Determine array size in constructor initializer

In the code below I would like array to be defined as an array of size x when the Class constructor is called. How can I do that?

```
class Class
{
public:
  int array[];
  Class(int x) : ??? { }
}
```

c++ arrays constructor initialization

edited Nov 8 '10 at 19:02
sbi 114k 28 158 315

asked Apr 15 '09 at 14:08
zaratustra 1,786 5 24 37

1 If you plan to use C++ regularly, I strongly recommend you familiarize yourself with the standard template library. It makes working with collections of data *much* easier. – Brian Apr 15 '09 at 14:38

1 As an aside, vectors make it relatively easy to work with the array without knowing the size in advance. It isn't necessary to know the size in advance; you can append elements to the end of a vector in (amortized) O(1) time anyhow using push_back. – Brian Apr 15 '09 at 14:43

Using vectors brings new problems, since the class I'm trying to vectorize has protected "new" operators. But that wasn't what I asked so nevermind. – zaratustra Apr 16 '09 at 17:12

@zaratustra: Why would that make a vector not work? It might need a custom allocator, but I doubt even that. – TBohne Jul 6 '12 at 0:21

10 Answers

You can't initialize the size of an array with a non-const dimension that can't be calculated at compile time (at least not in current C++ standard, AFAIK).

I recommend using `std::vector<int>` instead of array. It provides array like syntax for most of the operations.

edited Jul 6 '12 at 0:18
jedwards 12.4k 15 39

answered Apr 15 '09 at 14:11
Cătălin Pitiș 10.9k 2 22 48

What would the syntax for using a vector in that situation be like? – zaratustra Apr 15 '09 at 14:13

2 vector< int > array; Class(x) : array(x) {}; – DevSolar Apr 15 '09 at 14:15

Use the new operator:

```
class Class
{
   int* array;
   Class(int x) : array(new int[x]) {};
};
```

answered Apr 15 '09 at 14:13
John Dibling 63.2k 10 97 210

2 Don't forget to call delete[] in the constructor if you use this code. – Brian Apr 15 '09 at 14:14

5 If you do this you will also need a copy constructor , an assignment operator and a destructor. Useing a std::vector gives you exactly the same functionality but requires none of these. – anon Apr 15 '09 at 14:15

+1 This actually answers the OP's question by initializing the size of an array, though it actually is replacing `int *array` which isn't an array, with the list in the constructor. @(anon) besides a copy constructor, an assignment operator, and a destructor, this truly answers the question. – Link TheProgrammer Oct 25 '14 at 11:41

I don't think it can be done. At least not the way you want. You can't create a statically sized array (array[]) when the size comes from dynamic information (x).

You'll need to either store a pointer-to-int, and the size, and overload the copy constructor, assignment operator, and destructor to handle it, or use std::vector.

```
class Class
{
  ::std::vector<int> array;
  Class(int x) : array(x) { }
};
```

edited Nov 3 '10 at 16:09

answered Apr 15 '09 at 14:16
AFoglia
4,226 17 33

Sorry for necroing this old thread. There is actually a way to find out the size of the array compile-time. It goes something like this:

```
#include <cstdlib>

template<typename T>
    class Class
    {
        T* _Buffer;

        public:
        template<size_t SIZE>
            Class(T (&static_array)[SIZE])
            {
                _Buffer = (T*)malloc(sizeof(T) * SIZE);

                memcpy(_Buffer, static_array, sizeof(T) * SIZE);
            }

            ~Class()
            {
                if(_Buffer)
                {
                    free(_Buffer);
                    _Buffer = NULL;
                }
            }
    };

int main()
{
    int int_array[32];
    Class<int> c = Class<int>(int_array);

    return 0;
}
```

Alternatively, if you hate to malloc / new, then you can create a size templated class instead. Though, I wouldn't really recommend it and the syntax is quite ugly.

```
#include <cstdio>

template<typename T, size_t SIZE>
    class Class
    {
        private:
            T _Array[sz];
        public:
            Class(T (&static_array)[SIZE])
            {
                memcpy(_Array, static_array, sizeof(T) * SIZE);
            }
    };

int main()
{
    char int_array[32];
    Class<char, sizeof(int_array)> c = Class<char, sizeof(int_array)>(int_array);
    return 0;
}
```

