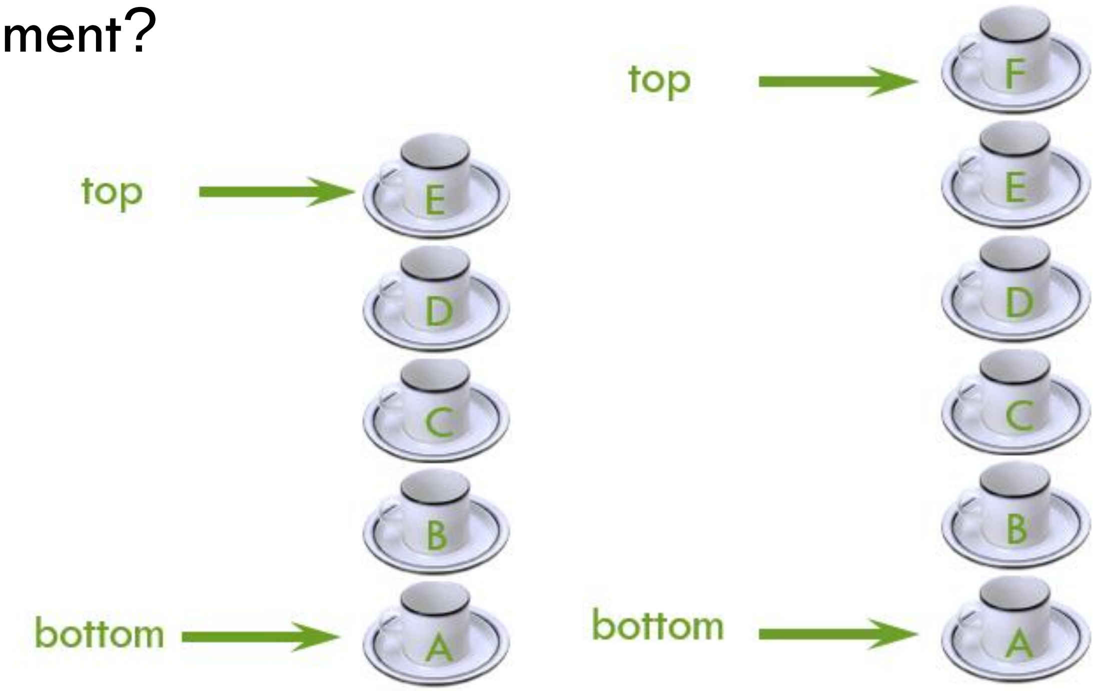
CS F211: DATA STRUCTURES & ALGORITHMS (2ND SEMESTER 2024-25) STACK, QUEUE AND DEQUE ADT

Chittaranjan Hota, PhD
Senior Professor, Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

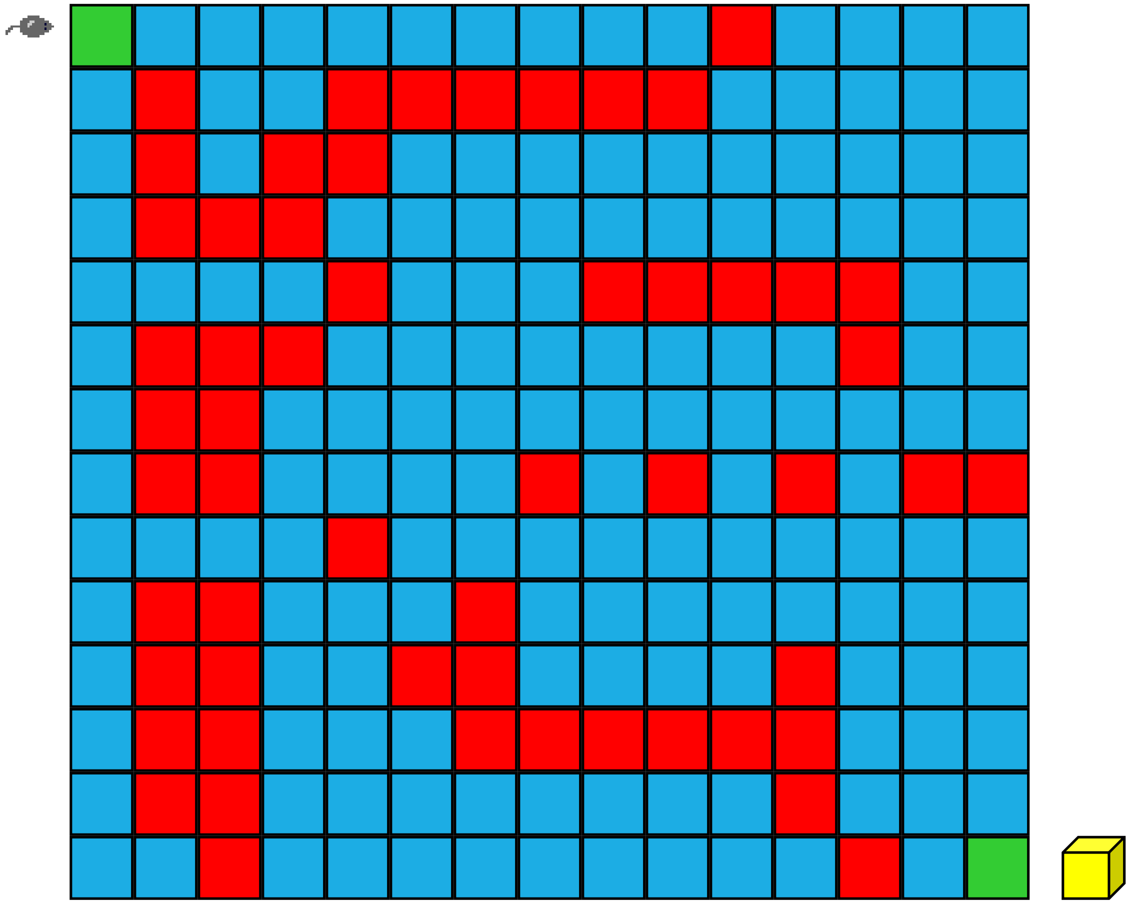
STACK ABSTRACT DATA TYPE (STACK ADT)

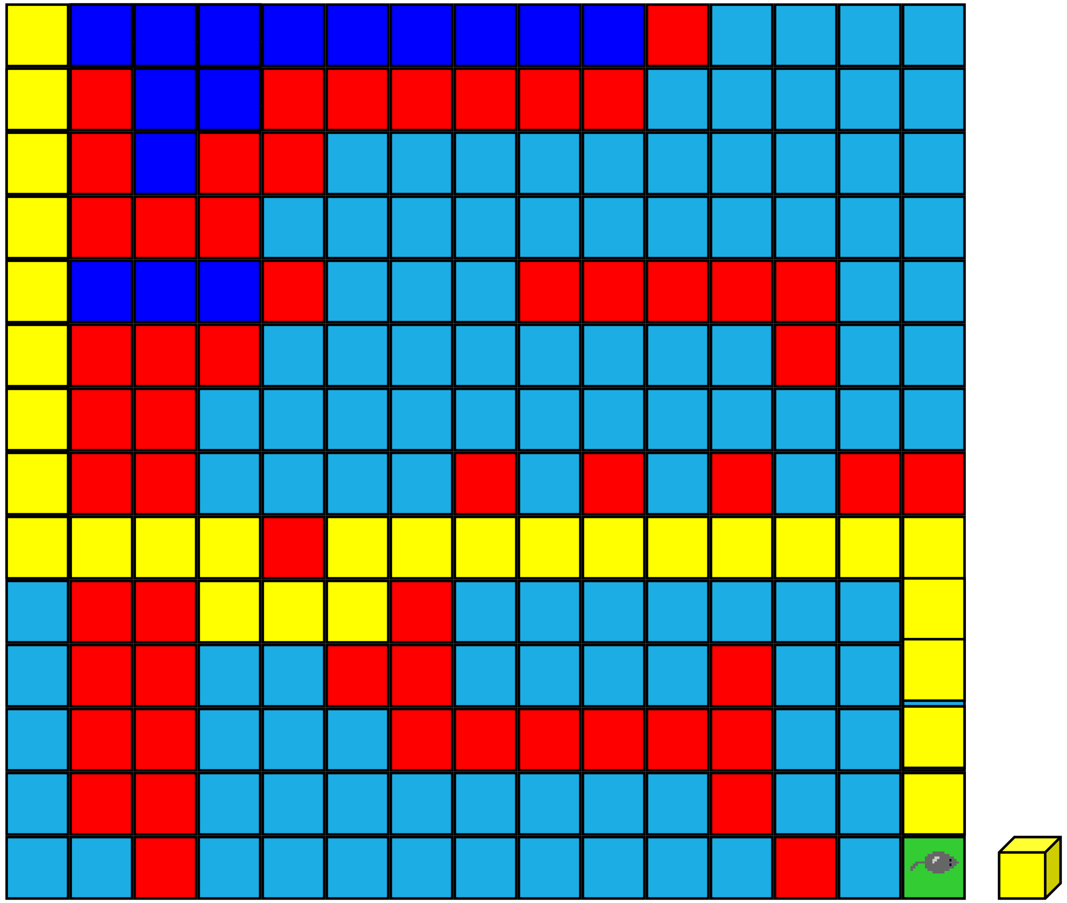
- What is a Stack?
- What type of policy does a stack implement?



Example usage: Matching parenthesis, Expression evaluation, Function call stack, Stock span, Backtracking, etc.

