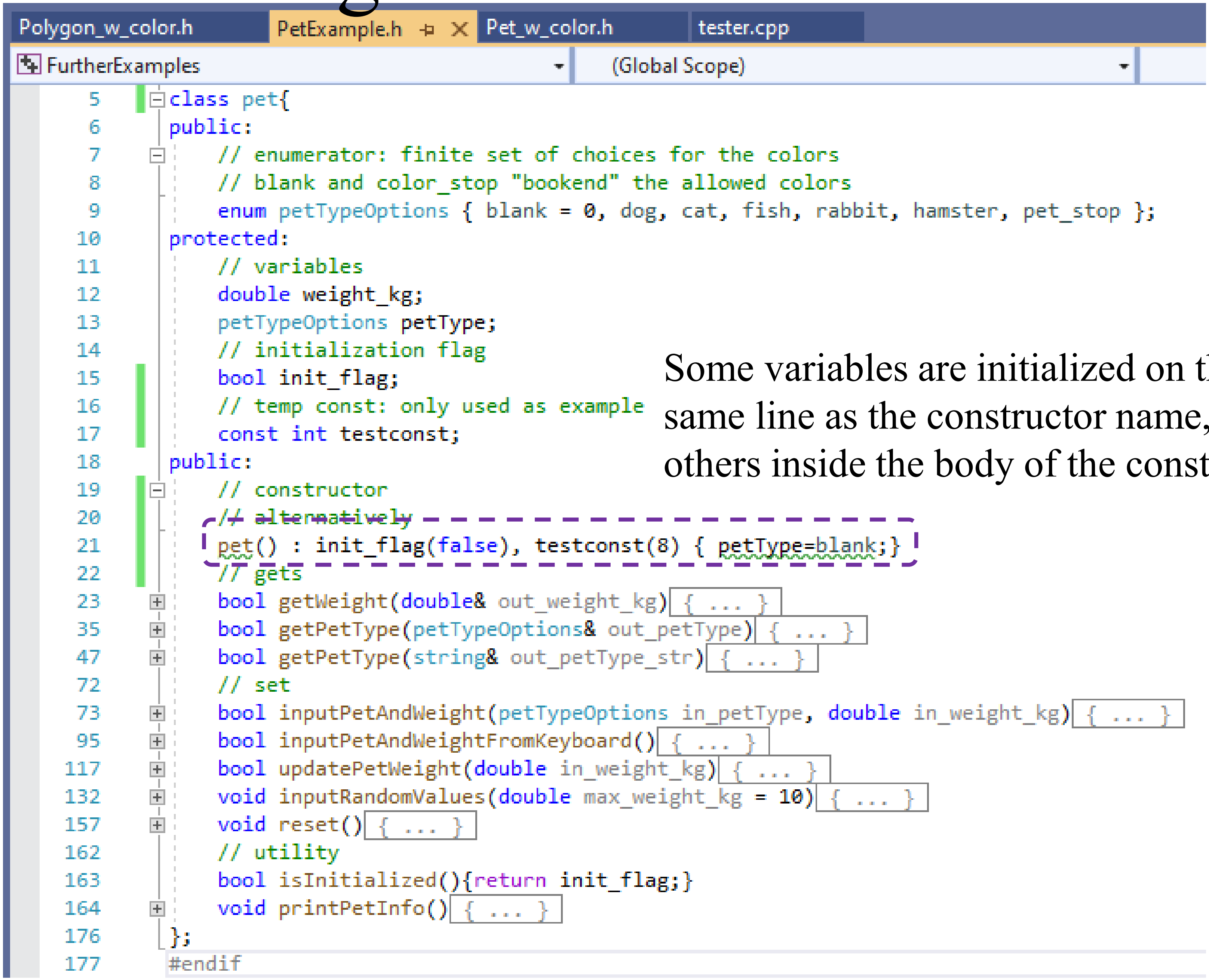
# Initializing values with constructors



```cpp
class pet{
public:
    // enumerator: finite set of choices for the colors
    // blank and color_stop "bookend" the allowed colors
    enum petTypeOptions { blank = 0, dog, cat, fish, rabbit, hamster, pet_stop };
protected:
    // variables
    double weight_kg;
    petTypeOptions petType;
    // initialization flag
    bool init_flag;
    //_temp_const:_only used as example
    const int testconst;
public:
    // constructor
    // alternatively
    pet() : init_flag(false), testconst(8) { petType=blank;}
    // gets
    bool getWeight(double& out_weight_kg) { ... }
    bool getPetType(petTypeOptions& out_petType) { ... }
    bool getPetType(string& out_petType_str) { ... }
    // set
    bool inputPetAndWeight(petTypeOptions in_petType, double in_weight_kg) { ... }
    bool inputPetAndWeightFromKeyboard() { ... }
    bool updatePetWeight(double in_weight_kg) { ... }
    void inputRandomValues(double max_weight_kg = 10) { ... }
    void reset() { ... }
    // utility
    bool isInitialized(){return init_flag;}
    void printPetInfo() { ... }
};
#endif
```

Special case: a ***const** member variable* may only be assigned a value **before** the constructor body

