Linked Lists
Part Two
Recap from Last Time
Linked Lists at a Glance

- 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 at a Glance

- 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 at a Glance

- 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 at a Glance

• 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 at a Glance

- 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 List Cells

• A linked list is a chain of \textbf{cells}.
• Each cell contains two pieces of information:
  • Some piece of data that is stored in the sequence, and
  • A \textbf{link} to the next cell in the list.
• We can traverse the list by starting at the first cell and repeatedly following its link.
Representing a Cell

- For simplicity, let's assume we're building a linked list of strings.
- We can represent a cell in the linked list as a structure:

```c
struct Cell {
    Type value;
    Cell* next;
};
```
- The structure is defined recursively!
Building a Linked List
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == ") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getLine("Next entry? ");
    if (line == "") break;
    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;
    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}
return result;

result

---

dikdik!
Cell* result = NULL;

while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}

return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;
    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == ") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}

return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line =="") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}

return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;

result

quokka!
dikdik!
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;
    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}
return result;
Cell* result = NULL;
while (true) {
    string line = getLine("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;
    cell->next = result;
    result = cell;
}

return result;}
Cell* result = NULL;
while (true) {
    string line = getline("Next entry? ");
    if (line == "") break;

    Cell* cell = new Cell;
    cell->value = line;

    cell->next = result;
    result = cell;
}

return result;
Traversing a Linked List

- Once we have a linked list, we can traverse it by following the links one at a time.

```c
for (Cell* ptr = list; ptr != NULL; ptr = ptr->next) {
    /* ... use ptr ... */
}
```
Once we have a linked list, we can traverse it by following the links one at a time.

```c
for (Cell* ptr = list; ptr != NULL; ptr = ptr->next) {
    /* ... use ptr ... */
}
```
Traversing a Linked List

- Once we have a linked list, we can traverse it by following the links one at a time.

```c
for (Cell* ptr = list; ptr != NULL; ptr = ptr->next) {
    /* ... use ptr ... */
}
```

![Diagram of a linked list traversal](image)
Traversing a Linked List

- Once we have a linked list, we can traverse it by following the links one at a time.

```c
for (Cell* ptr = list; ptr != NULL; ptr = ptr->next) {
    /* ... use ptr ... */
}
```
Traversing a Linked List

- Once we have a linked list, we can traverse it by following the links one at a time.

```c
for (Cell* ptr = list; ptr != NULL; ptr = ptr->next) {
    /* … use ptr … */
}
```
Traversing a Linked List

- Once we have a linked list, we can traverse it by following the links one at a time.
  
  ```c
  for (Cell* ptr = list; ptr != NULL; ptr = ptr->next) {
    /* ... use ptr ... */
  }
  ```
Traversing a Linked List

- Once we have a linked list, we can traverse it by following the links one at a time.

```c
for (Cell* ptr = list; ptr != NULL; ptr = ptr->next) {
    /* … use ptr … */
}
```
Once More With Recursion

- Linked lists are defined recursively, and we can traverse them using recursion!

```c
void recursiveTraverse(Cell* list) {
    if (list == NULL) return;
    /* ... do something with list ... */
    recursiveTraverse(list->next);
}
```
Freeing a Linked List

• All good things must come to an end, and we eventually need to reclaim the memory for a linked list.

• Here's an **Extremely Bad Idea**:

```c
for (Cell* ptr = list; ptr != NULL; ptr = ptr->next) {
    delete ptr;
}
```
Freeing a Linked List

• All good things must come to an end, and we eventually need to reclaim the memory for a linked list.

• Here's an **Extremely Bad Idea**:  
  ```cpp
  for (Cell* ptr = list; ptr != NULL; ptr = ptr->next) {
    delete ptr;
  }
  ```
Freeing a Linked List

- All good things must come to an end, and we eventually need to reclaim the memory for a linked list.

- Here's an **Extremely Bad Idea**:

  ```
  for (Cell* ptr = list; ptr != NULL; ptr = ptr->next) {
    delete ptr;
  }
  ```
Freeing a Linked List

• All good things must come to an end, and we eventually need to reclaim the memory for a linked list.

• Here's an **Extremely Bad Idea**:  

```cpp
for (Cell* ptr = list; ptr != NULL; ptr = ptr->next) {
    delete ptr;
}
```

![Diagram showing the concept of freeing a linked list]

```
ptr

2

... 
```
Freeing a Linked List

- All good things must come to an end, and we eventually need to reclaim the memory for a linked list.

- Here's an *Extremely Bad Idea*:

```c
for (Cell* ptr = list; ptr != NULL; ptr = ptr->next) {
    delete ptr;
}
```

```plaintext
   ^         ^
   ptr      ???
```
Freeing a Linked List Properly

- To properly free a linked list, we have to be able to
  - Destroy a cell, and
  - Advance to the cell after it.
- How might we accomplish this?
Linked Lists: The Tricky Parts

• Suppose that we want to write a function that will add an element to the front of a linked list.

