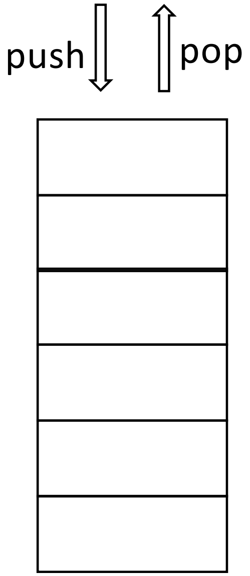
DS:
Stack & Queue

Liwei

Stack

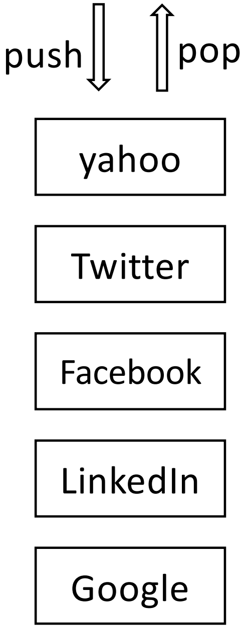
What is stack

- Property of stack
  - Follows LIFO (Last In First Out) method



Why should we learn stack?

- When we need to create an application which utilizes last incoming data first
- Example: implementation of back button in brower



Common operations in Stack

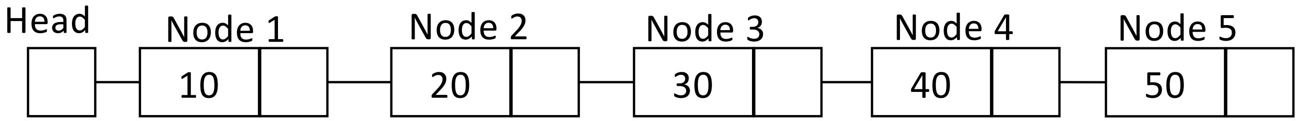
- CreateStack
- Push
- Pop
- Peek
- IsEmpty
- IsFull

Implementation options of Stack

- Array
  - Pros: Easy to implement
  - Cons: Fixed size

10	20	30	40			

- Linked List
  - Pros: Variable size
  - Cons: Moderate in implementation



Create Stack (Array Implementation)

```
CreateStack(int size)
    Create blank array of size              O(1)
    Initialize variable topOfStack to -1    O(1)
```

Time Complexity: O(1)

Push operation of Stack (Array)

```
push(value)
    if stack is full                          O(1)
        return error message                  O(1)
    else                                      O(1)
        insert value at the top of the array  O(1)
        topOfStack ++                         O(1)
```

Time Complexity: O(1)

Pop operation of Stack (Array)

```
pop()
    if stack is empty                                  O(1)
        return error message                           O(1)
    else                                               O(1)
        print top of stack                             O(1)
        topOfStack --                                  O(1)
```

Time Complexity: O(1)

Peek operation of Stack (Array)

```
peek()
    if stack is empty                   O(1)
        return error message            O(1)
    else                                O(1)
        print top of stack              O(1)
```

Time Complexity: O(1)

isEmpty operation of Stack (Array)

```
isEmpty ()
    if topOfStack is -1                      O(1)
        return true                          O(1)
    else                                     O(1)
        return false                         O(1)
```

Time Complexity: O(1)

isFull operation of Stack (Array)

```
isFull ()
    if topOfStack equals arr.size            O(1)
        return true                          O(1)
    else                                     O(1)
        return false                         O(1)
```

Time Complexity: O(1)

Push operation of Stack (Linked List)

```
push(nodeValue)
    create a node                        O(1)
    node.value = nodeValue;              O(1)
    node.next = head;                    O(1)
    head = node;                         O(1)
                                         O(1)
```

Time Complexity: O(1)

Pop operation of Stack (Linked List)

```
pop()
    if isEmpty()                          O(1)
        return error message              O(1)
    else
        tmpNode = head                    O(1)
    head = head.next                      O(1)
    return tmpNode.value                  O(1)
```

Time Complexity: O(1)

Peek operation of Stack (Linked List)

```
peek ()
    return head.value                    O(1)
```

Time Complexity: O(1)

isEmpty operation of Stack (Linked List)

```
isEmpty ()
    if (head equals null)                    O(1)
        return true                          O(1)
    else                                     O(1)
        return false                         O(1)
```

Time Complexity: O(1)

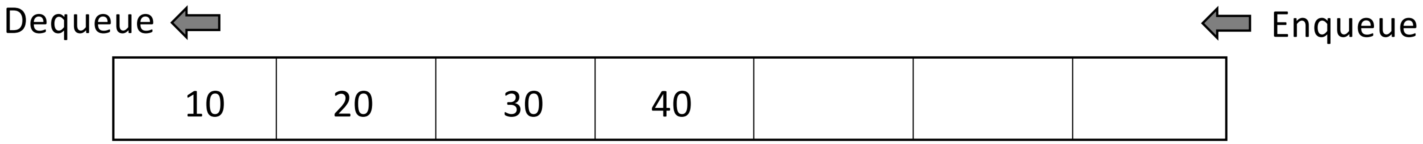
When to use/avoid Stack

- When to use
  - Helps manage the data in particular way (LIFO)
  - Cannot be easily corrupted (No one can insert data in middle)
- When to avoid
  - Random access not possible –
    if you have done some mistake, its costly to rectify.

