

C/C++ Programming: Further Examples



The screenshot shows a C++ IDE with a project structure on the left and a code editor 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 code editor displays the content of `RectangleExample.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, 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){ ... }
};
```

Overloading a member function

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

From file: RectangleExample.h

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();  
}
```

C:\ 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) { ... }
```

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

```
// selects random values for each side, within [0, max_val]  
void inputRandomSides(double min_val, double max_val) {
```

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

```
{  
    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 required 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;  
}
```

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
```

public: Overloading a member function

```
// 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**:

public: Overloading a member function

```
// 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:

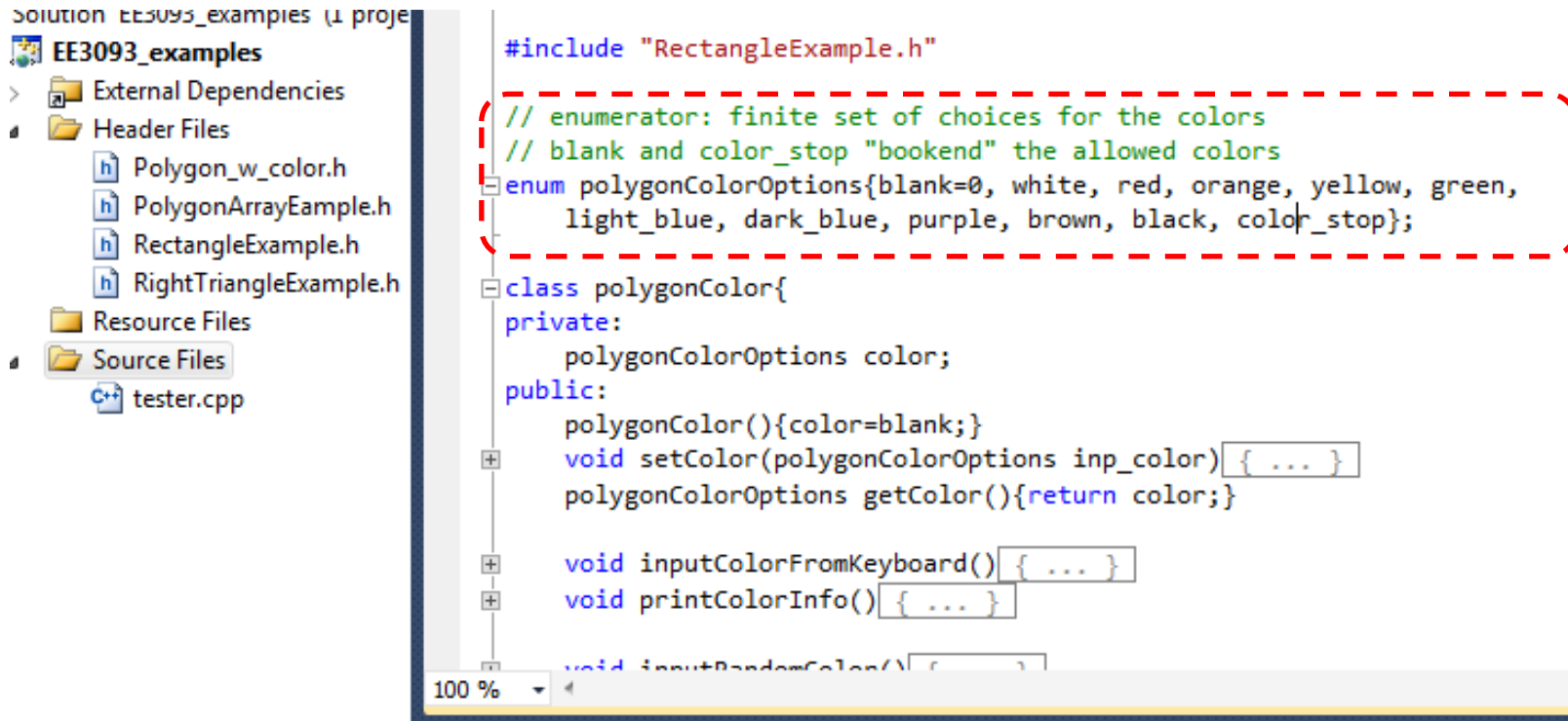
```
test_rc.inputRandomSides(10);
```

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

Any question?



