CSE 373: Data Structures and Algorithms

Brad Chamberlain
University of Washington
Spring 1999

What is a Data Structure?

data structure –
**Observation**

- Data is an attribute common to all programs
  - programs *process, manipulate, store, display, gather*
  - data may be *information, numbers, images, sound*
- Each program must decide how to store data
- Choice influences program at every level:
  - execution speed
  - memory requirements
  - maintenance (debugging, extending, etc.)

---

**Course Goals**

- To introduce several standard data structures
- To teach how data structures are evaluated
- To determine when each structure is useful
- To give you the ability to design, build, and evaluate your own data structures
What about Algorithms?

*algorithm* – a description of a process useful for completing a specific task

Algorithms are often closely tied to the selection of a data structure (in this class anyway).

---

C Data Types

- **basic types:**
  - `char`, `int`, `double`, etc.
  - pointers (e.g., `char *`, `int *`, `double *`)
- **compound types:**
  - arrays (e.g., `int [26]`, `double [100][100]`)
  - structures (e.g., `struct`{
      `int x,y;`
      `double len;`
  })

---
Building Data Structures

typedef char name[32];
typedef enum {BIOCHM, ECON, EE, MATH, PREMAJ} dept;
typedef struct _student {
    name first, last;
    int UWID;
    name email;
    char college;
    dept major;
    int class;
} student;
typedef student class[80];

Data Structure class

class cse373;

cse373

<table>
<thead>
<tr>
<th>Roberto Benigni</th>
<th>James Coburn</th>
<th>Judi Dench</th>
<th>Gwyneth Paltrow</th>
<th>Jackie Chan</th>
</tr>
</thead>
<tbody>
<tr>
<td>9525413</td>
<td>9624331</td>
<td>9515423</td>
<td>9731429</td>
<td>9716423</td>
</tr>
<tr>
<td>benigni</td>
<td>jcoburn</td>
<td>judid</td>
<td>gpaltrrow</td>
<td>jchan</td>
</tr>
<tr>
<td>C</td>
<td>J</td>
<td>C</td>
<td>C</td>
<td>J</td>
</tr>
<tr>
<td>MATH</td>
<td>EE</td>
<td>MATH</td>
<td>PREMAJ</td>
<td>EE</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Abstract Data Types (ADTs)

Is the `class` type an ADT?

Example: `FindMajor()`

```java
void FindMajor(class, dept);  
  - takes a class and a department as arguments  
  - prints all the students in the class in that major
```

How would `FindMajor()` be implemented for our current `class` implementation?

Could changing `class` improve the performance of `FindMajor()`?
ADT Tensions

Ideal: a fast, elegant ADT that uses little memory

Generates tensions:
- time vs. space
- performance vs. elegance
- generality vs. simplicity
- one operation's performance vs. another's

Another Example

• Spring registry ADT for UW – stores which students are taking which classes
• Supported operations:
  int TakingClass(int UVID, int SLN);
  • tells if the student is taking the course specified by SLN
  void PrintSchedule(int UVID);
  • prints the schedule of the student
  class MakeClassList(int SLN);
  • creates a class type for the given SLN
A Naive Implementation

```c
const int num_courses = 7000;
const int num_students = 33000;

typedef int registry[num_students][num_classes];
```

Evaluating this Implementation

What are the advantages of this implementation?

What are the disadvantages?

How could we improve the implementation?
The Myth of ADTs

Not a perfect black box:
- knowing how an ADT will be used can lead to a good choice of implementation
- also, knowledge of an ADT's implementation may change how a client uses it

*But... ADTs are still a useful concept*

*Use motivates design*