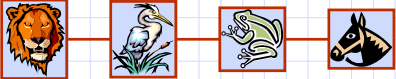


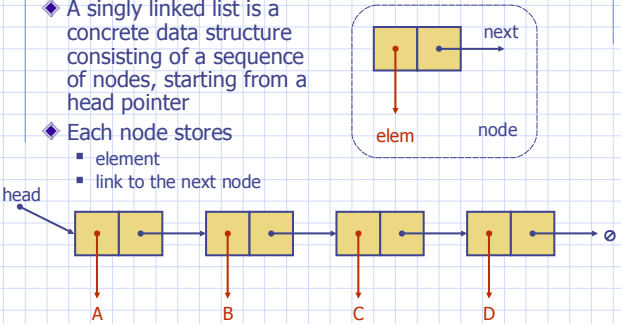
Linked Lists



© 2013 Goodrich, Tamassia, Goldwasser Linked Lists 1

Singly Linked List

- ◆ A singly linked list is a concrete data structure consisting of a sequence of nodes, starting from a head pointer
- ◆ Each node stores
 - element
 - link to the next node



© 2013 Goodrich, Tamassia, Goldwasser Linked Lists 2

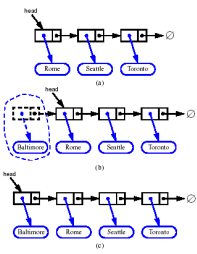
The Node Class for List Nodes

```
public class Node {
    // Instance variables:
    private Object element;
    private Node next;
    /** Creates a node with null references to its element and next node. */
    public Node() {
        this(null, null);
    }
    /** Creates a node with the given element and next node. */
    public Node(Object e, Node n) {
        element = e;
        next = n;
    }
    // Accessor methods:
    public Object getElement() {
        return element;
    }
    public Node getNext() {
        return next;
    }
    // Modifier methods:
    public void setElement(Object newElem) {
        element = newElem;
    }
    public void setNext(Node newNext) {
        next = newNext;
    }
}
```

© 2013 Goodrich, Tamassia, Goldwasser Linked Lists 3

Inserting at the Head

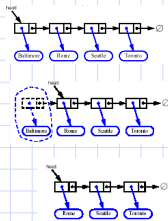
1. Allocate a new node
2. Insert new element
3. Have new node point to old head
4. Update head to point to new node



© 2013 Goodrich, Tamassia, Goldwasser Linked Lists 4

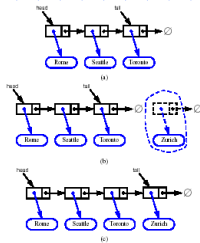
Removing at the Head

- 1. Update head to point to next node in the list
- 2. Allow garbage collector to reclaim the former first node



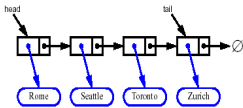
Inserting at the Tail

- 1. Allocate a new node
- 2. Insert new element
- 3. Have new node point to null
- 4. Have old last node point to new node
- 5. Update tail to point to new node



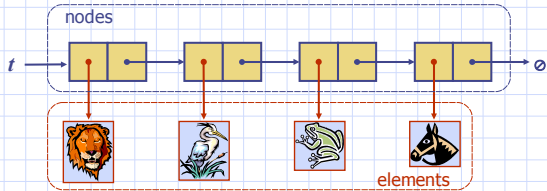
Removing at the Tail

- ◆ Removing at the tail of a singly linked list is not efficient!
- ◆ There is no constant-time way to update the tail to point to the previous node



Stack as a Linked List

- ◆ We can implement a stack with a singly linked list
- ◆ The top element is stored at the first node of the list
- ◆ The space used is $O(n)$ and each operation of the Stack ADT takes $O(1)$ time

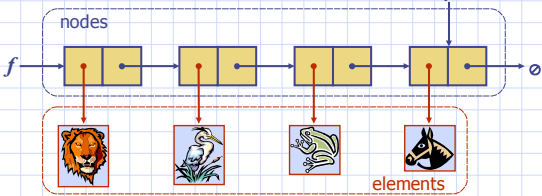


Linked-List Stack in Python

```
1 class LinkedStack:
2     """LIFO Stack implementation using a singly linked list for storage."""
3
4     # ----- nested _Node class -----
5     class _Node:
6         """Lightweight, nonpublic class for storing a singly linked node."""
7         __slots__ = '_element', '_next' # streamline memory usage
8
9     def __init__(self, element, next):
10         """Initialize node's fields:
11         self._element = element # reference to user's element
12         self._next = next # reference to next node
13
14     # ----- stack methods -----
15     def __init__(self):
16         """Creates an empty stack"""
17         self._head = None # reference to the head node
18         self._size = 0 # number of stack elements
19
20     def __len__(self):
21         """Return the number of elements in the stack"""
22         return self._size
23
24     def is_empty(self):
25         """Return True if the stack is empty"""
26         return self._size == 0
27
28     def push(self, e):
29         """Add element e to the top of the stack"""
30         self._head = self._Node(e, self._head) # create and link a new node
31         self._size += 1
32
33     def top(self):
34         """Return (but do not remove) the element at the top of the stack.
35         Raise Empty exception if the stack is empty.
36         """
37         if self.is_empty():
38             raise Empty('Stack is empty')
39         return self._head._element # top of stack is at head of list
40
41     def pop(self):
42         """Remove and return the element from the top of the stack (i.e., LIFO).
43         Raise Empty exception if the stack is empty.
44         """
45         if self.is_empty():
46             raise Empty('Stack is empty')
47         answer = self._head._element
48         self._head = self._head._next # bypass the former top node
49         self._size -= 1
50         return answer
```

Queue as a Linked List

- ◆ We can implement a queue with a singly linked list
 - The front element is stored at the first node
 - The rear element is stored at the last node
- ◆ The space used is $O(n)$ and each operation of the Queue ADT takes $O(1)$ time



Linked-List Queue in Python

```
1 class LinkedQueue:
2     """FIFO queue implementation using a singly linked list for storage."""
3
4     # ----- nested _Node class -----
5     class _Node:
6         """Lightweight, nonpublic class for storing a singly linked node"""
7         (omitted here; identical to that of LinkedStack._Node)
8
9     def __init__(self):
10         """Creates an empty queue"""
11         self._head = None
12         self._tail = None # number of queue elements
13         self._size = 0
14
15     def __len__(self):
16         """Return the number of elements in the queue"""
17         return self._size
18
19     def is_empty(self):
20         """Return True if the queue is empty"""
21         return self._size == 0
22
23     def first(self):
24         """Return (but do not remove) the element at the front of the queue"""
25         if self.is_empty():
26             raise Empty('Queue is empty')
27         return self._head._element # front aligned with head of list
28
29     def dequeue(self):
30         """Remove and return the first element of the queue (i.e., FIFO).
31         Raise Empty exception if the queue is empty.
32         """
33         if self.is_empty():
34             raise Empty('Queue is empty')
35         answer = self._head._element
36         self._head = self._head._next
37         self._size -= 1
38         if self.is_empty():
39             self._tail = None # special case as queue is empty
40             # removed head had been the tail
41         return answer
42
43     def enqueue(self, e):
44         """Add an element to the back of queue"""
45         newest = self._Node(e, None) # node will be new tail node
46         if self.is_empty():
47             self._head = newest # special case: previously empty
48         else:
49             self._tail._next = newest
50             self._tail = newest # update reference to tail node
51             self._size += 1
```