Subject: Proposal on *Index: access via hash value
Posted by Mindtraveller on Tue, 31 Mar 2009 11:37:52 GMT
View Forum Message <> Reply to Message

Recently I`ve rewritten a number of classes from VectorMap towards heavily Index usage. But IMO this class lacks access by hash value. To show why it is important I`ll give an example of typical situation.
Let`s imagine I have a number of CollectionElement class objects. Each object is identified by it`s unique value. Hash value is derived from this unique id. I do also have a number of another objects which contain information based on CollectionElement-s. To keep this system serializable, I refer to CollectionElements by their unique ids. The problem arises when I try to get requested object by this id. *Index class has no means to reach object by it`s hash. Once found, I would certainly use FindNext() - so multi-keying isn`t problem here.
You may say that there`s a *Map class, and it gives me access by id value. But IMO in this sutiation it is more than needed. Using *Map will make me duplicate keys: internal CollectElement object id is duplicated as the *Map key. It doesn`t seem quite well.
May be it will be really iseful to add something like
bool AIndex::Check(unsigned _hash);
int AIndex::FindFirst(_hash);
?

Subject: Re: Proposal on *Index: access via hash value
Posted by mirek on Tue, 31 Mar 2009 12:01:36 GMT
View Forum Message <> Reply to Message

I am not quite sure where Index fits. You have

Index<CollectionElement> ?

Mirek

Subject: Re: Proposal on *Index: access via hash value
Posted by Mindtraveller on Tue, 31 Mar 2009 13:04:21 GMT
View Forum Message <> Reply to Message

Yes

Subject: Re: Proposal on *Index: access via hash value
Posted by mirek on Tue, 31 Mar 2009 13:23:36 GMT
View Forum Message <> Reply to Message

How do you finds items then? Using fake CollectionElement as key?

Mirek

Subject: Re: Proposal on *Index: access via hash value
Posted by mrjt on Tue, 31 Mar 2009 14:21:41 GMT
View Forum Message <> Reply to Message

luzr wrote on Tue, 31 March 2009 14:23How do you finds items then? Using fake CollectionElement as key?

Mirek
I think thats the problem isn't it?

You have something like this no?

```
struct CollectionElement : public Moveable<CollectionElement>
{
  String unique_id;
  Value data; // Whatever
};

struct Referer : public Moveable<Referer>
{
  String unique_id;
  Value data; // Whatever
  String collection_id;
}

Index<CollectionElement> collection;
Vector<Referer> ref;

int FindRefCollection(Referer &ref) {
  unsigned collection_hash = GetHashValue(ref.collection_id);
  // Now we need to find the CollectionElement by hash only
  // but the closest you can get is Find(T &x, unsigned hash)
  return -1;
}
```

Since AIndex::hash is protected you can of course just inherit and add the memebr yourself, but it wouldn't be ideal.

Subject: Re: Proposal on *Index: access via hash value
Posted by Mindtraveller on Tue, 31 Mar 2009 18:17:33 GMT
View Forum Message <> Reply to Message

mrjt, yes, I do have very close situation.

UPD: Finally U++ forced me to switch back to VectorMap. It seems like Index is not designed for such cases (but these cases happen very often IMO). Two big drawbacks:
1) Unable to find element by hash
2) Unable to change element, due to I have only (const T &) in return. But this seems too strict. Changing the object doesn`t necessary change it`s hash value. This approach forces one to declare all members as mutable.

Subject: Re: Proposal on *Index: access via hash value
Posted by mirek on Wed, 01 Apr 2009 03:26:46 GMT
View Forum Message <> Reply to Message

Well, I think that VectorMap is not that bad in this situation. Also, quite often it is possible to reorganize the code and remove 'id' from T.

As an option, you can consider using HashBase. It is not really documented (?yet), but it implements just hashing without values.

Also worth noting is that hashing itself costs about 20 bytes per element. In this light, wasting 16 bytes to store String key separately is not really critical.

Mirek

Subject: Re: Proposal on *Index: access via hash value
Posted by mrjt on Wed, 01 Apr 2009 07:49:11 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Tue, 31 March 2009 19:17mrjt, yes, I do have very close situation.

UPD: Finally U++ forced me to switch back to VectorMap. It seems like Index is not designed for such cases (but these cases happen very often IMO). Two big drawbacks:
1) Unable to find element by hash
2) Unable to change element, due to I have only (const T &) in return. But this seems too strict. Changing the object doesn`t necessary change it`s hash value. This approach forces one to declare all members as mutable.
I don't understand these problems.
1) Why don't you just use the hash as the VectorMap key (VectorMap<unsigned, CollectionElement>())? Then you can search by hash directly, although I don't really see the benefit of this for VectorMap.
2) Unless I've misunderstood then this is just wrong. both the Get() and operator[] methods have non-const variants. You can even get direct non-const access to the internal Value vector. If you want to change the Key then you use GetKey and SetKey. This just isn't a problem.

And as I stated previously if you still really want to use Index the code required to add find by hash

is minimal (30-40 lines, mostly a copy of the Index class declaration.)