# CSE 326
# Lecture 1

Spring Quarter 1997



## Course Information

- Instructor: Richard Ladner
  311 Sieg Hall
  543-9347, ladner@cs.washington.edu
  Office Hours: Tue - Thurs 11-12
- TA: Jim Fix
  326 Sieg Hall
  fix@cs.washington.edu
  Office Hours: TBA
- Text: Data Structures and Algorithm Analysis in C++, by Mark Allen Weiss



## Course Policies

- Midterm on Friday, May 2, 1997
- Final on Wednesday, June 11,1997, 2:30
- Written homeworks due every Friday
  Homeworks due at the beginning of class and no late homeworks accepted.
- A number of programming projects
  - Projects turned in electonically before 11:30 pm.
  - No late projects accepted.
- Grading
  - midterm 20%
  - final 35%
  - homeworks 20%
  - projects 25%



## Course Mechanics

- CSE 326 WWW site
- CSE 326 class directory /cse/courses/cse326
- CSE 326 mailing list
- Course Laboratory is 329 Sieg Hall
  - Unix workstation lab

## Background Survey

- Computer Science and Enginering Courses

  - CSE 143: Ada or C++

  -CSE 321: Yes or No or current

  -CSE 378: Yes or No or current

  -CSE 341: Yes or No or current



## Goals of Course

- Improve ability to think and solve problems abstractly
  - abstraction is a powerful tool in problem solving
- Improve ability to analyze algorithms
  - proof of correctness
  - time complexity analysis
- Become facile with some of the basic data structures used in computer science and engineering
  - at the abstract data type level (ADT)
  - at the algorithm level
  - at the program design level where C++ used as a tool
- Become modestly skilled with the Unix operating system and X-windows. These will be used in future CSE courses.



## Basic Terminology

- Abstract Data Type - ADT

  Mathematical description of an object and set of related operations on the object.
- Algorithm

  A high level, language independent, decription of a step-by-step process for solving a problem.
- Data Structure

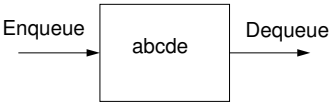
  A family of algorithms for implementing the operations of a given abstract data type. Often, the term "data structure" refers to abstract data type that is of general use in a diversity of applications. Data structures are the infrastructure of programs, the plumbing and wiring that you don't see.
- Implementation of a data structure

  The computer code that realizes the ADT and its data structure. The code can be written in C++ or other object oriented language.



## FIFO Queue ADT

- Queue operation
  - Create
  - Destroy
  - Enqueue
  - Dequeue
  - Is_Empty
  - Is_Full



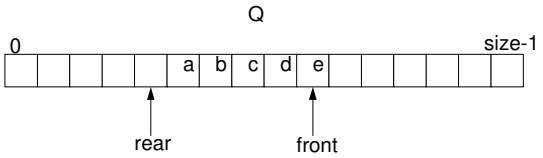

- ADT "axioms" defining the queue (example)

  - If x is Enqueued before y is Enqueued then x will be Dequeued before y will be Dequeued.

  This distinuishes the FIFO Queue ADT from the LIFO Stack ADT where, if x is Pushed before y is Pushed then y will be Popped before x will be Popped.

## One Data Structure for a Queue

- The circular array.




- Enqueue(Q,x):

  Q[rear] := x
  rear := (rear - 1) mod size;
- Dequeue(Q)

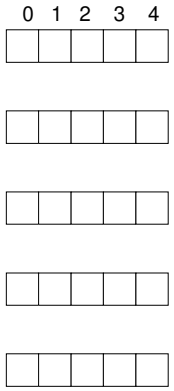
  local integer x;
  x := Q[front];
  front := (front -1) mod size;
  return(x)
- Is_Empty(Q)

  return (front = rear)
- Is_Full(Q)

  return (front = (rear - 1) mod size)



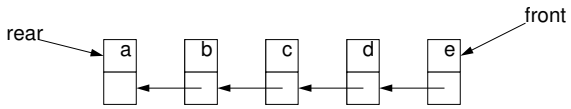
## Queue Example






## Another Data Structure for a Queue

- Linked list




- Enqueue(x)

  local pointer p;

  p := new;
  p.data := x;
  p.next:= nil;
  rear.next := p;
  rear := p
- Dequeue()

  local integer x;

  x := front.data;
  front := front.next;
  return(x)

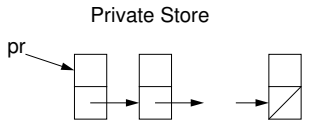


## Private Store

- Private store is implement as a list pr of nodes:
  - Enqueue(x)

    local pointer p;
    ***if pr = nil then p := new***
    ***else {p := pr; pr := pr.next};***
    p.data := x;
    p.next := nil;
    rear.next := p;
    rear := p
  - Dequeue()

    local pointer p; local integer x;
    ***p := front;***
    x := front.data;
    front := front.next;
    ***p.next := pr;***
    ***pr := p:***
    return(x)



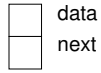
## Memory Management

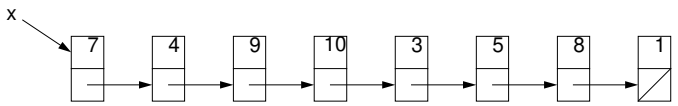
- Private store - get blocks from a private store when possible, return blocks to the private store.
  + efficiently uses blocks of a specific size
  - may use too much storage if private store grows unnecessarly large
- Global allocator's store - always get and return blocks to global allocator.
  + blocks of various sizes can be merged into larger blocks
  - allocator may ineffectively handle different size blocks
  - allocator may be slow compared to a private store
- Garbage Collection - run time system recovers inaccessible blocks from time to time. Used in Smalltalk, Lisp, and Java.
  + no need to bother with returning blocks at all
  - when a garbage collection occurs there may be an undesirable response time



## Recursion Example

- Finding the maximum in a list.




- max(x);

```
local integer m;
if x = nil then return(-infinty)
  else  { m := max(x.next);
     if m > x.data then return(m)
        else return(x.data) }
```



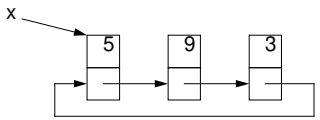
## Guides in Recursion

- Recursive algorithms on well designed data structures are often short, elegant, and efficient.
- In some cases recursive algorithms can be converted to interative algorithms improving efficiency.
- Guides:
  1. There must be some base case or cases.
  2. Progress toward the base cases must be made at each recursive call.
  3. In designing a recursive algorithm alway assume that the recursive calls work.
  4. Never duplicate a recursive call on the same data. (This is different than number 2 above.)



## Causes of Lack of Progress

- Inadequate basis

```
bad(n);
  if n = 0 then return(0)
    else return(bad(n/3 + 1) + n)
```

- Data has a cycle, so basis is never reached.