Linked Lists

Singly Linked List

Circular Linked List

(no null pointer)

Doubly Linked Circular List
Outline

- Sequential vs. Linked
- Linked List
- Building a Linked List
- Traversing a Linked List
- Implementation (Circular)
Sequential vs. Linked

- **Sequential data structures**
  - Put one object next to another
  - A block of consecutive memory in the computer
  - Python: list of objects
  - Arbitrary access, "get me the $i^{th}$ object"
  - Fixed size, or dynamic but less efficient

- **Linked data structures**
  - Each object has link to another (or perhaps several)
  - Python: link is a reference to another object
  - Dynamic size
  - Flexible and widely-used way of organizing data
  - More challenging to code and debug
# Sequential vs. Linked

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>&quot;The&quot;</td>
</tr>
<tr>
<td>C1</td>
<td>&quot;cat&quot;</td>
</tr>
<tr>
<td>C2</td>
<td>&quot;sat&quot;</td>
</tr>
<tr>
<td>C3</td>
<td>-</td>
</tr>
<tr>
<td>C4</td>
<td>-</td>
</tr>
<tr>
<td>C5</td>
<td>-</td>
</tr>
<tr>
<td>C6</td>
<td>-</td>
</tr>
<tr>
<td>C7</td>
<td>-</td>
</tr>
<tr>
<td>C8</td>
<td>-</td>
</tr>
<tr>
<td>C9</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>&quot;cat&quot;</td>
</tr>
<tr>
<td>C1</td>
<td>C8</td>
</tr>
<tr>
<td>C2</td>
<td>-</td>
</tr>
<tr>
<td>C3</td>
<td>-</td>
</tr>
<tr>
<td>C4</td>
<td>&quot;The&quot;</td>
</tr>
<tr>
<td>C5</td>
<td>Co</td>
</tr>
<tr>
<td>C6</td>
<td>-</td>
</tr>
<tr>
<td>C7</td>
<td>-</td>
</tr>
<tr>
<td>C8</td>
<td>&quot;sat&quot;</td>
</tr>
<tr>
<td>C9</td>
<td>null</td>
</tr>
</tbody>
</table>

Python list vs. linked list
Linked List

- **Linked list**
  - Simplest linked data structure
  - Node is a recursive data structure
  - Each node contains:
    - An item (some data)
    - A pointer to next node in the list

```python
class Node:
    def __init__(self, s):
        self.item = s
        self.next = None
```

Three Node objects hooked together to form a linked list:

- "The" → "cat" → "sat" → null

Special pointer value null (None in Python) terminates the list. We denote with a dot.
Building a linked list

```python
first = Node()
first.item = "The"
```

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>-</td>
</tr>
<tr>
<td>C1</td>
<td>-</td>
</tr>
<tr>
<td>C2</td>
<td>-</td>
</tr>
<tr>
<td>C3</td>
<td>-</td>
</tr>
<tr>
<td>C4</td>
<td>&quot;The&quot;</td>
</tr>
<tr>
<td>C5</td>
<td>None/null</td>
</tr>
<tr>
<td>C6</td>
<td>-</td>
</tr>
<tr>
<td>C7</td>
<td>-</td>
</tr>
<tr>
<td>C8</td>
<td>-</td>
</tr>
<tr>
<td>C9</td>
<td>-</td>
</tr>
</tbody>
</table>
first = Node()
first.item = "The"

second = Node()
second.item = "cat"
Building a linked list

```python
first = Node()
first.item = "The"

second = Node()
second.item = "cat"

third = Node()
third.item = "sat"
```

