CSCI 104
Abstract Data Types

Mark Redekopp
David Kempe
MAN, I SUCK AT THIS GAME.
CAN YOU GIVE ME
A FEW POINTERS?

0x3A28213A
0x6339392C,
0x7363682E.

I HATE YOU.
Abstract Data Types

• DAPS defines an **abstract data type**, or ADT, as:
  – Specification/model for a group of values/data and the operations on those values
• The model allows us to separate...
  – The decision of what data structure to use and how it will be used in our higher level application
  – And the implementation of the specific data structure
• DAPS defines a **data structure** as:
  – An implementation of an ADT in a given programming language
• Each ADT we will examine in this course has certain:
  – Well defined operations and capabilities that are often useful
  – Time & space advantages
  – Time & space disadvantages
• You need to know those operations, advantages and disadvantages

---

Data Abstraction & Problem Solving with C++, Carrano and Henry will henceforth be abbreviated as DAPS
3 Popular ADTs

- List
- Dictionary/Map
- Set
- (Possible 4th: Priority Queue)
Lists

• Ordered collection of items, which may contain duplicate values, usually accessed based on their position (index)
  – Ordered = Each item has an index and there is a front and back (start and end)
  – Duplicates allowed (i.e. in a list of integers, the value 0 could appear multiple times)
  – Accessed based on their position ( list[0], list[1], etc. )

• What are some operations you perform on a list?
## List Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Input(s)</th>
<th>Output(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>Add a new value at a particular location shifting others back</td>
<td>Index : int</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td>Remove value at the given location</td>
<td>Index : int</td>
<td>Value at location</td>
</tr>
<tr>
<td>get / at</td>
<td>Get value at given location</td>
<td>Index : int</td>
<td>Value at location</td>
</tr>
<tr>
<td>set</td>
<td>Changes the value at a given location</td>
<td>Index : int</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>empty</td>
<td>Returns true if there are no values in the list</td>
<td></td>
<td>bool</td>
</tr>
<tr>
<td>size</td>
<td>Returns the number of values in the list</td>
<td></td>
<td>int</td>
</tr>
<tr>
<td>push_back / append</td>
<td>Add a new value to the end of the list</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>find</td>
<td>Return the location of a given value</td>
<td>Value</td>
<td>Int : Index</td>
</tr>
</tbody>
</table>
Maps / Dictionaries

- Stores key,value pairs
  - Example: Map student names to their GPA
- Keys must be unique (can only occur once in the structure)
- No constraints on the values
- What operations do you perform on a map/dictionary?
- No inherent ordering between key,value pairs
  - Can't ask for the 0th item...

Grade Inflation in the Ivy League!!

<table>
<thead>
<tr>
<th>Student</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Billy Bruin&quot;</td>
<td>2.5</td>
</tr>
<tr>
<td>&quot;Tommy Trojan&quot;</td>
<td>3.7</td>
</tr>
<tr>
<td>&quot;Harry Harvard&quot;</td>
<td>4.3</td>
</tr>
<tr>
<td>&quot;Dale Duck&quot;</td>
<td>2.5</td>
</tr>
</tbody>
</table>
## Map / Dictionary Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Input(s)</th>
<th>Output(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert / add</td>
<td>Add a new key,value pair to the dictionary (assuming its not there already)</td>
<td>Key, Value</td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td>Remove the key,value pair with the given key</td>
<td>Key</td>
<td></td>
</tr>
<tr>
<td>Get / lookup</td>
<td>Lookup the value associated with the given key or indicate the key,value pair doesn't exist</td>
<td>Key</td>
<td>Value associated with the key</td>
</tr>
<tr>
<td>In / Find</td>
<td>Check if the given key is present in the map</td>
<td>Key</td>
<td>bool (or ptr to pair/NUL)</td>
</tr>
<tr>
<td>empty</td>
<td>Returns true if there are no values in the list</td>
<td></td>
<td>bool</td>
</tr>
<tr>
<td>size</td>
<td>Returns the number of values in the list</td>
<td></td>
<td>int</td>
</tr>
</tbody>
</table>
Set

• A set is a dictionary where we only store keys (no associated values)
  – Example: All the courses taught at USC (ARLT 100, ..., CSCI 104, MATH 226, ...)

• Items (a.k.a. Keys) must be unique
  – No duplicate keys (only one occurrence)

• Not accessed based on index but on value
  – We wouldn't say, "What is the 0th course at USC?"

• In DAPS textbook Chapter 1, this is the 'bag' ADT

• What operations do we perform on a set?
## Set Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Input(s)</th>
<th>Output(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert / add</td>
<td>Add a new key to the set (assuming it's not there already)</td>
<td>Key</td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td>Remove</td>
<td>Key</td>
<td></td>
</tr>
<tr>
<td>In / Find</td>
<td>Check if the given key is present in the map</td>
<td>Key</td>
<td>bool (or ptr to item/NULL)</td>
</tr>
<tr>
<td>empty</td>
<td>Returns true if there are no values in the list</td>
<td></td>
<td>bool</td>
</tr>
<tr>
<td>size</td>
<td>Returns the number of values in the list</td>
<td></td>
<td>Int</td>
</tr>
<tr>
<td>intersection</td>
<td>Returns a new set with the common elements of the two input sets</td>
<td>Set1, Set2</td>
<td>New set with all elements that appear in both set1 and set2</td>
</tr>
<tr>
<td>union</td>
<td>Returns a new set with all the items that appear in either set</td>
<td>Set1, Set2</td>
<td>New set with all elements that appear in either set1 and set2</td>
</tr>
<tr>
<td>difference</td>
<td>Returns a set with all items that are just in set1 but not set2</td>
<td>Set1, Set2</td>
<td>New set with only the items in set1 that are not in set2</td>
</tr>
</tbody>
</table>
Intersection, Union, Difference

• May be familiar from CS 170

• Set intersection
  – $S_1 \cap S_2$

• Set Union
  – $S_1 \cup S_2$

• Set Difference
  – $S_1 - S_2$
What's Your ADT?

- Scores on a test
- Students in a class
- Courses & their enrollment
- Temperature Reading at a location
- Usernames and password
- Index in a textbook
- Facebook friends
- Adjacent countries of a map

- List
- Set (maybe List)
- Map (Key = course, Value = enrollment)
- List
- Map
- Map
- Set
- Set
Some Implementation Details

• List
  – An array acts as a list
  – Index provides ordering
    • First at location 0
    • Last at location n-1

• Set
  – Can use an array
  – Must check for duplicate on insertion
    • O(n) solution
  – Can we do better? Yes…

• Map
  – Can also use an array
  – Again check for duplicate key on insertion

```c
struct Pair{
    string key;
    double value;
};
```