

c++ truths

A C++ BLOG ON INTERMEDIATE TO ADVANCED TOPICS IN C++ PROGRAMMING LANGUAGE FEATURES, STANDARDS, IDIOMS, DESIGN PATTERNS, AND OBJECT-ORIENTED PROGRAMMING IN GENERAL.

tuesday, october 11, 2011

Multi-dimensional arrays in C++11

What new can be said about multi-dimensional arrays in C++? As it turns out, quite a bit! With the advent of C++11, we get new standard library class `std::array`. We also get new language features, such as template aliases and variadic templates. So I'll talk about interesting ways in which they come together.

It all started with a simple question of how to define a multi-dimensional `std::array`. It is a great example of *deceptively simple* things. Are the following the two arrays identical except that one is native and the other one is `std::array`?

```
1 int native[3][4];
2 std::array<std::array<int, 3>, 4> arr;
```

No! They are not. In fact, `arr` is more like an `int[4][3]`. Note the difference in the array subscripts. The native array is an array of 3 elements where every element is itself an array of 4 integers. 3 rows and 4 columns. If you want a `std::array` with the same layout, what you really need is:

```
1 std::array<std::array<int, 4>, 3> arr;
```

That's quite annoying for two reasons. First of all, it is quite verbose. Think of three or higher dimensional arrays. It's just ugly. Worse yet, the array bounds must be spelled out in reverse order.

Fear not, help is on its way! Meet [template aliases](#)! It is a way of defining synonyms for other types including partially defined templates. With C++03 typedef, you must use a fully specialized template to create an alias. C++11 still has that restriction on typedefs but template aliases will let you create "typedefs" for template partial specializations. It is a C++11 solution for the [Templated Typedef](#) idiom. Lets try to write one for a two dimensional `std::array`.

```
1 template <class T, size_t ROW, size_t COL>
2 using Matrix = std::array<std::array<T, COL>, ROW>;
3 Matrix<float, 3, 4> mat;
```

This one is much nicer. `Matrix` is a template alias for a partial specialization of `std::array`. With

about me



Sumant Tambe

Follow

I am interested in C++-standards-based middleware for distributed systems, model-driver software development, and scalable fault-tolerant distributed systems.

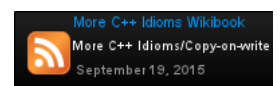
[View my complete profile](#)

rss feeds

766 readers
BY FEEDBURNER

ShareThis

+18 Recommend this on Google+



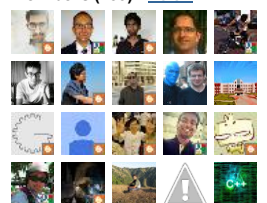
[↑ Grab this Headline Animator](#)



followers

Join this site with Google Friend Connect

Members (239) [More »](#)



Already a member? [Sign in](#)

that, defining a two dimensional matrix (mat) is really easy. Note that order of ROW and COL. An alias for two dimensional native array can also be defined similarly.

```
1 template <class T, size_t ROW, size_t COL>
2 using NativeMatrix = T[ROW][COL];
3 NativeMatrix<float, 3, 4> mat;
4
```

So far so good. Lets move on to multi-dimensional arrays. Using template aliases for arbitrary dimension arrays would need variadic templates. Unfortunately, it is not straightforward at all.

```
1 template <class T, size_t... DIMS>
2 using MultiDimArray = std::array<??? ...DIMS... ???>;
3 MultiDimArray<float, 3, 4, 5, 6, 7> floats;
4
```

DIMS parameter pack can be expanded where a comma separated list is expected. However, multi-dimensional std::array definitions are **hierarchical**, which prevents straight-forward parameter pack expansion.

One way I could make it work was using a little meta-program.

```
1 template <class T, size_t I, size_t... J>
2 struct MultiDimArray
3 {
4     using Nested = typename MultiDimArray<T, J...>::type;
5     // typedef typename MultiDimArray<T, J...>::type Nested;
6     using type = std::array<Nested, I>;
7     // typedef std::array<Nested, I> type;
8 };
9
10 template <class T, size_t I>
11 struct MultiDimArray<T, I>
12 {
13     using type = std::array<T, I>;
14     // typedef std::array<T, I> type;
15 };
16
17 MultiDimArray<float, 3, 4, 5, 6, 7>::type floats;
```



MultiDimArray is a pair of meta-functions to compute nested type for multi-dimensional std::array. The most general MultiDimArray is a variadic template of unsigned integers to pass an arbitrary number of dimensions. The terminating MultiDimArray specialization defines the simplest case of single dimensional std::array. I'm using the "using" syntax instead of typedefs. The equivalent typedef are written in comments.

Similarly, MultiDimNative can also be defined to create a native array of arbitrary dimensions.

```
1 template <class T, size_t I, size_t... J>
2 struct MultiDimNative
3 {
4     using Nested = typename MultiDimNative<T, J...>::type;
5     using type = Nested [I];
6 };
7
8 template <class T, size_t I>
9 struct MultiDimNative<T, I>
10 {
11     using type = T [I];
12 }
```

subscribe to

 Posts 

 Comments 

links

[My homepage](#)
[The C++ Standards Committee](#)
[C++ soup!](#)
[LightSleeper](#)
[Thinking Asynchronous in C++](#)
[More C++ links](#)

blog archive

► 2015 (1)
 ► 2014 (9)
 ► 2013 (6)
 ► 2012 (6)
 ▼ 2011 (4)
 ▼ October (1)
 Multi-dimensional arrays in C++11
 ► July (2)
 ► June (1)
 ► 2010 (2)
 ► 2009 (2)
 ► 2008 (8)
 ► 2007 (18)
 ► 2006 (20)
 ► 2005 (27)

```

13 };
14
15 MultiDimNative<double, 3, 4, 5, 6, 7>::type doubles;

