Subject: Understanding pick behaviour and containers
Posted by Mindtraveller on Thu, 01 May 2008 19:09:50 GMT
View Forum Message <> Reply to Message

Let`s say, one needs to have hashed container of some complex objects. OK, taking Index and putting some moveable and optional deep copy flavour to objects contained:

```
class AAA : public VectorMap<int, int>, public MoveableAndDeepCopyOption<AAA>
{
public:
 AAA() {}
 AAA(const AAA &, int) {}
 unsigned GetHashValue() const {return 0;/*proxy here*/;}
};

CONSOLE_APP_MAIN
{
 AAA a;
 a.Add(10,1);
 AAA a2(a);
 Cout() << a2.GetCount();

 Index<AAA> ai;
 ai.Add(a2);
 Cout() << " / " << ai[0].GetCount();
}
```

Output gives us 1 / 0. First value is OK, since first pick operation succeeded. Second one is zero, which is not right IMO: I added object to Index and it didn`t appear in the container.
When we talk about Vector or even VectorMap - it`s OK, we have AddPick() there, which works in this case.

So, why add operation didn`t work and how to solve this?

P.S. Why (pick_ != const) for MSC compiler?

Subject: Re: Understanding pick behaviour and containers
Posted by mirek on Thu, 01 May 2008 21:04:23 GMT
View Forum Message <> Reply to Message

Bug in docs (now fixed). Keys have to be Moveable and have deep copy constructor and assignment.

I sugguest this fix:

```
class AAA : public WithDeedCopy< VectorMap<int, int> >
```

Mirek

Subject: Re: Understanding pick behaviour and containers
Posted by Mindtraveller on Fri, 02 May 2008 06:27:26 GMT
View Forum Message <> Reply to Message

Mirek, also I want to suggest add
int AIndex::Find[Prev,Next,Last](unsigned _hash)
functions. What for?
Let`s see typical situation: I have a number of objects and I want them to be in quick access. For a number of reasons, Map doesn`t fit: i.e. key value is a natural part of internal object structure and cannot be taken out as different key object (it is possible but not good solution). Of course your hashed Index container is an ideal solution here.
Of course in some situations I want object to be found quickly and I can compute hash. But all I

But I don`t need this x variable, I just want object(s) who`s hash value is _hash. Also this function could make testing (are any objects with such hash value) easier.

Subject: Re: Understanding pick behaviour and containers
Posted by mirek on Fri, 02 May 2008 12:17:48 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Fri, 02 May 2008 02:27Mirek, also I want to suggest add
int AIndex::Find[Prev,Next,Last](unsigned _hash)
functions. What for?
Let`s see typical situation: I have a number of objects and I want them to be in quick access. For a number of reasons, Map doesn`t fit: i.e. key value is a natural part of internal object structure and cannot be taken out as different key object (it is possible but not good solution). Of course your hashed Index container is an ideal solution here.
Of course in some situations I want object to be found quickly and I can compute hash. But all I have in AIndex is AIndex::Find[Prev,Next,Last](const T& x, unsigned _hash)
But I don`t need this x variable, I just want object(s) who`s hash value is _hash. Also this function could make testing (are any objects with such hash value) easier.

Maybe you might want to use "HashBase" directly?

Mirek

Subject: Re: Understanding pick behaviour and containers
Posted by Mindtraveller on Fri, 02 May 2008 13:24:40 GMT
View Forum Message <> Reply to Message

HshBase seems too low-level class since I`d have to duplicate all the AIndex functionality. The only thing is needed - to make "gate" Find(unsigned) function to the one in protected member

HashBase.

Subject: Re: Understanding pick behaviour and containers
Posted by mirek on Mon, 05 May 2008 10:08:55 GMT
View Forum Message <> Reply to Message

Sorry, but I am not sure I get it.

I thought the problem is that you cannot extract the key from the structure (e.g. to use parallel Index).

Anyway, in that case I think the best you can achieve is to have hashvalue of key and do the testing yourself. I do not see what Index functionality you can use in this case.

Maybe some small code-snippet would enlight me

Mirek