CSCI 104
Overview

Mark Redekopp
Administrative Issues

• Preparation
  – Basic if, while, for constructs
  – Arrays, linked-lists
  – Structs, classes
  – Dynamic memory allocation and pointers
  – Recursion

• Syllabus
  – http://bytes.usc.edu/cs104 (Not bits.usc.edu)

• Expectations
  – I'll give you my best, you give me yours...
    • Attendance, participation, asking questions, academic integrity, take an interest
  – Treat CS104 right!
  – Let's make this fun
More Helpful Links

• Remedial modules
  – http://ee.usc.edu/~redkekopp/csmodules.html

• Class website
  – http://bytes.usc.edu/cs104 (not bits.usc.edu)
An Opening Example

• Consider a paper phonebook
  – Stores names of people and their phone numbers

• What operations do we perform with this data
  – You: Lookup/search
  – Phone Company: Add, Remove

• How is the data stored and ordered and why?
  – Sorted by name to make lookup faster...
  – How fast? That's for you to figure out...

• What if it was sorted by phone number or just random? What is the worst case number of records you'd have to look at to find a particular person's phone number?
Opening Example (cont.)

• Would it ever be reasonable to have the phonebook in a random or unsorted order?
  – What if the phonebook was for the residence of a town with only a few residents
  – What if there was a phonebook for Mayflies (life expectancy of 1-24 hours)
    • Might want to optimize for additions and removals
    • Plus, a mayfly doesn't have fingers to dial their phones so why would they even be trying to search the phonebook

• Main Point: The best way to organize data depends on how it will be used.
  – Frequent search
  – Frequent addition/removals
  – Addition/removal patterns (many at once or one at a time)
Why Data Structures Matter?

- Modern applications process vast amount of data
- Adding, removing, searching, and accessing are common operations
- Various data structures allow these operations to be completed with different time and storage requirements

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Insert</th>
<th>Search</th>
<th>Get-Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted List</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Balanced Binary Search Tree</td>
<td>$O(lg\ n)$</td>
<td>$O(lg\ n)$</td>
<td>$O(lg\ n)$</td>
</tr>
<tr>
<td>Heap</td>
<td>$O(lg\ n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Recall $O(n)$ indicates that the actual run-time is bounded by some expression $a*n$ for some $n > n_0$ (where a and $n_0$ are constants)
Importance of Complexity

<table>
<thead>
<tr>
<th>N</th>
<th>O(1)</th>
<th>O(log₂N)</th>
<th>O(n)</th>
<th>O(n*log₂N)</th>
<th>O(n²)</th>
<th>O(2^n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>4.3</td>
<td>20</td>
<td>86.4</td>
<td>400</td>
<td>1,048,576</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>7.6</td>
<td>200</td>
<td>1,528.8</td>
<td>40,000</td>
<td>1.60694E+60</td>
</tr>
<tr>
<td>2000</td>
<td>1</td>
<td>11.0</td>
<td>2000</td>
<td>21,931.6</td>
<td>4,000,000</td>
<td>#NUM!</td>
</tr>
</tbody>
</table>
Abstract Data Types

• Beginning programmers tend to focus on the code and less on the data and its organization
• More seasoned programmers focus first on
  – What data they have
  – How it should be organized
  – How it will be accessed
• An **abstract data type** describes what data is stored and what operations are to be performed
• A **data structure** is a specific way of storing the data implementing the operations
• Example **ADT**: **List**
  – Data: items of the same type in a particular order
  – Operations: insert, remove, get item at location, set item at location, find
• Example **data structures** implementing a **List**:
  – Linked list, array, etc.
Transition to Object-Oriented

• Object-oriented paradigm fits nicely with idea of ADTs
  – Just as ADTs focus on data and operations performed on it so objects combine data + functions

• Objects (C++ Classes) allows for more legible, modular, maintainable code units

• Suppose you and a friend are doing an electronic dictionary app. Your friend codes the dictionary internals and you code the user-interface.
  – You don't care how they implement it just that it supports the desired operations and is fast enough
  – Abstraction: Provides a simplified interface allowing you to reason about the higher level logic and not the low level dictionary ops.
  – Encapsulation: Shields inside from outside so that internals can be changed w/o affecting code using the object
Course Goals

• Learn about good programming practice with object-oriented design
  – Learn good style and more advanced C++ topics such as templates, inheritance, polymorphism, etc.

• Learn basic and advanced techniques for implementing data structures and analyzing their efficiency
  – May require strong fundamentals including mathematical analysis
  – This is why we couple CS 104 and CS 170