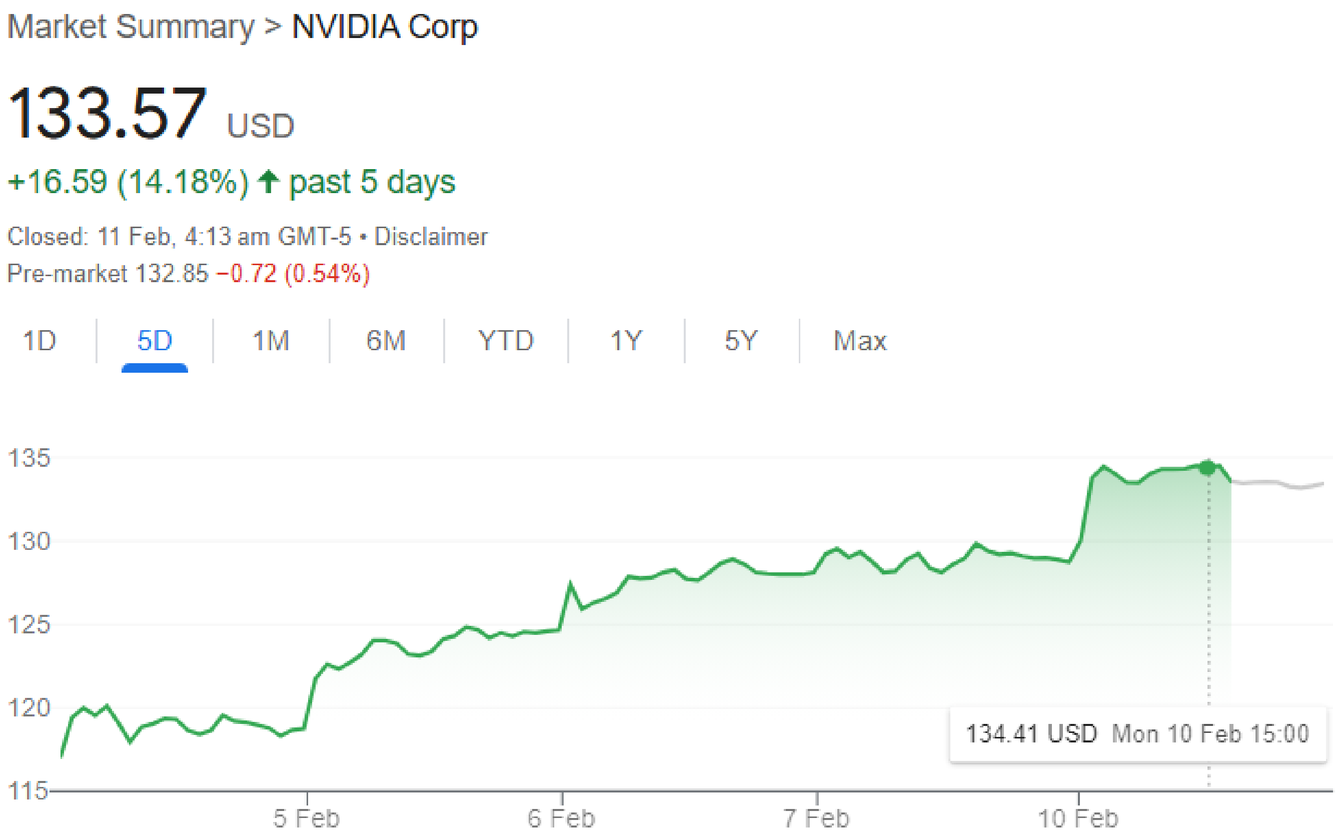
ANOTHER EX. USAGE: RAT IN A MAZE (BACKTRACKING)





STACK USAGES CONTINUED...

```
#include <iostream>
using namespace std;
void functionB() {
  cout    << "Inside Function B"   << endl;
  return;
}
void functionA() {
  cout    << "Inside Function A"   << endl;
  functionB();
  cout   << "Inside Function A"    << endl;
}
int main() {
  cout << "Inside Main function" << endl;
  functionA();
  cout <<  "Main fun finished"   <<  endl;
  return 0;
}
```



Market Summary > NVIDIA Corp

133.57 USD

+16.59 (14.18%) ↑ past 5 days

Closed: 11 Feb, 4:13 am GMT-5 • Disclaimer

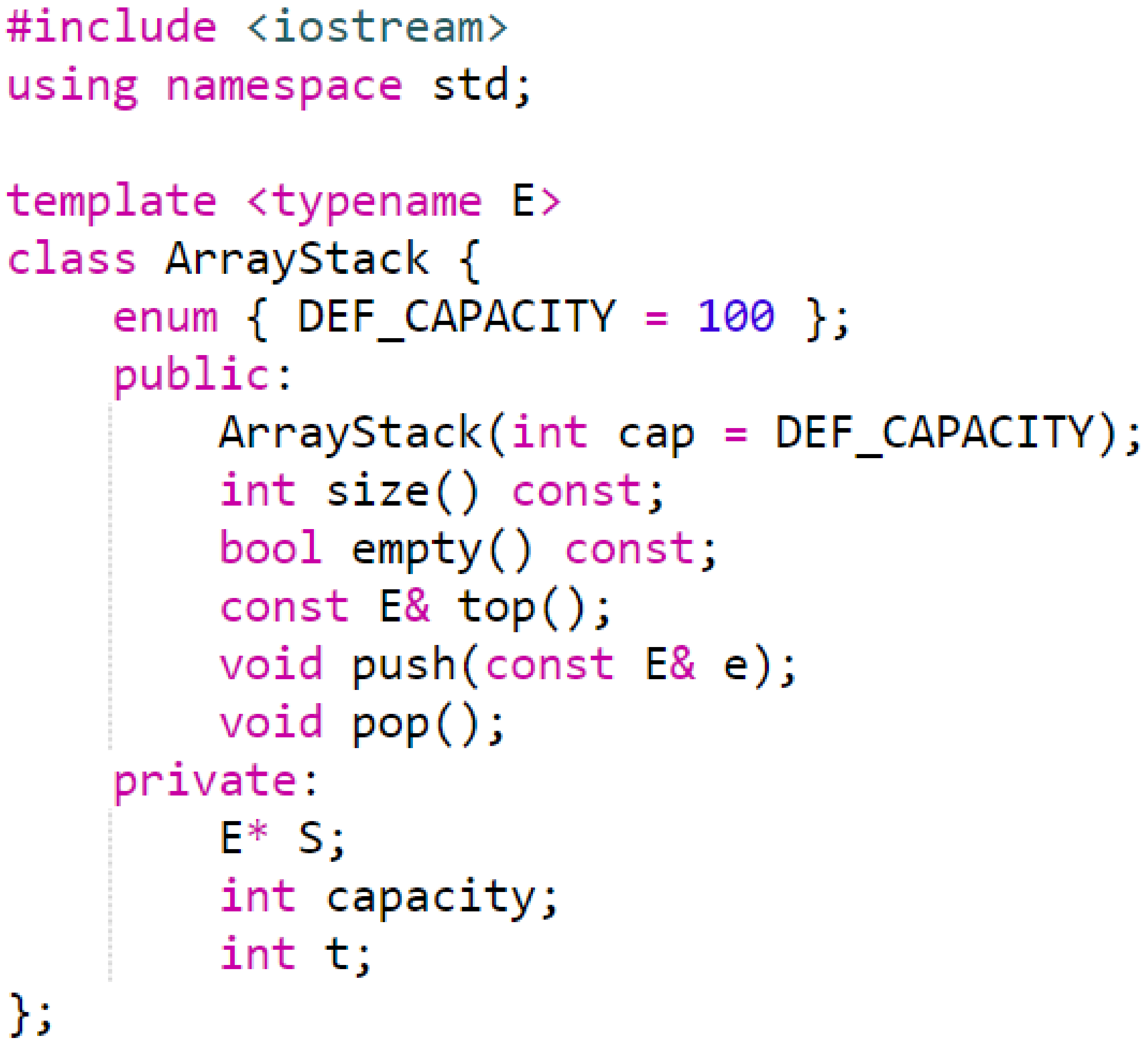
Pre-market 132.85 −0.72 (0.54%)

134.41 USD Mon 10 Feb 15:00

(Stock span: Nvidia)

STACK INTERFACE

- What is an ADT?
- Exs: Graph ADT (introductory classes), Stock trading (BST, Heaps, Hash maps, etc.)
- STACK ADT:
    - Data?
    - Operations?
    - Auxiliary operations?



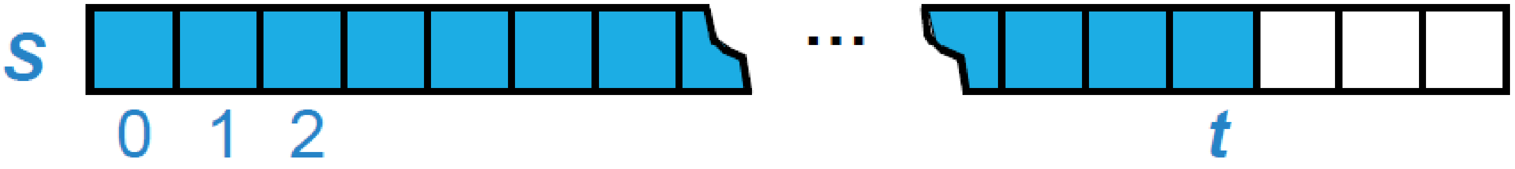
```
#include <iostream>
using namespace std;

template <typename E>
class ArrayStack {
    enum { DEF_CAPACITY = 100 };
    public:
        ArrayStack(int cap = DEF_CAPACITY);
        int size() const;
        bool empty() const;
        const E& top();
        void push(const E& e);
        void pop();
    private:
        E* S;
        int capacity;
        int t;
};
```

ARRAY-BASED STACK IMPLEMENTATION

- A simple way of implementing the Stack ADT uses an array.
- We add elements from left to right.
- A variable keeps track of the index of the top element.

```
Algorithm push(o)
    if $t = S.size() - 1$ then
        throw StackFull
    else
        $t \leftarrow t + 1$
        $S[t] \leftarrow o$
```

```
Algorithm pop()
    if empty() then
        throw StackEmpty
    else
        $t \leftarrow t - 1$
        return $S[t + 1]$
```

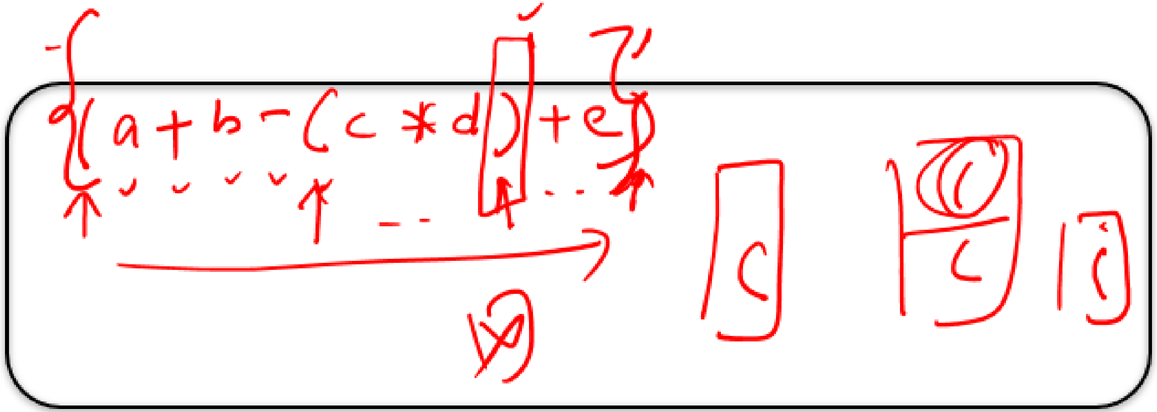


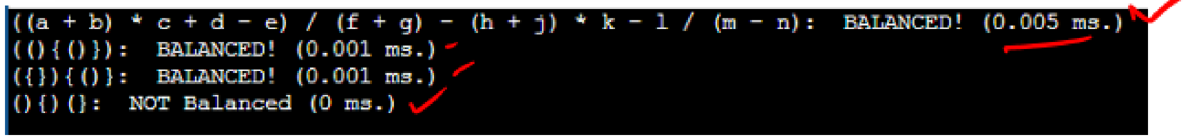
```
Algorithm size()
    return $t + 1$
```

