Announcements

Problem Set #1

policy for regrade requests
Announcements

Problem Set #1
   policy for **regrade requests**

Problem Set #2
   100 + 10 points
   call trace
   c++ questions
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

Lecture 07: Linked Lists
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
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
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

5 — 7 — 2 — 6

Tail

NULL pointer
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

**InsertKey**
front, end, by index, by value

**DeleteKey**
front, end, by index, by value

**GetKey/Search**
by value, by index

**Traverse**
visit all nodes (e.g. print)

**Destroy**

When implementing methods for linked lists, always draw the state of the linked list.
p = create_node(key)

if (head == NULL)
    head = tail = p
else
    tail->nxt = p
    tail = p
p = create_node(key)

if (head == NULL)
    head = tail = p
else
    tail->nxt = p
    tail = p

O(1)
Insert at Front

\[
p = \text{create\_node(key)}
\]

\[
\text{if (head == NULL)}
\quad \text{head = tail = p}
\]
\[
\text{else}
\quad p->nxt = head
\quad \text{head = p}
\]
Insert at Front

\[ p = \text{create\_node} (\text{key}) \]

\[
\text{if } (\text{head} == \text{NULL}) \\
\quad \text{head} = \text{tail} = p
\]

\[
\text{else} \\
\quad p->\text{nxt} = \text{head} \\
\quad \text{head} = p
\]

\( O(1) \)
// 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
// 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

Useful when making ordered insertions

try it yourself
Insert by Value

Useful when making ordered insertions

try it yourself

$O(n)$
Delete at Front

```c
if head == NULL
    throw exception
else if size == 1
    delete head
    head = tail = NULL
else
    p = head
    head = p->nxt
    delete p
```
Delete at Front

```c
if head == NULL
    throw exception
else if size == 1
    delete head
    head = tail = NULL
else
    p = head
    head = p->nxt
    delete p
```
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
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`
Delete by Value

```c
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 Value

\[ O(n) \]

\[
p = \text{head}
\]
\[
\text{while } p \text{ && } p->\text{data} != \text{key}
\]
\[
\quad q = p
\]
\[
\quad p = p->\text{nxt}
\]
\[
\text{if not } p
\]
\[
\quad \text{throw exception}
\]
\[
\text{else if } \text{head} == p
\]
\[
\quad \text{call delete at front}
\]
\[
\text{else if } \text{tail} == p
\]
\[
\quad \text{call delete at end}
\]
\[
\text{else}
\]
\[
\quad q->\text{nxt} = p->\text{nxt}
\]
\[
\text{delete } p
\]
Delete by Index

Try it yourself …

similar to delete by Value
Delete by Index

Try it yourself …

similar to delete by Value

$O(n)$
Try it yourself ...

return value at ‘Index’
Try it yourself …

return value at ‘Index’
Try it yourself …

find a key in the list
Try it yourself …

find a key in the list

$O(n)$
Destroy

```c
p = head
while p
    q = p
    p = p->nxt
    delete q
head = tail = NULL
```
```
p = head
while p
    q = p
    p = p->nxt
    delete q
head = tail = NULL
```
Circular Lists

Can also have a circular doubly linked list