Chapter 3
Linked Lists, Vectors & Analysis of their behavior

CptS 223 - Spring 2017 - Aaron Crandall

Today's Agenda

- Announcements
- Quiz results
- Some Linux fun
- Chapter 3 structures
  - Linked Lists vs. Vectors & time complexity
  - Software engineering discussion & the STL
  - Skipping stacks in class (you know this one!)
  - Queues (if we have time)

Announcements

- I'll be shipping Homework #1 soon - it's a written thing, not coding
- Open labs are (so far):
  - T/Th @5:00-8:00 in Sloan 353's center room
- Google is here tomorrow & Wednesday
  - The schedule is on our Blackboard site
- VCEA Technical Career Fair
  - October 3rd, 10am-3pm in Beasley Coliseum
  - Definitely the best opportunity to get a tech internship
  - Ensure your resume is ready and you have good clothes!
  - Contact Sandi Brabb in the PPEL for hiring help/ideas: https://vcea.wsu.edu/ppel/

Neat stuff!

USS Zumwalt

Zumwalt bridge/command center

REALLY neat stuff!

What OS runs the latest destroyer’s comps?

Hint: it’s not Microsoft Windows

We think it might be because of this incident: “Sunk by Windows NT”

While Microsoft continues to trumpet the success of its NT operating system over Unix-based systems, the US Navy is having second thoughts about putting NT at the helm. A system failure on the USS Yorktown last September temporarily paralyzed the cruiser, ...

"For about two-and-a-half hours, the ship was what we call 'dead in the water,'" said Commander John Singley of the Atlantic Fleet Surface Force.

http://archive.wired.com/science/discoveries/news/1998/07/13987

A few points about the article

- “... problem appeared to be more political than technical”
- “... politics were played in the assigning of the contract -- there was not a discussion of engineers”
- “... when the software attempted to divide by zero, a buffer overrun occurred -- crashing the entire network and causing the ship to lose control of its propulsion system.”
- “... when reliability is of utmost importance, Unix-like systems are preferable” (report by Navy reviewers after the fact)

What OS runs the latest destroyer? :-)

Hint: it’s not Microsoft Windows

Though, it *is* commanded by Captain Kirk

Quiz #1 review

- When you state the Big-O complexity for an algorithm, what does that normally represent?
- What does the time complexity Big-Omega (Big-Ω) for an algorithm represent?
- If T(N) = log^2 N + 1.5N^3 + 2N + 1000, what is the Big-O of T(N)?
- Given this set of nested for loops, what is the Big-O time you'd expect from the code?

```
for( i = 0; i < N; i++ )
  for( j = 0; j < M; j++ )
```

Last of quiz #1 review

- For a Bubblesort function that takes an array of N elements and sorts it, what is the space complexity of the function? void bubbleSort(int array[], int n)
- Which of these are features of the OS kernel running on most computers?

- [ ] Managing hardware
- [ ] Giving you a word processor
- [ ] Starting programs
- [ ] Providing a user interface
- [ ] Managing memory
- [ ] Being the shell
- [ ] Scheduling processors on the CPU

Opportunity to ask about git & the assignments here

https://git.eecs.wsu.edu

git clone

git checkout [branchname]

editing files

git add

git commit

git push

Diving into Chapter 3

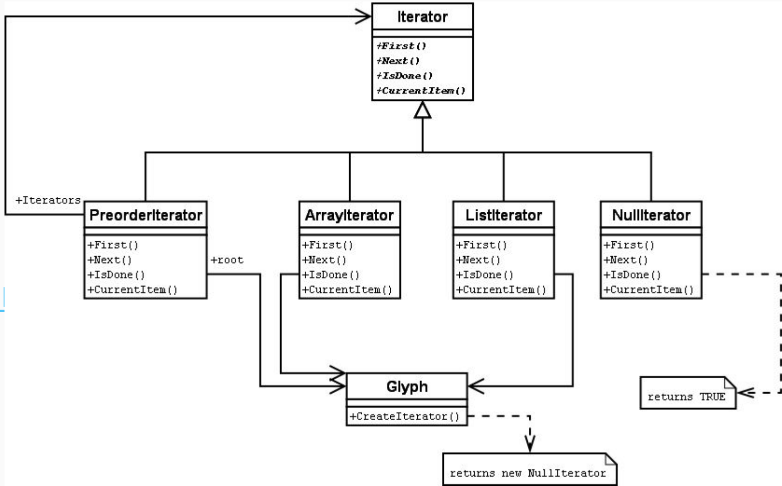
- Abstract Data Types (ADTs)
    - Mathematical abstractions of behavior and storage for data
    - No specification for implementation in the concept
    - Lists, sets, graphs vs. integers, reals, booleans

Common concept of ADTs

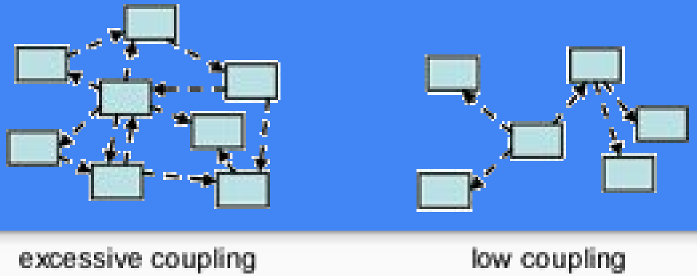
- Common operations:
  - Add
  - Remove
  - Size
  - Contains
  - Union
  - Find
- How are these translated for some common ADTs?
  - Stacks, queues, lists, vectors

C++ (and the STL) allow for hiding implementation details

- Objects allow the hiding of implementation details
  - Proper API design
  - Proper coupling and cohesion
- An API is: Application Program Interface
  - Externally exposed ways to manipulate an object
  - Can be formally defined, and should be!
  - See: UML and other modeling languages
    - Unified Modeling Language
    - http://www.tutorialspoint.com/uml/uml_over



Coupling and Cohesion



- Software engineering philosophy for structural design
- Coupling:
  - How objects are tied to one another's behavior
  - Coupling is the manner and degree of interdependence between software modules
  - A measure of how closely connected two routines or modules are
  - The strength of the relationships between modules
  - It's a common desire to reduce coupling in your software design, hence ADTs in their own objects with simple APIs that do not expose implementation details nor data access

Cohesion is normally the opposite of coupling (why it’s not a single scale…?)

