CS 106X
Lecture 16: Linked Lists
Wednesday, February 15, 2017

Programming Abstractions (accelerated)
Fall 2016
Stanford University
Computer Science Department

Lecturer: Chris Gregg

reading:
Programming Abstractions in C++, Chapter 11
Today's Topics

• Logistics
  • Midterm information:
    • Thursday February 23rd
    • Midterm Review Session: Monday, 7:30-8:30pm, Location TBA
    • There will be practice midterms on the website Friday.

• Linked Lists
  • Could you architect a Queue?
  • Nodes
  • Linked Lists
  • The Towers of Gondor
  • Do nodes have names?
  • Big O?
  • Stack made from a Linked List
  • Queue made with a Linked List
Your job: Architect a Queue
class QueueInt { // in QueueInt.h  
public:  
QueueInt (); // constructor  

void enqueue(int value); // append a value  
int dequeue(); // return the first-in value  

private:  
Vector<int> data; // member variables  
};
You're next!

back

1 2 3 4 5 6 7 8

front

dequeue()
You're next!

dequeue()
Excuse Me, Coming Through

back

front

1 2 3 4 5 6 7
Excuse Me, Coming Through

enqueue(42)
enqueue(42)
Excuse Me, Coming Through

enqueue(42)
enqueue(42)
Excuse Me, Coming Through

enqueue(42)
Excuse Me, Coming Through

enqueue(42)
Excuse Me, Coming Through

enqueue(42)
enqueue(42)
enqueue(42)
Queue as Vector: Big O

Enqueue: O(n)

Dequeue: O(1)
And Now for Something Completely Different

int * data

enqueue(7)
And Now for Something Completely Different

int * data

enqueue(7)

7

8 9
And Now for Something Completely Different

int * data

enqueue(6)
And Now for Something Completely Different

int * data

enqueue(6)
And Now for Something Completely Different

**Diagram:**
- `int * data`
- `enqueue(6)`

**Nodes:**
- Node 6
- Node 7
- Node 8
- Node 9

**Connections:**
- Node 6 to Node 7
- Node 7 to Node 8
- Node 8 to Node 9
And Now for Something Completely Different

Now we have a way to add to the front in $O(1)$ time!
A linked list is a chain of nodes.
Each node contains two pieces of information:
- Some piece of data that is stored in the sequence
- A link to the next node in the list.
We can traverse the list by starting at the first cell and repeatedly following its link.
Linked Lists

- A **linked list** is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.
A linked list is a data structure for storing a sequence of elements.

Each element is stored separately from the rest.

The elements are then chained together into a sequence.
Linked Lists

- A **linked list** is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.
• A **linked list** is a data structure for storing a sequence of elements.

• Each element is stored separately from the rest.

• The elements are then chained together into a sequence.
A **linked list** is a data structure for storing a sequence of elements.

- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.
A linked list is a data structure for storing a sequence of elements. Each element is stored separately from the rest. The elements are then chained together into a sequence.
A **linked list** is a data structure for storing a sequence of elements.

Each element is stored separately from the rest.

The elements are then chained together into a sequence.
Why Linked Lists?

- Can efficiently splice new elements into the list or remove existing elements anywhere in the list.
- Never have to do a massive copy step;
- Has some tradeoffs; we'll see this later.
For simplicity, let's assume we're building a linked list of strings.

We can represent a node in the linked list as a structure:

```c
struct Node {
    string value;
    /* ? */ next;
};
```
For simplicity, let's assume we're building a linked list of strings.

We can represent a node in the linked list as a structure:

```c
struct Node {
    string value;
    Node* next;
};
```
For simplicity, let's assume we're building a linked list of strings.