[[ ]] {{ }} (( ))

STACK USAGE EXAMPLE-1: MATCHING PARENTHESIS

```
Let S be an empty stack
for i=0 to n-1 do
  if X[i] is an opening grouping symbol then
     S.push(X[i])
  else
    if X[i] is a closing grouping symbol then
       if S.empty() then
             return false {nothing to match with}
       if S.pop() does not match the type of X[i] then
             return false {wrong type}
if S.empty() then
   return true {every symbol matched}

else

  return false {some symbols were never matched}
```

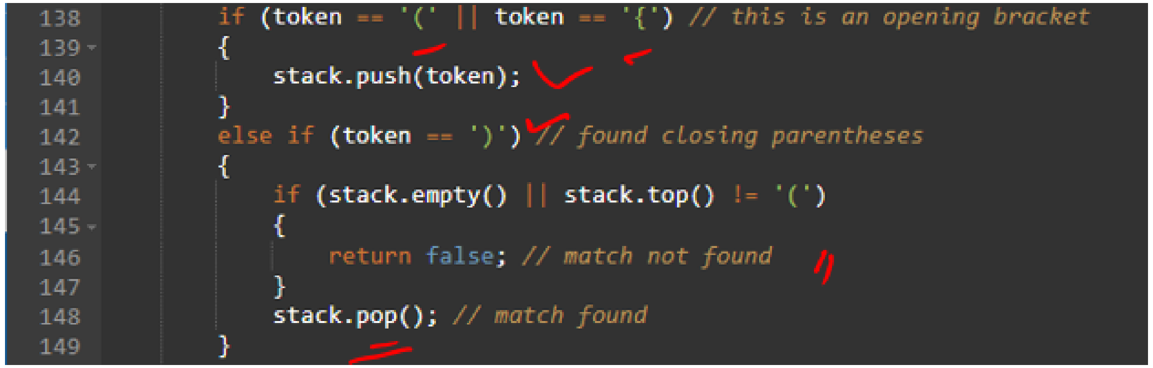




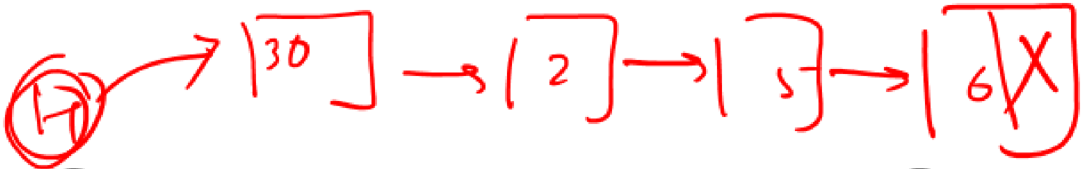
```
((a + b) * c + d - e) / (f + g) - (h + j) * k - l / (m - n):  BALANCED!  (0.005 ms.)
(){()}):  BALANCED!  (0.001 ms.)
({}){()}:  BALANCED!  (0.001 ms.)
(){}(}:  NOT Balanced  (0 ms.)
```

(Output)

Lab-6 (Next week's Lab)



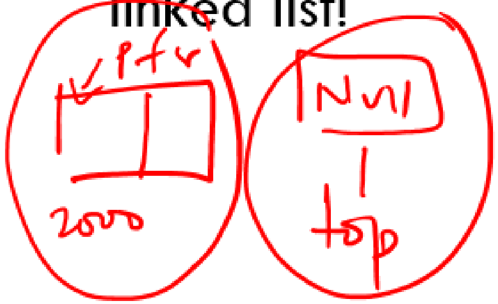
```
        if (token == '(' || token == '{') // this is an opening bracket
        {
            stack.push(token);
        }
        else if (token == ')') // found closing parentheses
        {
            if (stack.empty() || stack.top() != '(')
            {
                return false; // match not found
            }
            stack.pop(); // match found
        }
```



LINKED LIST-BASED STACK IMPLEMENTATION

LIFO

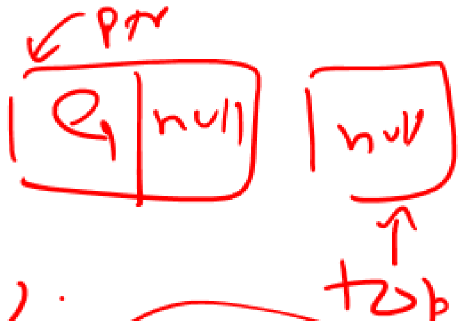
- Let us see how to do push into and pop from an empty stack implemented using a linked list!



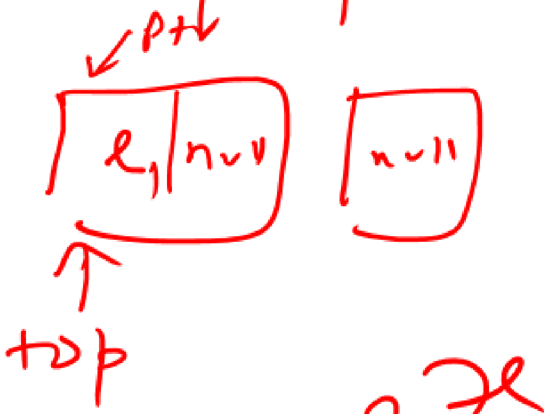
Node * top = null;

ptr -> data = e1;

ptr -> next = *top;



*top = ptr

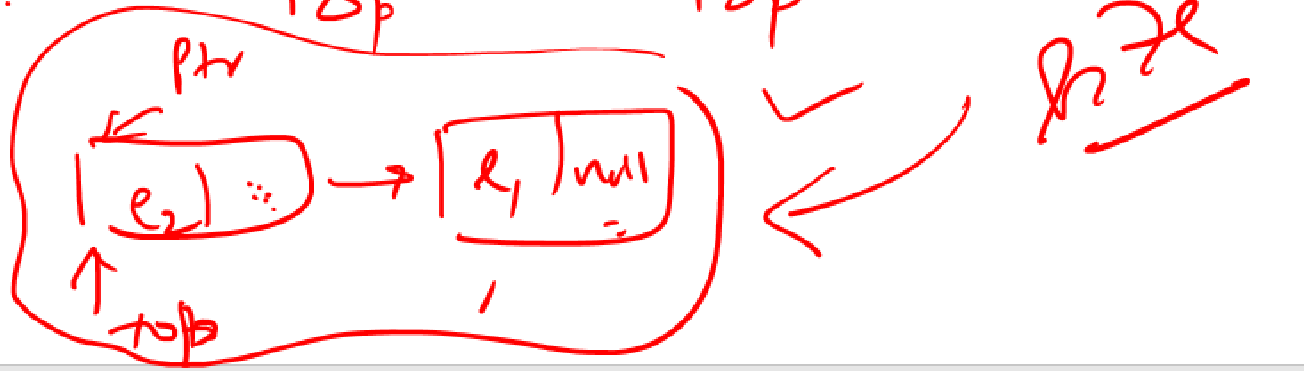


func Node * ptr = new Node();

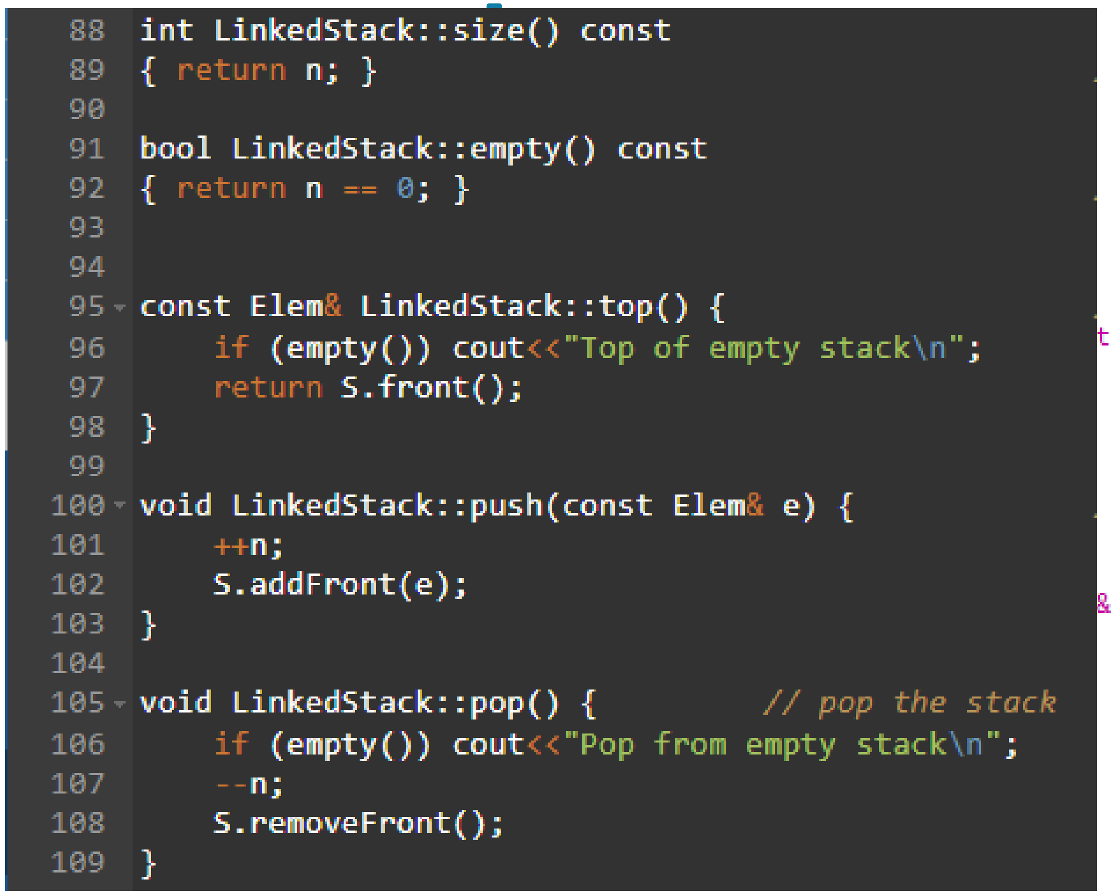
delete
return data;

Node * tmp = top;
data = tmp -> data;
top = tmp -> next;

