Iterators

About

Iterators are a way that we can traverse data structures without needing an index number (or when items aren't stored at a specific index).

Topics

1. Iterator basics

2. For each loops

Iterator Basics

1. Iterator Basics

For some structures, we can access elements at random with the subscript operator. This is true for the STL vector, which is implemented as a dynamic array behind-the-scenes…

```
vector<string> myList = { /* stuff */ };
for (   unsigned int i = 0;
        i < myVec.size();
        i++ )
{
    cout << myVec[i] << endl;
}
```

Notes

1. Iterator Basics

But remember with our Linked List class, when we wanted to get an item at position *n*, we would have to traverse there every time.

```
T& At( int index )
{
    int counter = 0;
    Node<T>* ptrCurrent = ptrFront;

    while ( counter < index )
    {
        ptrCurrent = ptrCurrent->ptrNext;
        counter++;
    }

    return ptrCurrent->data;
}
```

If we made a for-loop on the outside to get item 0, then 1, then 2, then 3, each time we call At() it would have to traverse from the beginning to that index all over again.

Notes

1. Iterator Basics

It would be much more efficient if we could keep some sort of reference to the data at that point, to continue from if we needed to…

```
Data* current = linkedList.At( 3 );
cout << *current << endl;

// later…

current++;
cout << *current << endl;
```

We could do this with a pointer, but there is a special class that is specifically meant for this kind of functionality.

Notes

1. Iterator Basics

Iterators!

An iterator acts a lot like a pointer, but is not exactly the same.

```
vector<string> myVec = { /* stuff */ };
for (   vector<string>::iterator it = myVec.begin();
        it != myVec.end();
        it++ )
{
    cout << *it << endl;
}
```

Notes

1. Iterator Basics

We begin a for loop with an iterator by declaring the iterator variable. This includes:

```
vector<string>::iterator it;
```

Data-type: An iterator for a vector type

Variable name: it

```
vector<string> myVec = { /* stuff */ };
for (   vector<string>::iterator it = myVec.begin();
        it != myVec.end();
        it++ )
{
    cout << *it << endl;
}
```

Notes

1. Iterator Basics

For the first part of our for loop, we assign its value

```
it = myVec.begin()
```

Assign it to the first item of myVec

```
vector<string> myVec = { /* stuff */ };
for (   vector<string>::iterator it = myVec.begin();
        it != myVec.end();
        it++ )
{
    cout << *it << endl;
}
```

Notes

1. Iterator Basics

With indices, we'd usually say "loop while i < size", but in this case, we loop while "*it* is not the end".

```
it != myVec.end()
```

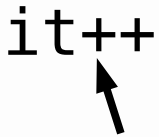
Loop while *it* is not at the end.

```
vector<string> myVec = { /* stuff */ };
for (   vector<string>::iterator it = myVec.begin();
        it != myVec.end();
        it++ )
{
    cout << *it << endl;
}
```

Notes

1. Iterator Basics

And we move forward to the next item in the vector with the ++ operator.



Move forward one position.

```
vector<string> myVec = { /* stuff */ };
for (   vector<string>::iterator it = myVec.begin();
        it != myVec.end();
        it++ )
{
    cout << *it << endl;
}
```

Notes

1. Iterator Basics

Then, to get the data that the iterator is assigned to, we use the * operator, just like a pointer.

```
cout << *it << endl;
```

Output the element of the vector stored in *it*.

```
vector<string> myVec = { /* stuff */ };
for (   vector<string>::iterator it = myVec.begin();
        it != myVec.end();
        it++ )
{
    cout << *it << endl;
}
```

Notes

1. Iterator Basics

This is the most common usage of an iterator you will probably use, but they can get more complicated. You can also create your own iterators by inheriting from std::iterators types.

```
vector<string> myVec = { /* stuff */ };
for (   vector<string>::iterator it = myVec.begin();
        it != myVec.end();
        it++ )
{
    cout << *it << endl;
}
```

Notes

For Each loops

2. For Each loops

In Java, C#, and other languages, “for each” loops are pretty common, but they were only recently added to C++ in C++11.

```
for s in stuff:
    print( s )
```

Python

```
foreach ( $stuff as $s )
{
    echo( $s );
}
```

PHP

```
foreach( string s in stuff )
{
    Console.WriteLine( s );
}
```

C#

Notes

2. For Each loops

in C++, a for each loop looks like this:

```
for ( string s : stuff )
{
    cout << s << endl;
}
```

Notes

Conclusion

We did a thing.

Next time we do more things.