How do I initialize a stl vector of objects who themselves have non-trivial constructors?

suppose I have the following class:

```
class MyInteger {
private:
  int n_;
public:
  MyInteger(int n) : n_(n) {};
  // MORE STUFF
};
```

And suppose this class don't have a default trivial constructor `MyInteger()`. I must always supply an `int` to initialize it for some reason. And then suppose that somewhere in my code I need a `vector<MyInteger>`. How do I initialize each `MyInteger` component in this `vector<>`?

I have two situations (probably the solution is the same, but I'll state them anyway), a normal variable inside a function:

```
int main(){
    vector<MyInteger> foo(10);  //how do I initialize each
                                //MyInteger field of this vector?
    doStuff(foo);
}
```

and as data in a class:

```
class MyFunClass {
private:
   vector<MyInteger> myVector;

public:
   MyFunClass(int size, int myIntegerValue) : myVector(size) {};
   // what do I put here if I need the
   // initialization to call MyInteger(myIntegerValue) for all
   // components of myVector?
};
```

Is it possible to do it just in the initialization list or must I write the initialization by hand in the MyFunClass(int, int) constructor?

This seems so very basic, and yet I somehow missed it inmy book and can't find in the web.

c++ constructor initialization initialization-list

asked May 26 '11 at 17:45

Rafael S. Calsaverini
3,716 4 32 90

2 Should the integer be the same for all instances inside the vector? – leftaroundabout May 26 '11 at 17:50

It's good enough. I just want to learn how to initialize the vector. The real code where this doubt appeared is not exactly this. I just wanted a simple code with the same problem. – Rafael S. Calsaverini May 26 '11 at 17:53

In my real code I have a vector of graphs and a class which is a collection of graphs, and I'm using a `vector<Graph>` as a container. The problem is that the number of graphs in the collection must be the same as the size of the Graphs, so I must pass to the Graph constructor the same `int` I pass to the vector constructor. – Rafael S. Calsaverini May 26 '11 at 17:56

4 Answers

There are many ways to get there. Here are some of them (in no particular order of presence).

Use `vector(size_type n, const T& t)` constructor. It initializes vector with `n` copies of `t`. For example:

```
#include <vector>

struct MyInt
```

```
{
    int value;
    MyInt (int value) : value (value) {}
};

struct MyStuff
{
    std::vector<MyInt> values;

    MyStuff () : values (10, MyInt (20))
    {
    }
};
```

Push elements into vector one by one. This might be useful when values should be different.
For example:

```
#include <vector>

struct MyInt
{
    int value;
    MyInt (int value) : value (value) {}
};

struct MyStuff
{
    std::vector<MyInt> values;

    MyStuff () : values ()
    {
        values.reserve (10); // Reserve memory not to allocate it 10 times...
        for (int i = 0; i < 10; ++i)
        {
            values.push_back (MyInt (i));
        }
    }
};
```

Another option is constructor initialization list, if C++0x is an option:

```
#include <vector>

struct MyInt
{
    int value;
    MyInt (int value) : value (value) {}
};

struct MyStuff
{
    std::vector<MyInt> values;

    MyStuff () : values ({ MyInt (1), MyInt (2), MyInt (3) /* ... */})
    {
    }
};
```

Of course, there is an option to provide default constructor and/or use something other than `std::vector` .

Hope it helps.

answered May 26 '11 at 17:59
user405725

+1 for the C++0x initialization. You beat me to it! – juanchopanza May 26 '11 at 18:11

Hum! Does g++ support C++0x yet? – Rafael S. Calsaverini May 26 '11 at 18:38

@Rafael: Yes. See gcc.gnu.org/projects/cxx0x.html – user405725 May 26 '11 at 18:44

If the elements of the vector are not default-constructible, then there are certain things you cannot do with the vector. You cannot write this (example 1):

```
vector<MyInteger> foo(10);
```

You can, however, write this (example 2):

```
vector<MyInteger> foo(10, MyInteger(37));
```

(This only requires a copy constructor.) The second argument is an initializer for the elements of the vector.

In your case, you could also write:

```
vector<MyInteger> foo(10, 37);
```

...since MyInteger has a non-explicit constructor taking "int" as argument. So the compiler will cast 37 to MyInteger(37) and give the same result as example 2.

You might want to study the documentation on std::vector.

answered May 26 '11 at 17:53

Nemo
37.6k 6 59 103

```
vector<MyInteger> foo(10, MyInteger(MY_INT_VALUE));

MyFunClass(int size, int myIntegerValue) : myVector(size, MyInteger(myIntegerValue)) {};
```

answered May 26 '11 at 17:53

mbykov
154 4

Besides all answers which answered the question very well, in a case that your class MyInteger is not copy-constructible, you could use this trick : instead of creating `vector< MyInteger>` , you could create `vector< shared_ptr< MyInteger > >`

answered May 26 '11 at 18:19

ВЈовић
32k 16 76 156
