Lectures 5 - CS163

1. Topic 4 - Stacks & Queues
2. Circular Linked Lists
3. Doubly Linked Lists, Additional Linked Lists
4. Next time: Recursion vs iteration

PRACTICE !!!

Announcements:

* Program 2 is on Stacks and Queues

Stack Operations

1. Push: `int push(const data & to_add);`

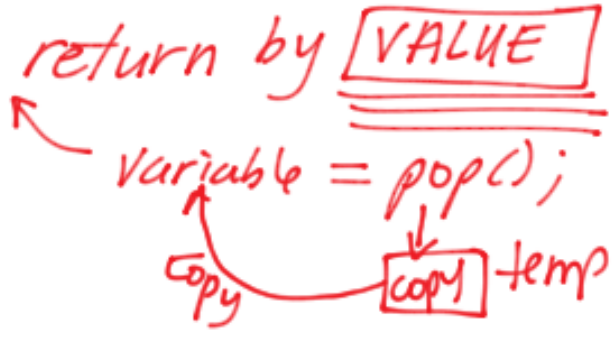
why not: `int push();` ? what's being added?

why not: `void push(data);` ? avoid pass by value when working with classes or structs

2. Pop: `int pop();` // and
`int pop(data & data_at_top);`

why not: `data pop();` ? return by VALUE

why not: `int pop(data);` ?



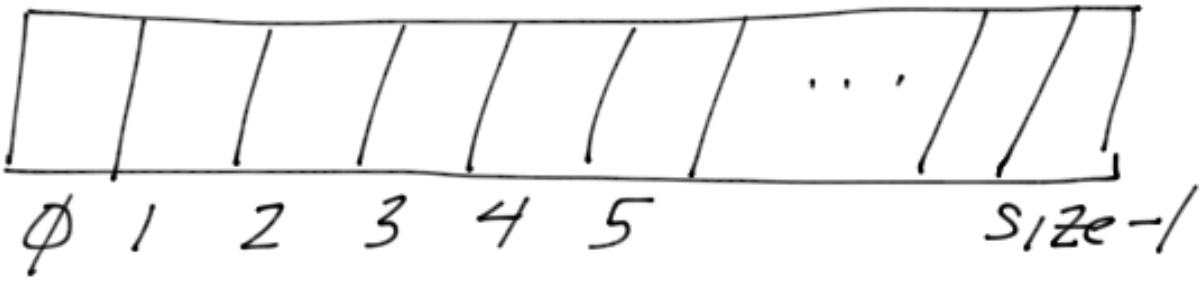
3. Peek (retrieve at top): `int peek(data & at_top);`

why not: `int peek();` ?

why not: `data peek();` ?

Stacks – Data Structures

Arrays



- where do you push? At the top index
- where is the top? It can start at 0 and go forward OR at size-1 and go backwards
- does the data shift or move? NEVER

- Option 1: use top as an index

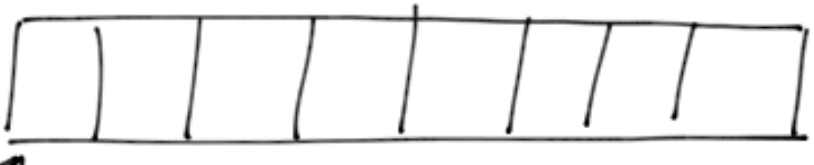
```
array[top] = _____;
++top;
```

array[top].set(to_add);

if the data is a class object

- Option 2: use top as a pointer

```
top = array; //setup
```

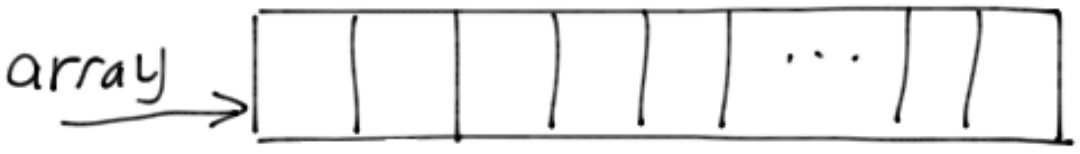



```
*top = ____
```

top->set(to_add);

Details

```
data * array;
array = new data[size];
```



Subscripts:

```
array[top] = ____
++top;
```

```
top = array;
*top = ____
++top;
```

or/ `*top++ ____`

Array of class objects

```
array[top].set(to_add);
++top;
```

```
(*top++).set(to_add);
```