- When objects have high cohesion, they normally do one thing well
    - They have a single job and do not “reach into” other objects or modules, except via well defined APIs
    - Cohesion means that the module/object doesn’t expose its own data to outsiders
    - Remember the whole public: and private: sections in your class definitions?
        - Yeah, that’s to allow you to make a hidden API and data store
- A well designed object should be drop-in replaceable by another one with the same API and behavior implementation, even if the internal implementations are radically different: see databases for an example

High cohesion and low coupling has many benefits, here’s a couple of examples

1) Implementation by different groups
   a) Just follow the spec, stick to the API, and your modules will integrate, right?
2) DoD is very interested in APIs and specifications because of contractors
3) NASA & other robust coding environments
   a) Requires every module be done three times with different algorithms, all same result
   b) Each implementation should be a drop-in replacement for the others
4) Code reuse is a goal of software engineering
   a) It’s also incredibly difficult to do and made more so by highly coupled code

All of this is the basis of ADTs

- A good general concept with:
  - A good API based on the basic operations
  - High cohesion in the implementation with proper data hiding
  - Low coupling between modules
    - Which is why you can just create a vector<T> without knowing how it’s built
    - vector< int > list;
    - vector< vector< int > > matrix;

Vector operations

- How long to accomplish these? How is a vector implemented in STL?
  - add (end, start, insert)
  - delete (@location)... on average it takes?
  - printList
  - findKth
- What happens when the vector gets full?
  - I sense an n^2 issue hiding under the surface!

Linked list (singly)

- How long to accomplish these?
    - add (start, end, insert)
    - find(x)
    - findKth(k)
    - printList
    - remove(k)
- What kind of linked list is provided by the STL?

STL vector vs. list

- Operations:
  - int size()
  - void clear()
  - bool empty() (is_empty?)
  - void push_back(Object & x)
  - void pop_back( )
  - const Object & back( )
  - const Object & front( )
- Only for list
  - void push_front(const Object & x)
  - void pop_front( )
- Only for vector:
  - Object & operator[] (int idx) - no check!
  - Object & at(int idx) - safe
  - int capacity( )
  - void reserve(int newCapacity) - force size

Iterators

- Used to issues commands to the middle of the list
  - Allows you to increment and decrement your way through the data
  - BUT! Doesn't allow you direct access to the implementation:
    - Good coupling and cohesion behavior
- iterator begin()
- itr++ ,   *itr,   itr1==itr2,   itr1 != itr2
- iterator insert(obj),   erase (pos), erase(start, end) -> can do a range
  - Erase works for both list and vector, but what's the cost for each?

Example: delete every other item

- Remove every other item from your list of numbers:
  - 6, 5, 1, 4, 2 ----> 5, 4
- Runs in linear time for a list, but.... Quadratic time for a vector! N^2

~~5~~	73	~~-2~~	10	~~3~~	7

- I ran into a discussion about how the CPU cache reduces this impact in some implementations, but we haven’t talked much about the memory hierarchy yet. That’ll come with B+ Trees.

Implemented with a vector

```
template <typename Container>
void removeEveryOtherItem( Container & vec) {
    typename Container::iterator itr = lst.begin( );
    while( itr != vec.end( ) ) {
        itr = vec.erase( itr );
        if( itr != vec.end( ) )
            ++itr;
    }
}
```

Implemented with a list

```
template <typename Container>
void removeEveryOtherItem( Container & lst ) {
    typename Container::iterator itr = lst.begin( );
    while( itr != lst.end( ) ) {
        itr = lst.erase( itr );
        if( itr != lst.end( ) )
            ++itr;
    }
}
```

Implemented with a list

```
template <typename Container>
void removeEveryOtherItem( Container & lst ) {
	typename Container::iterator itr = lst.begin( );
	while( itr != lst.end( ) ) {
		itr = lst.erase( itr );
		if( itr != lst.end( ) )
			++itr;
	}
}
```

Wait! That doesn’t change anything in the implementation. Stop messing with us, Crandall!

- What actually matters is what we initialize and pass to the function.
- The resulting implementation detail behavior is entirely hidden from our algorithm:

list<int> myData;

vs.

vector<int> myData;

Both are passed just the same:

removeEveryOtherItem(myData);

But one operates in linear time (N), the other in quadratic (N^2)

Chapter 3 continues, but we won't

- Implementation details of vector and list
  - We don't do that in class (boring to do as a group!)
  - You still should read through it and work through the algorithms in your heads
  - These "complex" types aren't that complex under the hood and you could implement them yourselves without a huge amount of work
- Show online documentation for STL types:

  Vector Stack Queue List

If we still have time...

- Skipping stacks, you’ve done those
- Onto queues
  - Array based implementation
  - Linked list-based implementation
- To the whiteboard!
- Queueing theory - all kinds of fun stuff and HIGHLY valuable to companies
  - See: phone/cell, routers, FedEx/UPS, anyone with a call center, order handling
- Finish reading Chapter 3. We’ll be doing Chapter 4 on Wednesday
  - Chapter 4 will take quite a while, so we’re nearing the end of the sprint to the fun stuff

GO TO THE HACKATHON

CRIMSON CODE