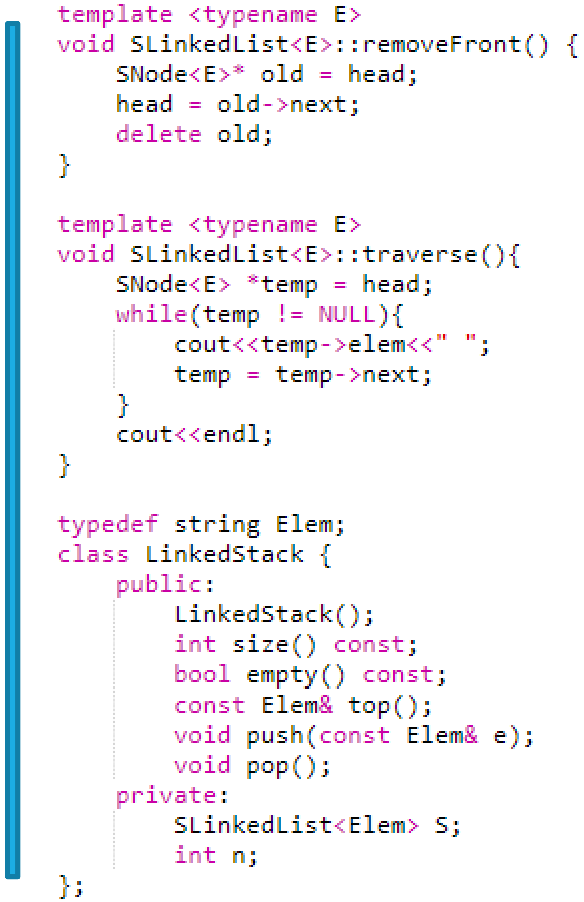
pop



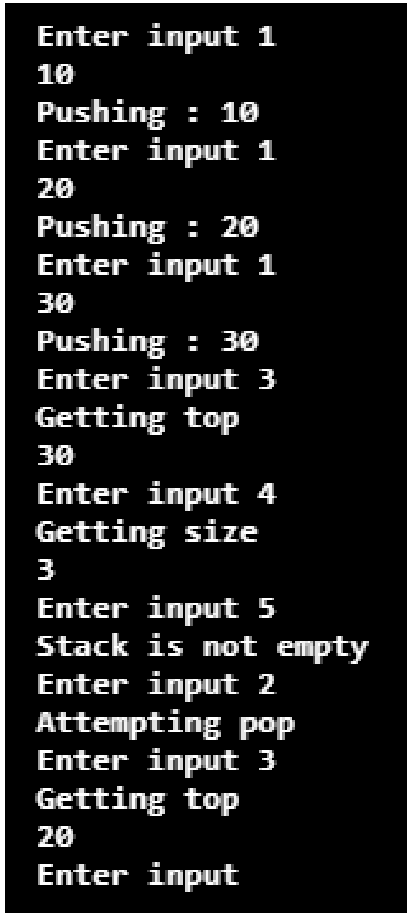
IMPLEMENTING A STACK WITH A GENERIC LINKED LIST



```
int LinkedStack::size() const
{ return n; }

bool LinkedStack::empty() const
{ return n == 0; }


const Elem& LinkedStack::top() {
    if (empty()) cout<<"Top of empty stack\n";
    return S.front();
}

void LinkedStack::push(const Elem& e) {
    ++n;
    S.addFront(e);
}

void LinkedStack::pop() {          // pop the stack
    if (empty()) cout<<"Pop from empty stack\n";
    --n;
    S.removeFront();
}
```

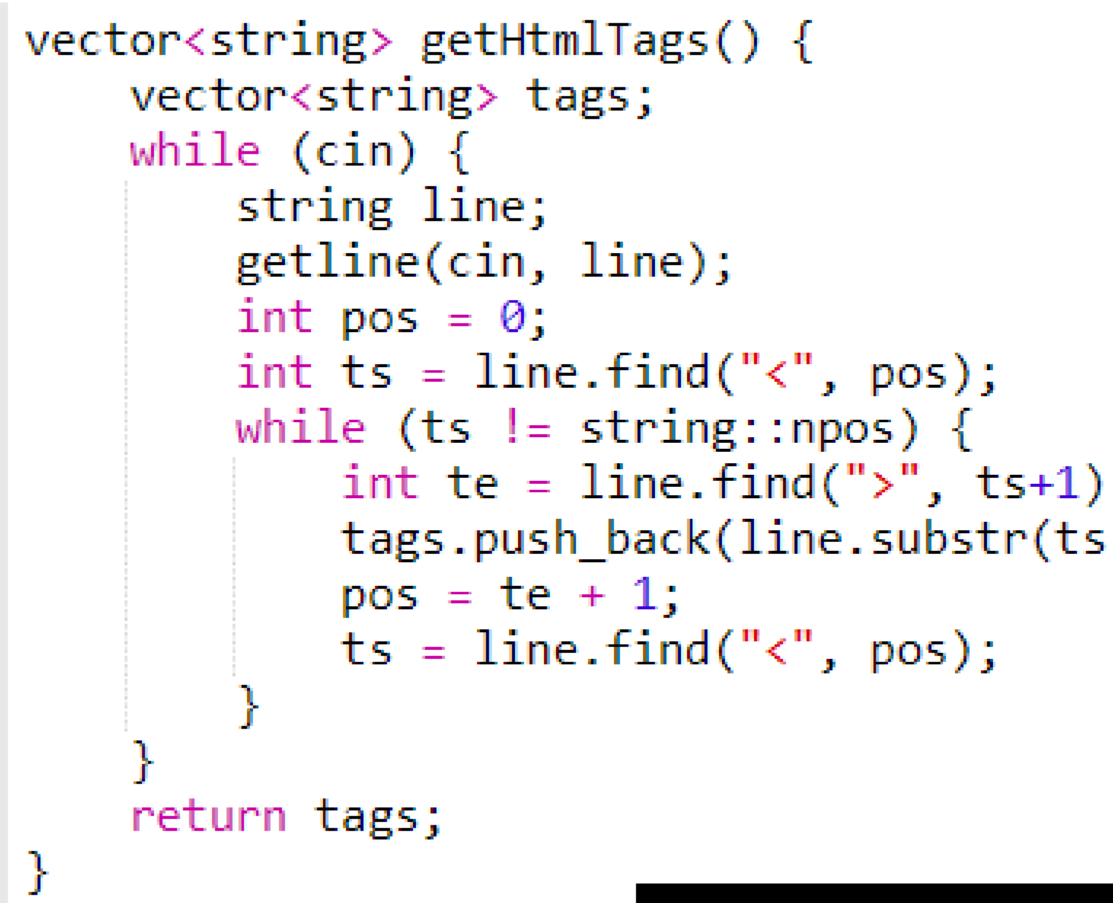


```
template <typename E>
void SLinkedList<E>::removeFront() {
    SNode<E>* old = head;
    head = old->next;
    delete old;
}

template <typename E>
void SLinkedList<E>::traverse(){
    SNode<E> *temp = head;
    while(temp != NULL){
        cout<<temp->elem<<" ";
        temp = temp->next;
    }
    cout<<endl;
}

typedef string Elem;
class LinkedStack {
    public:
        LinkedStack();
        int size() const;
        bool empty() const;
        const Elem& top();
        void push(const Elem& e);
        void pop();
    private:
        SLinkedList<Elem> S;
        int n;
};
```

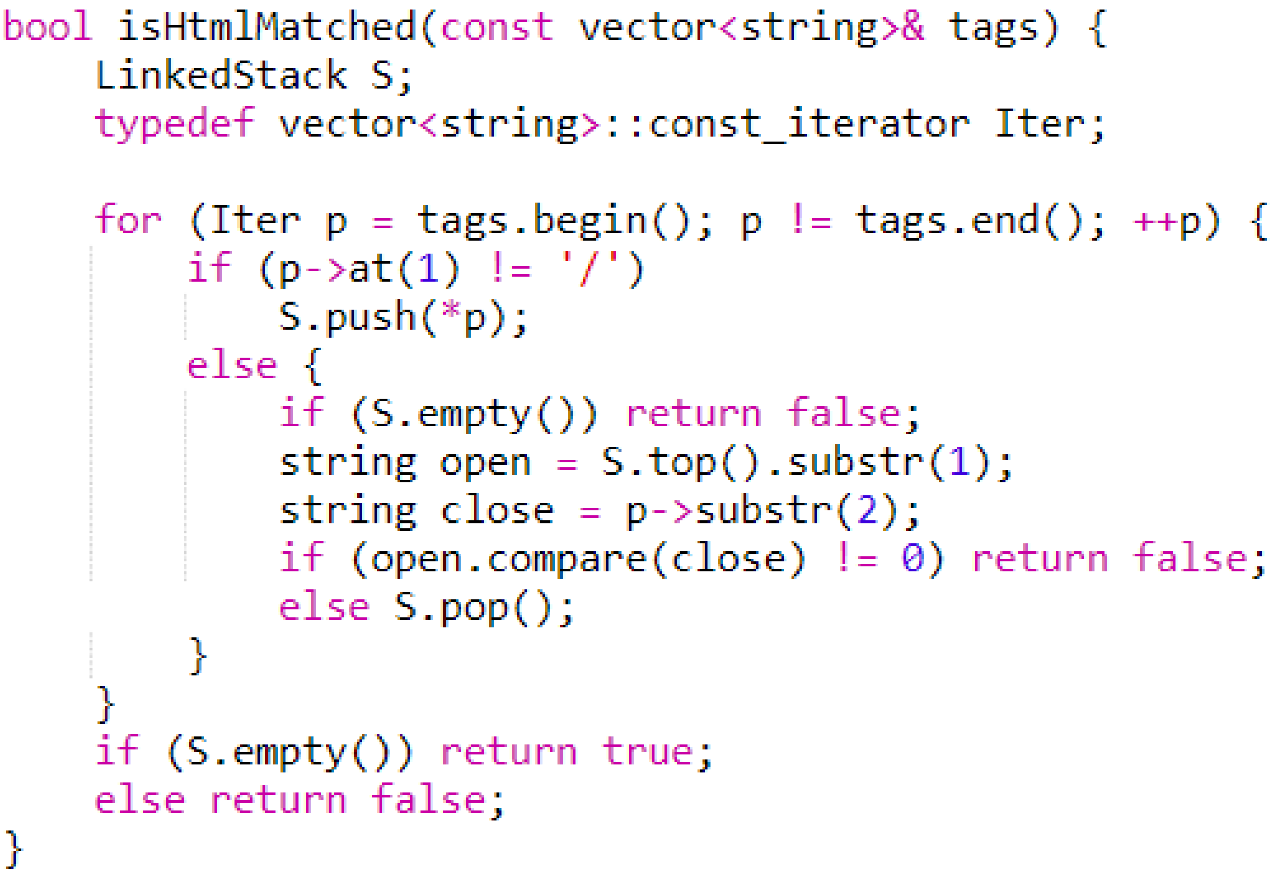


```
Enter input 1
10
Pushing : 10
Enter input 1
20
Pushing : 20
Enter input 1
30
Pushing : 30
Enter input 3
Getting top
30
Enter input 4
Getting size
3
Enter input 5
Stack is not empty
Enter input 2
Attempting pop
Enter input 3
Getting top
20
Enter input
```