Queue

What is Queue

- Property of queue
  - New addition of members happens at end of the queue
  - First person in the queue is the first to get out from queue
  - Follows FIFO (First In First Out) method



Dequeue

Enqueue

10	20	30	40			

Why should we learn Queue?

- When we need to create an application which utilizes first incoming data first
- Example: implementation of “billing couter”

Common operations in Queue

- createQueue
- enQueue
- deQueue
- peekInQueue
- isEmpty
- isFull

Implementation options of Queue

- Array
  - Linear Queue
  - Circular Queue
- Linked List
  - Linear Queue
  - ~~Circular Queue~~

Create Linear Queue (Array Implementation)

```
CreateQueue (int size)
    Create blank array of size                          O(1)
    Initialize variable topOfQueue to -1                O(1)
    Initialize variable beiginningOfQueue to -1         O(1)
```

Time Complexity: O(1)

enQueue operation of Queue (Array)

```
enQueue (value)
    if queue is full                              O(1)
        return error message                      O(1)
    else                                          O(1)
        arr [++topOfQueue] = value                O(1)
```

Time Complexity: O(1)

Dequeue operation of Queue (Array)

```
deQueue()
    if queue is empty                                   O(1)
        return error message                            O(1)
    else                                                O(1)
        print arr[++beginningOfQueue]                   O(1)
        If(beginningOfQueue > endOfQueue)               O(1)
                beginningOfQueue = topOfQueue = -1;     O(1)
```

Time Complexity: O(1)

Peek operation of Queue (Array)

```
peek()
    if queue is empty                       O(1)
        return error message                O(1)
    else                                    O(1)
        print arr[beginningOfQueue+1]       O(1)
```

Time Complexity: O(1)

IsEmpty operation of Queue (Array)

```
isEmpty ()
    if beginningOfQueue == -1              O(1)
        return true                        O(1)
    else                                   O(1)
        return false                       O(1)
```

Time Complexity: O(1)

IsFull operation of Queue (Array)

```
isFull  ()
    if topOfQueue == arr.size -1                O(1)
        return true                             O(1)
    else                                        O(1)
        return false                            O(1)
```

Time Complexity: O(1)

Why learn circular queue?

- deQueue operation causes blank cells

Create Circular Queue (Array)

```
CreateQueue (int size)
    Create blank array of size                    O(1)
    Initialize variable topOfQueue to -1          O(1)
    Initialize variable start to  0               O(1)
```

Time Complexity: O(1)

enQueue of Circular Queue (Array)

```
enQueue (value)
    if queue is full
        return error message
    else
        if(topOfQueue+1 == size)      // If top is already at last cell of array,
                                        reset it to first cell
                topOfQueue = 0
        else
                topOfQueue++
    arr[topOfQueue] = value
```

Time Complexity: O(1)

deQueue of Circular Queue (Array)

```
deQueue (value)
    if queue is empty
        return error message
    else
        print (arr[start])
        if(start == topOfQueue)
                topOfQueue=-1;   // If there only one element in Queue
                start =0;
        else if (start+1==size)   // If start has reached the end of array, start again from 0
                start =0;
        else
                start++;
```

Time Complexity: O(1)

Peek of Circular Queue (Array)

```
peek ()
    if (isQueueEmpty())
        print “queue is empty”
    else
        print arr[start]
```

Time Complexity: O(1)

IsEmpty of Circular Queue (Array)

```
IsEmpty ()
    if (topOfQueue == -1)
        return true
    else
        Return false
```

Time Complexity: O(1)

IsFull of Circular Queue (Array)

```
IsFull()
    if (topOfQueue+1 == start)    // If we have completed a circle,
                                  we can say that Queue is full
        return true
    else if ((start == 0)&&(topOfQueue+1 == size))
        return true               // trivial case of queue being full
    else
        Return false
```

Time Complexity: O(1)

Create Linear Queue (Linked List)

```
CreateQueue ()
    Create a blank SingleLinkedList                O(1)
    (head = null, tail = null)
```

Time Complexity: O(1)

enQueue operation of Linear Queue (LL)

```
enQueue ()
    create a node
    node.value = nodeValue
    node.next = null;
    if tail equals null   //if queue is empty
        head = tail = node;
    else                  // if queue is not empty
        tail.next = node;
        tail = node;
```

Time Complexity: O(1)

deQueue operation of Linear Queue (LL)

```
enQueue (nodeValue)
    if head equals null
        return error message
    tmpNode = head
    head = head.next
    return tmpNode;
```

Time Complexity: O(1)

peek operation of Linear Queue (LL)

```
peek ()
    if head equals null
        return error message
    else
        return head.value;
```

Time Complexity: O(1)

isEmpty operation of Linear Queue (LL)

```
isEmpty ()
    if header equals null
        return true
    else
        return false
```

Time Complexity: O(1)

deletion operation of Linear Queue (LL)

```
deleteQueue ()
    head = tail = null
```

Time Complexity: O(1)

When to use/avoid queue

- When to use
  - Helps manage the data in particular way (FIFO)
  - Not easily corrupted (no one can easily insert data in middle)
- When to avoid
  - Random access not possible
    (if we have done some mistake, its costly to recitify.)