

Re: Question about arrays in objects

Source: <http://coding.derkeiler.com/Archive/C/ CPP/alt.comp.lang.learn.c-cpp/2004-01/0871.html>

From: Joec (joec_at_annuna.com)

Date: 01/17/04

Date: Sat, 17 Jan 2004 03:11:45 GMT

Leor Zolman wrote:

> *On Sat, 17 Jan 2004 01:52:28 GMT, Joec <joec@annuna.com> wrote:*

>

>

>>*I want to create a 2d array as an object member.*

>>

>>*like:*

>>

>>*class x{*

>>*private:*

>>*const int num;*

>>*int array[num];*

>>*...*

>>

>>*do I have to use a pointer like:*

>>*int *px;*

>>

>>*then put the array in the constructor and use the pointer?*

>

>

> *A couple of things need clarification here:*

>

> *1. You said "2d" array, but you show a definition for a 1d array (and
> have only a single dimension value, "num"). Do you really want a
> 1-dimensional, or a 2-dimensional array?*

>

> *2. Do you intend for the dimension(s) of your array (be it 1d or 2d)
> to be "compile time constant", or do you want the dimension(s) to be
> supplied at run time at the time the object containing this array is
> to be instantiated? That will drive your implementation technique.*

>

> *Just to keep the examples simple, let's use a 1d array (although the
> pointer-based solution I'll discuss below gets more complicated with
> multiple dimensions). If the dimension is to be a compile-time
> constant (not what I think you're looking for, but I'm just trying to
> cover all the bases), then the best way to define it is as an*

> *enumeration constant:*
>
> *class x {*
> *enum {num = 100};*
> *int array[num][num];*
> *};*
>
> *There's at least one other way to do this, using a static const int*
> *(or size_t, rather, if you want to be a stickler about it), but I*
> *think the enum way is the cleanest.*
>
> *Now, if you want the size of the array to be specified at run-time and*
> *handled in the constructor, you just can't use a plain old array. A*
> *std::vector would be a good choice, because it'll handle its own*
> *memory management for you, and you wouldn't need any asterisks in your*
> *program (well, at least not for this).*
>
> *If you'd prefer to "roll your own" dynamic array implementation, then*
> *that is where using a pointer would come in. Note that you can't use a*
> *built-in array, because array dimensions must be compile-time*
> *constants. A class data member defined as*
> *const int num;*
> *the way you wrote it above wouldn't work because the compiler can't*
> *predict its value at compile time. num would have to be set in the*
> *constructor using a member initializer, e.g.,*
> *x::x(int size) : num(size) { ... }*
> *and thus it would not count as a "compile time constant".*
>
> *In your class, you'll need at least a pointer to int. Most folks*
> *would also store the dimension in the object, in order to permit*
> *bounds checking or size querying. The class definition would then have*
> *at least this:*
>
> *class x {*
> *public:*
> *x(size_t size); // constructor*
> *~x(); // destructor*
> *// other member functions...*
> *private:*
> *size_t n_elements;*
> *int *datap;*
> *};*
>
> *and the job of the constructor would be to set n_elements to the value*
> *of the "size" parameter, and to allocate memory for holding that many*
> *ints and save a pointer to that memory in datap. Many folks prefer to*
> *do all that in the initializer list of the constructor, because there*
> *are often performance benefits in C++ to doing as much as you can in*
> *the initializer list rather than in the body of the constructor*
> *(though it wouldn't matter much in this case):*
>
>