EX-2: MATCHING TAGS

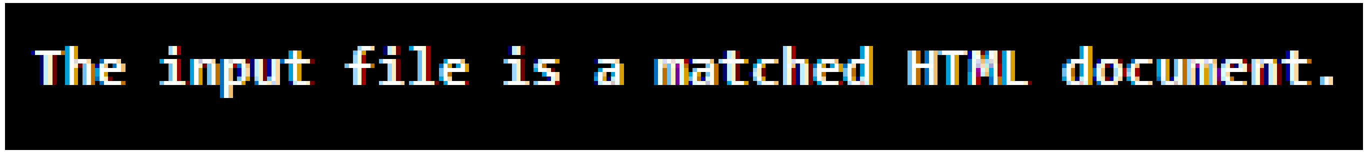


```
vector<string> getHtmlTags() {
    vector<string> tags;
    while (cin) {
        string line;
        getline(cin, line);
        int pos = 0;
        int ts = line.find("<", pos);
        while (ts != string::npos) {
            int te = line.find(">", ts+1)
            tags.push_back(line.substr(ts
            pos = te + 1;
            ts = line.find("<", pos);
        }
    }
    return tags;
}
```



```
bool isHtmlMatched(const vector<string>& tags) {
    LinkedStack S;
    typedef vector<string>::const_iterator Iter;

    for (Iter p = tags.begin(); p != tags.end(); ++p) {
        if (p->at(1) != '/')
            S.push(*p);
        else {
            if (S.empty()) return false;
            string open = S.top().substr(1);
            string close = p->substr(2);
            if (open.compare(close) != 0) return false;
            else S.pop();
        }
    }
    if (S.empty()) return true;
    else return false;
}
```

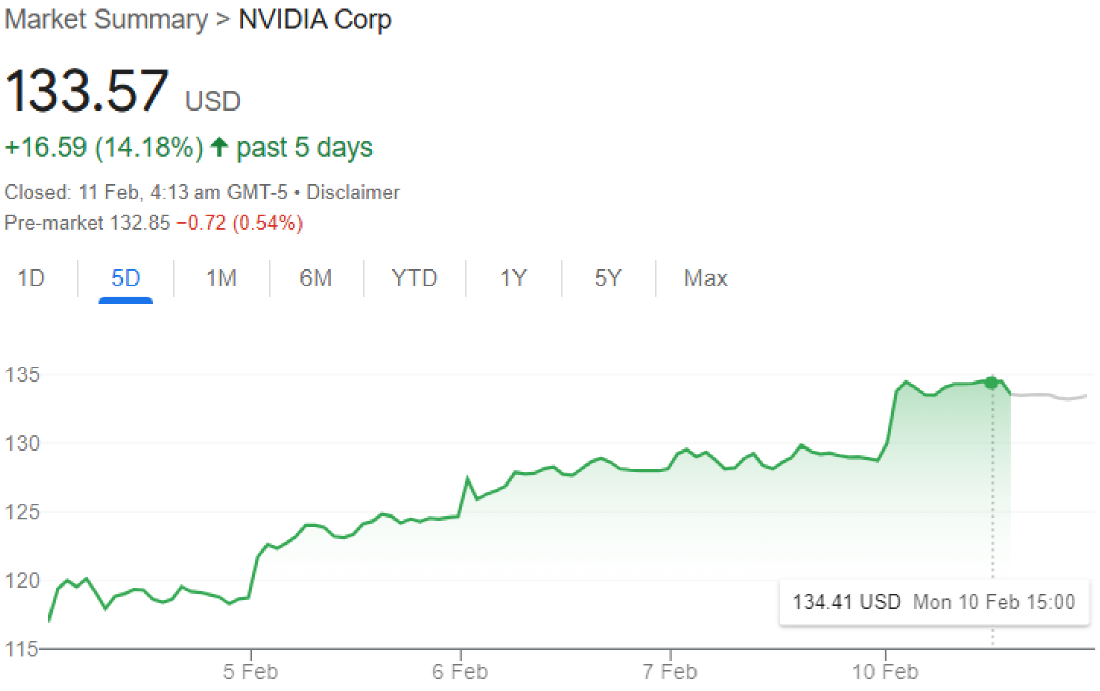
Lab6: Next week



```
The input file is a matched HTML document.
```

STACK USAGE EXAMPLE-3: STOCK SPAN

- Stock span can be defined as the number of consecutive days before the current day where the price of the stock was equal to or less than the current price.



Market Summary > NVIDIA Corp

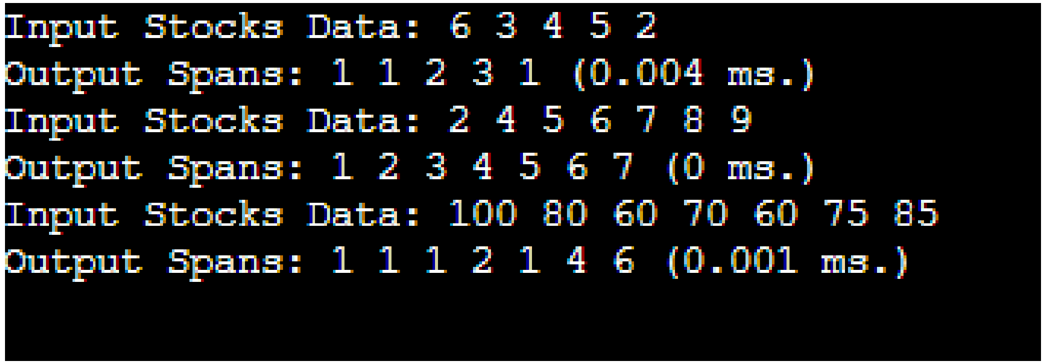
133.57 USD

+16.59 (14.18%) ↑ past 5 days

Closed: 11 Feb, 4:13 am GMT-5 • Disclaimer

Pre-market 132.85 −0.72 (0.54%)

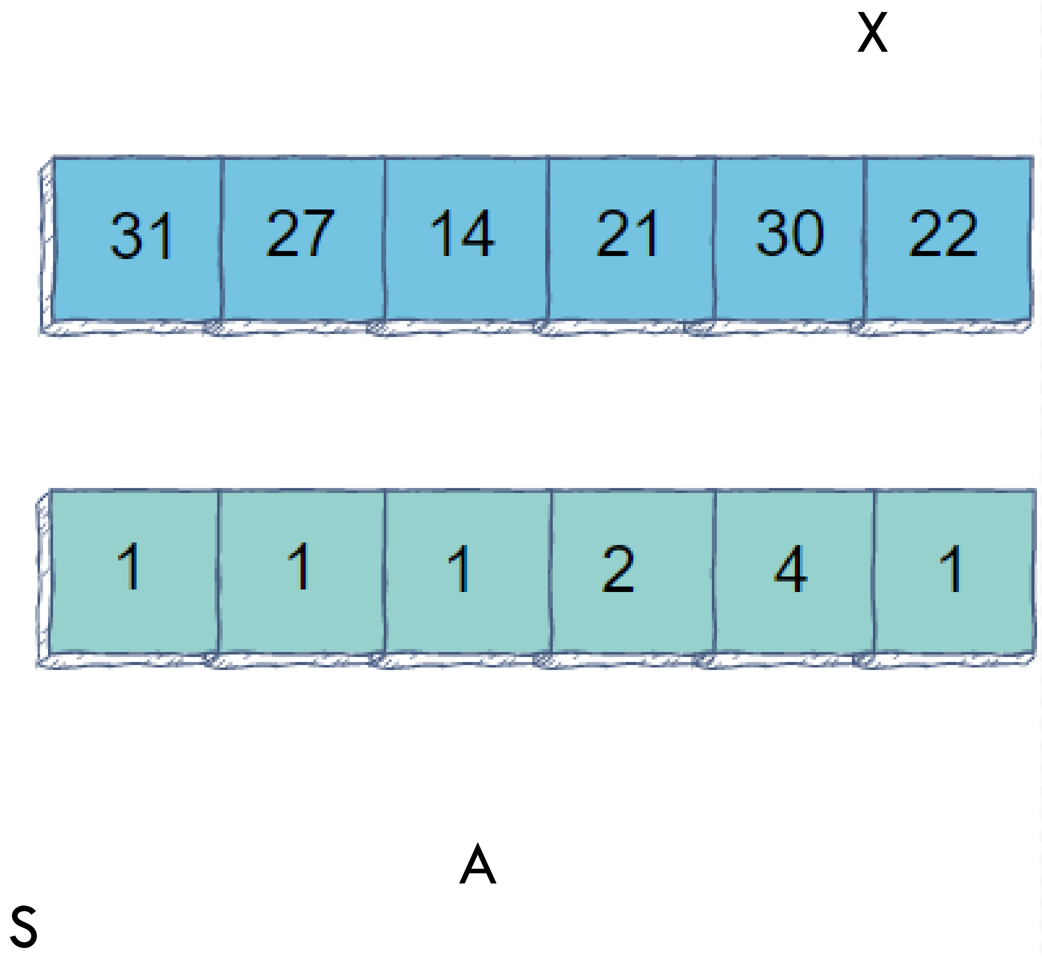
(Stock span: Nvidia)



```
Input Stocks Data: 6 3 4 5 2
Output Spans: 1 1 2 3 1 (0.004 ms.)
Input Stocks Data: 2 4 5 6 7 8 9
Output Spans: 1 2 3 4 5 6 7 (0 ms.)
Input Stocks Data: 100 80 60 70 60 75 85
Output Spans: 1 1 1 2 1 4 6 (0.001 ms.)
```

