Stacks and Queues

Data structures that restrict which elements can be accessed at a given time

Array or a linked list implementations

Stack - Last In First Out (LIFO)

Cafeteria plates.

push(value) - Add to stack

pop() - Remove from stack

Queue - First In First Out (FIFO)

Line at grocery store

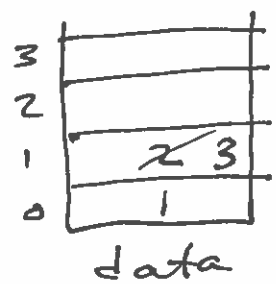
Enqueue(value) - Add to queue

Dequeue() - Remove from queue

Enq. at the tail.

Deq at the head.

Array-based implementation of stack



push(1)
push(2)
pop()
push(3)

Position where element is added is called top.

Pop, push to top

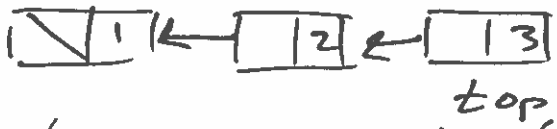
data[0..top-1] are contents of stack

When top = 0, stack is empty

When top = maxSize, stack is full

When top > maxSize, called stack overflow error

Linked List



When top = NULL, stack is empty

Stack

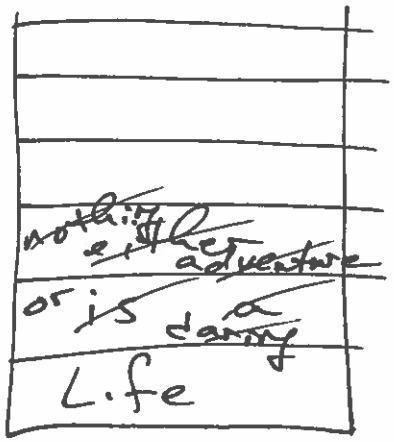
```
private:
    top
    data
    maxSize
public:
    Stack()
    isFull()
    isEmpty()
    push(value)
    pop()

push(value){
    if( !isFull() ) {
        data[top] = value
        top++
    }
```

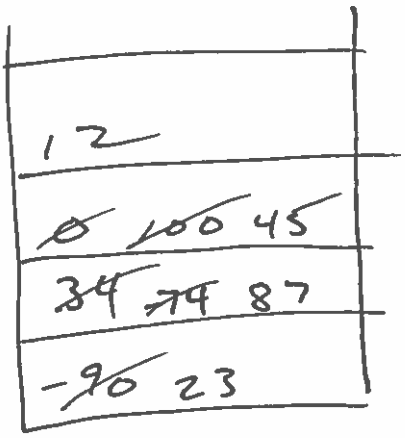
```
}

pop() {
    top--
    return data[top]
}
```

Life Is Either a daring adventure or nothing



either is a adventure daring nothing

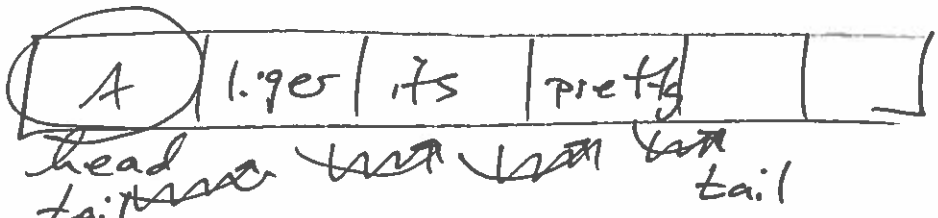


12, 45, 87, 23

34, 0, -74, -90, 100

Queue - Array implementation

Given a sentence, add words to queue

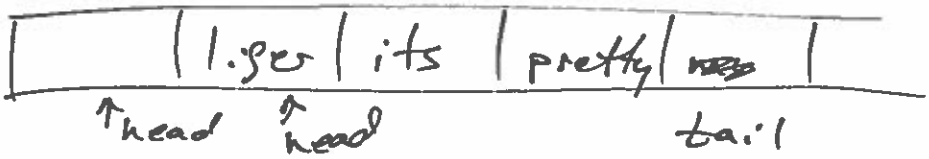


A liger its pretty much my favorite animal

enqueue at the tail

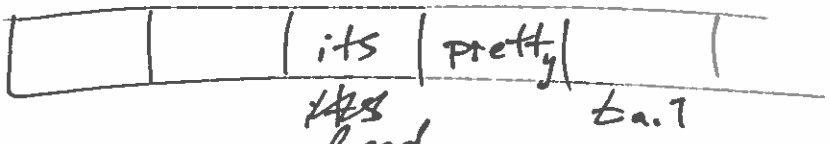
dequeue at head

deq() return "A"



Shift everything over 1, leave the head in same spot.

Move the head



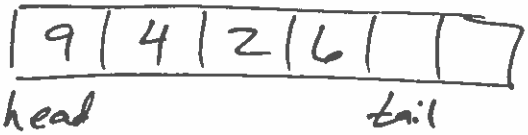
deq() return "liger"

Moving the head creates a circular queue, efficient - no shifting. Trickier to implement

Circular array queue

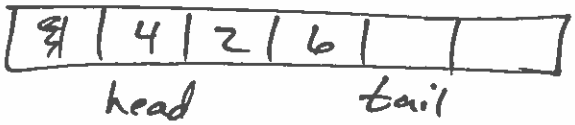
head and tail position wrap around as enq and deq happen

Example:



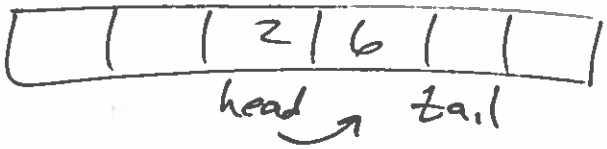
```
dequeue()
  if ( !isEmpty() )
      value = data[head]
      currentSize--
      if ( head == maxSize - 1 )
          head = 0
      else
          head++
  else
      Print "queue is empty"
  return value
```

value = 9
currentSize = 3
head = 1



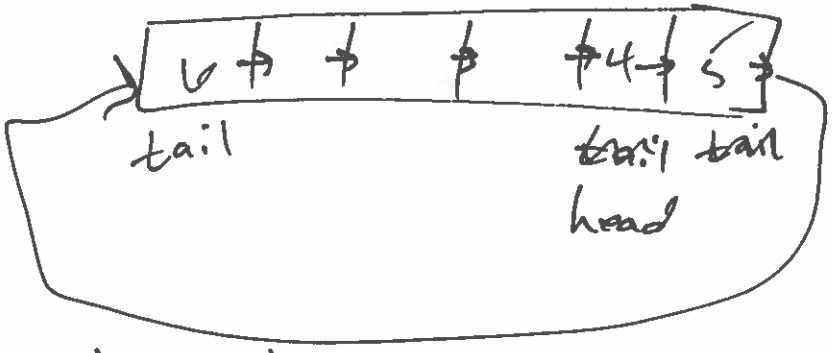
deq()

value = 4
currentSize = 2
head = 2



deq() return 2

deq() return 6



enq(4)
enq(5)
enq(6)