More Hashing and Template Functions

9-23-2003

Opening Discussion

- What did we talk about last class?
- Do you have any questions about the assignment?
- Java has a class called StringTokenizer that can greatly help when parsing out text.
- Talk about the "customers" view of the application.

Example Code

- Let's now look at some code for hashing and operator overloading.
- One interesting operator to overload is the () operator. It allows you to use an object the way you normally use a function. Classes that have this overloaded are typically called functors.

Resolving Collisions without Linking

- Linking is less than ideal for resolving conflicts because it makes the code more complex and adds some overhead to the structure. It isn't exactly fast either if the lists start getting long.
- The latter part could be fixed by handling collisions with more complex data structures, but that makes the former worse. Instead we'd like to just put things somewhere else in the hash.

Open Addressing

- The alternative to linking is to put everything in the hash table. We do this by making our hash function give us a sequence of numbers. The first is the "normal" hash value. If that is full though we look at other ones.
- Open addressing has problems with deleting because the search times can be longer than the number of collisions with a given space.

Linear Probing

- $h(k,i)=(h'(k)+i) \bmod m$
- Start at $h(k)$ then move forward one at a time.
- This has the advantage of simplicity but we have problems with clustering. If a block of entries is full then the search times for both inserts and searches can be long.

Quadratic Probing

- $h(k,i)=(h'(k)+c_1 i+c_2 i^2) \bmod m$
- You can't use just any value of m, c_1 and c_2. The reason is that as i goes from 0 to m-1 you have to hit every element of the hash.
- The normal problem of clustering goes away, but you still have a problem with a collision that the sequences are the same.

Double Hashing

- $h(k,i)=(h_1(k)+ih_2(k)) \bmod m$
- So we start at a given position based on the key and take steps whose size depends on the key also.
- $h_2(k)$ must be relatively prime to m (they can share no common factors other than 1). This is easy if m is a power of 2 and $h_2(k)$ always returns an odd number.

Rehashing

- Obviously open addressing guarantees and you never have $\alpha>1$. However, it also means that the hash can actually fill up.
- Typically when you are using this scheme you keep α below a certain value and if it every gets above that you have to increase the size of the hash and completely rebuild it.

Template Functions

- We can template functions in a similar way to how we template classes. The main difference is that C++ will infer the type used if that type is an argument to the function. If you don't pass an argument of that type it must be specified.

```
template<class MyClass>
void func(MyClass &mc) {}

template<class Stuff>
void map(List<Stuff> &list) {}
```

Templates with Restrictions

- Templates can also be used when you don't want to allow any type to be used. The real general rule for templates is that you can use any type that has definitions for all the methods (including operators) that are called in the code.
- A templated sort might require operator<. The indexed file class for assignment #2 might require methods that can read and write with a FILE.

Iterators in the STL

- STL stands for the Standard Template Library. It is a significant component of the standard libraries in C++. As you can guess from the name, templates are used significantly to provide flexibility. As was just discussed, the template really only requires methods with the right names that can fit in.
- Part of the power of the STL comes from the use of iterators. These are classes that let you walk through a container in a uniform way.

More on Iterators

- Iterators have overloaded ++, --, and unary * operators. This allows you to walk through any container with a uniform syntax.
- The Iterator class itself is a public class inside the container.

```
template<class Cont,class Func>
void applyToList(Cont &c,Func &map) {
     for(Cont::iterator it=c.begin();
          it!=c.end(); it++) {
          map(*it);
     }
}
```

Minute Essay

- What are the advantages of open addressing? What are some problems with it?
- The test code for assignment #2 is due Thursday.