Lab 6: Next week's lab

COMPUTING STOCK SPAN CONTINUED…



31	27	14	21	30	22

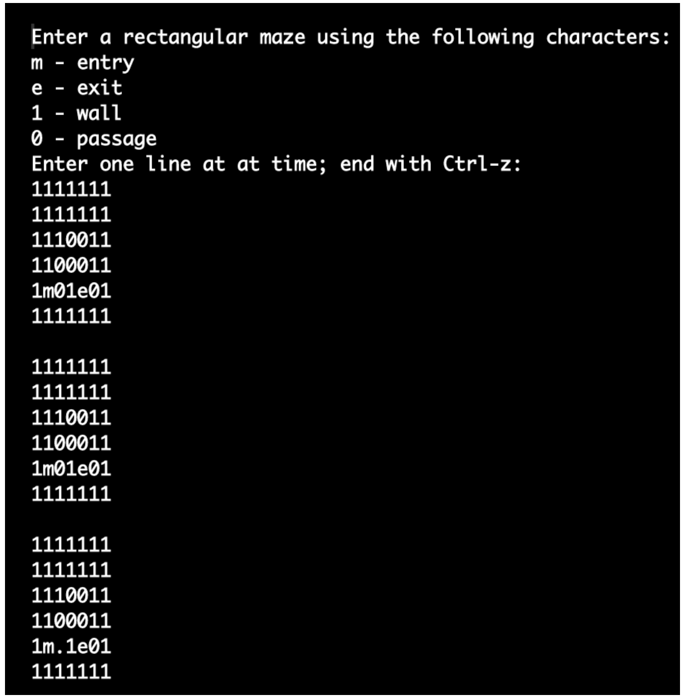
1	1	1	2	4	1

```
STOCK_SPAN(prices):
  n = length(prices)
  span = array of size n  // To store the stock spans
  stack = empty stack     // Stack to store indices

  for i from 0 to n-1:
    while stack is not empty AND prices[stack.top()] <= prices[i]:
      stack.pop()

    if stack is empty:
      span[i] = i + 1  // Span is from the beginning
    else:
      span[i] = i - stack.top() // Span is difference of indices

    stack.push(i) // Push the current index

  return span
```

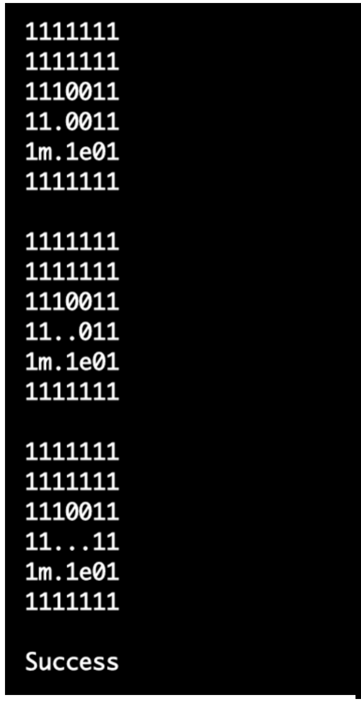
STACK USAGE EX-4: BACKTRACKING

```
1 1 1 1 1    1 1 1 1 1
1 1 0 0 1    1 1 0 0 1
1 0 0 0 1    1 0 1 0 1
m 0 1 e 0    m 0 1 e 0
```

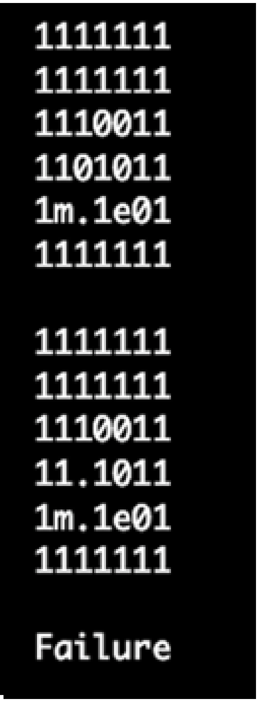
Chakravyuha from Mahabharata



```
Enter a rectangular maze using the following characters:
m - entry
e - exit
1 - wall
0 - passage
Enter one line at at time; end with Ctrl-z:
1111111
1111111
1110011
1100011
1m01e01
1111111

1111111
1111111
1110011
1100011
1m01e01
1111111

1111111
1111111
1110011
1100011
1m.1e01
1111111
```



```
1111111
1111111
1110011
11.0011
1m.1e01
1111111

1111111
1111111
1110011
11..011
1m.1e01
1111111

1111111
1111111
1110011
11...11
1m.1e01
1111111

Success
```



```
1111111
1111111
1110011
1101011
1m.1e01
1111111

1111111
1111111
1110011
11.1011
1m.1e01
1111111

Failure
```

Lab 6: Next week’s Lab

(3, 4, 5)

STACK STL IN C++ (REVERSING A VECTOR)

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
```

It is a container adaptor. Why?

```
#include <vector>
stack<int, vector<int>> myStack;
stack<int, list<int>> myStack;
stack<int, deque<int>> myStack;
```

Which one is default?

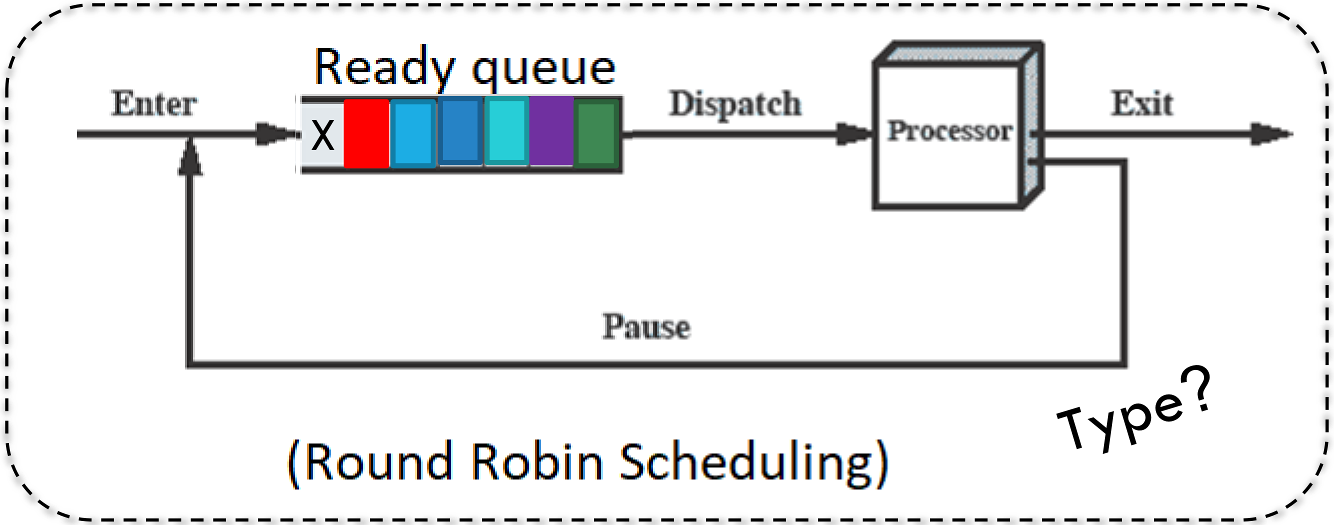
```
vector<int> rever(const vector<int>& inputVector) {
  stack<int> myStack;
  vector<int> reversedVector;
  for (int element : inputVector) {
   myStack.push(element);
  }
  while (!myStack.empty()) {
    reversedVector.push_back(myStack.top());
    myStack.pop();
  }
  return reversedVector;
}
```

(5)

(5, 4)

(5, 4, 3)

USE CASES OF QUEUES IN THE REAL WORLD



(Round Robin Scheduling)

- Is it NOT a linear data structure?
- What operations would you like to see in a Queue ADT?
- How is it different from Stack?

v = Q[0];

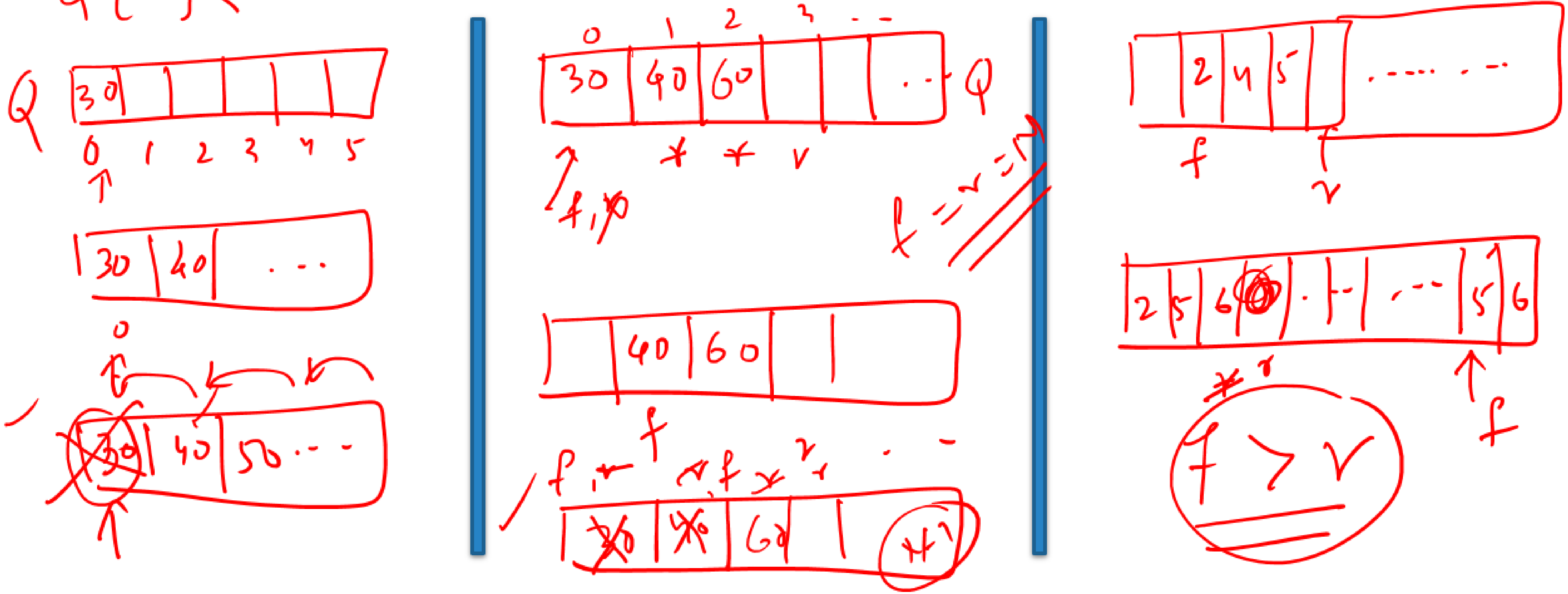
%

QUEUE IMPLEMENTED USING ARRAYS

f ≤ r

- Multiple Approaches: 1. Q[0]:front; 2. f, r, and n; 3. Circular array (with %)

Q[0] ← 30



OPERATIONS USING CIRCULAR ARRAY & INTERFACE

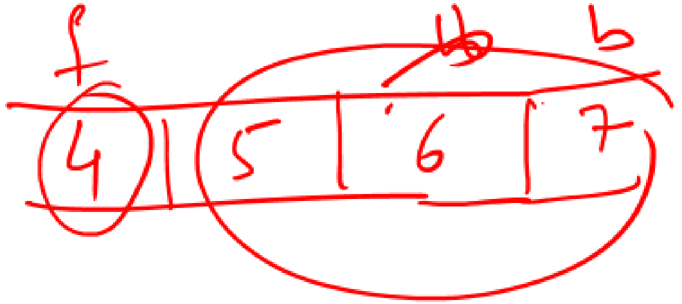
```
Algorithm size()
   return n
Algorithm empty()
   return (n == 0)
```

