CS 261 Lab #5

Is it really midterm time already?
Mix of multiple choice, matching, short answer, true/false, and code

Code needn’t be *perfect*, but should clearly be C
(missing a semi-colon is fine; only writing pseudo-code is not)

Will cover weeks 1 – 4
(everything up to and including binary search)

Header code will be provided
Be able to determine **Big O runtime** by examining an algorithm (in pseudocode and C) or equation.

Know the Big O runtimes of **common algorithms** (e.g., *binary search*).
What’s the **Big O runtime** for:

A method that takes $3n^2 + 6n + 50$ steps?
What’s the **Big O runtime** for:

A method that takes \(3n^2+6n+50\) steps?

\(O(n^2)\) because the \(n^2\) term dominates
What’s the **Big O runtime** for:

A method that takes $3n^2+6n+50$ steps?

*O*(n²) because the n² term dominates

Binary search?
What’s the **Big O runtime** for:

A method that takes $3n^2+6n+50$ steps?

- **$O(n^2)$** because the $n^2$ term dominates

Binary search?

- **$O(\log n)$** because it halves the search space on each iteration
What’s the Big O runtime for:

A method that takes $3n^2 + 6n + 50$ steps?

$\mathcal{O}(n^2)$ because the $n^2$ term dominates

Binary search?

$\mathcal{O}(\log n)$ because it halves the search space on each iteration

```java
for (int i = n; i > 0; i = i / 2) {
    // constant-time operations
}
```
What’s the Big O runtime for:

A method that takes $3n^2 + 6n + 50$ steps?

$O(n^2)$ because the $n^2$ term dominates

Binary search?

$O(\log n)$ because it halves the search space on each iteration

```java
for (int i = n; i > 0; i = i / 2) {
    // constant-time operations
}
```

$O(\log n)$ because the counter it halved on each iteration
What's the **Big O runtime** for:

```java
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        // constant-time operations
    }
}
```
What’s the Big O runtime for:

```java
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        // constant-time operations
    }
}
```

**O(n²)** because the outer loop will run $n$ times, and each time the inner loop can run up to $n$ times.
What’s the **Big O runtime** for:

```cpp
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        // constant-time operations
    }
}
```

**$O(n^2)$** because the outer loop will run $n$ times, and each time the inner loop can run up to $n$ times

```cpp
iterator = list->frontSentinel->next;
while (iterator != list->backSentinel) {
    if (iterator->value == value)
        return 1;
    iterator = iterator->next;
}
```
What’s the Big O runtime for:

for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        // constant-time operations
    }
}

$O(n^2)$ because the outer loop will run $n$ times, and each time the inner loop can run up to $n$ times

```
iterator = list->frontSentinel->next;
while (iterator != list->backSentinel) {
    if (iterator->value == value)
        return 1;
    iterator = iterator->next;
}
```

$O(n)$ because it needs to check each element
Know the **properties** and **operations** of the data types we’ve covered (e.g., stack, queue, dynamic array, etc.)

Be able to **compare the Big O runtimes** of common operations on different data types

Understand situations when **one data type** is preferable to another
What are the three operations of a **stack** ADT?

Which ADT would be good for **finite-length undo**?
What are the three operations of a **stack** ADT?

*push, pop, & top*

Which ADT would be good for **finite-length undo**?
What are the three operations of a stack ADT?

push, pop, & top

Which ADT would be good for finite-length undo?

dequeue

(need to remove old entries to have finite length)
What’s the **ordering** property of a **stack**?
What’s the **ordering** property of a **stack**?

**last in, first out**
What’s the **ordering** property of a **stack**?

**last in, first out**

What about a **queue**?
What’s the **ordering** property of a **stack**?

*last in, first out*

What about a **queue**?

*first in, first out*
What’s the **ordering** property of a **stack**?

*last in, first out*

What about a **queue**?

*first in, first out*

Does the **bag** ADT have an **ordering** property?
What’s the **ordering** property of a **stack**?

**last in, first out**

What about a **queue**?

**first in, first out**

Does the **bag** ADT have an **ordering** property?

