Announcements

Problem Set #1
- Policy for regrade requests

Problem Set #2
- 100 ± 10 points
  - Call trace
  - C++ questions

Programming Assignment #1
- Until 5pm (no late submissions)
  - Anonymize your name on Autolab (scoreboard)

CSC 212
Data Structures and Abstractions
Spring 2016

Previously ...

Pointers and Arrays
Example with Dynamic Arrays

Today ...

More on Pointers
Linked Lists

Pointers in C++

- Declaration, dereferencing
- Address of, reference (save for later)

Linked Lists

Arrays and Linear Sequences

- Can’t make insertions efficiently at both ends
- Can’t make deletions efficiently

Linked Lists

- Collection of sequential data stored at non-contiguous nodes
- Nodes are connected by links
  - Every node has a pointer to the next node
- Can grow and shrink dynamically

Singly Linked List

- Head
- Tail
- NULL pointer
```cpp
class List {
    private:
        Node *head;
        Node *tail;
    public:
        List();
        ~List();
        void insert_end(int);
};

class Node {
    private:
        int data;
        Node *nxt;
    public:
        Node(int d);
        ~Node();
        friend class List;
};
```

### Basic Methods

- **Insert at End**
  ```cpp
  p = create_node(key)
  if (head == NULL)
      head = tail = p
  else
      tail->nxt = p
      tail = p
  O(1)
  ```

- **Insert at Front**
  ```cpp
  p = create_node(key)
  if (head == NULL)
      head = tail = p
  else
      p->nxt = head
      head = p
  O(1)
  ```

- **Insert by Index**
  ```cpp
  O(n)
  // inserting key at position k
  // starting at 0
  if size < k or k < 0
      throw exception
  else if k == 0
      call insert at front
  else if k == size
      call insert at end
  else
      q = head
      for i=0, i<(k-1), i++
          q = q->nxt
      p = create_node(key)
      p->nxt = q->nxt
      q->nxt = p
  ```

- **Insert by Value**
  ```cpp
  O(n)
  Useful when making ordered insertions
  try it yourself
  ```

- **Delete at Front**
  ```cpp
  O(1)
  if head == NULL
      throw exception
  else if size == 1
      delete head
      head = tail = NULL
  else
      p = head
      head = p->nxt
      delete p
  ```

- **Delete at End**
  ```cpp
  O(n)
  if head == NULL
      throw exception
  else if size == 1
      delete head
      head = tail = NULL
  else
      p = head
      while p->nxt->nxt
          p = p->nxt
      tail = p
      delete tail->nxt
      tail->nxt = NULL
  ```

When implementing methods for linked lists, always draw the state of the linked list.
Delete by Value

O(n)

p = head
while p && p->data != key
    q = p
    p = p->nxt
if not p
    throw exception
else if head == p
    call delete at front
else if tail == p
    call delete at end
else
    q->nxt = p->nxt
    delete p

Delete by Index

O(n)

Try it yourself …

similar to delete by Value

Get

O(n)

Try it yourself …

return value at ‘Index’

Search

O(n)

Try it yourself …

find a key in the list

Destroy

O(n)

p = head
while p
    q = p
    p = p->nxt
    delete q
head = tail = NULL

Circular Lists

Can also have a circular doubly linked list