Scoping examples



```
#include "RectangleExample.h"

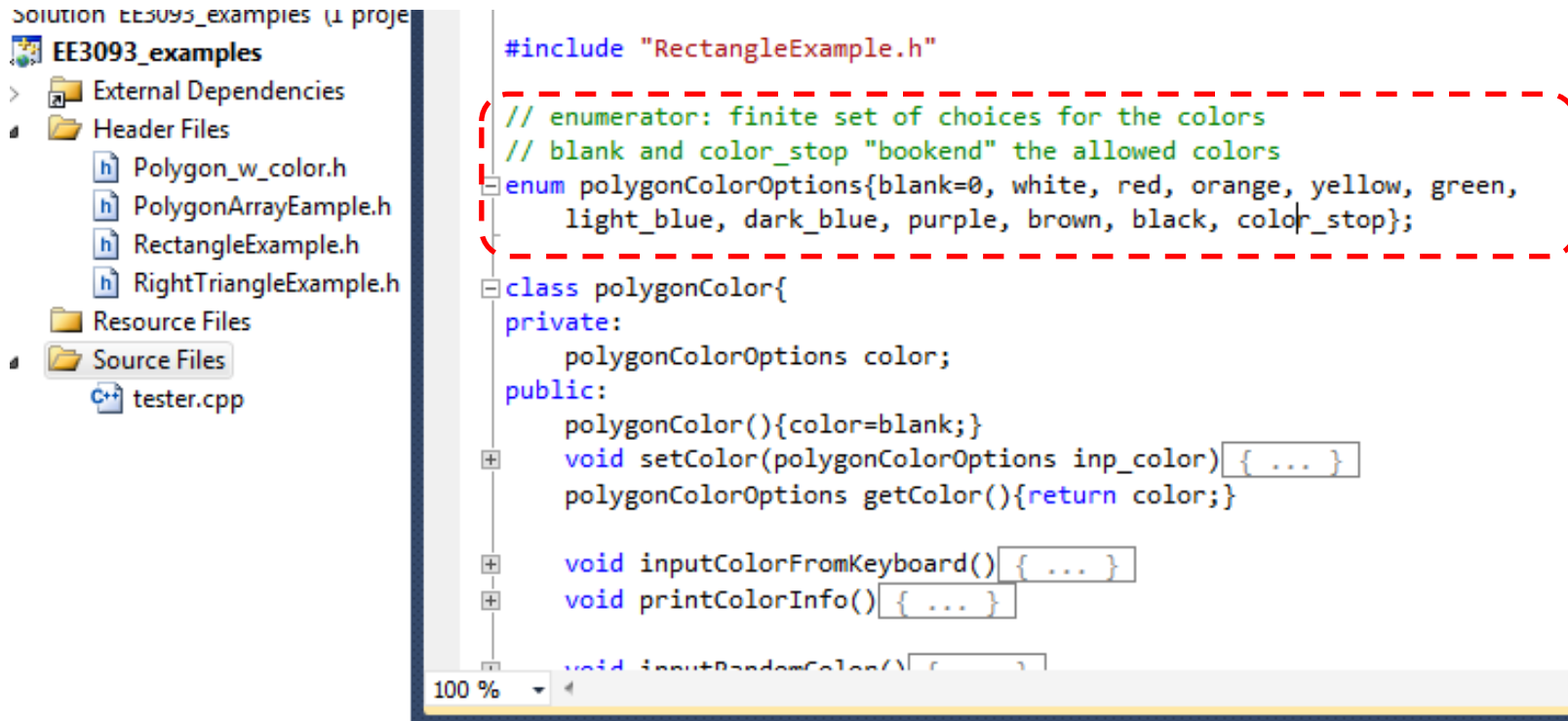
// enumerator: finite set of choices for the colors
// blank and color_stop "bookend" the allowed colors
enum polygonColorOptions{blank=0, white, red, orange, yellow, green,
    light_blue, dark_blue, purple, brown, black, color_stop};

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-style) 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-style) 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)

```
Polygon_w_color.h  Pet_w_color.h  tester.cpp
FurtherExamples  polygonColor  setColor(polygonCo

