Lecture 15: Linked Structures

Lewis Caroll
Through the Looking Glass

"The name of the song is called 'Haddocks' Eyes.' "

"Oh, that's the name of the song, is it?" Alice said, trying to feel interested.

"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "

"Then I ought to have said 'That's what the song is called' ?" Alice corrected herself.

"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means,' but that is only what it's called, you know!"

"Well, what is the song, then?" said Alice, who was by this time completely bewildered.

"I was coming to that," the Knight said. "The song really is 'A-sitting On A Gate,' and the tune's my own invention."

Linked vs. Sequential Allocation

Goal: process a collection of objects.

Sequential allocation: put object one after another.
- TOY: consecutive memory cells.
- Java: array of objects.

Linked allocation: include in each object a link to the next one.
- TOY: link is memory address of next object.
- Java: link is reference to next object.

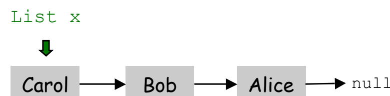
Key distinctions.
- Sequential allocation: random access, fixed size.
- Linked allocation: sequential access, variable size.

Linked Lists

Linked list of strings.
- A recursive data structure.
- A string plus a pointer to another linked list (or empty list).
- Unwind recursion: linked list is a sequence of strings.

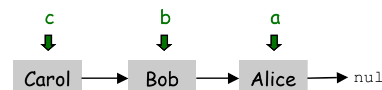
Linked lists in Java.
- Easy to define as a Java class.
- A reference to a `String`.
- A reference to another `List`.
- Use special value `null` to terminate list.

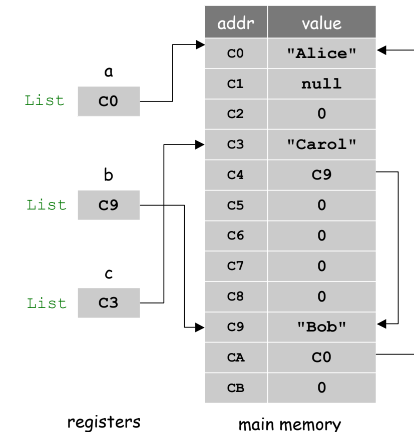
```
public class List {
   private String name;
   private List next;
}
```



Linked List Demo

```
List a = new List();
a.name = "Alice";
a.next = null;
List b = new List();
b.name = "Bob";
b.next = a;
List c = new List();
c.name = "Carol";
c.next = b;
```

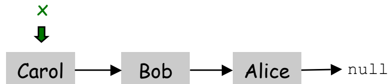




Traversing a List

Paradigm for traversing a null-terminated linked list.

```
for (List x = c; x != null; x = x.next) {
   System.out.println(x.name);
}
```




```
% java List
```


Stack and Queue ADTs

Fundamental data type.

- Set of operations (add, remove, test if empty) on generic data.
- Intent is clear when we insert.
- Which object do we remove?

Stack.

- Remove the object most recently added. ("last in first out")
- Analogy: cafeteria trays, surfing Web.

Queue.

- Remove the object least recently added. ("first in first out")
- Analogy: Registrar's line.

Multiset.

- Remove any object.
- Law professor calls on arbitrary student.

Queue

Queue operations.

- `enqueue` Insert a new object onto queue.
- `dequeue` Delete and return the object least recently added.
- `isEmpty` Is the queue empty?

```
public static void main(String[] args) {
   Queue q = new Queue();
   q.enqueue("Vertigo");
   q.enqueue("Just Lose It");
   q.enqueue("Pieces of Me");
   q.enqueue("Pieces of Me");
   System.out.println(q.dequeue());
   q.enqueue("Drop It Like It's Hot");
   while(!q.isEmpty())
      System.out.println(q.dequeue());
}
```

A simple queue client

More Applications of Queues

Real world applications.

- iTunes playlist.
- Echo filter to store last ten waves.
- Dispensing requests on a shared resource (printer, processor).
- Asynchronous data transfer (file IO, pipes, sockets).
- Data buffers (iPod, TiVo).
- Graph processing (stay tuned).

