CS 106B
Lecture 16: Linked Lists

Monday, May 7, 2018

programming Abstractions
Spring 2018
Stanford University
Computer Science Department

Lecturer: Chris Gregg

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

• Logistics
  • Midterm graded — regrade requests due by next Monday
  • You should have received an email to check Gradescope for your grade.

• Linked Lists
  • Could you architect a Queue?
  • Nodes
  • Linked Lists
  • The Towers of Gondor
  • Do nodes have names?
  • Big O?
  • Stack and Queue made from 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!

1 2 3 4 5 6 7 8

back   front

depqueue()
You're next!

dequeue()
Excuse Me, Coming Through

back
1 2 3 4 5 6 7
front
enqueue(42)
enqueue(42)
enqueue(42)
Excuse Me, Coming Through

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

enqueue(42)
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

8 9

enqueue(7)
And Now for Something Completely Different

int * data

enqueue(7)
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

int * data

enqueue(6)
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.

```
1 ← 2 ← 3
```
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.
- To add a node at the end, we chain the last to the end.
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.
- To add a node at the end, we chain the last to the end.
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.
- To add a node in the middle, we simply add one to the middle (we have to find that location, though!)
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.
- To add a node in the middle, we simply add one to the middle (we have to find that location, though!)
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.
- To remove a node, we connect the previous node to the next node.
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:

```cpp
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:

```cpp
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:

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

• The structure is defined recursively!
Always!

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.
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;
// 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;
}
```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);
```

```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;
}
```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);
```

```c
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;
}
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);
```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);
```

```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("Palo Alto", 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;
}
```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);
```

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

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

  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);

  // Linked List Trace
  head = createTower("San Francisco", head);
}
// 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);
Lord of the Linked Lists
How is the Stack Implemented?
struct Node{
    int value;       /* The value of this elem */
    Node *next;      /* Pointer to the next node */
};
Stack

Node
*head

8

9
Stack

Node
*head

push(7);
Goal of Push

Node
*head

Node
7

Node
8

Node
9
Push Attempt

push(7);
push(7);

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

push(7);

Node *head

Node *temp

head = temp;
BRAAWWRRRR!
push(7);
push(7);

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

temp -> next = head;
push(7);

Node *head

Node *temp

head = temp;
push(7);

Node *head

8

9

7

exit function
Stack is a Linked List

Node
*head

8
9

7

pop();
Stack is a Linked List

Node *head

int toReturn = head->value;

pop();

int toReturn

7
Stack is a Linked List

Node *head

head = head->next;

pop();

int toReturn

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

Node
*head

pop();
Stack is a Linked List

Node *head

int toReturn = head->value;

pop();
Node * head

Node * temp

Stack is a Linked List

Node * temp = head;

int toReturn

8

9

pop();

7
Stack is a Linked List

head = temp->link;

toReturn = 7;

pop();
Stack is a Linked List

Node *head

Node *temp

pop();

int toReturn

7

delete temp;
Stack is a Linked List

Node *head

8
9

pop();

int toReturn

7

return toReturn;
#pragma once

class IntStack {
public:
    IntStack(); // constructor
    ~IntStack();

    bool isEmpty();
    void push(int value);
    int peek();
    int pop();

private:
    struct Node {
        int value;
        Node* next;
    };

    Node* head;
};
void IntStack::push(int value) {
    Node* node = new Node;
    // pushing "value"
    node->value = value;

    node->next = head;
    head = node;
}

int IntStack::pop() {
    if (isEmpty()) {
        throw "Error! Trying to pop from empty stack!";
    }

    Node* result = head;
    head = head->next;

    int value = result->value;
    delete result;
    return value;
}
Stack Implementation: Big O?

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

Yay!
Queue?

Node
*back

Node
8

Node
9
Queue Enqueue?

Node
*back

8
7
9
Queue Enqueue?

Node
*back

8
9

7

O(1)
Queue Dequeue? 

Node *back 

[Diagram showing queue nodes with numbers 7, 8, and 9, with arrows indicating the dequeue process]
Queue Dequeue? 

Node *back

8

7

9
Queue Dequeue?

Node
*back

8
9
7
Queue Dequeue?

Node
*back

8

9

7
Queue Dequeue?

Node
*back

8
7
Queue Dequeue?

Node
*back

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

Node *front

Node *back

1 2 3
Actual Queue: Enqueue

Node * back

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

Node *front

Node *back

Node cp

front->next = cp;
Actual Queue: Enqueue

Node *front

Node *back

Node *cp

front = cp;
Actual Queue: Enqueue

Node *front

Node *back

Actual Queue: Enqueue

return;
Actual Queue: Enqueue

Node *front

Node *back

1 ➔ 2 ➔ 3 ➔ 4

O(1)
Deque
Actual Queue: Dequeue

Node *front

Node *back

1 → 2 → 3 → 4
Actual Queue: Dequeue

Node *front

Node *cp

Node * cp = front;

Node * back
Actual Queue: Dequeue

Node *front

Node *back

Node *cp

front = cp->next;
Actual Queue: Dequeue

Node *front

Node *back

Node *cp

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

Node *front

Node *back

int toReturn

Node * cp

Node * cp

2

3

4

delete cp;
Actual Queue: Dequeue

return toReturn;
Actual Queue: Dequeue

Node *front

Node *back

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 *next;
    };
    Node * back; // has a pointer to the first node
    Node * front; // and a pointer to the last node
};
void QueueInt::enqueue(int v) {
    Node *temp = new Node;
    temp->value = v;
    back->next = temp;
    back = temp;
}

int QueueInt::dequeue() {
    int toReturn = front->value;
    Node * temp = front;
    front = temp->next;
    if (front == nullptr) {
        back = nullptr;
    }
    delete temp;
    return toReturn;
}
Linked Lists are Excellent

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst Case Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack Push</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Stack Pop</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Queue Enqueue</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Queue Dequeue</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>