1  #ifndef PolygonWcolor_h
2      #define PolygonWcolor_h
3
4  #include "RectangleExample.h"
5  #include "RightTriangleExample.h"
6
7
8  class polygonColor{
9      public:
10         ( // enumerator: finite set of choices for the colors
11         | // blank and color_stop "bookend" the allowed colors
12         | enum polygonColorOptions{blank=0, white, red, orange, yellow, green,
13         | light_blue, dark_blue, purple, brown, black, color_stop};
14     private:
15         polygonColorOptions color;
16     public:
17         polygonColor(){color=blank;}
18         void setColor(polygonColorOptions inp_color){ ... }
28         polygonColorOptions getColor(){return color;}
29
30         void inputColorFromKeyboard(){ ... }
52         void printColorInfo(){ ... }
95
96         void inputRandomColor(){ ... }
105     };
106
107     class rectangleWcolor{
108     public:
109         rectangle the_rectangle;
110         polygonColor the_color;
```

Scoping examples

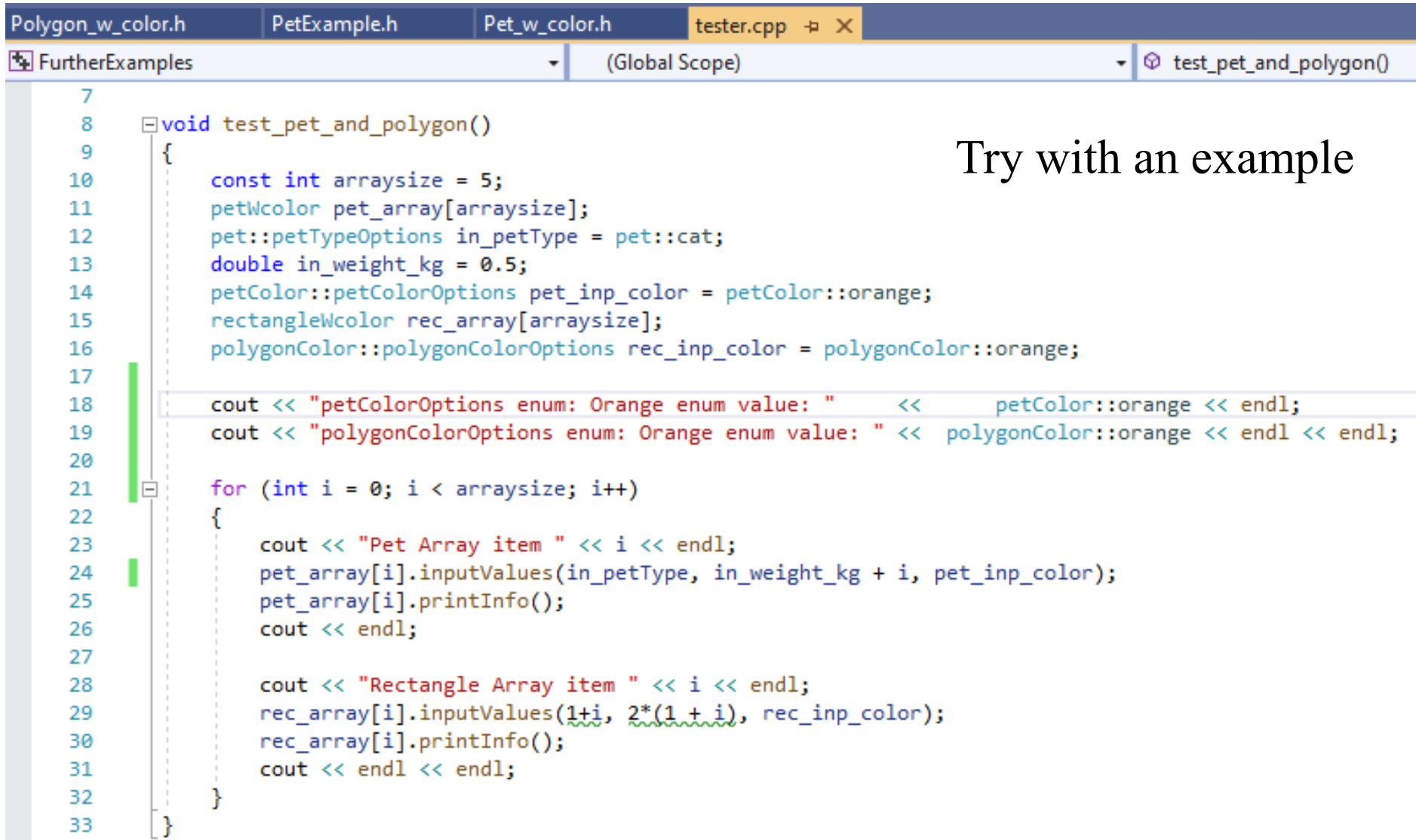
- Define the *enum* **within** the corresponding class (petColor)