```

To make sure that the layout of all the arrays is the same I wrote a small test program.

```

1  template <class T, size_t I, size_t J>
2  union data
3  {
4      T native[I][J];
5      NativeMatrix<T, I, J> native_matrix;
6      Matrix<T, I, J> matrix;
7      typename MultiDimArray<T, I, J>::type multidim_array;
8      typename MultiDimNative<T, I, J>::type multidim_native;
9  };
10
11 template <class T>
12 void print(T & t, size_t rows, size_t columns)
13 {
14     for(size_t i = 0; i < rows; ++i)
15     {
16         for(size_t j = 0; j < columns; ++j)
17             printf("%d ", t[i][j]);
18
19         printf("\n");
20     }
21     printf("\n");
22 }
23
24 int main()
25 {
26     constexpr size_t I = 3;
27     constexpr size_t J = 4;
28
29     data<int, I, J> d;
30     size_t val = 0;
31
32     for(size_t i = 0; i < I; ++i)
33     {
34         for(size_t j = 0; j < J; ++j)
35             d.native[i][j] = val++;
36     }
37
38     print(d.native, I, J);
39     print(d.native_matrix, I, J);
40     print(d.matrix, I, J);
41     print(d.multidim_array, I, J);
42     print(d.multidim_native, I, J);
43
44     return 0;
45 }

```

This program defines a union of 5 different arrays, all of which have the same dimensions and layout. It populates a native two dimensional array and prints the contents using all other multi-dimensional arrays defined in the same union. As of today, only clang accepts this program.

That's it for now!

posted by [sumant tambe](#) at 8:59 pm

17 comments:

 **ralph zhang** said...

That's quite interesting, but what compiler are you using? Gcc doesn't seem to support that even in 4.7

Wed Oct 12, 02:09:00 AM PDT



sumant said...

I'm using clang.

Wed Oct 12, 08:37:00 AM PDT

fahad munir said...

Sumant its a good blog for beginners

im also a beginner

its my blog

<http://fahad-cprogramming.blogspot.com>

i followed kindly follow back to my blog thanks

Fri Dec 09, 12:40:00 PM PST

xander345 said...

if you like c++ you can compile it online here: <http://codecompiler.info/>

32, 64 - windows & Linux - and more programming languages

Mon Jan 30, 10:55:00 AM PST

basimah said...

Dear Sumant,

I am a beginner in C++. I want to implement a grid file for records which may have 1-32 dimensions. For 32-dimensions how can I declare arrays. Please help.

Wed Feb 15, 01:27:00 AM PST

basimah said...

Dear Mr. Sumant,

Would you please help to declare 32-dimension arrays or any structure which can be used easily for direct access for 1-32 dimension data. I want to build a grid file system for direct access.

Wed Feb 15, 01:28:00 AM PST

anonymous said...

I thought it was going to be some boring old post, but it really compensated for my time.

I will post a link to this page on my blog. I am sure my visitors will find that very useful.

[best train institute in jaipur](#)

Tue Apr 24, 09:52:00 AM PDT

sas said...

It amazing

Note: it works using GCC 4.7 on windows but you need to add the option
-std=c++0x

Dear Sumant did you manage to make the varidic template solution work?

I test it on GCC 4.7 It does not work

thanks

Fri May 11, 02:44:00 AM PDT

anonymous said...

You can make it cleaner with alias:

```

template
struct GetMultiDimArray
{
using type = std::array::type, d1>;
};

template
struct GetMultiDimArray
{
using type = std::array, d1>;
};

template
using MultiDimArray = typename GetMultiDimArray::type;

// Usage:
MultiDimArray mdarr {1, 2, 3, 4, 5, 6};

```

Sat May 26, 05:42:00 AM PDT

 **sse said...**

Thank you very much for posting and sharing this great article. It is so interesting. I want to know some other information about this site. So please give me this news quickly. I always will be aware of you. [army combat boots](#)

Fri Nov 02, 04:18:00 AM PDT

 **sse said...**

No doubt this is often a wonderful post I got plenty of data when reading smart luck. Theme of web log is superb there's nearly everything to scan, sensible post.... [Brighton Tiles](#)

Tue Nov 06, 07:37:00 AM PST

 **jatinder singh said...**

This comment has been removed by the author.

Thu Nov 22, 11:25:00 PM PST

anonymous said...

<http://cplusplus-in.blogspot.com/>

Mon Dec 17, 06:25:00 AM PST

sayeed said...

Thanks Sumant, its a useful blog for the [beginner](#). but how do i download and install clang?

Fri Dec 21, 02:54:00 PM PST

 **felix sander said...**

I am glad to found such useful post. I really increased my knowledge after read your post which will be beneficial for me.
www.easywritinghelp.com

Sat Feb 02, 03:12:00 AM PST

holiday packages to mecca said...

This is true that without c++ no one can be a good developer because this language is the key of programming.

Sat Feb 16, 02:45:00 AM PST

anonymous said...

hi, thanks for the post, but you've syntax error in the template at the end instead of '>'

9/23/2015

C++ Truths: Multi-dimensional arrays in C++11

you wrote '<'.

Tue Mar 18, 09:01:00 AM PDT

Post a Comment

[Newer Post](#)

[Home](#)

[Older Post](#)

Subscribe to: [Post Comments \(Atom\)](#)