```
Algorithm dequeue ()
   if empty() then
      throw QueueEmpty
     else
      data = Q[f];
      f ← (f + 1) mod N;
      n = n – 1;
return data;
```

```
Algorithm enqueue (P)
{
   if size() == N then
      throw QueueFull

   else {

      Q[r] ← P;
      r ← (r+1) mod N;
      n = n + 1;
 }
}
```

```
template <typename E>

class Queue {
public:
   int size() const;
   bool empty() const;
   const E& front() const
      throw(QueueEmpty);
   void enqueue (const E& e);
   void dequeue()
      throw(QueueEmpty);
};
```

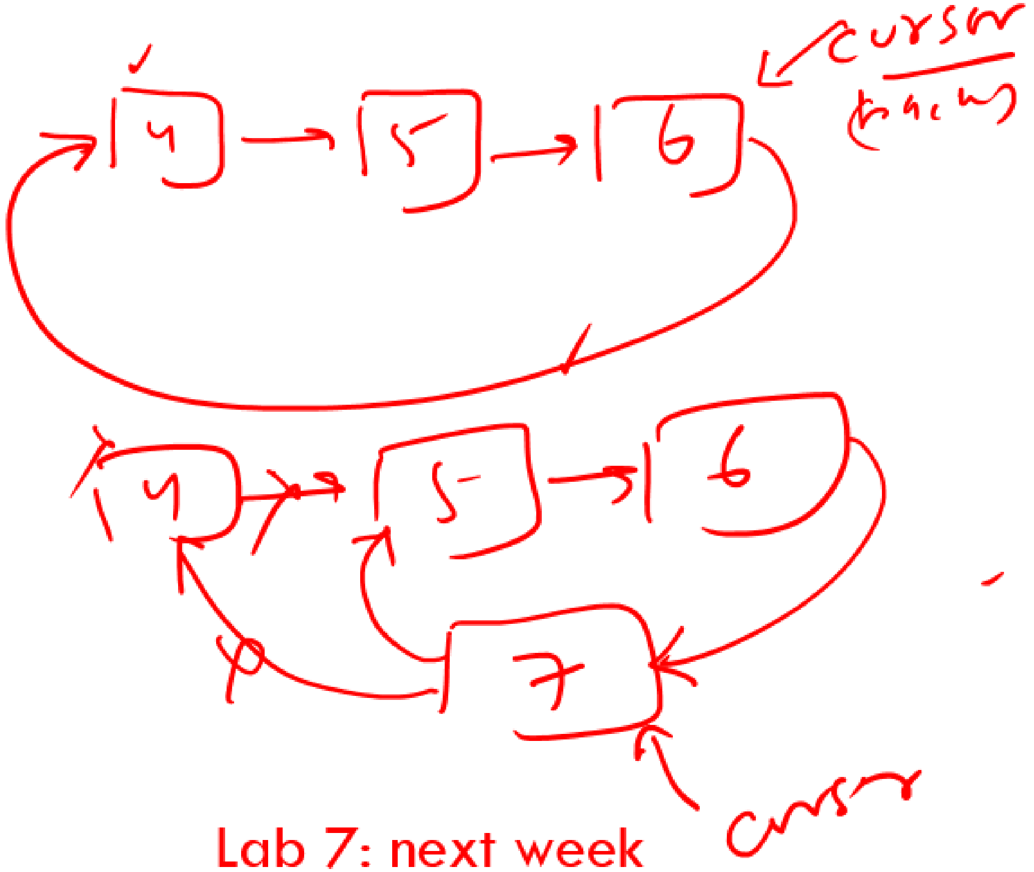
USING CIRCULAR LINKED LIST



```
typedef string Elem;
class LinkedQueue {

public:
    LinkedQueue();
    int size() const;
    bool empty() const;
    const Elem& front() const throw(QueueEmpty);

    void enqueue(const Elem& e);

    void dequeue() throw(QueueEmpty);
private:

CircleList C;

int n;

