how do you make a heterogeneous boost::map?

Asked 16 years, 1 month ago Modified 7 years, 7 months ago Viewed 24k times

33

I want to have a map that has a homogeneous key type but heterogeneous data types.

I want to be able to do something like (pseudo-code):

```
boost::map<std::string, magic_goes_here> m;
m.add<int>("a", 2);
m.add<std::string>("b", "black sheep");

int i = m.get<int>("a");
int j = m.get<int>("b"); // error!
```

I could have a pointer to a base class as the data type but would rather not.

I've never used boost before but have looked at the fusion library but can't figure out what I need to do.

Thanks for your help.

c++ boost dictionary

Share
Improve this question
Follow

edited Jul 11, 2009 at 19:37
Piotr Dobrogost
42.3k ● 45 ● 241 ● 373

asked Oct 30, 2008 at 19:23
Kurt
2,395 ● 2 ● 30 ● 31

If you know all possible types that you might want to stuff in the map then `boost::variant` will work great. If you want literally any type, then `boost::any` is the way to go. – Kurt Mar 7, 2014 at 5:21

6 Answers

Sorted by: Highest score (default)

40

```
#include <map>
#include <string>
#include <iostream>
#include <boost/any.hpp>

int main()
{
    try
    {
        std::map<std::string, boost::any> m;
        m["a"]  = 2;
        m["b"]  = static_cast<char const *>("black sheep");

        int i = boost::any_cast<int>(m["a"]);
        std::cout << "I(" << i << ")\n";

        int j = boost::any_cast<int>(m["b"]); // throws exception
        std::cout << "J(" << j << ")\n";
    }
    catch(...)
    {
        std::cout << "Exception\n";
    }

}
```

Share

Improve this answer

Follow

edited Oct 30, 2008 at 23:57

answered Oct 30, 2008 at 23:12

Loki Astari

264k ● 86 ● 342 ● 571

1 Instead of the cumbersome_cast, you can use `std::decay` . – Kerrek SB Mar 17, 2012 at 1:17

is not it just using std::map rather than boost map as initialized here ? – Tab Nov 8, 2013 at 5:33

10

How can I build a <favorite container> of objects of different types?

> You can't, but you can fake it pretty well. In C/C++ all arrays are homogeneous (i.e., the elements are all the same type). However, with an extra layer of indirection you can give the appearance of a heterogeneous container (a heterogeneous container is a container where the contained objects are of different types).
>
> There are two cases with heterogeneous containers.
>
> The first case occurs when all objects you want to store in a container are publicly derived from a common base class. [...]
>
> The second case occurs when the object types are disjoint — they do not share a common base class.
> The approach here is to use a handle class. The container is a container of handle objects (by value or by pointer, your choice; by value is easier). Each handle object knows how to "hold on to" (i.e., maintain a pointer to) one of the objects you want to put in the container. You can use either a single handle class with several different types of pointers as instance data, or a

> hierarchy of handle classes that shadow the various types you wish to contain (requires the container be of handle base class pointers). The downside of this approach is that it opens up the handle class(es) to maintenance every time you change the set of types that can be contained. The benefit is that you can use the handle class(es) to encapsulate most of the ugliness of memory management and object lifetime. Thus using handle objects may be beneficial even in the first case.

Share
Improve this answer
Follow

edited Jun 20, 2020 at 9:12
Community Bot
1 ● 1

answered Oct 30, 2008 at 19:28
Leon Timmermans
30.2k ● 2 ● 64 ● 110

8

Would boost::any do the trick for you?

Share Improve this answer Follow

answered Oct 30, 2008 at 19:28
activout.se
6,096 ● 4 ● 29 ● 37

7

Thank you David, that was what I needed. Here's the working solution.

```
#include <iostream>
using std::cout;
using std::endl;

#include <map>
#include <boost/any.hpp>

using boost::any_cast;
typedef std::map<std::string, boost::any> t_map;


int main(int argc, char **argv)
{

  t_map map;
  char *pc = "boo yeah!";

  map["a"] = 2.1;
  map["b"] = pc;

  cout << "map contents" << endl;
  cout << any_cast<double>(map["a"]) << endl;
  cout << any_cast<char*>(map["b"]) << endl;
```

```
    return 0;
}
```

Share Improve this answer Follow

answered Oct 31, 2008 at 0:04

Kurt
2,395 ● 2 ● 30 ● 31

1 this should be `const char *pc = "boo yeah!";` – Piotr Dobrogost Jul 11, 2009 at 19:37

5

If you want a limited set of types to be supported, Boost.Variant should do the trick.

Share Improve this answer Follow

answered Oct 30, 2008 at 20:22

Ferruccio
101k ● 38 ● 229 ● 303

0

boost any surely works, but I think using Int to type Technology as the key type of fusion map is a better solution. No type erasure and possibly faster

Share Improve this answer Follow

answered May 14, 2017 at 14:00

FaceBro
859 ● 2 ● 14 ● 30