**nope**, but the **ordered bag** does
What are the **average** and **worst-case** Big O runtimes for the deque and bag interfaces on a **dynamic array** versus a linked list?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>average</td>
<td>worst</td>
</tr>
<tr>
<td>contains</td>
<td>average</td>
<td>worst</td>
</tr>
<tr>
<td>addFirst</td>
<td>average</td>
<td>worst</td>
</tr>
<tr>
<td>addLast</td>
<td>average</td>
<td>worst</td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>add</strong></td>
<td>$O(1+)$</td>
<td></td>
</tr>
<tr>
<td><strong>contains</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>addFirst</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>addLast</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The $O(1+)$ notation for the add operation in a dynamic array indicates that the runtime is generally $O(1)$, but can degrade to $O(n)$ in the worst case due to resizing.
What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td><strong>O(1+)</strong></td>
<td><strong>O(n)</strong></td>
</tr>
<tr>
<td>contains</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addFirst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addLast</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>worst</td>
</tr>
<tr>
<td>add</td>
<td>O(1+)</td>
<td>O(n)</td>
</tr>
<tr>
<td>contains</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addFirst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addLast</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>add</strong></td>
<td>O(1+)</td>
<td>O(1)</td>
</tr>
<tr>
<td><strong>contains</strong></td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td><strong>addFirst</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>addLast</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>add</strong></td>
<td>$O(1+)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td></td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>contains</strong></td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>addFirst</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>addLast</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the deque and bag interfaces on a dynamic array versus a linked list?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>worst</td>
</tr>
<tr>
<td><strong>add</strong></td>
<td>O(1+)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>contains</strong></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>addFirst</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>addLast</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the deque and bag interfaces on a **dynamic array** versus a **linked list**?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>add</strong></td>
<td>$O(1+)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>contains</strong></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>addFirst</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>addLast</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the the deque and bag interfaces on a dynamic array versus a linked list?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>average</strong></td>
<td><strong>worst</strong></td>
<td><strong>average</strong></td>
</tr>
<tr>
<td>add</td>
<td>O(1+)</td>
<td>O(1)</td>
</tr>
<tr>
<td>contains</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>addFirst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addLast</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the deque and bag interfaces on a **dynamic array** versus a **linked list**?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>add</strong></td>
<td>O(1+)</td>
<td>O(1)</td>
</tr>
<tr>
<td><strong>contains</strong></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>addFirst</strong></td>
<td>O(1+)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>addLast</strong></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the `deque` and `bag` interfaces on a **dynamic array** versus a **linked list**?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>worst</td>
</tr>
<tr>
<td><strong>add</strong></td>
<td>O(1+)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>contains</strong></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>addFirst</strong></td>
<td>O(1+)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>addLast</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>worst</td>
</tr>
<tr>
<td><strong>add</strong></td>
<td>O(1+)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>contains</strong></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>addFirst</strong></td>
<td>O(1+)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>addLast</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ADTs
What are the average and worst-case Big O runtimes for the the deque and bag interfaces on a dynamic array versus a linked list?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>worst</td>
</tr>
<tr>
<td>add</td>
<td>$O(1+)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>contains</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>addFirst</td>
<td>$O(1+)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>addLast</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th></th>
<th>linked list</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>worst</td>
<td>average</td>
<td>worst</td>
</tr>
<tr>
<td>add</td>
<td>$O(1+)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>contains</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>addFirst</td>
<td>$O(1+)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>addLast</td>
<td>$O(1+)$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the deque and bag interfaces on a dynamic array versus a linked list?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>worst</td>
</tr>
<tr>
<td><strong>add</strong></td>
<td>O(1+)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>contains</strong></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>addFirst</strong></td>
<td>O(1+)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>addLast</strong></td>
<td>O(1+)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

<table>
<thead>
<tr>
<th>Method</th>
<th>Dynamic Array Average</th>
<th>Dynamic Array Worst</th>
<th>Linked List Average</th>
<th>Linked List Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>O(1+)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>contains</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>addFirst</td>
<td>O(1+)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>addLast</td>
<td>O(1+)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
What are the **average** and **worst-case** Big O runtimes for the the deque and bag interfaces on a **dynamic array** versus a **linked list**?

