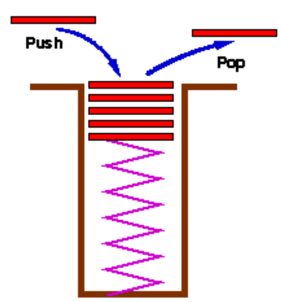
CSE 373 Lecture 5: Lists, Stacks, and Queues

- We will review:
  - Stack ADT and applications
  - Queue ADT and applications
  - Introduction to Trees
- Covered in Chapters 3 and 4 in the text

Stacks

- Stack ADT: Same as List except Insert/Delete allowed only at the *beginning of the list* (the top of the stack).
  - Both operations now take O(1) time!
- "LIFO" – Last in, First out
- Push: Insert element at top
- Pop: Delete (and optionally return) the top element



Stack ADT

- Operations:
  - push(Object x) // Insert item at top of stack
  - pop( ) // Remove topmost item from stack
  - top( ) // Return topmost item without altering stack
  - topAndPop ( ) // Return and remove topmost item from stack
  - isEmpty( ) // Return TRUE if stack is empty
  - MakeEmpty( ) // Make stack empty
- Implementations:
  - Linked list with Header, Header's Next points to top of stack
  - Array-based: Pre-allocate array; top is Array[TopofStack]
    - Push x: Increment TopofStack; set Array[TopofStack] = x
- Run time for each of these operations?

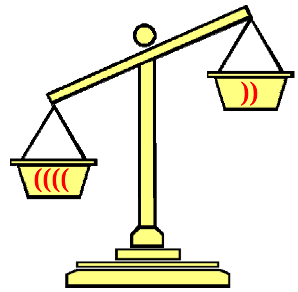
Stack ADT

Go through the Java/C++ code for these in your text

- Operations:
  - push(Object x) // Insert item at top of stack
  - pop( ) // Remove topmost item from stack
  - top( ) // Return topmost item without altering stack
  - topAndPop ( ) // Return and remove topmost item from stack
  - isEmpty( ) // Return TRUE if stack is empty
  - MakeEmpty( ) // Make stack empty
- Run time: All operations are O(1) (except MakeEmpty for Linked List implementation which takes $\Theta(N)$)

Applications of Stacks I: Compilers/Word Processors

- Compilers and Word Processors: Balancing symbols
  - E.g. 2*(i + 5*(17 – j/(6*k)) is not balanced – ")" is missing
- In-Class Exercise: Write a Balance-Checker using Stacks and analyze its running time.



Applications of Stacks I: Compilers/Word Processors

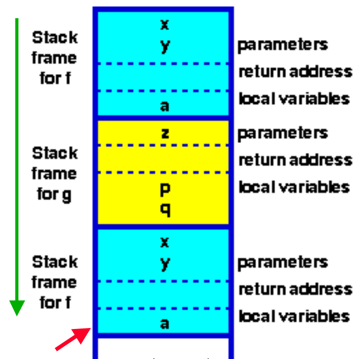
- Balance-Checker using Stacks:
  1. Make an empty stack and start reading symbols
  2. If input is an opening symbol, Push onto stack
  3. If input is a closing symbol:
     If stack is empty, report error
     Else
       Pop the stack
       Report error if popped symbol is not a matching open symbol
  4. If End-of-File and stack is not empty, report error
- Run time for N symbols in the input text: O(N)

Applications of Stacks II: Function Calls

- Handling function calls in programming languages
  - E.g. Two functions `f` and `g` calling each other: Need to store current environment (input parameters, local variables, address to return to, etc.)

```
function f( int x, int y) {
   int a;
   if ( term_cond ) return …;
   a = ….;
   return g( a );
   }

function g( int z ) {
   int p, q;
   p = …. ; q = …. ;
   return f(p,q);
   }
```



Current environment (Top)

A New Twist to Lists: Queues

- Queue = List ADT with inserts only at one end and deletes only at other end
- Queues are "FIFO" – first in, first out
- Instead of Push and Pop, we have Enqueue and Dequeue
- Applications?
- Why not just use stacks?

A New Twist to Lists: Queues

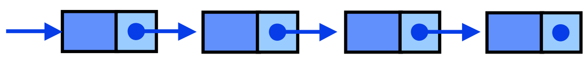
- Queue = List ADT with inserts only at one end and deletes only at other end
- Queues are "FIFO" – first in, first out
- Instead of Push and Pop, we have Enqueue and Dequeue
- Applications? Why not just use stacks?
  - Items can get buried in stacks and not appear at the top for a long time – "not fair to old items"
  - A queue ensures "fairness". For example:
    - Callers waiting on a customer hotline
    - Jobs sent to a printer, etc.

Queue ADT

- Operations:
  - Enqueue(Object x) // Insert item at the back of the list
  - Dequeue( ) // Delete and return front item in list
  - getFront( ) // Return front item in list
  - IsEmpty( ) // Return TRUE if queue is empty
  - isFull( ) // Return TRUE if queue is full
  - MakeEmpty( ) // Make Queue empty
- Implementations:
  - Linked list implementation is natural
    - What pointers do you need to keep track of for O(1) implementation of Enqueue and Dequeue?