We can represent a node in the linked list as a structure:

```c
struct Node {
    string value;
    Node* next;
};
```

The structure is defined recursively!
Draw a picture
In a scene that was brilliantly captured in Peter Jackson’s film adaptation of *The Return of the King*, Rohan is alerted to the danger to Gondor by a succession of signal fires moving from mountain top to mountain top. This scene is a perfect illustration of the idea of message passing in a linked list.
THE LORD OF THE RINGS
THE RETURN OF THE KING
Step 1: Make this linked list

Step 2: Light the fires....

Lighting the fire of San Francisco!
struct Tower {
    string name;  /* The name of this tower */
    Tower *link;  /* Pointer to the next tower */
};
// add the first tower
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;
// add the first tower
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;
// add the first tower
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;
// add the first tower
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;
Lord of the Linked Lists
```c
// main
Tower * head = new Tower;
hdr->name = "San Jose";
hdr->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```c
struct Tower{
    string name;
    Tower * link;
};
```

```c
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower * createTower(string name, Tower * link) {
    Tower * tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```c
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower * createTower(string name, Tower * link) {
    Tower * tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

class Tower
{struct
   string name;
   Tower * link;
};

Tower *createTower(string name, Tower *link) {
   Tower *tp = new Tower;
   tp->name = name;
   tp->link = link;
   return tp;
}
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```cpp
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

Linked List Trace
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
// main
Tower * head = new Tower;
head->name = “San Jose”;
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower * createTower(string name, Tower * link) {
    Tower * tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

Linked List Trace
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
// main
Tower * head = new Tower;
head->name = “San Jose”;
head->link = NULL;

head = createTower(“Santa Clara”, head);
head = createTower(“Mountain View”, head);
head = createTower(“Palo Alto”, head);
head = createTower(“Menlo Park”, head);
head = createTower(“Redwood City”, head);
head = createTower(“Millbrae”, head);
head = createTower(“Bayshore”, head);
head = createTower(“San Francisco”, head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```
// main
Tower * head = new Tower;
head->name = “San Jose”;
head->link = NULL;

head = createTower(“Santa Clara”, head);
head = createTower(“Mountain View”, head);
head = createTower(“Palo Alto”, head);
head = createTower(“Menlo Park”, head);
head = createTower(“Redwood City”, head);
head = createTower(“Millbrae”, head);
head = createTower(“Bayshore”, head);
head = createTower(“San Francisco”, head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;
head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);

struct Tower{
    string name;
    Tower * link;
};

Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
void signal(Tower *start) {
    if (start != NULL) {
        cout << "Lighting " << start->name << endl;
        signal(start->link);
    }
}

signal(head);
How is the Stack Implemented?
struct Node{
    int value;    /* The value of this element */
    Node *link;   /* Pointer to the next node */
};
Node *
data

8

9
class StackInt { // in StackInt.h
public:
    StackInt (); // constructor

    void push(int value); // append a value
    int pop(); // return the first-in value

private:
    struct Node {
        int value;
        Node * link;
    };
    Node * data; // member variables
};
void StackInt::push(int v) {
    Node * temp = new Node;
    temp->value = v;
    temp->link = data;
    data = temp;
}

int StackInt::pop() {
    int toReturn = data->value;
    Node * temp = data;
    data = temp->link;
    delete data;
    return toReturn;
}
Node *
  data

8

9
Node *
data

push(7);
Goal of Push
Node *
data

push(7);
Stack is a Linked List

```
push(7);
```

```
Node * temp = new Node;
temp -> value = 7;
```
Stack is a Linked List

```
push(7);
```

```c
Node *data = temp;
```
Node *

data

push(7);
Node * temp = new Node;
temp -> value = 7;

push(7);
Node *
data

Node*
temp

push(7);

temp -> link = data;
Node *
data

Node *
temp

data = temp;

push(7);
Node *
data

7

8

9

exit function

push(7);
Stack is a Linked List

Node *

data

7

8

9

pop();
Stack is a Linked List

Node *
data

int toReturn

pop();

int toReturn = data->value;
Stack is a Linked List

Node *

data

8

9

data = data->link;

pop();

int toReturn

7
That didn’t work. Let’s try again...
Stack is a Linked List

Node *
  data

Node
  7

Node
  8

Node
  9

pop();
Stack is a Linked List

