

## Re: Hash table in C++? STL?

**Source:** [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.cpp/2004-04/2746.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.cpp/2004-04/2746.html)

---

**From:** Siemel Naran ([SiemelNaran\\_at\\_REMOVE.att.net](mailto:SiemelNaran_at_REMOVE.att.net))

**Date:** 04/15/04

Date: Thu, 15 Apr 2004 05:31:12 GMT

"Claudio Puviani" <[puviani@hotmail.com](mailto:puviani@hotmail.com)> wrote in message news:MMhfc.43722

> "Kevin Goodsell" <[usenet2.spamfree.fusion@neverbox.com](mailto:usenet2.spamfree.fusion@neverbox.com)> wrote

>> *Does anyone know what will need to be provided in  
>> order to hash on a particular type? std::map requires  
>> a less-than comparison operation -- what operations  
>> would a hash-based map need?*

> *A hashing function and an equality operation are the minimum requirements,  
but*

> *requiring a relational comparison (such as less\_than) instead of equality  
would*

> *allow for certain optimizations on the part of the implementers (such as  
using*

> *a tree-like structure instead of lists for the buckets). I don't know  
whether*

> *the standard will require equality or a relational comparator.*

>

> *Sadly, the hashing function is where all the complexity lies and I'd be*

> *pleasantly surprised if more than 1% of programmers knew what constitutes  
a*

> *good hashing function. With a bad hashing function (and a naive bucket*

> *strategy), performance can quickly degenerate to much worse than a  
balanced*

> *tree on large data sets.*

I don't think I'm in that 1%. A naive guess is to add up all the letters and modulo by the bucket size, which they say should be prime, but why is this?

```
size_t hash(const char * s, int buckets) {  
    size_t h = 0;  
    for ( ; *s; ++s) h += *s;  
    return h%bucket.  
}
```

But we need hash only the first maxchars characters.

## comp.lang.c++. Re: Hash table in C++? STL?

To answer Kevin's question about the hash function, it should be like this in general

```
class hash {
public:
    typedef char char_type;
    typedef typename std::vector<T>::size_type hash_type;
    explicit hash(size_t maxchars);
    hash_type operator()(char_type begin, char_type end, size_t numbuckets)
    const;
private:
    size_t maxchars;
};
```

The hash may increase numbuckets dynamically as it grows, so numbuckets is a parameter to operator(). But the number of chars to hash is a constant, and should not change as we increase or decrease the number of buckets, so it is argument to the constructor and gets stored in the class.

In our code above we rely on an implicit conversion from char\_type to hash\_type (which seems not reasonable), and hash\_type::operator+= (which seems reasonable as hash\_type is just an unsigned integer).