Data Structures with C++ : CS189

Lecture 5-2: Iterators

Recap

- List and Vector are ADTs, or "shapes" for data
- The two are good at different things
  - Vector is fast for lookups but slow for changes
  - List is slow for lookups but fast for changes
  - That is a generalization
- Vectors are powered by allocating and re-allocating an array
- Lists are powered by newing and deleting Nodes

Abstract Accessing

- The user doesn't know a Vector is an array and a List is nodes
  - Information hiding!
- So how can the user access the data if they don't know the format?
  - Vector has an index, but not everyone does
- An "iterator" is the abstraction of "one piece of data"
  - In List, only we know it is the mData inside a ListNode

Iterator Struct

```
list<int> X;
.
.
for( auto iter = X.begin();
iter != X.end();
iter++)
{
        cout << (*iter)
}
// "auto" here means
"Whatever type begin just
returned.  I'll be that."
```

- Since the concept of "something that iterates a List" is so specific to List, its iterator is again an inner class
- But we do want the user to have this one, so it is public
  - The node struct was private
- So its name is actually List<>::iterator
  - Every ADT has an iterator, and they can't all have the same name of just "iterator"
  - The syntax for the name of a class inside a templated class is weird. That's what "auto" is for

Using Iterators

- In that loop you saw Begin and End
  - for(auto iter = X.begin(); iter != X.end(); iter++)
  - ^^^ Memorize that
- The user doesn't know they want "head to tail" or "0 to size - 1". They want to go from "Begin to end"
- Begin is the first good piece of data
- End is the first bad piece of data
  - For vector, that's "size"
  - For list, that's "tail"
- An iterator outside of begin and end gives "unspecified behavior"

const

```
// Can't change the rock pointed at
const Rock *X;
// Can't change the pointer
Rock *const X;
// Method safe to call with const pointer
void Func() const
// Safe reference that can't change target
const Rock &X;
// Unchangable value
int SIZE = 11;
```

- In general "const" means "can't change"
- Its meaning depends on which of five-ish different places you put it
- The syntax is pretty clear, but one is connected to a fundamental ADT fact
  - When you put something in a container, it gets copied
  - When you ask for something from a container, you get a reference to the actual thing in there
  - Be careful you don't accidentally copy data out by sticking it in a temp variable
  - Always using pointers instead of objects in containers keeps you from being confused

Go to Canvas

Lecture part is short today because historically the List assignment generates a lot of questions.

End

Look at the syllabus and take some time to skim the chapter we'll do next