8.5b — Non-static member initialization

BY ALEX ON FEBRUARY 12TH, 2016 | LAST MODIFIED BY ALEX ON FEBRUARY 18TH, 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:

```cpp
class Square
{
private:
    double m_length = 1.0; // m_length has a default value of 1.0
    double m_width = 1.0; // m_width has a default value of 1.0

public:
    Square()
    {
    // This constructor will use the default values above since they aren't overridden here
    }

    void print()
    {
        std::cout << "length: " << m_length << ", width: " << m_width << '\n';
    }
};

int main()
{
    Square x; // x.m_length = 1.0, x.m_width = 1.0
    x.print();

    return 0;
}
```

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:

```cpp
class Square
{
private:
    double m_length = 1.0;
    double m_width = 1.0;

public:

    // note: No default constructor provided in this example

    Square(double length, double width)
        : m_length(length), m_width(width)
    {
        // m_length and m_width are initialized by the constructor (the default values aren't
used)
    }

    void print()
    {
        std::cout << "length: " << m_length << ", width: " << m_width << '\n';
    }

};

int main()
{
    Square x; // will not compile, no default constructor exists, even though members have def
ault initialization values

    return 0;
}
```

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:

```
class Square
{
private:
    double m_length = 1.0;
    double m_width = 1.0;

public:

    Square(double length, double width)
        : m_length(length), m_width(width)
    {
        // m_length and m_width are initialized by the constructor (the default values aren't
used)
    }

    void print()
    {
        std::cout << "length: " << m_length << ", width: " << m_width << '\n';
    }

};

int main()
{
    Square x(2.0, 3.0);
    x.print();

    return 0;
}
```

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.

```
#include <string>
#include <iostream>
class Ball
{
```

```
private:
    std::string m_color;
    double m_radius;

public:
        // Default constructor with no parameters
        Ball()
        {
        m_color = "black";
        m_radius = 10.0;
        }

        // Constructor with only color parameter (radius will use default value)
    Ball(const std::string &color)
    {
        m_color = color;
        m_radius = 10.0;
    }

        // Constructor with only radius parameter (color will use default value)
    Ball(double radius)
    {
        m_color = "black";
        m_radius = radius;
    }

        // Constructor with both color and radius parameters
    Ball(const std::string &color, double radius)
    {
        m_color = color;
        m_radius = radius;
    }

    void print()
    {
        std::cout << "color: " << m_color << ", radius: " << m_radius << '\n';
    }
};

int main()
{
        Ball def;
        def.print();

    Ball blue("blue");
    blue.print();

    Ball twenty(20.0);
    twenty.print();

    Ball blueTwenty("blue", 20.0);
    blueTwenty.print();

    return 0;
}
```

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

6 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).

```
#include <iostream>
#include <string>


/*
- Using non-static member initialization.
Important note:
```

```
-    The compiler will _NOT_ automatically create an "empty" default constructor since othe
r
    constructors are provided.
-    If no constructors are provided, the automatically created constructor would be publi
c.
-    To instantiate an object without any values, we would have to use our own default cons
tructor
    that takes no parameters (or all of whose parameters have default values).
*/

class Ball
{
private:
    std::string m_color = "black";
    double m_radius = 10.0;

public:
    // Default constructor
    Ball()
    {
        // No initialization (or default values) needed as we are using non-static member
        // initialization
    }

    // Constructor with only color parameter
    Ball(const std::string &color)
    {
        m_color = color;
    }

    // Constructor with only radius parameter
    Ball(double radius)
    {
        m_radius = radius;
    }

    // Constructor with both parameters
    Ball(std::string color, double radius)
    {
        m_color = color;
        m_radius = radius;
    }

    void print()    // print member variables
    {
        std::cout << "color: " << m_color << ", radius: " << m_radius << '\n';
    }
};


int main()
{
    Ball empty;
    empty.print();

    Ball blue("blue");
    blue.print();

    Ball twenty(20.0);
    twenty.print();

    Ball blueTwenty("blue", 20.0);
    blueTwenty.print();

    return 0;
}
```

Alex
February 9, 2016 at 10:37 am · Reply

I've updated the quiz question with some of your suggestions. Thanks!

Lokesh
February 10, 2016 at 5:34 am · Reply

I kindly disagree with:

```
class Ball
{
private:
    std::string m_color = "black";
    double m_radius = 10.0;

public:
        // no default constructor needed, since we are using non-static member in
itialization instead

// ...

};
```

I think we need a default constructor(we have to define our own) because although the private member variables have default values, these are non-static. So, we cannot instantiate an object without any values(no parameters) that will automatically take on the default values provided for the member variables. So, although we have used non-static member initialization, we need to define an "empty" constructor in order for us to instantiate an object with no parameters. As I have said earlier, we need a default constructor as so:

```
class Ball
{
private:
    std::string m_color = "black";
    double m_radius = 10.0;

public:
    // Default constructor
    Ball()
    {
        // No initialization (or default values) needed as we are using non-stati
c member
        // initialization
    }


    // Other constructors here ...
};
```

Secondly, you cannot use "default" as an identifier(in quiz-1 solution) because it is a keyword in C++.

Alex
February 11, 2016 at 9:51 am · Reply

Agreed with your assessment. Not sure how I messed that up. The article has been fixed.

Bassam
February 17, 2016 at 2:19 am · Reply

Hi! Alex. Thank you for this very useful explanation. I think you should use double instead of int in the constructor parameters for the second and third example.

```
Square(int length, int width)
```

Alex
February 18, 2016 at 1:12 pm · Reply

Quite right. Fixed!