```
Polygon_w_color.h  RightTriangleExample.h  RectangleExample.h  PetExample.h  Pet_w_color.h  X  tester.cpp
FurtherExamples  petColor

5  class petColor{
6  public:
7      // enumerator: finite set of choices for the colors
8      // blank and color_stop "bookend" the allowed colors
9      enum petColorOptions{blank=0, white, orange, brown, black, color_stop};
10 private:
11     petColorOptions color;
12 public:
13     petColor(){color=blank;}
14     //set
15     bool setColor(petColorOptions inp_color){ ... }
30     bool inputColorFromKeyboard(){ ... }
48     void inputRandomColor(){ ... }
57     void reset(){ ... }
62     // get
63     petColorOptions getColor(){return color;}
64     // utilities
65     bool isInitialized(){ ... }
72     void printColorInfo(){ ... }
97 };
98
99 class petWcolor{
100 public:
101     pet the_pet;
102     petColor the_color;
103 public:
104     // set
105     void inputFromKeyboard(){ ... }
122     bool inputValues(pet::petTypeOptions in_petType, double in_weight_kg, petColor::petColorOptions inp_color){ ... }
136     void inputRandomValues(double max_val = 10){ ... }
141     void reset(){ ... }
146     //utilities
147     void printInfo(){ ... }
159     bool isInitialized(){ ... }
166 };
```

Scoping examples

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



```
Polygon_w_color.h  PetExample.h  Pet_w_color.h  tester.cpp  X
FurtherExamples  (Global Scope)  test_pet_and_polygon()

7
8 void test_pet_and_polygon()
9 {
10     const int arraysize = 5;
11     petWcolor pet_array[arraysize];
12     pet::petTypeOptions in_petType = pet::cat;
13     double in_weight_kg = 0.5;
14     petColor::petColorOptions pet_inp_color = petColor::orange;
15     rectangleWcolor rec_array[arraysize];
16     polygonColor::polygonColorOptions rec_inp_color = polygonColor::orange;
17
18     cout << "petColorOptions enum: Orange enum value: " << petColor::orange << endl;
19     cout << "polygonColorOptions enum: Orange enum value: " << polygonColor::orange << endl << endl;
20
21     for (int i = 0; i < arraysize; i++)
22     {
23         cout << "Pet Array item " << i << endl;
24         pet_array[i].inputValues(in_petType, in_weight_kg + i, pet_inp_color);
25         pet_array[i].printInfo();
26         cout << endl;
27
28         cout << "Rectangle Array item " << i << endl;
29         rec_array[i].inputValues(1+i, 2*(1+i), rec_inp_color);
30         rec_array[i].printInfo();
31         cout << endl << endl;
32     }
33 }
```

Try with an example

Scoping examples

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