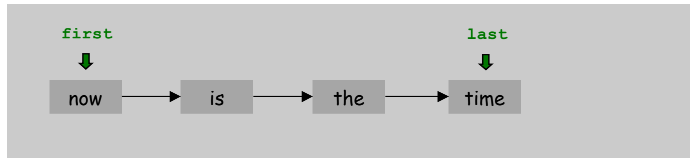
Simulations of the real world.

- Traffic analysis of Lincoln tunnel.
- Waiting times of customers in McDonalds .
- Determining number of cashiers to have at a supermarket.

Queue: Linked List Implementation

Linked list implementation.

- Maintain linked list of elements.
- Let `first` be reference to first node on list.
- Let `last` be reference last node on list.

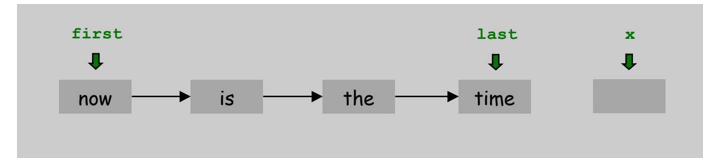


Queue: Linked List Implementation

Linked list implementation.

- Maintain linked list of elements.
- Let `first` be reference to first node on list.
- Let `last` be reference last node on list.

Insert.



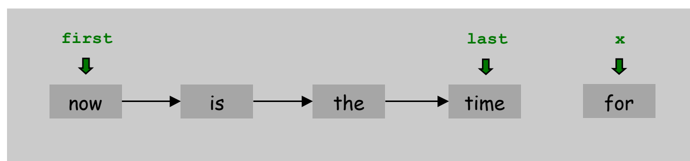

```
List x = new List();
x.item = "for";
last.next = x;
last = x;
```


Queue: Linked List Implementation

Linked list implementation.

- Maintain linked list of elements.
- Let `first` be reference to first node on list.
- Let `last` be reference last node on list.

Insert.



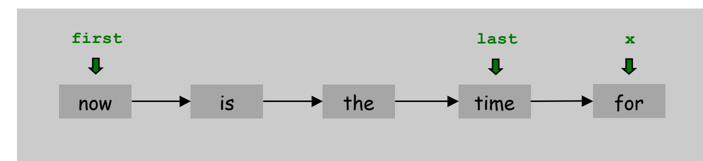

```
List x = new List();
x.item = "for";
last.next = x;
last = x;
```


Queue: Linked List Implementation

Linked list implementation.

- Maintain linked list of elements.
- Let `first` be reference to first node on list.
- Let `last` be reference last node on list.

Insert.



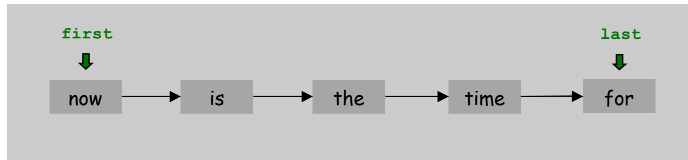

```
List x = new List();
x.item = "for";
last.next = x;
last = x;
```

Queue: Linked List Implementation

Linked list implementation.

- Maintain linked list of elements.
- Let `first` be reference to first node on list.
- Let `last` be reference last node on list.

Insert.



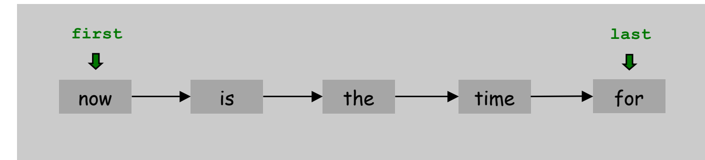

```
List x = new List();
x.item = "for";
last.next = x;
last = x;
```


Queue: Linked List Implementation

Linked list implementation.

- Maintain linked list of elements.
- Let `first` be reference to first node on list.
- Let `last` be reference last node on list.