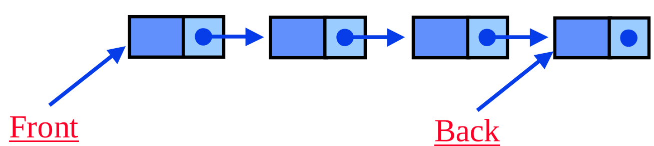
Queue Implementations: Linked List

- Implementations:
  - Linked list implementation is natural
  - What pointers do you need to keep track of for O(1) implementation of Enqueue and Dequeue?



Queue Implementations: Linked List

- Implementations:
  - Linked list implementation is natural
  - What pointers do you need to keep track of for O(1) implementation of Enqueue and Dequeue?



Queue Implementations: Array-Based

- Implementations:
  - Array-based: can use List operations Insert and Delete, but O(N) time for Dequeue

Dequeue (Delete) Enqueue (Insert)

0	1	2	3	...	N-1		MAX
A_1	A_2	A _ 3	A _ 4	…	A _ N		

- How can you make array-based Enqueue and Dequeue O(1) time?

Queue Implementations: Array-Based

- Array-based Enqueue and Dequeue in O(1) time:
  - Use Front and Back indices
    - Enqueue x: increment Back and set Array[Back] = x
    - Dequeue: return Array[Front] and increment Front
  - To make full use of available space (due to Dequeues):
    - Wrap around Front/Back to 0 after MAX
    - *Circular array* implementation

Front Back

0	1	2	3	...	N-1		MAX
A_1	A_2	A _ 3	A _ 4	…	A _ N		

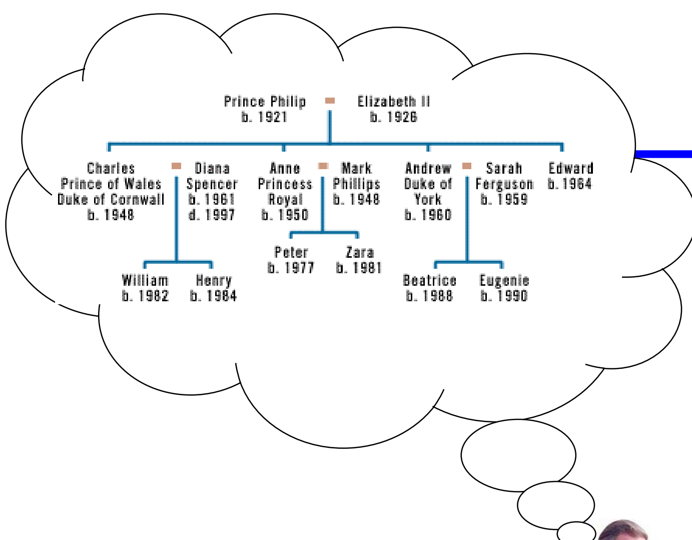
Applications of Queues

- **File servers**: Users needing access to their files on a shared file server machine are given access on a FIFO basis
- **Printer Queue**: Jobs submitted to a printer are printed in order of arrival
- Phone calls made to **customer service hotlines** are usually placed in a queue
- Expected wait-time of real-life queues such as customers on phone lines may be too hard to solve analytically   use queue ADT for **simulating real-life queues**

Queues are for commoners…pray tell, how does a prince represent his royal family hierarchy?

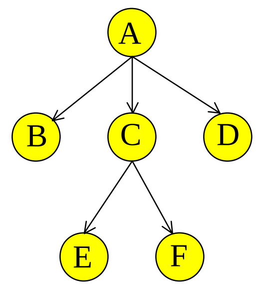
Storing Hierarchical Information

- ✦ Lists, Stacks, and Queues represent linear sequences
- ✦ Data often contain hierarchical relationships that cannot be expressed as a linear ordering
  - ➪ File directories or folders on your computer
  - ➪ Moves in a game
  - ➪ Employee hierarchies in organizations and companies
  - ➪ Classification hierarchies (e.g. phylum, family, genus, species)
  - ➪ Family trees



Tree Jargon

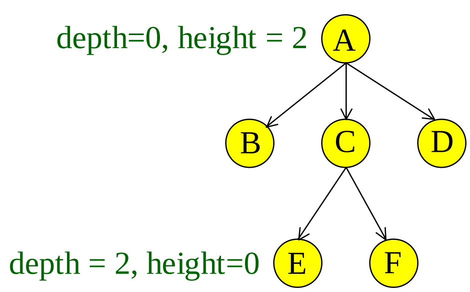
- Basic terminology:
  - nodes and edges
  - root
  - subtrees
  - parent
  - children, siblings
  - leaves
  - path
  - ancestors
  - descendants
  - path length



Note: Arrows denote directed edges
Trees always contain directed edges but arrows are often omitted.

