Iterators

In C++, iterators provide an interface for client code access to data in a way that abstracts away the internals of the data structure.

An instance of an iterator is a current location in a pass through the data structure:

<table>
<thead>
<tr>
<th>Type</th>
<th>Cur. Location</th>
<th>Current Data</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linked List</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hypercube</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The iterator minimally implements three member functions:

- `operator*`, Returns the current data
- `operator++`, Advance to the next data
- `operator!=`, Determines if the iterator is at a different location

Implementing an Iterator

A class that implements an iterator must have two pieces:

1. [Implementing Class]: Must implement:
   - `operator*`, Returns the current data
   - `operator++`, Advance to the next data
   - `operator!=`, Determines if the iterator is at a different location

2. [Implementing Class’ Iterator]:
   A separate class (usually an internal class) that extends `std::iterator` and implements an iterator. This requires:

Using an Iterator

```
#include <vector>
#include <string>
#include <iostream>

struct Animal {
    std::string name, food;
    bool big;
    Animal(std::string name = "blob", std::string food = "you", bool big = true) :
        name(name), food(food), big(big) { /* nothing */ }
};

int main() {
    Animal g("giraffe", "leaves", true),
        p("penguin", "fish", false),
        b("bear");
    std::vector<Animal> zoo;
    zoo.push_back(g);
    zoo.push_back(p);   // std::vector's insertAtEnd
    zoo.push_back(b);
    for (std::vector<Animal>::iterator it = zoo.begin();
         it != zoo.end(); it++ ) {
        std::cout << (*it).name << " " << (*it).food << std::endl;
    }
    return 0;
}
```

Q: What does the above code do?

Locations of `::begin` and `::end` iterators:

<table>
<thead>
<tr>
<th>Type</th>
<th>::begin()</th>
<th>::end()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linked List</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For-Each loop with Iterators

```
for ( const Animal & animal : zoo ) {
    std::cout << animal.name << " " << animal.food << std::endl;
}
```
Trees!
“The most important non-linear data structure in computer science.”
- David Knuth, The Art of Programming, Vol. 1

We will primarily talk about binary trees:
- What’s the longest English word you can make using the vertex labels in the tree (repeats allowed)?
- Find an edge that is not on the longest path in the tree. Give that edge a reasonable name.
- One of the vertices is called the root of the tree. Which one?
- Make a “word” containing the names of the vertices that have a parent but no sibling.
- How many parents does each vertex have?
- Which vertex has the fewest children?
- Which vertex has the most ancestors?
- Which vertex has the most descendants?
- List all the vertices in b’s left subtree.
- List all the leaves in the tree.

Definition: Binary Tree
A binary tree T is either:

Tree Property: Tree Height

Tree Property: Full

Tree Property: Perfect

CS 225 – Things To Be Doing:
1. Programming Exam A starts tomorrow (Thursday!)
2. MP3 has been released; extra credit deadline is Monday!
3. lab_quacks in lab this week
4. Daily POTDs