Delete.



val now

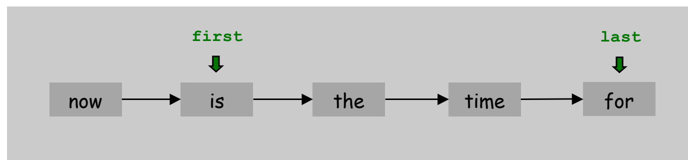
```
String val = first.item;
first      = first.next;
return val;
```


Queue: Linked List Implementation

Linked list implementation.

- Maintain linked list of elements.
- Let `first` be reference to first node on list.
- Let `last` be reference last node on list.

Delete.



val now

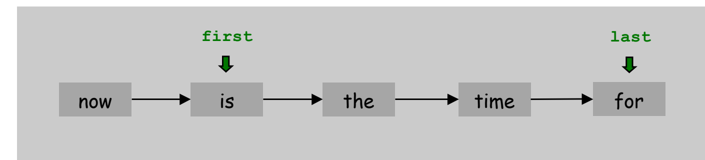
```
String val = first.item;
first      = first.next;
return val;
```


Queue: Linked List Implementation

Linked list implementation.

- Maintain linked list of elements.
- Let `first` be reference to first node on list.
- Let `last` be reference last node on list.

Delete.



val now

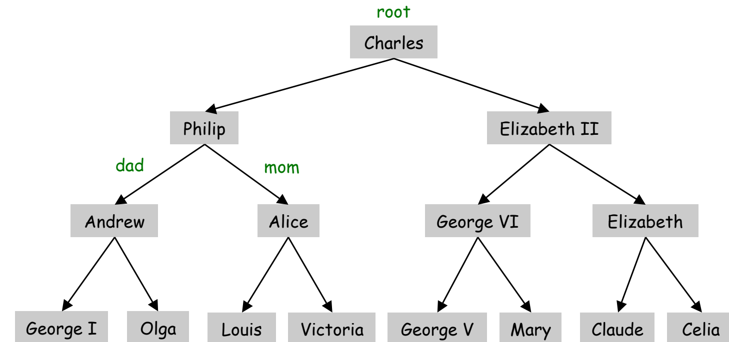
```
String val = first.item;
first      = first.next;
return val;
```

Queue: Linked List Implementation

```
public class Queue {
   private List first;      reference to first element in queue
   private List last;       reference to last  element in queue

   private class List { String item; List next; }    nested class

   public boolean isEmpty() { return (first == null); }

   public void enqueue(String anItem) {
      List x = new List();
      x.item = anItem;
      x.next = null;
      if (isEmpty()) { first      = x; last = x; }
      else           { last.next = x; last = x; }
   }

   public String dequeue() {
      String val = first.item;
      first      = first.next;   ← delete first element
      return val;
   }
}
```


Binary Trees

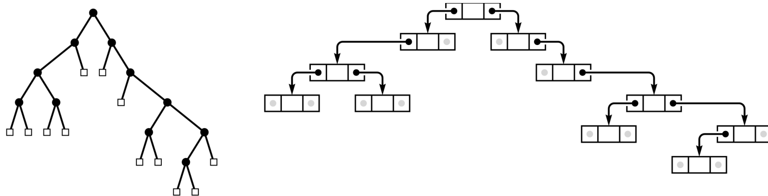
Binary tree.
- Organize homogeneous collection of values (all same type).
- Associate two pointers with each value.
- Use pointers to access each branch of the tree.



Binary Tree: Java Implementation

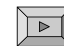
Java implementation of a binary tree of strings is:
- A reference to the `String`.
- A reference to the left `Tree`.
- A reference to the right `Tree`.