<table>
<thead>
<tr>
<th></th>
<th>dynamic array</th>
<th>linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>worst</td>
</tr>
<tr>
<td>add</td>
<td>$O(1+)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>contains</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>addFirst</td>
<td>$O(1+)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>addLast</td>
<td>$O(1+)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
Be able to **write and understand C code** that uses or builds upon our ADTs

Be able to **show the state of an ADT** after a series of operations have been performed on it
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

<table>
<thead>
<tr>
<th>capacity</th>
<th>count</th>
<th>beginning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>capacity</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>count</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beginning</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
```

<table>
<thead>
<tr>
<th>capacity</th>
<th>count</th>
<th>beginning</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

<table>
<thead>
<tr>
<th>capacity</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>1</td>
</tr>
<tr>
<td>beginning</td>
<td>0</td>
</tr>
</tbody>
</table>
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>capacity</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>count</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beginning</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

```
<table>
<thead>
<tr>
<th>capacity</th>
<th>count</th>
<th>beginning</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0</td>
<td>5.0</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
```

- Capacity: 5
- Count: 3
- Beginning: 0
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

| capacity | 5 |
| count    | 3 |
| beginning| 1 |
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

| capacity | 5 |
| count    | 2 |
| beginning| 1 |
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>capacity</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>count</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beginning</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```c
define struct dynArrDeque d;  
initDynArrDeque(&d, 5);  
addBackArrDeque(&d, 3.0);  
addBackArrDeque(&d, 5.0);  
addBackArrDeque(&d, 1.0);  
removeFrontArrDeque(&d);  
addBackArrDeque(&d, 2.0);  
addFrontArrDeque(&d, 8.0);  
removeBackArrDeque(&d);  
removeFrontArrDeque(&d);  
```

<table>
<thead>
<tr>
<th></th>
<th>3.0</th>
<th>5.0</th>
<th>1.0</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>capacity</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beginning</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0</td>
<td>5.0</td>
<td>1.0</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

capacity: 5

count: 3

beginning: 1
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
```

```plaintext
<table>
<thead>
<tr>
<th></th>
<th>capacity</th>
<th>count</th>
<th>beginning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```c
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

<table>
<thead>
<tr>
<th>capacity</th>
<th>count</th>
<th>beginning</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

The state of the dynamic array is as follows:

- **Capacity**: 5
- **Count**: 3
- **Beginning**: 0

The array contents are displayed in the table.
Show the state of a dynamic array after each of the following operations:

```
struct dynArrDequeue d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

<table>
<thead>
<tr>
<th>capacity</th>
<th>count</th>
<th>beginning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

<table>
<thead>
<tr>
<th>capacity</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>4</td>
</tr>
<tr>
<td>beginning</td>
<td>0</td>
</tr>
</tbody>
</table>
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.0</td>
<td>5.0</td>
<td>1.0</td>
<td>2.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **capacity**: 5
- **count**: 3
- **beginning**: 0
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```
Show the state of a dynamic array after each of the following operations:

```c
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

<table>
<thead>
<tr>
<th>capacity</th>
<th>count</th>
<th>beginning</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
After the following instructions execute, how many times would you need to \textit{pop} to get the value 8 off of the stack?

\begin{verbatim}
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
\end{verbatim}
After the following instructions execute, how many times would you need to `pop` to get the value 8 off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```c
struct dynArrStack s;

dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```
After the following instructions execute, how many times would you need to pop to get the value 8 off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
**dynArrStackPush(&s, 3);**
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```
After the following instructions execute, how many times would you need to \textbf{pop} to get the value $8$ off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
\textcolor{blue}{dynArrStackPush(&s, 3)};
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

\[3\]
After the following instructions execute, how many times would you need to pop to get the value 8 off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

3
After the following instructions execute, how many times would you need to pop to get the value 8 off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);  // Highlighted line

dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

3
After the following instructions execute, how many times would you need to pop to get the value 8 off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
**dynArrStackPush(&s, 2);**
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

3
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
```

```c
[blue]dynArrStackPush(&s, 2);
```

```c
dynArrStackTop(&s);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```
After the following instructions execute, how many times would you need to pop to get the value 8 off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

2
7
3
After the following instructions execute, how many times would you need to \textbf{pop} to get the value \textbf{8} off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
\textcolor{blue}{dynArrStackPush(&s, 8)};
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

Table:

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
After the following instructions execute, how many times would you need to \texttt{pop} to get the value \texttt{8} off of the stack?

\begin{verbatim}
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
\end{verbatim}
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
```

```
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
```
After the following instructions execute, how many times would you need to \textbf{pop} to get the value \textbf{8} off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5):
  dynArrStackPush(&s, 1);
  dynArrStackPush(&s, 2);
  dynArrStackPop(&s);
  dynArrStackPush(&s, 9);
```
After the following instructions execute, how many times would you need to \textbf{pop} to get the value \textbf{8} off of the stack?