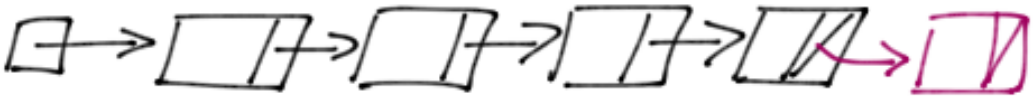
or

```
top->set(to_add);
++top;
```

Stacks - Linear Linked Lists

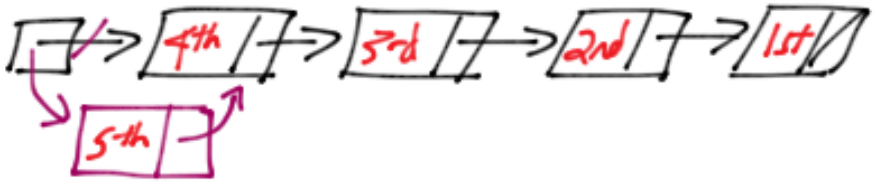
choice 1:

where do we add?



?

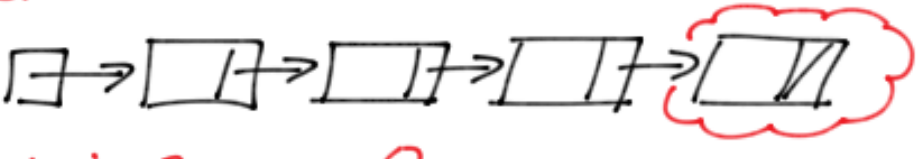
choice 2:



?

Before you make the decision — think about where we remove (pop).....

choice 1:



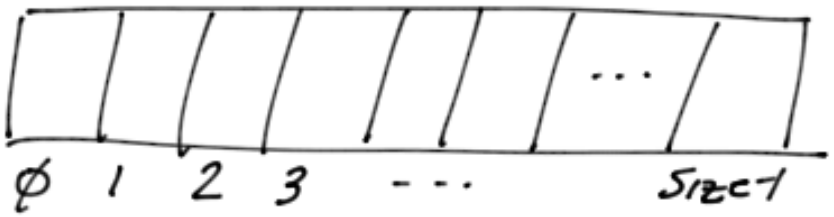
how?

choice 2:

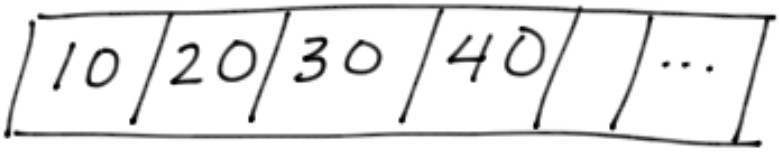
?

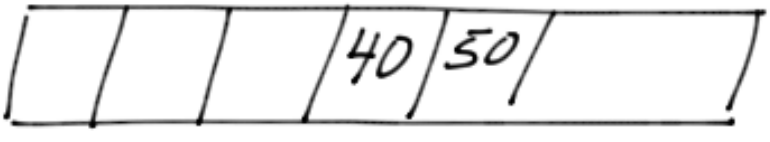
There is only ONE correct interpretation!

Queues - Using Arrays (Linear array)




```
enqueue(10);
enqueue(20);
enqueue(30);
enqueue(40);
dequeue();
dequeue();
dequeue();
enqueue(50);
```

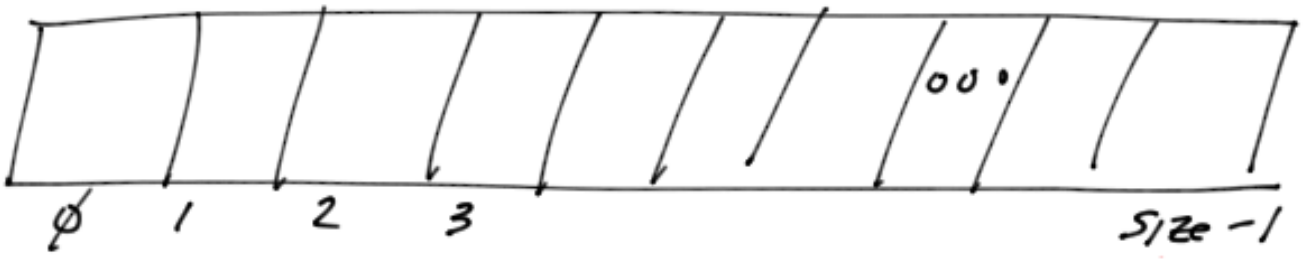




rightward Drift

Therefore, due to rightward drift Linear arrays are not viable data structures.