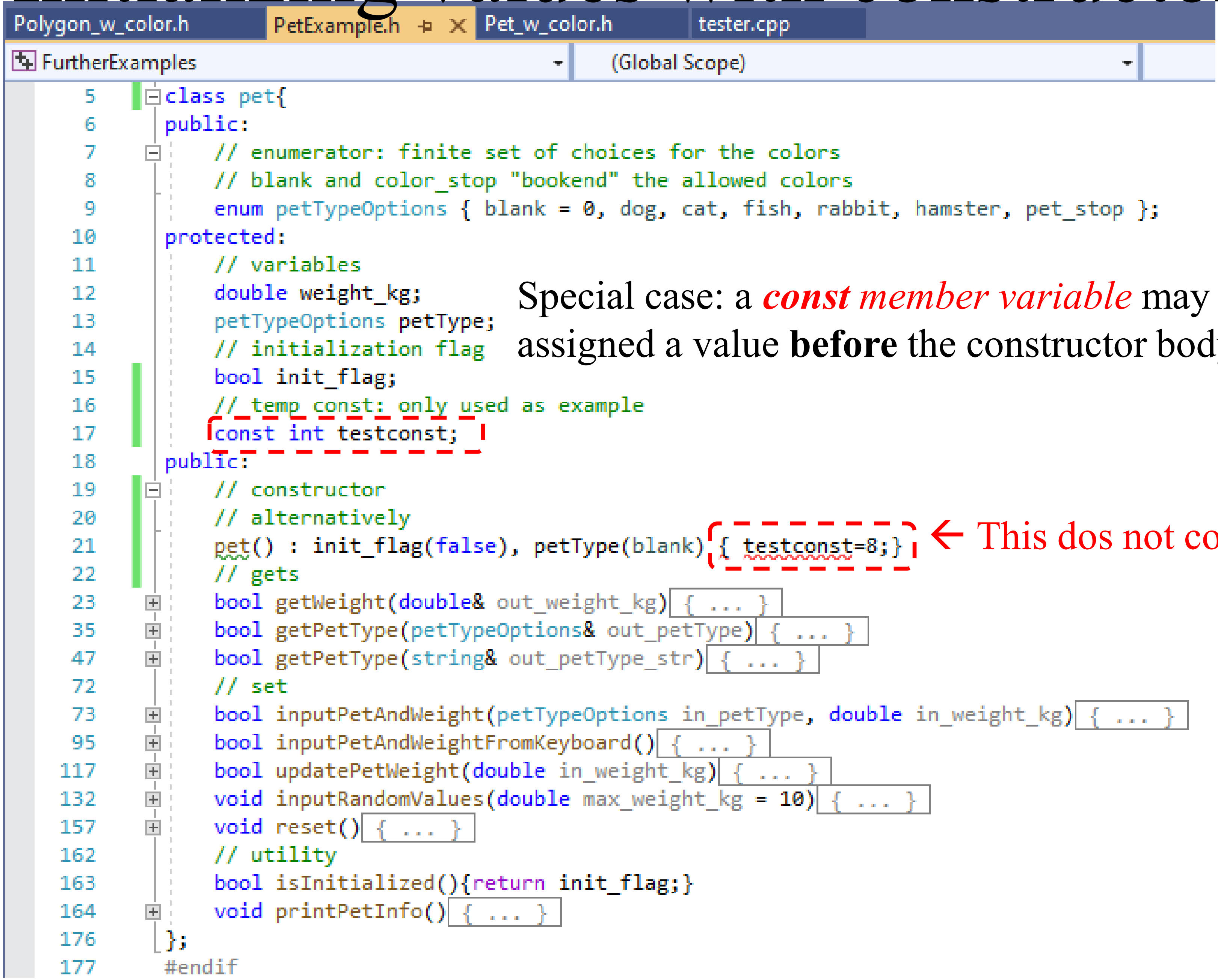
# Initializing values with constructors



```cpp
class pet{
public:
    // enumerator: finite set of choices for the colors
    // blank and color_stop "bookend" the allowed colors
    enum petTypeOptions { blank = 0, dog, cat, fish, rabbit, hamster, pet_stop };
protected:
    // variables
    double weight_kg;
    petTypeOptions petType;
    // initialization flag
    bool init_flag;
    // temp const: only used as example
    const int testconst;
public:
    // constructor
    // alternatively
    pet() : init_flag(false), testconst(8) { petType=blank;}
    // gets
    bool getWeight(double& out_weight_kg){ ... }
    bool getPetType(petTypeOptions& out_petType){ ... }
    bool getPetType(string& out_petType_str){ ... }
    // set
    bool inputPetAndWeight(petTypeOptions in_petType, double in_weight_kg){ ... }
    bool inputPetAndWeightFromKeyboard(){ ... }
    bool updatePetWeight(double in_weight_kg){ ... }
    void inputRandomValues(double max_weight_kg = 10){ ... }
    void reset(){ ... }
    // utility
    bool isInitialized(){return init_flag;}
    void printPetInfo(){ ... }
};
#endif
```

Some variables are initialized on the same line as the constructor name, others inside the body of the constructor.

# Initializing values with constructors



```
class pet{
public:
    // enumerator: finite set of choices for the colors
    // blank and color_stop "bookend" the allowed colors
    enum petTypeOptions { blank = 0, dog, cat, fish, rabbit, hamster, pet_stop };
protected:
    // variables
    double weight_kg;
    petTypeOptions petType;
    // initialization flag
    bool init_flag;
    // temp const: only used as example
    const int testconst;
public:
    // constructor
    // alternatively
    pet() : init_flag(false), petType(blank){ testconst=8;}
    // gets
    bool getWeight(double& out_weight_kg) { ... }
    bool getPetType(petTypeOptions& out_petType) { ... }
    bool getPetType(string& out_petType_str) { ... }
    // set
    bool inputPetAndWeight(petTypeOptions in_petType, double in_weight_kg) { ... }
    bool inputPetAndWeightFromKeyboard() { ... }
    bool updatePetWeight(double in_weight_kg) { ... }
    void inputRandomValues(double max_weight_kg = 10) { ... }
    void reset() { ... }
    // utility
    bool isInitialized(){return init_flag;}
    void printPetInfo() { ... }
};
#endif
```

Special case: a ***const*** *member variable* may only be assigned a value **before** the constructor body

← This dos not compile

# Any question?

Otherwise, let’s go on