CS 225

#12: Iterators

February 11, 2018 · *Fagen-Ulmschneider, Zilles*

Data Structures Review

- List ADT
  - Linked Memory Implementation ("Linked List")
    - O(1) insert/remove at front/back
    - O(1) insert/remove after a given element
    - O(n) lookup by index
  - Array Implementation ("Array List")
    - O(1) insert/remove at front/back
    - O(n) insert/remove at any other location
    - O(1) lookup by index

	Queue	Stack
Operations + Data Order:		
Implementation:		
Runtime:		

Example 1

```
Queue<int> q;
q.enqueue(3);
q.enqueue(8);
q.enqueue(4);
q.dequeue();
q.enqueue(7);
q.dequeue();
q.dequeue();
q.enqueue(2);
q.enqueue(1);
q.enqueue(3);
q.enqueue(5);
q.dequeue();
q.enqueue(9);
```

Example 2

```
Queue<char> q;
q.enqueue('m');
q.enqueue('o');
q.enqueue('n');
…
q.enqueue('d');
q.enqueue('a');
q.enqueue('y');
q.enqueue('i');
q.enqueue('s');
q.dequeue();
q.enqueue('h');
q.enqueue('a');
```

Three designs for data storage in data structures:

1. T & data
2. T * data
3. T data

Tradeoffs between our data store strategies:

1. Who manages the lifecycle of the data?
2. Is it possible to store a NULL as the data?
3. If the data is manipulated by user code while stored in our data structure, are the changes reflected within our data structure?
4. What is the relative speed compared to other methods?

	Storage by Reference	Storage by Pointer	Storage by Value
Lifecycle management of data?			
Possible to insert NULL?			
External data manipulation?			
Speed			

Accessing Every Element in Our List / Queue / [Anything]

Suppose we want to look through every element in our data structure. What if we don't know what our data structure even looks like?

	Linked List
	Array
	Hypercube

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:

Type	Cur. Location	Current Data	Next
Linked List			
Array			
Hypercube			

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]:

2. [Implementing Class' Iterator]:
   A separate class (usually an internal public member class) that extends `std::iterator` and implements an iterator.

Using an Iterator

stlList.cpp

```
 1  #include <vector>
 2  #include <string>
 3  #include <iostream>
 4
 5  struct Animal {
 6    std::string name, food;
 7    bool big;
 8    Animal(std::string name = "blob", std::string food = "you",
    bool big = true) :
 9      name(name), food(food), big(big) { /* nothing */ }
10  };
11
12  int main() {
13    Animal g("giraffe", "leaves", true),
             p("penguin", "fish", false), b("bear");
14    std::vector<Animal> zoo;
15
16    zoo.push_back(g);
17    zoo.push_back(p);   // std::vector's insertAtEnd
18    zoo.push_back(b);
19
20    for ( std::vector<Animal>::iterator it = zoo.begin();
                                        it != zoo.end(); it++ ) {
21      std::cout << (*it).name << " " << (*it).food << std::endl;
22    }
23
24    return 0;
25  }
```

Q: What does the above code do?

For-Each loop with Iterators

stlList-forEach.cpp

```
20  for ( const Animal & animal : zoo ) {
21    std::cout << animal.name << " " << animal.food << std::endl;
22  }
```

CS 225 – Things To Be Doing:

1. Programming Exam A starts Thursday
2. MP2 due tonight; MP3 released tomorrow
3. lab_quacks released on Wednesday
4. Daily POTDs