Data Structures with C++ : CS189

Lecture 3-2: STL and Functors

Recap

- Some ways of using data were so common that early programmers gave them names
- List, vector, set, map, stack, queue, unordered_map, priority_queue, and more
  - cplusplus.com is the place to go to find all the methods each of these have

Iterators

Iterators

- You do not know how these work inside
  - Information Hiding!
- So how could you loop through a List to output the data?
  - Array index, pointer, struct?
  - You couldn't even get to the private data if you knew how they worked anyway
- An "Iterator" is an abstract way to refer to "Something in a ADT"
  - Begin, next, next, next, next, end
- Just like how ADTs share some commands, most have Iterators that work the same

Iterator Syntax

```
for(
auto iter = myList.begin();
iter != myList.end();
iter++ )
{
        cout << (*iter);
// Operator asterisk is as if
it were iter.GetData()
}

for (auto one : myList){}
"For each thing in the list"
```

- You will type the line on the left a million times in your life
  - It is the STL version of a for loop
  - for(int i = 0; i < count; i++){}
- "auto" means "VS, please figure out the type for me"
  - It looks at the return type of what's on the right and assumes that's what you want
- "*iter" is meant to look like a dereference operator, but remember that iter is not a pointer
  - It's actually an operator overload

Functors

Functors

- Sorting ints is easy
- Sorting strings is easy
- How do you sort Students?
- What if you want to sort Students two different ways?
- STL abstraction is essential
  - ADT is the abstraction of a bunch of data
  - Iterator is the abstraction of one piece of data
  - Functor is the abstraction of an operation

Why Functors?

- Some ADTs require their data to be sortable
  - set and map store data smallest to largest
- ints work because they already understand <
  - A templated class is just copy-paste-find-replace
  - "if(x < y)" is somewhere in the sort code
- We could put a Student in a set if we gave it operator<
  - But then we couldn't chose to sort on gpa sometimes and age other times

How Functors Work

```
// Inside the Student class

struct CompareByAge {
bool operator()
        (const Student &A,
        const Student &B)
        const
        {
        return A.age <
                B.age;
        }
};
```

- Back in the day, you would pass a pointer to a function to choose sort style
- C++ lets us overload operators.
  - Including ()
- Calling a function in C looks exactly the same as operator() in C++
  - Backwards compatibility!
- Still don't want to give () to Student, so we give it to one of our functors
- Student can have many of these
  - "Student::CompareByAge" because of scope

Using Functors: Priority Queue

```
priority_queue< Student,
        vector<Student>,
        Student::ByAge
> myPQ;

// Type,
// Internal memory choice,
// Name of functor
```

- A queue let the first data in be the first data out
- PQ changes that to "the most important data is the first one out"
- "Most important" is where the functor comes in
  - By default, "biggest" using < is the most important
    - ints also have access to built-in functors like "less" and "greater" in <functional>

Go to Canvas Quiz

Remember, you are still not supposed to know how these work inside. But even 175 needs to know how to use them.

End

Diving in to our first ADT next week