CSE 326: Data Structures
Lecture #0
Introduction

Bart Niswonger

Summer Quarter 2001

Come up and say hello!

Today's Outline

- Administrative Cruft
- Overview of the Course
- Queues
- Stacks
- Project #1

Course Information

- Instructor: Bart Niswonger
  226D Sieg Hall
  bart@cs.washington.edu
  Office hours: M 11:50-12:50, W 1:50-2:50 & whenever door is open
- TA:
  Ashish Sabharwal ashish@cs
  Office hours: Tues 10:50-11:50 in 226A/B
- Text: *Data Structures & Algorithm Analysis in C++*, 2nd edition, by Mark Allen Weiss

Tutorials

- Unix I
  - Tuesday, June 19th
  - 10:50-11:50 + lab time, Sieg 322
- Templates
  - Thursday, June 21st
  - 10:50-11:50 + lab time, GUG 410
    - In place of section
- Unix Development
  - Tuesday, June 26th
  - 10:50-11:50 + lab time, Sieg 322

REQUIRED

Course Policies

- Roughly weekly written homework due at the start of class on the due date
- Projects (4 total) due by 10PM on the due date
  - you have 4 late days for projects
- Grading
  - homework: 15%
  - projects: 25%
  - midterm: 20%
  - final: 30%
  - best of these: 10%

PARTICIPATION!

Course Mechanics

- 326 Web page: www/education/courses/326/01su
- 326 course directory: /cse/courses/cse326
- 326 mailing list: cse326@cs.washington.edu
  - subscribe to the mailing list using majordomo, see homepage
- Course labs are 232 and 329 Sieg Hall
  - lab has NT machines w/X servers to access UNIX
- All programming projects graded on UNIX/g++

Goals of the Course

- Become familiar with some of the fundamental data structures in computer science
- Improve ability to solve problems abstractly
  - data structures are the building blocks
- Improve ability to analyze your algorithms
  - prove correctness
  - gauge (and improve) time complexity
- Become modestly skilled with the UNIX operating system and X-windows (you'll need this in upcoming courses)

Observation

- All programs manipulate data
  - programs *process, store, display, gather*
  - data can 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.)

What is a Data Structure?

data structure -

What is an Abstract Data Type?

Abstract Data Type (ADT) -

1) An opportunity for an acronym

2) Mathematical description of an object and the set of operations on the object

Data Structures : Algorithms

- Algorithm
  - A high level, language independent description of a step-by-step process for solving a problem
- Data Structure
  - A set of algorithms which implement an ADT

Why so many data structures?

Ideal data structure:

fast, elegant, memory efficient

Generates tensions:

- time *vs.* space
- performance *vs.* elegance
- generality *vs.* simplicity
- one operation's performance *vs.* another's

Dictionary ADT

- list
- binary search tree
- AVL tree
- Splay tree
- Red-Black tree
- hash table

Code Implementation

- Theoretically
  - abstract base class describes ADT
  - inherited implementations implement data structures
  - can change data structures transparently (to client code)
- Practice
  - different implementations sometimes suggest different interfaces (generality *vs.* simplicity)
  - performance of a data structure may influence form of client code (time *vs.* space, one operation *vs.* another)

ADT Presentation Algorithm

- Present an ADT
- Motivate with some applications
- Repeat until browned entirely through
  - develop a data structure for the ADT
  - analyze its properties
    - efficiency
    - correctness
    - limitations
    - ease of programming
- Contrast data structure's strengths and weaknesses
  - understand when to use each one

Queue ADT

- Queue operations
  - create
  - destroy
  - enqueue
  - dequeue
  - is_empty

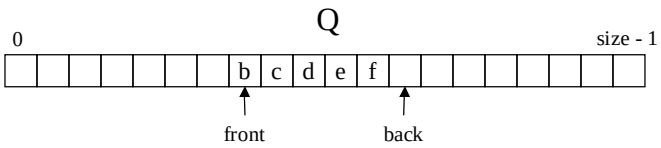
- Queue property: if x is enQed before y is enQed, then x will be deQed before y is deQed

  FIFO: First In First Out

Applications of the Q

- Hold jobs for a printer
- Store packets on network routers
- Hold memory “freelists”
- Make waitlists fair
- Breadth first search

Circular Array Q Data Structure




```
void enqueue(Object x) {
   Q[back] = x
   back = (back + 1) % size
}
Object dequeue() {
   x = Q[front]
   front = (front + 1) % size
   return x
}
```

When is the Q empty?