\begin{verbatim}
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
\textbf{dynArrStackPush}(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
\end{verbatim}
After the following instructions execute, how many times would you need to \texttt{pop} to get the value 8 off of the stack?

\begin{verbatim}
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
\textcolor{blue}{dynArrStackPush(&s, 1)};
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
\end{verbatim}

\begin{tabular}{|c|c|c|c|c|}
\hline
1  & 1  & 5  & 8  & 2  \\
\hline
3  & 7  & 3  &    &    \\
\hline
\end{tabular}
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
**dynArrStackPop(&s);**
dynArrStackPush(&s, 9);
```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
After the following instructions execute, how many times would you need to pop to get the value 8 off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1):
  dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

3
After the following instructions execute, how many times would you need to pop to get the value 8 off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

The value 8 is on the stack at position 3. Therefore, you need to pop it once to get it off the stack.
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```
After the following instructions execute, how many times would you need to \textit{pop} to get the value 8 off of the stack?

\begin{verbatim}
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
\end{verbatim}
After the following instructions execute, how many times would you need to `pop` to get the value 8 off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

```
5
8
2
7
3
```
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```c
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

**Code**

<table>
<thead>
<tr>
<th></th>
<th>pop</th>
<th>pop</th>
<th>pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>8</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Write a function to print each value of a linked list:

```c
struct SLink {
    TYPE value;
    struct SLink *next;
};

struct List {
    struct SLink *frontSntl;
    struct SLink *backSntl;
};

void _printList(struct List *list) {
}
```
Write a function to print each value of a linked list:

```c
struct SLink {
    TYPE value;
    struct SLink *next;
};

struct List {
    struct SLink *frontSntl;
    struct SLink *backSntl;
};

void _printList(struct List *list) {
    struct SLink *current;
}
```
Write a function to print each value of a linked list:

```c
struct SLink {
    TYPE value;
    struct SLink *next;
};

struct List {
    struct SLink *frontSntl;
    struct SLink *backSntl;
};

void _printList(struct List *list) {
    struct SLink *current;
    current = list->frontSntl->next;
}
```
Write a function to print each value of a linked list:

```c
struct SLink {
    TYPE value;
    struct SLink *next;
};

struct List {
    struct SLink *frontSntl;
    struct SLink *backSntl;
};

void _printList(struct List *list) {
    struct SLink *current;
    current = list->frontSntl->next;
    while (current != list->backSntl) {
        // Print the value
        current = current->next;
    }
}
```
Write a function to print each value of a linked list:

```c
struct SLink {
    TYPE value;
    struct SLink *next;
};

struct List {
    struct SLink *frontSntl;
    struct SLink *backSntl;
};

void _printList(struct List *list) {
    struct SLink *current;
    current = list->frontSntl->next;
    while (current != list->backSntl) {
        printf("Value = %d\n", current->value);
    }
}
```
Write a function to print each value of a linked list:

```c
struct SLink {
    TYPE value;
    struct SLink *next;
};

struct List {
    struct SLink *frontSntl;
    struct SLink *backSntl;
};

void _printList(struct List *list) {
    struct SLink *current;
    current = list->frontSntl->next;
    while (current != list->backSntl) {
        printf("Value = %d\n", current->value);
        current = current->next;
    }
}
```
Write the Iterator functions \texttt{next()} and \texttt{hasNext()} for a linked list:

```c
struct ListIterator {
    struct List *list;
    struct SLink *current;
};

int hasNext(struct ListIterator *itr) {
}

TYPE next (struct ListIterator *itr) {
}
```
Write the Iterator functions `next()` and `hasNext()` for a linked list:

```c
struct ListIterator {
    struct List *list;
    struct SLink *current;
};

int hasNext(struct ListIterator *itr) {
    if (itr->current->next != itr->list->backSntl) {

    }
}

TYPE next (struct ListIterator *itr) {

}
```
Write the Iterator functions `next()` and `hasNext()` for a linked list:

```c
struct ListIterator {
    struct List *list;
    struct SLink *current;
};

int hasNext(struct ListIterator *itr) {
    if (itr->current->next != itr->list->backSntl) {
        itr->current = itr->current->next;
    }
}

TYPE next (struct ListIterator *itr) {
}
```
Write the Iterator functions `next()` and `hasNext()` for a linked list:

```c
struct ListIterator {
    struct List *list;
    struct SLink *current;
};

int hasNext(struct ListIterator *itr) {
    if (itr->current->next != itr->list->backSntl) {
        itr->current = itr->current->next;
        return 1;
    }
}