```
7
8 void test_pet_and_polygon()
9 {
10     const int arraysize = 5;
11     petWcolor pet_array[arraysize];
12     pet::petTypeOptions in_petType = pet::cat;
13     double in_weight_kg = 0.5;
14     petColor::petColorOptions pet_inp_color = petColor::orange;
15     rectangleWcolor rec_array[arraysize];
16     polygonColor::polygonColorOptions rec_inp_color = polygonColor::orange;
17
18     cout << "petColorOptions enum: Orange enum value: " << petColor::orange << endl;
19     cout << "polygonColorOptions enum: Orange enum value: " << polygonColor::orange << endl << endl;
20
21     for (int i = 0; i < arraysize; i++)
22     {
23         cout << "Pet Array item " << i << endl;
24         pet_array[i].inputValues(in_petType, in_weight_kg + i, pet_inp_color);
25         pet_array[i].printInfo();
26         cout << endl;
27
28         cout << "Rectangle Array item " << i << endl;
29         rec_array[i].inputValues(1+i, 2*(1+i), rec_inp_color);
30         rec_array[i].printInfo();
31         cout << endl << endl;
32     }
33 }
```

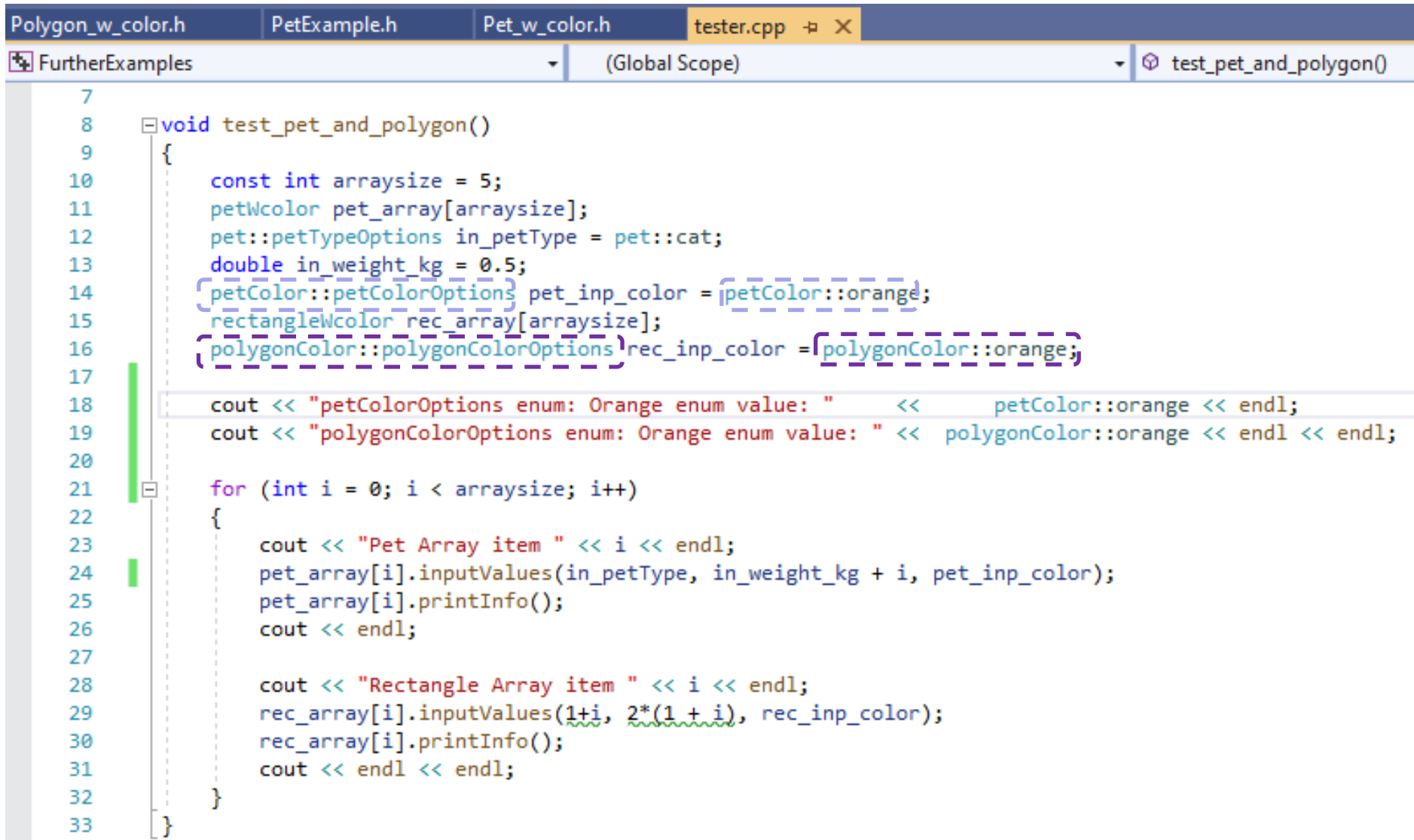
Scoping examples

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