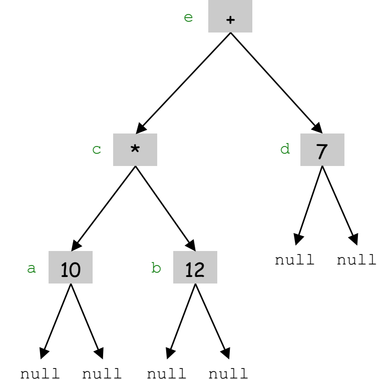
```
public class Tree {
   private String s;
   private Tree left;
   private Tree right;
}
```



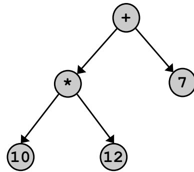
Parse Tree Demo

```
Tree a  = new Tree();
a.s     = "10";
a.left  = null;
a.right = null;

Tree b  = new Tree();
b.s     = "12";
b.left  = null;
b.right = null;

Tree c  = new Tree();
c.s     = "*";
c.left  = a;
c.right = b;

Tree d  = new Tree();
d.s     = "7";
d.left  = null;
d.right = null;

Tree e  = new Tree();
e.s     = "+";
e.left  = c;
e.right = d;
```





Parse Tree Evaluation: Java Implementation



((10 * 12) + (7)) = 127

Parse tree.

- Abstract representation of expression.
- Applications: compilers, computational linguistics.

Evaluating a parse tree.

- If string is an integer return it.
- Else, evaluate both subtrees recursively and return sum or product.

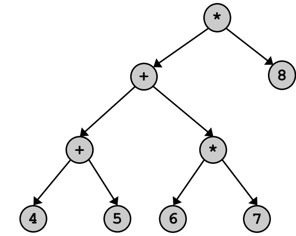
```
public class ParseTree {
   private String s;          // represent data as a string, e.g., "+" or "1234"
   private ParseTree left;    // left subtree
   private ParseTree right;   // right subtree

   public int eval() {
      if      (s.equals("+")) return left.eval() + right.eval();
      else if (s.equals("*")) return left.eval() * right.eval();
      else return Integer.parseInt(s);
   }                          // convert from string to integer
}
```


Preorder Traversal

How do we print out the information?

- Print string.
- Print left subtree recursively.
- Print right subtree recursively.

No parentheses!



((4 + 5) + (6 * 7)) * 8

```
public String toString() {
   if (s.equals("+") || s.equals("*"))
      return s + " " + left + " " + right;
   else
      return s;
}
```

Preorder traversal: * + + 4 5 * 6 7 8

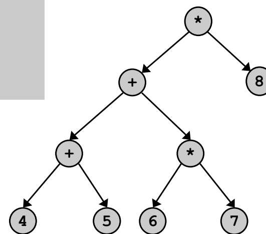
Parse Tree Construction

How do we read it back in and create the tree?

- Read string from standard input.
- If + or * operator, construct left and right subtrees recursively.

```
public ParseTree() {                     // constructor
   s = StdIn.readString();
   if (s.equals("+") || s.equals("*")) {
      left  = new ParseTree();
      right = new ParseTree();
   }
}
```

```
% java ParseTree
* + + 4 5 * 6 7 8
408
```

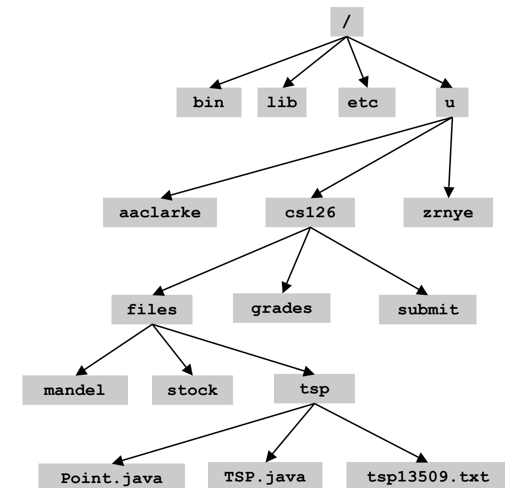


((4 + 5) + (6 * 7)) * 8

Other Types of Trees

Other types of trees.

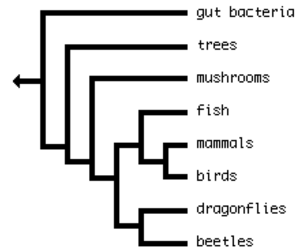
- Family tree.
- Parse tree.
- Unix file hierarchy.