TYPE next (struct ListIterator *itr) {
}
```
Write the Iterator functions `next()` and `hasNext()` for a linked list:

```c
struct ListIterator {
    struct List *list;
    struct SLink *current;
};

int hasNext(struct ListIterator *itr) {
    if (itr->current->next != itr->list->backSntl) {
        itr->current = itr->current->next;
        return 1;
    } else {
        return 0;
    }
}

TYPE next (struct ListIterator *itr) {
}
```
Write the Iterator functions `next()` and `hasNext()` for a linked list:

```c
struct ListIterator {
    struct List *list;
    struct SLink *current;
};

int hasNext(struct ListIterator *itr) {
    if (itr->current->next != itr->list->backSntl) {
        itr->current = itr->current->next;
        return 1;
    } else {
        return 0;
    }
}

TYPE next (struct ListIterator *itr) {
    return itr->current->value;
}
```
Use the provided \texttt{\_binarySearch()} function to \textbf{implement a contains()} function that runs in $O(\log n)$ time

\begin{verbatim}
int \_binarySearch(TYPE *data, int count, TYPE value);

int contains (struct DynArr *da, TYPE value) {

}
\end{verbatim}
Use the provided _binarySearch() function to implement a contains() function that runs in O(log n) time

```c
int _binarySearch(TYPE *data, int count, TYPE value);

int contains (struct DynArr *da, TYPE value) {
    int index;
}```
Use the provided \_binarySearch() function to implement a `contains()` function that runs in \(O(\log n)\) time.

```c
int _binarySearch(TYPE *data, int count, TYPE value);

int contains (struct DynArr *da, TYPE value) {
    int index;
    index = _binarySearch(da->data, da->size, value);
}
```
Use the provided \_binarySearch() function to **implement a contains() function** that runs in $O(\log n)$ time

```c
int contains (struct DynArr *da, TYPE value) {
    int index;
    index = \_binarySearch(da->data, da->size, value);
    if (index < da->size) {
    }
}
```
Use the provided \_binarySearch() function to implement a contains() function that runs in O(log n) time

```c
int \_binarySearch(TYPE \*data, int count, TYPE value);

int contains (struct DynArr \*da, TYPE value) {
    int index;
    index = \_binarySearch(da->data, da->size, value);

    if (index < da->size) {
        if (da->data[index] == value) {

        }
    }
}
```
Use the provided `_binarySearch()` function to **implement a contains() function** that runs in $O(\log n)$ time

```c
int _binarySearch(TYPE *data, int count, TYPE value);

int contains (struct DynArr *da, TYPE value) {
    int index;
    index = _binarySearch(da->data, da->size, value);

    if (index < da->size) {
        if (da->data[index] == value) {
            return 1;
        }
    }
}
```
Use the provided `_binarySearch()` function to **implement a contains() function** that runs in $O(\log n)$ time.

```c
int _binarySearch(TYPE *data, int count, TYPE value);

int contains (struct DynArr *da, TYPE value) {
    int index;
    index = _binarySearch(da->data, da->size, value);
    if (index < da->size) {
        if (da->data[index] == value) {
            return 1;
        }
    }
    return 0;
}
```
This was practice...
midterm questions will be different!

---

Miss Lenhart couldn't be here today, so she asked me to substitute.

Mr. Munroe

I've put out your tests. Please get started.

Mr. Munroe, Miss Lenhart never taught us this.

That's because Miss Lenhart doesn't understand how important certain kinds of math are.

But this just looks --

This material is more vital than anything you've ever learned.

But --

No buts.

This is a matter of life and death.

---

1. The velociraptor spots you 40 meters away and attacks, accelerating at 4 m/s² up to its top speed of 25 m/s. When it spots you, you begin to flee, quickly reaching your top speed of 6 m/s. How far can you get before you're caught and devoured?

2. You are at the center of a 20m equilateral triangle with a raptor at each corner. The top raptor has a wounded leg and is limited to a top speed of 10 m/s.

(NOT TO SCALE)

The raptors will run toward you. At what angle should you run to maximize the time you stay alive?

3. Raptors can open doors, but they are slowed by them. Using the floor plan on the next page, plot a route through the building, assuming raptors take 5 minutes to open the first door and halve the time for each subsequent door. Remember, raptors run at 10 m/s and they do not know fear.
That’s all!

Any questions?