"Circular" Arrays



with a "Circular Array", we alter how indices are incremented:

Linear Array

```
array[index] = .....
++index;
```

Circular Array

```
array[index] = ....
index = (index+1) % size;
```

(index+1): Go To The next index

% size: make sure resulting index is within the range 0 → size-1

Efficiency: `index = (index+1) % size;`

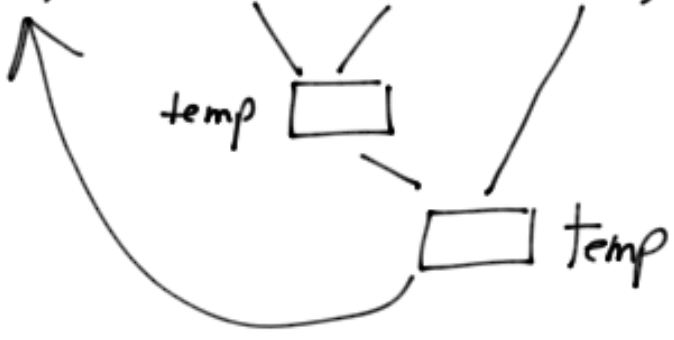
(index+1): ++index

% size: use compound assign. %=

Result: `++index %= size;`

Compare:

```
index = (index + 1) % size;
```



3 ops + 5 fetches

```
++index %= size;
```

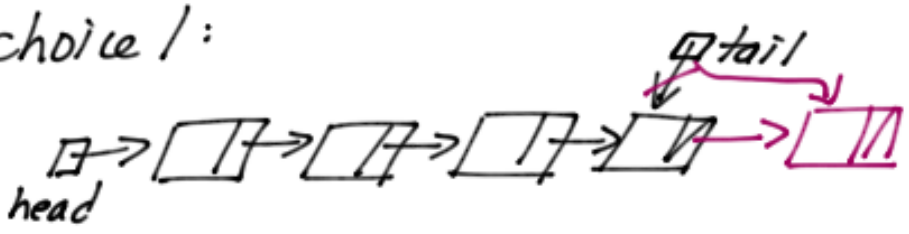
First

2 ops + 2 fetches

Queues - Linear Linked List

where to enqueue?

choice 1:




```
tail->next = new node;
tail = tail->next;
tail->next = NULL;
```

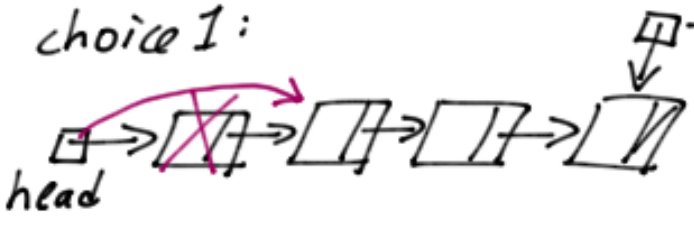
vs.

choice 2:


```
temp = new node;
temp->next = head;
head = temp;
```

First think about dequeue?

choice 1:




```
temp = head;
head = head->next;
delete temp; <- releases
                that first node
```

choice 2:

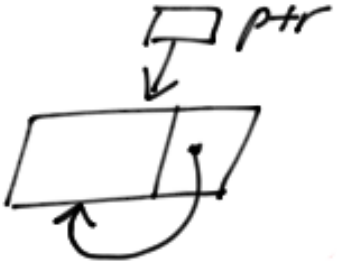

```
if (head == tail)
{   delete tail;   head = tail = NULL
}
else { current = head;
  while (current->next != tail)
      current = current->next;
  delete tail;  tail = current;
                tail->next = NULL;
```

Choice #1 is the ONLY VIABLE SOLUTION!

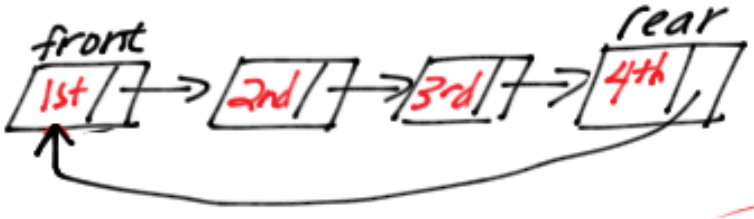
Queues - Circular Linked Lists

case 1 : Empty List

case 2 : 1 item

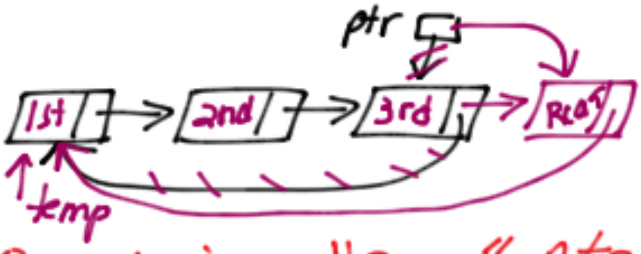


case 3 : More items



Enqueue (case 3)

```
temp = ptr->next;
ptr->next = new node;
ptr=ptr->next;
ptr->next = temp;
```



∴ ptr is a "rear" ptr

Dequeue (case 3)

```
temp = ptr->next->next;
delete ptr->next;
ptr->next = temp;
```