CSE 326: Data Structures

Winter 2009
Steve Seitz
Lecture 1

Today’s Outline

• Introductions
• Administrative Info
• What is this course about?
• Review: queues and stacks

Course Information

Web page:
http://www.cs.washington.edu/326


Errata: handout.
Communication

Instructors
› cse326-instr@cs.washington.edu
› (or our individual addresses)

Announcements
› cse326a_sp08@u.washington.edu
› (you are automatically subscribed @u)

Discussion
› Discussion board linked off home page

Written homeworks

Written homeworks (8 total)
› Due at the start of class on due date
› Typically assigned/due on Fridays
› No late homeworks accepted

Projects

• Programming projects (3 total, with phases)
  › In Java
  › Eclipse encouraged
  › Turned in electronically
  › Can have one “late day” for extra 24 hours
    Must email TA in advance

• Work in teams only on explicit team projects
  › Appropriate discussions encouraged – see website

Overall grading

Grading
25% - Written Homework Assignments
25% - Programming Assignments
20% - Midterm Exam (in class, fifth week)
25% - Final Exam
5% - Best of Programming or Exams
Project/Homework Guides

On the website - note especially:
› Gilligan’s Island rule applies
› Homeworks: Use pseudocode, not code. A human being is reading your homeworks.
   • See website for pseudocode example.
› Projects: code is only 40% of your grade!
› Spend time commenting your code as you write - it will help you be a better programmer.

Section

Meeting times
› AA – Thurs 9:30 - 10:20 – MGH 228 (Soyoung)
› AB – Thurs 2:30 - 3:20 – MGH 228 (Eric)
› AC – Thurs 1:30 - 2:20 – EEB 045 (Brent)
What happens there?
› Answer questions about current homework
› Previous homeworks returned and discussed
› Discuss the project (getting started, getting through it, answering questions)
› Finer points of Java and environs
› Reinforce lecture material

Homework for Today!!

Reading in Weiss
Chapter 1 – (Review) Mathematics and Java
Chapter 2 – (Next lecture) Algorithm Analysis
Chapter 3 – (Project #1) Lists, Stacks, & Queues

Today’s Outline

• Introductions
• Administrative Info
• What is this course about?
• Review: Queues and stacks
Steve’s view of CSE

• 100 level courses, some 300 level
  › how to do stuff

• This course
  › Really cool ways to do stuff

• 400 level courses
  › How to do really cool stuff

Common tasks

• Many possible solutions
  › Choice of algorithm, data structures matters
  › What properties do we want?

  speed
  low memory
  correctness/robustness
  extendibility
  simplicity or elegance

Example: Fibonacci

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fib</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>...</td>
</tr>
</tbody>
</table>

```c
int fib( int n )
{
    if( n <= 2 )
        return 1;
    else
        return fib( n - 1 ) + fib( n - 2 );
}
```
Why should we care?

Moore's Law
Large data sizes
Libraries
- don't cover everything
- choose between functions

Program Abstraction

Problem defn: sort N numbers

Algorithm: quicksort, bubble, merge

Implementation: quicksort.java, quick.c

How to be an expert

Tricks of the trade
- Knowledge — what's out there
- Analysis — how to compare algo
- Style — design skills

Data Abstraction

Abstract Data Type (ADT):
- Stack: push, pop, ...

Data Structure:
- linkedlist, array

Implementation:
- stack.java, ...
**Terminology**

- **Abstract Data Type (ADT)**
  - Mathematical description of an object with set of operations on the object. Useful building block.
- **Algorithm**
  - A high level, language-independent, description of a step-by-step process.
- **Data structure**
  - A specific organization of the data to accompany algorithms for an abstract data type.
- **Implementation of data structure**
  - A specific implementation in a specific language.

**Today’s Outline**

- Introductions
- Administrative Info
- What is this course about?
- Review: queues and stacks

**First Example: Queue ADT**

- **FIFO: First In First Out**
- **Queue operations**
  - `create`
  - `destroy`
  - `enqueue`
  - `dequeue`
  - `is_empty`

- Enqueue: `G`
- Dequeue: `A`

**Queues in practice**

- Print jobs
- File serving
- Phone calls and operators

(Later, we will consider “priority queues.”)
Array Queue Data Structure

```
enqueue(Object x) {
    Q[back] = x
    back = (back + 1)
}
dequeue() {
    x = Q[0]
    shiftLeftOne()
    return x
}
```

- What's missing in these functions?
- How to find K-th element in the queue?
- What is complexity of these operations?

Circular Array Queue Data Structure

```
enqueue(Object x) {
    assert(!is_full())
    Q[back] = x
    back = (back + 1)
}
dequeue() {
    assert(!is_empty())
    x = Q[front]
    front = (front + 1)
    return x
}
```

- How test for empty/full list?
- How to find K-th element in the queue?
- What is complexity of these operations?
- Limitations of this structure?

Linked List Queue Data Structure

```
void enqueue(Object x) {
    if (is_empty())
        front = back = new Node(x)
    else {
        back->next = new Node(x)
        back = back->next
    }
}
bool is_empty() {
    return front == null
}
```

Circular Array vs. Linked List

- Simple & easy to implement
- No need to manage pointers
- No extra space for parts

- Larger size
- More efficient
- Can handle larger amounts of data
Second Example: Stack ADT

- LIFO: Last In First Out
- Stack operations
  - create
  - destroy
  - push
  - pop
  - top
  - is_empty

Stacks in Practice

- Function call stack
- Removing recursion
- Balancing symbols (parentheses)
- Evaluating postfix or “reverse Polish” notation

Reminder: homework

Reading in Weiss
Chapter 1 – (Review) Mathematics and Java
Chapter 2 – (Next lecture) Algorithm Analysis
Chapter 3 – (Project #1) Lists, Stacks, & Queues