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>&quot;cat&quot;</td>
</tr>
<tr>
<td>C1</td>
<td>null</td>
</tr>
<tr>
<td>C2</td>
<td>-</td>
</tr>
<tr>
<td>C3</td>
<td>-</td>
</tr>
<tr>
<td>C4</td>
<td>&quot;The&quot;</td>
</tr>
<tr>
<td>C5</td>
<td>null</td>
</tr>
<tr>
<td>C6</td>
<td>-</td>
</tr>
<tr>
<td>C7</td>
<td>-</td>
</tr>
<tr>
<td>C8</td>
<td>&quot;sat&quot;</td>
</tr>
<tr>
<td>C9</td>
<td>null</td>
</tr>
</tbody>
</table>
Building a linked list

```
first = Node()
first.item = "The"

second = Node()
second.item = "cat"

third = Node()
third.item = "sat"

first.next = second
```

```
<table>
<thead>
<tr>
<th>Memory address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Co</td>
<td>&quot;cat&quot;</td>
</tr>
<tr>
<td>C1</td>
<td>null</td>
</tr>
<tr>
<td>C2</td>
<td>-</td>
</tr>
<tr>
<td>C3</td>
<td>-</td>
</tr>
<tr>
<td>C4</td>
<td>&quot;The&quot;</td>
</tr>
<tr>
<td>C5</td>
<td>Co</td>
</tr>
<tr>
<td>C6</td>
<td>-</td>
</tr>
<tr>
<td>C7</td>
<td>-</td>
</tr>
<tr>
<td>C8</td>
<td>&quot;sat&quot;</td>
</tr>
<tr>
<td>C9</td>
<td>null</td>
</tr>
</tbody>
</table>
```
Building a linked list

```python
first = Node()
first.item = "The"

second = Node()
second.item = "cat"

third = Node()
third.item = "sat"

first.next = second
second.next = third
```

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>&quot;cat&quot;</td>
</tr>
<tr>
<td>C1</td>
<td>C8</td>
</tr>
<tr>
<td>C2</td>
<td>-</td>
</tr>
<tr>
<td>C3</td>
<td>-</td>
</tr>
<tr>
<td>C4</td>
<td>&quot;The&quot;</td>
</tr>
<tr>
<td>C5</td>
<td>Co</td>
</tr>
<tr>
<td>C6</td>
<td>-</td>
</tr>
<tr>
<td>C7</td>
<td>-</td>
</tr>
<tr>
<td>C8</td>
<td>&quot;sat&quot;</td>
</tr>
<tr>
<td>C9</td>
<td>null</td>
</tr>
</tbody>
</table>
Traversing a List

- **Iterate over all elements in a linked list**
  - Assume list is null terminated
  - Assume `first` instance variable points to start of list
  - Print all the strings in the list

```python
current = first
while current != None:
    print(current.item)
    current = current.next
```
Traversing a list

```python
current = first
while current != None:
    print(current.item)
    current = current.next
```

```
current

"The"

"cat"

"sat"
```
current = first
while current != None:
    print(current.item)
    current = current.next

"The" next "cat" next "sat"
Traversing a list

```python
current = first
while current != None:
    print(current.item)
    current = current.next
```

The cat sat

```
"The"  "cat"  "sat"
```

first
Traversing a list

```python
current = first
while current != None:
    print(current.item)
    current = current.next
```

```
The
"cat"
"sat"
```
Traversing a list

current = first
while current != None:
    print(current.item)
    current = current.next
current = first
while current != None:
    print(current.item)
    current = current.next
Traversing a list

```python
current = first
while current != None:
    print(current.item)
    current = current.next
```

The cat sat

```
| "The"   | "cat"  | "sat" |
```

first
null
Playing with a Linked List

What things might we want to do with a list?
- Construct a node
- Add a node to the end
- Insert a node at a certain position
- Remove a node from a position
- Print out the list of nodes
Playing with a Linked List

- The definition of the class:

```python
class LinkedList:
    # Constructor for an empty linked list.
    def __init__(self):
        self.length = 0
        self.start = None
```
Playing with a Linked List

- What things might we want to do with a list?
  - Construct a linked list
What things might we want to do with a list?
  - Add a node to the end
What things might we want to do with a list?

- Insert a node at a certain position
Playing with a Linked List

- What things might we want to do with a list?
  - Remove a node from a position
Playing with a Linked List

- What things might we want to do with a list?
  - Print out the list of nodes
Summary

- Sequential vs. Linked
- Linked List
- Building a Linked List
- Traversing a Linked List
- Implementation (Circular)
Your Turn

- Open Moodle, go to CSCI 136, Section 01
- Open the dropbox for today: Activity 1
- Drag and drop your program file to the Moodle dropbox

You get: 1 point if you turn in something, 2 points if you turn in something that is correct.

- On the class website is a file, Quote.py. There are three blank lines in the method insertWord that you need to fill in with real code to make it work. If you have done it correctly, when you run it, the output should be:

A rose is a rose.
A rose is just a rose.