```plaintext
Node *
data

8
9
7

int toReturn = data->value;

pop();

int toReturn
7
```
Stack is a Linked List

```c
Node * temp = data;
```

```
Node* temp = data;
```

```
int toReturn
```

```
pop();
```
Stack is a Linked List

Node *

data

Node *
temp

8

9

7

int toReturn

7

data = temp->link;

pop();
Stack is a Linked List

Node *
data

8

9

Node *
temp

delete temp;

pop();

int toReturn

7
Stack is a Linked List

Node *
data

8
9

pop();

int toReturn

return toReturn;
class StackInt { // in StackInt.h
public:
    StackInt (); // constructor
    void push(value); // append a value
    int pop(); // return the first-in value

private:
    struct Node {
        int value;
        Node * link;
    };
    Node * data; // member variables
};
void StackInt::push(int v) {
    Node * temp = new Node;
    temp->value = v;
    temp->link = data;
    data = temp;
}

int StackInt::pop() {
    int toReturn = data->value;
    Node * temp = data;
    data = temp->link;
    delete temp;
    return toReturn;
}
Stack Implementation: Big O?

Big O of `push()`? O(1)
Big O of `pop()`? O(1)

Yay!
Queue?

Node *
  data

8

9
Queue Enqueue?
Queue Enqueue?

Node *
  data

8

9

7

O(1)
Queue Dequeue?
Queue Dequeue?
Queue Dequeue?

Node *
  data

Node: 7

8 → 9

Diagram of a queue with nodes 7, 8, and 9.
Queue Dequeue?
Queue Dequeue?
Queue Dequeue?

\[ \Theta(n) \]
Always a Better Way
Actual Queue: Enqueue

Node * head

1

Node * tail

2

3

Actual	Queue:	Enqueue
Actual Queue: Enqueue

Node * cp = new Node;
cp->value = 4;
Actual Queue: Enqueue

```c
Node * head

Node * tail

tail->link = cp;

Node * head

Node * cp

Node * tail

1 -> 2 -> 3 -> 4

Node * cp
```
Actual Queue: Enqueue

Node * head

1 → 2 → 3 → tail = cp;

Node * tail

4

Node * cp
Actual Queue: Enqueue

Node * head

return;

Node * tail

1 2 3 4
Actual Queue: Enqueue

Node * head

Node * tail

O(1)
Deque
Actual Queue: Dequeue

Node *
head

Node *
tail

1 → 2 → 3 → 4
Actual Queue: Dequeue

Node *
head

Node *
cp

Node *
tail

Node *
cp = head;
Actual Queue: Dequeue

head = cp->link;
Actual Queue: Dequeue

Node * head

Node * cp

Node * tail

int toReturn

int toReturn = cp->value;
Actual Queue: Dequeue

Node * head

Node * cp

Node * tail

int toReturn

2

3

4

delete cp;
Actual Queue: Dequeue

Node * head

Node * tail

int toReturn

return toReturn;
Actual Queue: Dequeue

Node * head

Node * tail

O(1)
class QueueInt { // in QueueInt.h
public:
    QueueInt (); // constructor
    void enqueue(int value); // append a value
    int dequeue(); // return the first-in value

private:
    struct Node {
        int value;
        Node * link;
    };
    Node * head; // has a pointer to the first node
    Node * tail; // and a pointer to the last node
};
void QueueInt::enqueue(int v) {
    Node * temp = new Node;
    temp->value = v;
    tail->link = temp;
    tail = temp;
}

int QueueInt::dequeue() {
    int toReturn = head->value;
    Node * temp = head;
    head = temp->link;
    delete temp;
    return toReturn;
}
Stack Push $O(1)$
Stack Pop $O(1)$
Queue Enqueue $O(1)$
Queue Dequeue $O(1)$