```
7
8 void test_pet_and_polygon()
9 {
10     const int arraysize = 5;
11     petWcolor pet_array[arraysize];
12     pet::petTypeOptions in_petType = pet::cat;
13     double in_weight_kg = 0.5;
14     [petColor::petColorOptions] pet_inp_color = petColor::orange;
15     rectangleWcolor rec_array[arraysize];
16     [polygonColor::polygonColorOptions] rec_inp_color = polygonColor::orange;
17
18     cout << "petColorOptions enum: Orange enum value: " << petColor::orange << endl;
19     cout << "polygonColorOptions enum: Orange enum value: " << polygonColor::orange << endl << endl;
20
21     for (int i = 0; i < arraysize; i++)
22     {
23         cout << "Pet Array item " << i << endl;
24         pet_array[i].inputValues(in_petType, in_weight_kg + i, pet_inp_color);
25         pet_array[i].printInfo();
26         cout << endl;
27
28         cout << "Rectangle Array item " << i << endl;
29         rec_array[i].inputValues(1+i, 2*(1+i), rec_inp_color);
30         rec_array[i].printInfo();
31         cout << endl << endl;
32     }
33 }
```

Scoping examples

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



The screenshot shows a C++ IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'FurtherExamples' with a sub-project '(Global Scope)'. The code editor shows the file 'tester.cpp' with the following code:

```
7
8 void test_pet_and_polygon()
9 {
10     const int arraysize = 5;
11     petWcolor pet_array[arraysize];
12     pet::petTypeOptions in_petType = pet::cat;
13     double in_weight_kg = 0.5;
14     [petColor::petColorOptions] pet_inp_color = [petColor::orange];
15     rectangleWcolor rec_array[arraysize];
16     [polygonColor::polygonColorOptions] rec_inp_color = [polygonColor::orange];
17
18     cout << "petColorOptions enum: Orange enum value: " << petColor::orange << endl;
19     cout << "polygonColorOptions enum: Orange enum value: " << polygonColor::orange << endl << endl;
20
21     for (int i = 0; i < arraysize; i++)
22     {
23         cout << "Pet Array item " << i << endl;
24         pet_array[i].inputValues(in_petType, in_weight_kg + i, pet_inp_color);
25         pet_array[i].printInfo();
26         cout << endl;
27
28         cout << "Rectangle Array item " << i << endl;
29         rec_array[i].inputValues(1+i, 2*(1+i), rec_inp_color);
30         rec_array[i].printInfo();
31         cout << endl << endl;
32     }
33 }
```


Scoping examples

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

```
Polygon_w_color.h | PetExample.h | Pet_w_color.h | tester.cpp [X]
FurtherExamples | (Global Scope) | test_pet_and_polygon()

7
8 void test_pet_and_polygon()
9 {
10     const int arraysize = 5;
11     petWcolor pet_array[arraysize];
12     pet::petTypeOptions in_petType = pet::cat;
13     double in_weight_kg = 0.5;
14     petColor::petColorOptions pet_inp_color = petColor::orange;
15     rectangleWcolor rec_array[arraysize];
16     polygonColor::polygonColorOptions rec_inp_color = polygonColor::orange;
17
18     cout << "petColorOptions enum: Orange enum value: " << petColor::orange << endl;
19     cout << "polygonColorOptions enum: Orange enum value: " << polygonColor::orange << endl << endl;
20
21     for (int i = 0; i < arraysize; i++)
22     {
23         cout << "Pet Array item " << i << endl;
24         pet_array[i].inputValues(in_petType, in_weight_kg + i, pet_inp_color);
25         pet_array[i].printInfo();
26         cout << endl;
27
28         cout << "Rectangle Array item " << i << endl;
29         rec_array[i].inputValues(1+i, 2*(1+i), rec_inp_color);
30         rec_array[i].printInfo();
31         cout << endl << endl;
32     }
33 }
```

Same label

Different scope

Scoping examples

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

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

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

Different
(numerical)
values

```
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.  
-----
```

Any question?



Otherwise, let's go on

Printing variable to screen

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

```
17
18 cout << "petColorOptions enum: Orange enum value: " << petColor::orange << endl;
19 cout << "polygonColorOptions enum: Orange enum value: " << polygonColor::orange << endl << endl;
20
```

```
C:\ 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

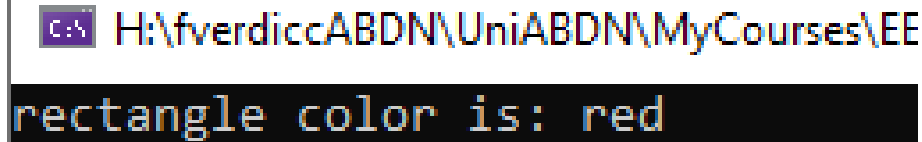
- A member function to output a *string* object with the colour name

