Today's Outline

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

Course Information

Web page:
http://www.cs.washington.edu/332

(or buy 2nd edition—1/3 price on Amazon!)

Communication

Instructors
- cse332-instr@cs.washington.edu
- (or our individual addresses)

Announcements
- cse332a_wi14@u, cse332b_wi14@u
- (you are automatically subscribed @u)

Discussion
- Discussion board linked off home page

Written homeworks

Written homeworks (8 total)
- Assigned each Wednesday
- Due at the start of class following Wednesday
- No late homeworks accepted
Projects

- Programming projects (3 total, with phases)
  › In Java
  › Eclipse encouraged
  › Turned in electronically
  › Can use a "late day" for 1 project of your choice
  **Must email TA in advance**

Project 1 out today

- Soundblaster! Reverse a song
  › a.k.a., “backmasking”
- Use a stack
  › Implement as array and as linked list
- **Read the website**
  › Detailed description of assignment
  › Detailed description of how programming projects are graded
- Phase A due Monday, Jan 13 (11:59pm)
  › Electronic submission

---

Overall grading

Grading
- 25% - Written Homework Assignments
- 30% - Programming Assignments
- 20% - Midterm Exam (Feb 10)
- 25% - Final Exam (March 17)

---

Collaboration

Read policy on website carefully
- HWs must be done solo
  › But you can discuss problems with others as long as you follow the Gilligan’s island rule
- Project 1 is solo (out today)
- Project 2 & 3 with a partner

---

Section

Meet on Thursdays
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, eclipse, etc.
  › 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?

Example: Fibonacci

```
int fib( int n )
{
    if( n <= 2 )
        return 1;
    else
        return fib( n - 1 ) + fib( n - 2 );
}
```

Why should we care?

• Computers are getting faster
  › No need to optimize

• Libraries: experts have done it for you
How to be an expert

- Tricks of the trade
  - Knowledge
  - Analysis
  - Style

Program Abstraction

Problem defn:

Algorithm:

Implementation:

Data Abstraction

Abstract Data Type (ADT):

Data Structure:

Implementation:

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
  - enqueue
  - dequeue
  - is_empty

G enqueue F E D C B dequeue A
Queues in practice

- Print jobs
- File serving
- Phone calls and operators

(Later, we will consider “priority queues.”)

Array Queue Data Structure

```
\text{size - 1}
```

```
enqueue(Object x) {
    Q[back] = x
    back = (back + 1)
}

dequeue() {
    x = Q[0]
    shiftLeftOne()
    Back = (back - 1)
    return x
}
```

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

Circular Array Queue Data Structure

```
\text{size - 1}
```

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

dequeue() {
    assert(!is_empty())
    x = Q[0]
    shiftLeftOne()
    Back = (back - 1)
    return x
}
```

How to test for empty/full list?
How to find K-th element in the queue?
What to do when full?

Circular Array vs. Linked List

- Advantages of circular array?
- Advantages of linked list?

Linked List Queue Data Structure

```
front A B C D E back
```

```
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
}
```

Second Example: Stack ADT

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

```
A
B
C
D
E
F
```

```
A
B
C
D
E
F
```

Stacks in Practice

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

Assigned readings

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