C++ 2d map? Like a 2d array?

Is it possible to make a 2d map?

Like this:

```
map< int, int, string> testMap;
```

And filling the values would be like:

```
testMap[1][3] = "Hello";
```

Thanks for your time :)

c++ arrays map 2d

asked Jul 14 '13 at 19:26

mrg95
398 1 17 42

3 Answers

You can nest two maps:

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

int main()
{
    std::map<int,std::map<int,std::string>> m;

    m[1][3] = "Hello";

    std::cout << m[1][3] << std::endl;

    return 0;
}
```

answered Jul 14 '13 at 19:29

user2530166

This worked perfectly :) Thanks. I'll accept the answer asap. – mrg95 Jul 14 '13 at 19:32

yes, use `std::pair`

```
map< std::pair<int, int>, string> testMap;
testMap[std::make_pair(1,3)] = "Hello";
```

edited Nov 13 '13 at 14:32

answered Jul 14 '13 at 19:29

andre
4,981 1 26 51

Interesting. Normally, I might use this, however I'm actually using Qt creator so I don't think they have the standard library. They might have something similar, but idk. For example, instead of using std::map, I'm using QMap. Good thing the syntax was the same – mrg95 Jul 14 '13 at 19:33

1 Qt Creator is just an IDE (integrated development environment), not a compiler or different form of C++. So, even though Qt Creator is useful when dealing with the Qt library, which defines QMap, you can still write standard C++ (e.g. using std::map) in Qt Creator and compile it just fine. – user2530166 Jul 14 '13 at 19:37

This will be considerably more efficient than RyanMcK's solution. – Cory Nelson Jul 14 '13 at 19:39

2 @mc360pro a `std::pair` is not as heavy as adding an extra `std::map` . The look up time for a map of maps is `O(Lg(n)*Lg(n))` while a pair only take `O(Lg(n))` . – andre Nov 13 '13 at 14:36

2 It is definitely not `O(lg(n)*lg(n))` . It's `O(lg(n) + lg(n))` , or just `O(lg(n))` . The first map lookup takes `lg(n)` , and the second also takes `lg(n)` . Of course, this is assuming a square "array." This should probably be broken down into `m` and `n` instead, but it still simplifies in a similar fashion. – michaelgulak Mar 28 '15 at 4:16

In case it's helpful for anybody, here is code for a class that builds upon andre's answer, which allows access via bracket operators like a regular 2D array would:

```
template<typename T>
class Graph {
/*
    Generic Graph ADT that uses a map for large, sparse graphs which can be
    accessed like an arbitrarily-sized 2d array in logarithmic time.
*/
private:
    typedef std::map<std::pair<size_t, size_t>, T> graph_type;
    graph_type graph;
    class SrcVertex {
    private:
        graph_type& graph;
        size_t vert_src;
    public:
        SrcVertex(graph_type& graph): graph(graph) {}
        T& operator[](size_t vert_dst) {
            return graph[std::make_pair(vert_src, vert_dst)];
        }
        void set_vert_src(size_t vert_src) {
            this->vert_src = vert_src;
        }
    } src_vertex_proxy;
public:
    Graph(): src_vertex_proxy(graph) {}
    SrcVertex& operator[](size_t vert_src) {
        src_vertex_proxy.set_vert_src(vert_src);
        return src_vertex_proxy;
    }
};
```

edited Dec 9 '14 at 8:27

answered Dec 9 '14 at 1:57

Paul Nickerson
11 2