Anyways, I hope this was helpful :)

edited Jan 6 '11 at 10:16

answered Jan 6 '11 at 10:09
user563910
121 1 2

2 are you a wizard – zaratustra Feb 9 '11 at 13:43

In C++11 a superior method of determining the size of a built-in array is using a constexpr template function. For example: template < class T, std::size_t N > constexpr std::size_t size(const T (&array)[N]) { return N; } – Ricky65 Feb 19 '14 at 14:23

Instead of using a raw array, why not use a vector instead.

```
class SomeType {
  vector<int> v;
  SomeType(size_t x): v(x) {}
};
```

Using a vector will give you automatic leak protection in the face of an exception and many other benefits over a raw array.

edited Apr 15 '09 at 14:45
Brian
15k 7 51 111

answered Apr 15 '09 at 14:13
JaredPar
385k 72 809 1132

Do you mean "Using *a vector* will give you automatic leak protection"? :) – mkb Apr 15 '09 at 14:24

@mkb, that's twice today I've made fundamentally stupid comments. Must drink more coffee to wake up before i start posting ;) – JaredPar Apr 15 '09 at 14:28

Don't you understand there is not need to use vector, if one wants to use arrays it's a matter of efficiency, e.g. less space, no copy time (in such case if handled properly there is not even need to delete the array within a destructor), etc. wichever reasons one has.

the correct answer is: (quoted)

```
class Class
{
   int* array;
   Class(int x) : array(new int[x]) {};
};
```

Do not try to force one to use non optimal alternatives or you'll be confusing unexperienced programmers

answered Dec 6 '10 at 7:05
Miguel Enrique León Figue
21 1

You can't do it in C++ - use a std::vector instead:

```
#include <vector>

struct A {
   std::vector <int> vec;
   A( int size ) : vec( size ) {
   }
};
```

edited Apr 15 '09 at 14:25
jwfearn
10.5k 11 68 90

answered Apr 15 '09 at 14:13
anon

Declare your array as a pointer. You can initialize it in the initializer list later through through new.

Better to use vector for unknown size.

You might want to look at this question as well on variable length arrays.

edited Apr 15 '09 at 14:32

answered Apr 15 '09 at 14:15
Shree
1,262 5 21 40

better to use the vector for known size too – anon Apr 15 '09 at 14:17

have to agree on that – Shree Apr 15 '09 at 14:24

BAD idea. Doing the memory management on a pointer that acts like an array is not trivial in the presence of exceptions. Use std::vector or std::tr1::array. – Loki Astari Apr 15 '09 at 15:51

accepted, but this was just an option in response to the original question – Shree Apr 15 '09 at 16:11

Two options:

Use std::vector. This allows easy re-sizing of the array.
Use std::tr1::array. This has a static size.

Both can be correctly initialized in the constructors
initializer list.

answered Apr 15 '09 at 15:53
Loki Astari
134k 37 190 356

You folks have so overcomplicated this. Of course you can do this in C++. It is fine for him to use a normal array for efficiency. A vector only makes sense if he doesn't know the final size of the array ahead of time, i.e., it needs to grow over time.

If you can know the array size one level higher in the chain, a templated class is the easiest, because there's no dynamic allocation and no chance of memory leaks:

```
template < int ARRAY_LEN > // you can even set to a default value here of C++'11

class MyClass
  {
  int array[ARRAY_LEN]; // Don't need to alloc or dealloc in structure!  Works like you
imagine!
  }

// Then you set the length of each object where you declare the object, e.g.

MyClass<1024> instance; // But only works for constant values, i.e. known to compiler
```

If you can't know the length at the place you declare the object, or if you want to reuse the same object with different lengths, or you must accept an unknown length, then you need to allocate it in your constructor and free it in your destructor... (and in theory always check to make sure it worked...)

```
class MyClass
  {
  int *array;

  MyClass(int len) { array = calloc(sizeof(int), len); assert(array); }
  ~MyClass() { free(array); array = NULL; } // DON'T FORGET TO FREE UP SPACE!
  }
```

edited Sep 4 '12 at 21:25

answered Sep 4 '12 at 21:19
Jeff Diamond
11 2