• What might this function look like?
What went wrong?
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

```c
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}
```
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

list
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

list
```cpp
int main() {
    Cell* list = NULL;
    ListInsert(list, 137);
    ListInsert(list, 42);
    ListInsert(list, 271);
}

void listInsert(Cell* list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```
```c
int main() {
    Cell* list = NULL;
    ListInsert(list, 137);
    ListInsert(list, 42);
    ListInsert(list, 271);
}

void listInsert(Cell* list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```
int main() {
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

void listInsert(Cell* list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```cpp
int main() {
    Cell* list = NULL;
    ListInsert(list, 137);
    ListInsert(list, 42);
    ListInsert(list, 271);
}
```
```c
int main() {
    Cell* list = NULL;
    ListInsert(list, 137);
    ListInsert(list, 42);
    ListInsert(list, 271);
}
```

```c
void listInsert(Cell* list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```
```c
int main() {
    void listInsert(Cell* list, int value) {
        Cell* newCell = new Cell;
        newCell->value = value;
        newCell->next = list;
        list = newCell;
    }
}
```
```c
void listInsert(Cell* list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```
```c
void listInsert(Cell* list, int value) {
Cell* newCell = new Cell;
newCell->value = value;
newCell->next = list;
list = newCell;
}
```
int main() {
    Cell* list = NULL;
    ListInsert(list, 137);
    ListInsert(list, 42);
    ListInsert(list, 271);
}

void listInsert(Cell* list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
int main() {
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

void listInsert(Cell* list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

list

137
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

Why does nobody love me?
In order to resolve this problem, we must pass the linked list pointer by reference.

Our new function:

```cpp
void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    cell->value = value;
    cell->next = list;
    list = cell;
}
```
Pointers by Reference

• In order to resolve this problem, we must pass the linked list pointer by reference.

• Our new function:

```cpp
void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    cell->value = value;
    cell->next = list;
    list = cell;
}
```
Pointers by Reference

• In order to resolve this problem, we must pass the linked list pointer by reference.

• Our new function:

```c
void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```

This is a reference to a pointer to a Cell. It’s often useful to read this from the right to the left.
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

```c
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}
```
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

list
```c
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}
```

```c
void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```
```c
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```
```c
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}

void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```c
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```c
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```
```c
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```c
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}
```

```c
void listInsert(Cell*& list, int value) {
    Cell* newCell = new Cell;
    newCell->value = value;
    newCell->next = list;
    list = newCell;
}
```
int main() {
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}
int main()
{
    Cell* list = NULL;
    listInsert(list, 137);
    listInsert(list, 42);
    listInsert(list, 271);
}

Yay! list loves me!
Pointers by Reference

- If you pass a pointer into a function by value, you can change the contents at the object you point at, but not which object you point at.

- If you pass a pointer into a function by reference, you can also change which object is pointed at.
A Quick Interlude for Announcements
Apply to Section Lead!

http://cs198.stanford.edu
Announcements

- WiCS (Women in Computer Science) is holding a dessert night tonight!
- **6:30PM - 7:30PM** at the Gates Patio.
- RSVP: https://docs.google.com/forms/d/12hcDPU9FGxSOMoTfehGSvT57yUw_4dtV32ezYPlfkk4/viewform
Announcements

- Interested in applying engineering and technical skills to create social good? Attend the Stanford4Good Conference!
- This Saturday, 11AM - 3PM at the d.school.
Back to our regularly scheduled programming...
Implementing Queue

- Earlier, we implemented a queue using two stacks.
- The implementation supported enqueue and dequeue in average-case O(1).
- We can also implement a queue using linked lists!
- Idea:
  - To **enqueue**, append a new cell to the end of the list.
  - To **dequeue**, remove the first cell from the list.
Implementing Queue

- Earlier, we implemented a queue using two stacks.
- The implementation supported enqueue and dequeue in average-case O(1).
- We can also implement a queue using linked lists!
- Idea:
  - To **enqueue**, append a new cell to the end of the list.
  - To **dequeue**, remove the first cell from the list.
Implementing Queue

- Earlier, we implemented a queue using two stacks.
- The implementation supported enqueue and dequeue in average-case O(1).
- We can also implement a queue using linked lists!
- Idea:
  - To **enqueue**, append a new cell to the end of the list.
  - To **dequeue**, remove the first cell from the list.
Implementing Queue

- Earlier, we implemented a queue using two stacks.
- The implementation supported enqueue and dequeue in average-case $O(1)$.
- We can also implement a queue using linked lists!
- Idea:
  - To **enqueue**, append a new cell to the end of the list.
  - To **dequeue**, remove the first cell from the list.
Implementing Queue

- Earlier, we implemented a queue using two stacks.
- The implementation supported enqueue and dequeue in average-case $O(1)$.
- We can also implement a queue using linked lists!

Idea:
- To **enqueue**, append a new cell to the end of the list.
- To **dequeue**, remove the first cell from the list.
Implementing Queue

- Earlier, we implemented a queue using two stacks.
- The implementation supported enqueue and dequeue in average-case $O(1)$.
- We can also implement a queue using linked lists!
- Idea:
  - To **enqueue**, append a new cell to the end of the list.
  - To **dequeue**, remove the first cell from the list.
Implementing **Queue**

- Earlier, we implemented a queue using two stacks.
- The implementation supported enqueue and dequeue in average-case $O(1)$.
- We can also implement a queue using linked lists!
- **Idea:**
  - To **enqueue**, append a new cell to the end of the list.
  - To **dequeue**, remove the first cell from the list.
Implementing Queue

- Earlier, we implemented a queue using two stacks.
- The implementation supported enqueue and dequeue in average-case $O(1)$.
- We can also implement a queue using linked lists!
- Idea:
  - To enqueue, append a new cell to the end of the list.
  - To dequeue, remove the first cell from the list.
Implementing Queue

• Earlier, we implemented a queue using two stacks.
• The implementation supported enqueue and dequeue in average-case O(1).
• We can also implement a queue using linked lists!
• Idea:
  • To **enqueue**, append a new cell to the end of the list.
  • To **dequeue**, remove the first cell from the list.
Implementing Queue

- Earlier, we implemented a queue using two stacks.
- The implementation supported enqueue and dequeue in average-case O(1).
- We can also implement a queue using linked lists!

Idea:
- To **enqueue**, append a new cell to the end of the list.
- To **dequeue**, remove the first cell from the list.
A Surprising Fact

It is possible to build a Map that supports lookups that run, on average, in $O(1)$ time.

How on earth is this possible?
Find out on Friday!