More Tree Jargon

- Length of a path = number of edges
- Depth of a node N = length of path from root to N
- Height of node N = length of longest path from N to a leaf
- Depth and height of tree = ?



Definition and Tree Trivia

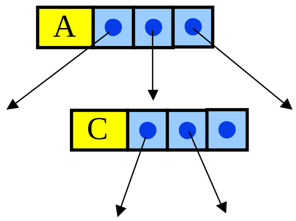
- Recursive Definition of a Tree:
  A tree is a set of nodes that is either:
  a. an empty set of nodes, or
  b. has one node called the root from which zero or more trees ("subtrees") descend.
- A tree with N nodes always has ___ edges
- Two nodes in a tree have at most ___ paths between them?
- Can a non-zero path from node N reach node N again?
- Does depth of nodes in a non-zero path increase or decrease?

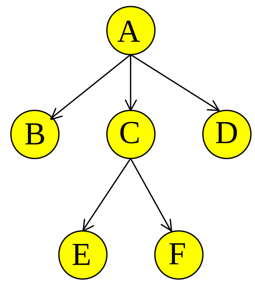
Definition and Tree Trivia

- Recursive Definition of a Tree:
  A tree is a set of nodes that is either:
  a. an empty set of nodes, or
  b. has one node called the root from which zero or more trees ("subtrees") descend.
- A tree with N nodes always has N-1 edges
- Two nodes in a tree have at most one path between them
- Can a non-zero path from node N reach node N again?
  - No! Trees can never have cycles.
- Does depth of nodes in a non-zero path increase or decrease?
  - Depth always increases in a non-zero path

Implementation of Trees: The Obvious

- Obvious Implementation: Node with value and links to children

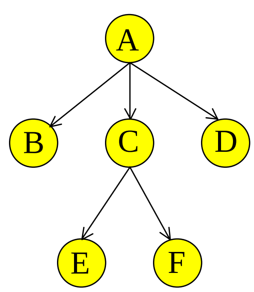




- Problem: Do not know number of children for each node in advance. Wastes space if maximum number of links assumed.

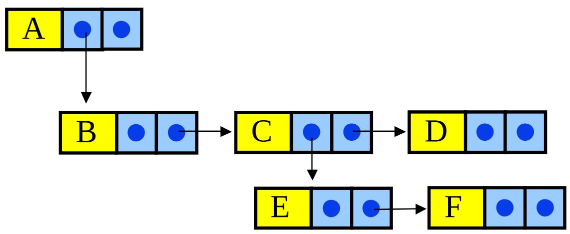
Implementation of Trees: 1st Child/Next Sib

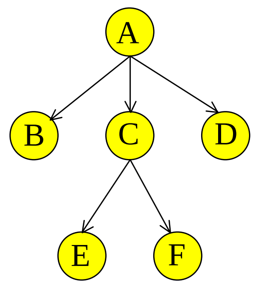
- Better Implementation: 1st Child/Next Sibling Representation
  - Each node has 2 pointers: one to its first child and one to next sibling
  - Can handle arbitrary number of children
  - Exercise: Draw the representation for this tree…



Implementation of Trees: 1st Child/Next Sib

- Better Implementation: 1st Child/Next Sibling Representation
  - Each node has 2 pointers: one to its first child and one to next sibling
  - Can handle arbitrary number of children
  - Exercise: Draw the representation for this tree…





Applications I: Arithmetic Expression Trees

Example Arithmetic Expression:

A + (B * (C / D))

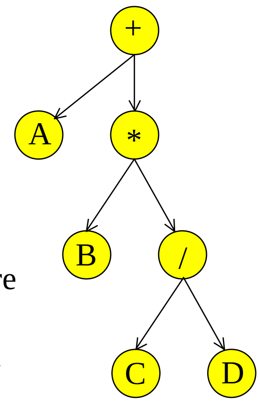
How would you express this as a tree?

Applications I: Arithmetic Expression Trees

Example Arithmetic Expression:

A + (B * (C / D))

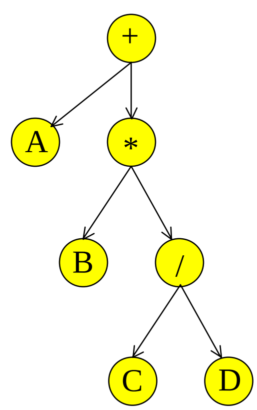
Tree for the above expression:



- Used in most compilers
- No parenthesis need – use tree structure
- Can speed up calculations e.g. replace / node with C/D if C and D are known
- Calculate by traversing tree (how?)

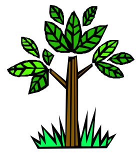
Traversing Trees

- ✦ Preorder: Root, then Children
  + A * B / C D
- ✦ Postorder: Children, then Root
  A B C D / * +
- ✦ Inorder: Left child, Root, Right child
  A + B * C / D



Next class:

Gardening 101: How to take care of your (binary) trees



To do:

Finish Homework no. 1 (due Wednesday, Jan 22)

Finish reading Chapter 3

Read Chapter 4