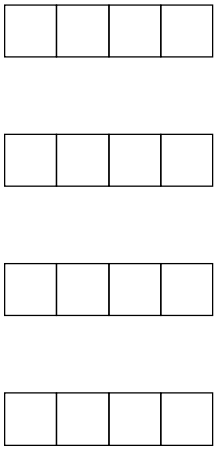
Are there error situations this code will not catch?

What are some limitations of this structure?

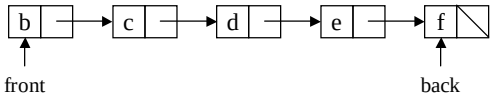
This is *pseudocode*. Do not correct my semicolons.

Q Example

enqueue R
enqueue O
dequeue
enqueue T
enqueue A
enqueue T
dequeue
dequeue
enqueue E
dequeue



Linked List Q Data Structure



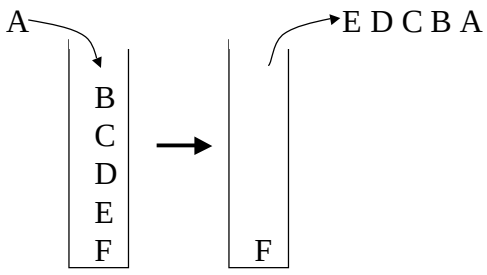

```
void enqueue(Object x) {
   if (is_empty())
        front = back = new Node(x)
   else
        back->next = new Node(x)
        back = back->next
}
```

```
Object dequeue() {
   assert(!is_empty)
   return_data = front->data
   temp = front
   front = front->next
   delete temp
   return temp->data
}
```

Circular Array vs. Linked List

LIFO Stack ADT

- Stack operations
  - create
  - destroy
  - push
  - pop
  - top
  - is_empty



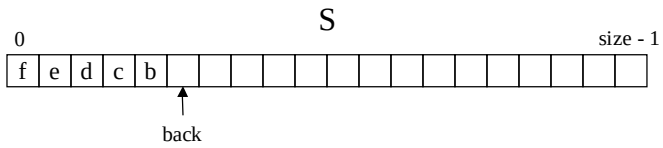
- Stack property: if x is on the stack before y is pushed, then x will be popped after y is popped

  LIFO: Last In First Out

Stacks in Practice

- Function call stack
- Removing recursion
- Balancing symbols (parentheses)
- Evaluating Reverse Polish Notation
- Depth first search

Array Stack Data Structure

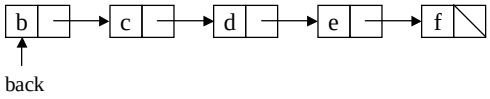



```
void push(Object x) {
   assert(!is_full())
   S[back] = x
   back++
}
Object top() {
   assert(!is_empty())
   return S[back - 1]
}
```

```
Object pop() {
   back--
   return S[back]
}

bool is_full() {
   return back == size
}
```

Linked List Stack Data Structure




```
void push(Object x) {
   temp = back
   back = new Node(x)
   back->next = temp
}
Object top() {
   assert(!is_empty())
   return back->data
}
```

```
Object pop() {
   assert(!is_empty())
   return_data = back->data
   temp = back
   back = back->next
   return return_data
}
```

Data structures you should already know

- Arrays
- Linked lists
- Trees
- Queues
- Stacks

To Do

- Sign up on the cse326 mailing list
- Check out the web page
- Log on to a PC in the course labs and access an instructional UNIX server
- Read Chapters 1 and 3 in the book

Coming Up

- Unix Tutorial
  - Tuesday (tomorrow) 10:50, Sieg 322
- Multi-Lists
- Priority Queues and Heaps
- Templates Tutorial
- First homework