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;
}
```



```
C:\> 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

```
Polygon_w_color.h  PetExample.h  Pet_w_color.h  tester.cpp
FurtherExamples  pet
1  #ifndef PetExample_h
2  #define PetExample_h
3  #include "RectangleExample.h"
4
5  class pet{
6  public:
7      // enumerator: finite set of choices for the colors
8      // blank and color_stop "bookend" the allowed colors
9      enum petTypeOptions { blank = 0, dog, cat, fish, rabbit, hamster, pet_stop };
10  protected:
11      double weight_kg = 0.0;
12      petTypeOptions petType;
13      bool init_flag = false;
14  public:
15      // constructor
16      pet()
17      {
18          // basic initialization
19          init_flag=false;
20          petType = blank;
21      }
22      // gets
23      bool getWeight(double& out_weight_kg) { ... }
24      bool getPetType(petTypeOptions& out_petType) { ... }
25      bool getPetType(string& out_petType_str) { ... }
26      // set
27      bool inputPetAndWeight(petTypeOptions in_petType, double in_weight_kg) { ... }
28      bool inputPetAndWeightFromKeyboard() { ... }
29      bool updatePetWeight(double in_weight_kg) { ... }
30      void inputRandomValues(double max_weight_kg = 10) { ... }
31      void reset() { ... }
32      // utility
33      bool isInitialized(){return init_flag;}
34      void printPetInfo() { ... }
35  };
```

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

```
Polygon_w_color.h  PetExample.h  Pet_w_color.h  tester.cpp
FurtherExamples  pet
1  #ifndef PetExample_h
2  #define PetExample_h
3  #include "RectangleExample.h"
4
5  class pet{
6  public:
7      // enumerator: finite set of choices for the colors
8      // blank and color_stop "bookend" the allowed colors
9      enum petTypeOptions { blank = 0, dog, cat, fish, rabbit, hamster, pet_stop };
10  protected:
11      // variables
12      double weight_kg;
13      petTypeOptions petType;
14      // initialization flag
15      bool init_flag;
16  public:
17      // constructor
18      pet()
19      {
20          // basic initialization
21          init_flag=false;
22          petType = blank;
23      }
24      // gets
25      bool getWeight(double& out_weight_kg) { ... }
26
27      bool getPetType(petTypeOptions& out_petType) { ... }
28
29      bool getPetType(string& out_petType_str) { ... }
30
31      // set
32      bool inputPetAndWeight(petTypeOptions in_petType, double in_weight_kg) { ... }
33
34      bool inputPetAndWeightFromKeyboard() { ... }
35
36      bool updatePetWeight(double in_weight_kg) { ... }
37
38      void inputRandomValues(double max_weight_kg = 10) { ... }
39
40      void reset() { ... }
41
42      // utility
43      bool isInitialized(){return init_flag;}
44
45      void printPetInfo() { ... }
46  };
```

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

Initializing values with constructors

```
Polygon_w_color.h  PetExample.h  Pet_w_color.h  tester.cpp
FurtherExamples  (Global Scope)

5  class pet{
6  public:
7      // enumerator: finite set of choices for the colors
8      // blank and color_stop "bookend" the allowed colors
9      enum petTypeOptions { blank = 0, dog, cat, fish, rabbit, hamster, pet_stop };
10 protected:
11     // variables
12     double weight_kg;
13     petTypeOptions petType;
14     // initialization flag
15     bool init_flag;
16 public:
17     // constructor
18     // alternatively
19     pet() : init_flag(false), petType(blank) {}
20     // gets
21     bool getWeight(double& out_weight_kg) { ... }
22
23     bool getPetType(petTypeOptions& out_petType) { ... }
24
25     bool getPetType(string& out_petType_str) { ... }
26
27     // set
28
29     bool inputPetAndWeight(petTypeOptions in_petType, double in_weight_kg) { ... }
30
31     bool inputPetAndWeightFromKeyboard() { ... }
32
33     bool updatePetWeight(double in_weight_kg) { ... }
34
35     void inputRandomValues(double max_weight_kg = 10) { ... }
36
37     void reset() { ... }
38
39     // utility
40
41     bool isInitialized(){return init_flag;}
42
43     void printPetInfo() { ... }
44
45 };
46
47 #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

