

8.5b — Non-static member initialization

BY ALEX ON FEBRUARY 12TH, 2016 | LAST MODIFIED BY ALEX ON APRIL 22ND, 2016

When writing a class that has multiple constructors (which is most of them), having to specify default values for all members in each constructor results in redundant code. If you update the default value for a member, you need to touch each constructor.

Starting with C++11, it's possible to give non-static class member variables a default initialization value directly:

```
1  class Square
2  {
3  private:
4      double m_length = 1.0; // m_length has a default value of 1.0
5      double m_width = 1.0; // m_width has a default value of 1.0
6
7  public:
8      Square()
9      {
10         // This constructor will use the default values above since they aren't overridden here
11     }
12
13     void print()
14     {
15         std::cout << "length: " << m_length << ", width: " << m_width << '\n';
16     }
17 };
18
19 int main()
20 {
21     Square x; // x.m_length = 1.0, x.m_width = 1.0
22     x.print();
23
24     return 0;
25 }
```

This program produces the result:

```
length: 1.0, width: 1.0
```

Non-static member initialization provides default values for your member variables that your constructors will use if the constructors do not provide an initialization values for the members themselves (via the member initialization list).

However, note that constructors still determine what kind of objects may be created. Consider the following case:

```
1  class Square
2  {
3  private:
4      double m_length = 1.0;
5      double m_width = 1.0;
6
7  public:
8
9      // note: No default constructor provided in this example
10
11     Square(double length, double width)
12         : m_length(length), m_width(width)
13     {
14         // m_length and m_width are initialized by the constructor (the default values aren't
15         used)
16     }
17 }
```

```

18     void print()
19     {
20         std::cout << "length: " << m_length << ", width: " << m_width << '\n';
21     }
22
23 };
24
25 int main()
26 {
27     Square x; // will not compile, no default constructor exists, even though members have default initialization values
28
29     return 0;
30 }

```

Even though we've provided default values for all members, no default constructor has been provided, so we are unable to create Square objects with no parameters.

If a default initialization value is provided and the constructor initializes the member via the member initializer list, the member initializer list will take precedence. The following example shows this:

```

1  class Square
2  {
3  private:
4      double m_length = 1.0;
5      double m_width = 1.0;
6
7  public:
8
9      Square(double length, double width)
10         : m_length(length), m_width(width)
11     {
12         // m_length and m_width are initialized by the constructor (the default values aren't
13         used)
14     }
15
16     void print()
17     {
18         std::cout << "length: " << m_length << ", width: " << m_width << '\n';
19     }
20
21 };
22
23 int main()
24 {
25     Square x(2.0, 3.0);
26     x.print();
27
28     return 0;
29 }

```

This prints:

```
length: 2.0, width: 3.0
```

Rule: Favor use of non-static member initialization to give default values for your member variables.

Quiz time

1) Update the following program to use non-static member initialization and member initializer lists.

```

1  #include <string>
2  #include <iostream>
3  class Ball
4  {

```

```
5 private:
6     std::string m_color;
7     double m_radius;
8
9 public:
10    // Default constructor with no parameters
11    Ball()
12    {
13        m_color = "black";
14        m_radius = 10.0;
15    }
16
17    // Constructor with only color parameter (radius will use default value)
18    Ball(const std::string &color)
19    {
20        m_color = color;
21        m_radius = 10.0;
22    }
23
24    // Constructor with only radius parameter (color will use default value)
25    Ball(double radius)
26    {
27        m_color = "black";
28        m_radius = radius;
29    }
30
31    // Constructor with both color and radius parameters
32    Ball(const std::string &color, double radius)
33    {
34        m_color = color;
35        m_radius = radius;
36    }
37
38    void print()
39    {
40        std::cout << "color: " << m_color << ", radius: " << m_radius << '\n';
41    }
42 };
43
44 int main()
45 {
46     Ball def;
47     def.print();
48
49     Ball blue("blue");
50     blue.print();
51
52     Ball twenty(20.0);
53     twenty.print();
54
55     Ball blueTwenty("blue", 20.0);
56     blueTwenty.print();
57
58     return 0;
59 }
```

This program should produce the result:

```
color: black, radius: 10
color: blue, radius: 10
color: black, radius: 20
color: blue, radius: 20
```

Show Solution

2) Why do we need to declare an empty default constructor in the program above, since all members are initialized via non-static member initialization?

Show Solution



[8.6 -- Overlapping and delegating constructors](#)

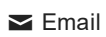


[Index](#)



[8.5a -- Constructor member initializer lists](#)

Share this:



Email



Facebook



Twitter



Google



Pinterest

[C++ PROGRAMMING](#) | [PRINT THIS POST](#)

12 comments to 8.5b — Non-static member initialization



Lokesh

February 8, 2016 at 3:08 am · [Reply](#)

I think the following program demonstrates all the rules about default constructors. I found the concept a bit twisty. I think you should use this particular example for quiz question no.1 as it serves as a summary(it just adds one or two concepts to what already exists in the question).

```
1  #include <iostream>
2  #include <string>
3
4
5  /*
6   - Using non-static member initialization.
7   Important note:
```