Other Types of Trees

Other types of trees.

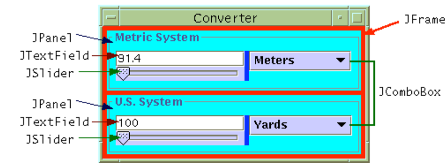
- Family tree.
- Parse tree.
- Unix file hierarchy.
- Phylogeny tree.

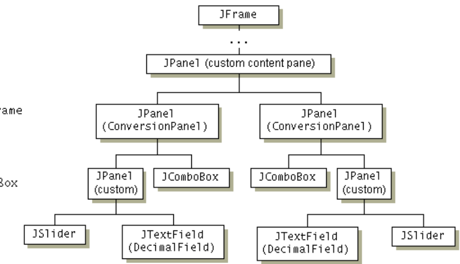


Other Types of Trees

Other types of trees.

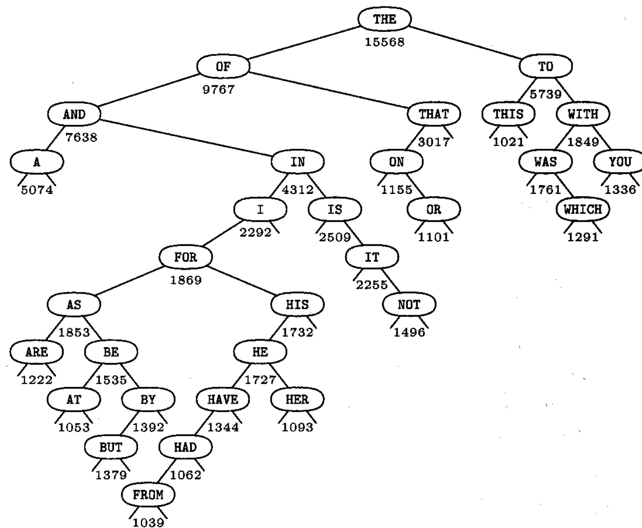
- Family tree.
- Parse tree.
- Unix file hierarchy.
- Phylogeny tree.
- GUI containment hierarchy.





Reference: http://java.sun.com/docs/books/tutorial/uiswing/overview/anatomy.html

Binary Search Tree



Other Types of Trees

Other types of trees.

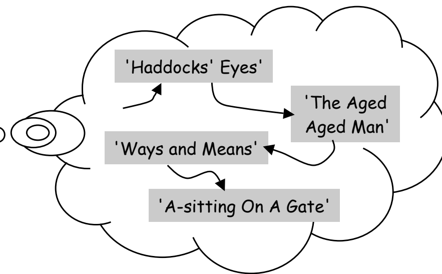
- Family tree.
- Parse tree.
- Unix file hierarchy.
- Phylogeny tree.
- GUI containment hierarchy.
- Binary search trees.
- NCAA basketball tournament.
- Barnes-Hut tree for fast N-body simulation.
- . . .

Conclusions

Sequential allocation: supports indexing, fixed size.
Linked allocation: variable size, supports sequential access.

Linked structures are a central programming abstraction.

- Linked lists.
- Binary trees.
- Graphs.
- Sparse matrices.



Alice should have done this!

Announcements

Thinking about majoring in Computer Science?

Or doing the Certificate in Applications of Computing?

Then: visit the all-new "Life in the Computer Science Department: A Guide for the Humble Undergraduate":

- http://www.cs.princeton.edu/academics/ugradpgm/life.html
- a handy FAQ that answers many many questions

And/Or: Come talk to me

AND CERTAINLY attend at least one of:

- C.S. open house for BSE freshmen Tuesday March 29, Friend Convocation Room, 5:45 (PM!): tours, demos, pizza (AB's welcome)
- C.S. open house for AB sophomores Tuesday April 5, C.S. Tea Room, 4 PM (but no pizza, and maybe fewer demos) (BSE's welcome)