```
Polygon_w_color.h  PetExample.h  Pet_w_color.h  tester.cpp
FurtherExamples    (Global Scope)

5  class pet{
6  public:
7      // enumerator: finite set of choices for the colors
8      // blank and color_stop "bookend" the allowed colors
9      enum petTypeOptions { blank = 0, dog, cat, fish, rabbit, hamster, pet_stop };
10 protected:
11     // variables
12     double weight_kg;
13     petTypeOptions petType;
14     // initialization flag
15     bool init_flag;
16     // temp const: only used as example
17     const int testconst;
18 public:
19     // constructor
20     // alternatively
21     pet() : init_flag(false), testconst(8) { petType=blank;}
22     // gets
23     bool getWeight(double& out_weight_kg) { ... }
24     bool getPetType(petTypeOptions& out_petType) { ... }
25     bool getPetType(string& out_petType_str) { ... }
26     // set
27     bool inputPetAndWeight(petTypeOptions in_petType, double in_weight_kg) { ... }
28     bool inputPetAndWeightFromKeyboard() { ... }
29     bool updatePetWeight(double in_weight_kg) { ... }
30     void inputRandomValues(double max_weight_kg = 10) { ... }
31     void reset() { ... }
32     // utility
33     bool isInitialized(){return init_flag;}
34     void printPetInfo() { ... }
35 };
36 #endif
```

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

Initializing values with constructors

```
Polygon_w_color.h  PetExample.h  Pet_w_color.h  tester.cpp
FurtherExamples  (Global Scope)

5  class pet{
6  public:
7      // enumerator: finite set of choices for the colors
8      // blank and color_stop "bookend" the allowed colors
9      enum petTypeOptions { blank = 0, dog, cat, fish, rabbit, hamster, pet_stop };
10 protected:
11     // variables
12     double weight_kg;
13     petTypeOptions petType;
14     // initialization flag
15     bool init_flag;
16     // temp const: only used as example
17     const int testconst;
18 public:
19     // constructor
20     // alternatively
21     pet() : init_flag(false), testconst(8) { petType=blank; }
22     // gets
23     bool getWeight(double& out_weight_kg) { ... }
24
25     bool getPetType(petTypeOptions& out_petType) { ... }
26
27     bool getPetType(string& out_petType_str) { ... }
28
29     // set
30
31     bool inputPetAndWeight(petTypeOptions in_petType, double in_weight_kg) { ... }
32
33     bool inputPetAndWeightFromKeyboard() { ... }
34
35     bool updatePetWeight(double in_weight_kg) { ... }
36
37     void inputRandomValues(double max_weight_kg = 10) { ... }
38
39     void reset() { ... }
40
41     // utility
42     bool isInitialized(){return init_flag;}
43     void printPetInfo() { ... }
44
45 };
46
47 #endif
```

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

Initializing values with constructors

```
Polygon_w_color.h  PetExample.h  X  Pet_w_color.h  tester.cpp
FurtherExamples  (Global Scope)

5  class pet{
6  public:
7      // enumerator: finite set of choices for the colors
8      // blank and color_stop "bookend" the allowed colors
9      enum petTypeOptions { blank = 0, dog, cat, fish, rabbit, hamster, pet_stop };
10 protected:
11     // variables
12     double weight_kg;
13     petTypeOptions petType;
14     // initialization flag
15     bool init_flag;
16     // temp const: only used as example
17     const int testconst;
18 public:
19     // constructor
20     // alternatively
21     pet() : init_flag(false), petType(blank) { testconst=8; }
22     // gets
23     bool getWeight(double& out_weight_kg) { ... }
24     bool getPetType(petTypeOptions& out_petType) { ... }
25     bool getPetType(string& out_petType_str) { ... }
26     // set
27     bool inputPetAndWeight(petTypeOptions in_petType, double in_weight_kg) { ... }
28     bool inputPetAndWeightFromKeyboard() { ... }
29     bool updatePetWeight(double in_weight_kg) { ... }
30     void inputRandomValues(double max_weight_kg = 10) { ... }
31     void reset() { ... }
32     // utility
33     bool isInitialized(){return init_flag;}
34     void printPetInfo() { ... }
35 };
36 #endif
```

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

← This does not compile

Any question?



Otherwise, let's go on