}
```

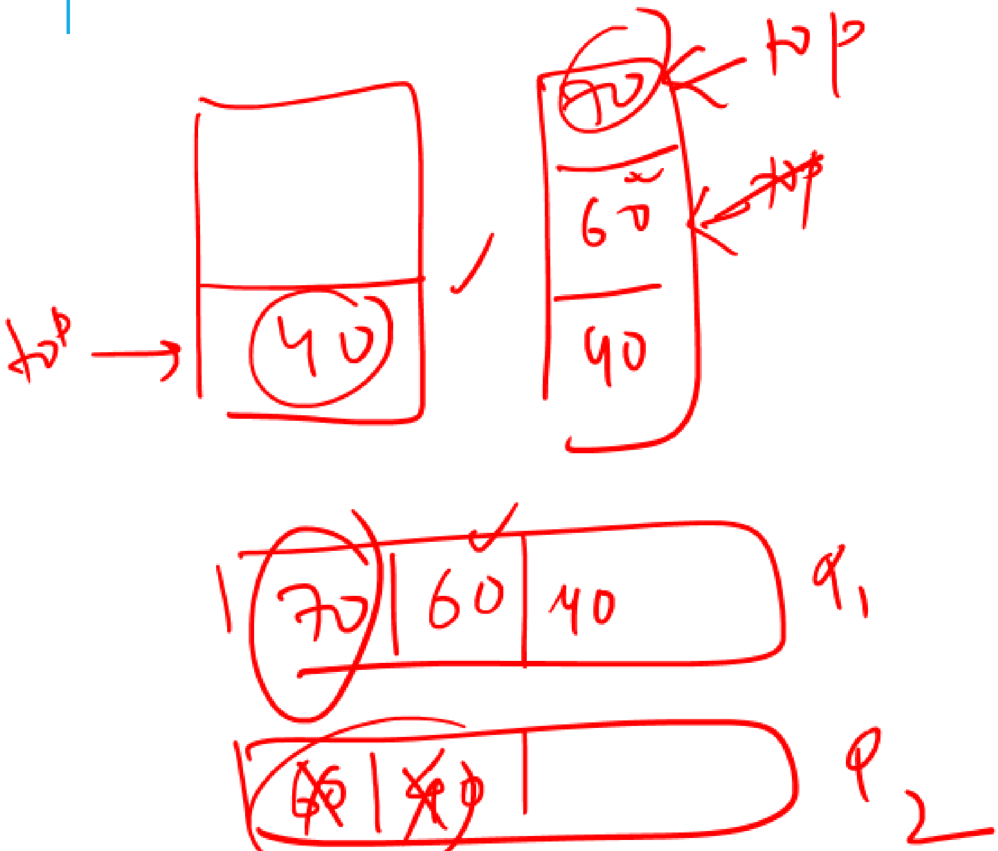
(class structure for Linked queue)

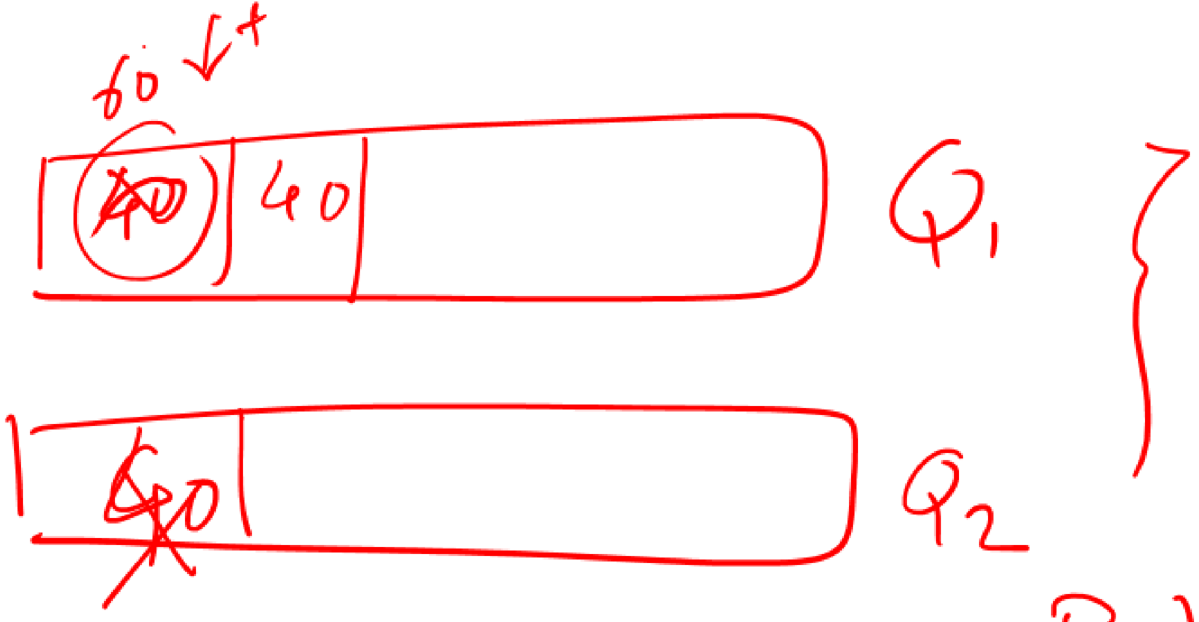
How will you do enqueue and dequeue?



Lab 7: next week

IMPLEMENTING STACK USING QUEUES





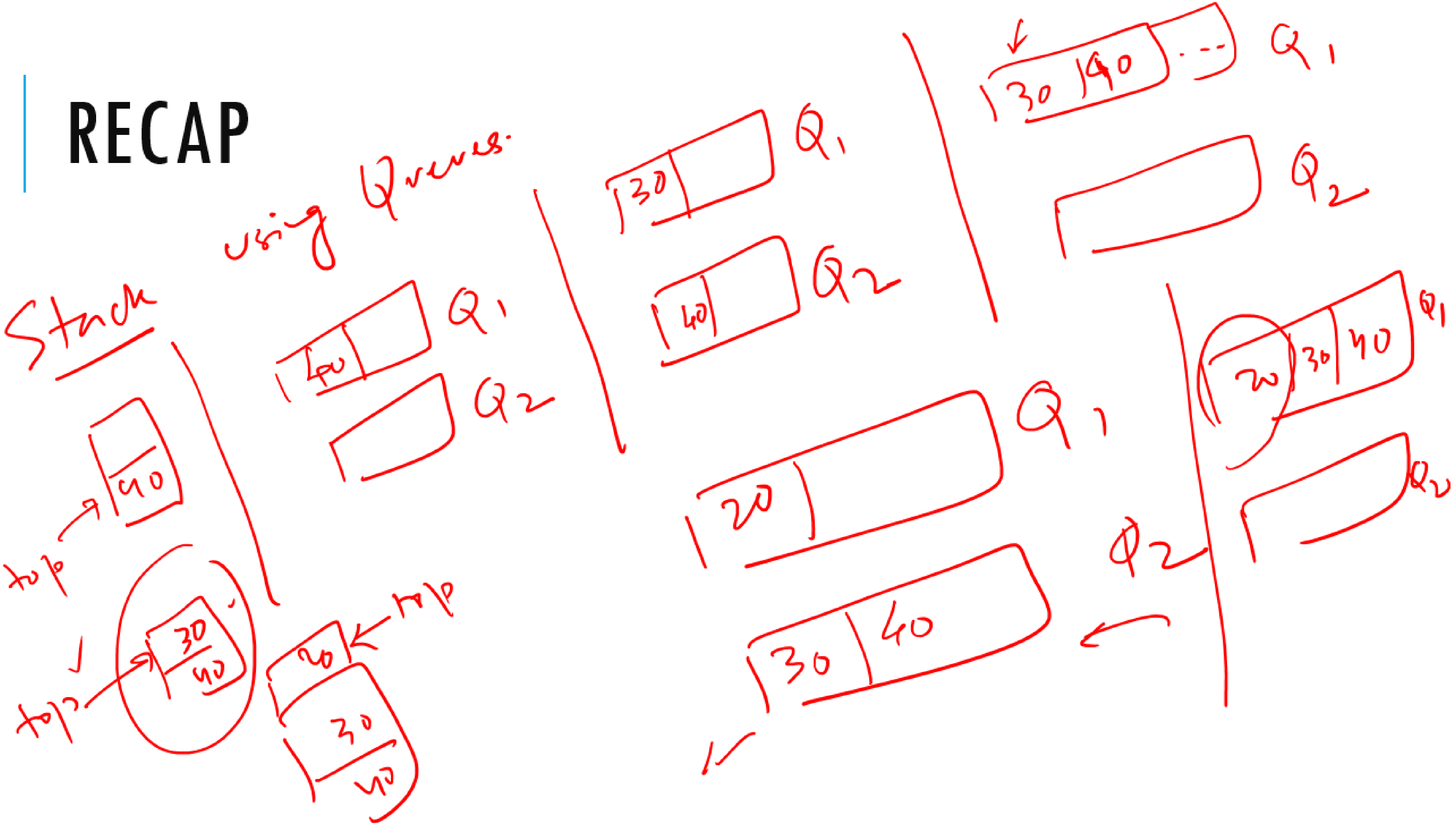
Lab 7: next week

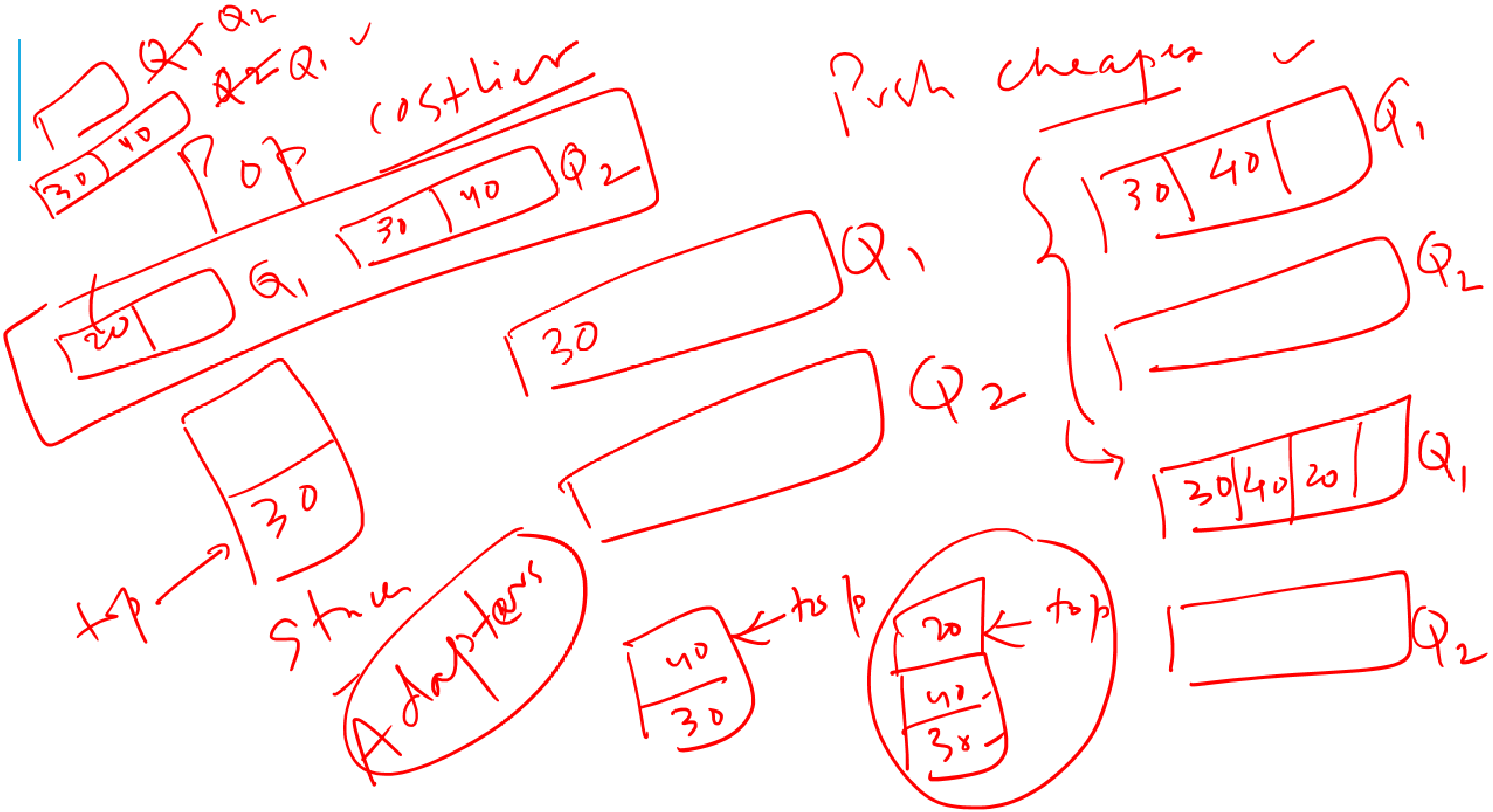
Push
O(n)

Costly.

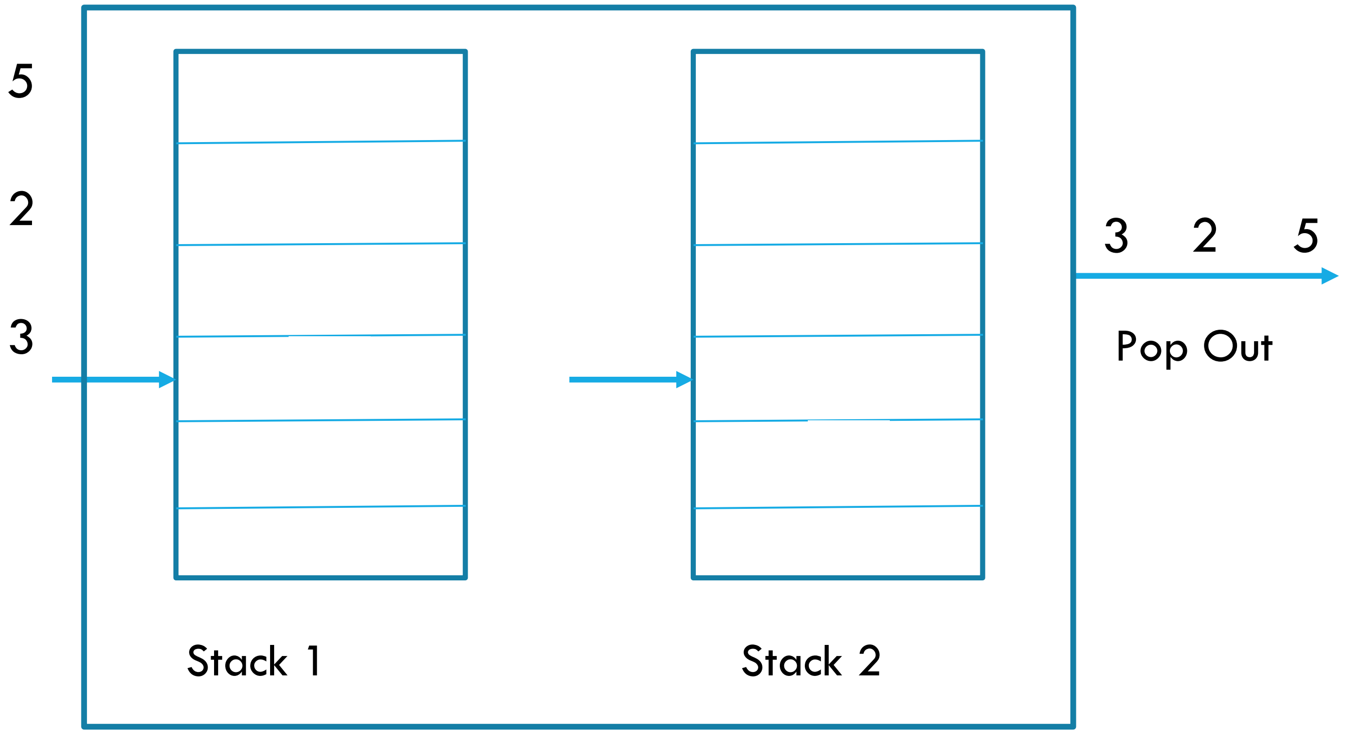
Pop
O(1)

RECAP


Stack
using Queues:
top
40
top
30
40
top
20
30
40
Q1
Q2
40
Q1
30
Q2
40
Q1
30
40
Q2
Q1
20
Q2
30
40
Q1
20
30
40
Q2


~~Q1~~ Q2
~~Q2~~ Q1
Pop costlier
30 | 40
Q2
30 | 40
Q1
20
30
Q1
Q2
30
top
Stack
Adapters
40
30
top
20
40
30
top
Push cheaper
30 | 40
Q1
Q2
30 | 40 | 20
Q1
Q2

A QUEUE USING TWO STACKS



Enqueue: push the elements into the Stack 1.

Dequeue: push all the elements from Stack 1 into Stack 2, and then pop from Stack 2.

Which operation is costly here?

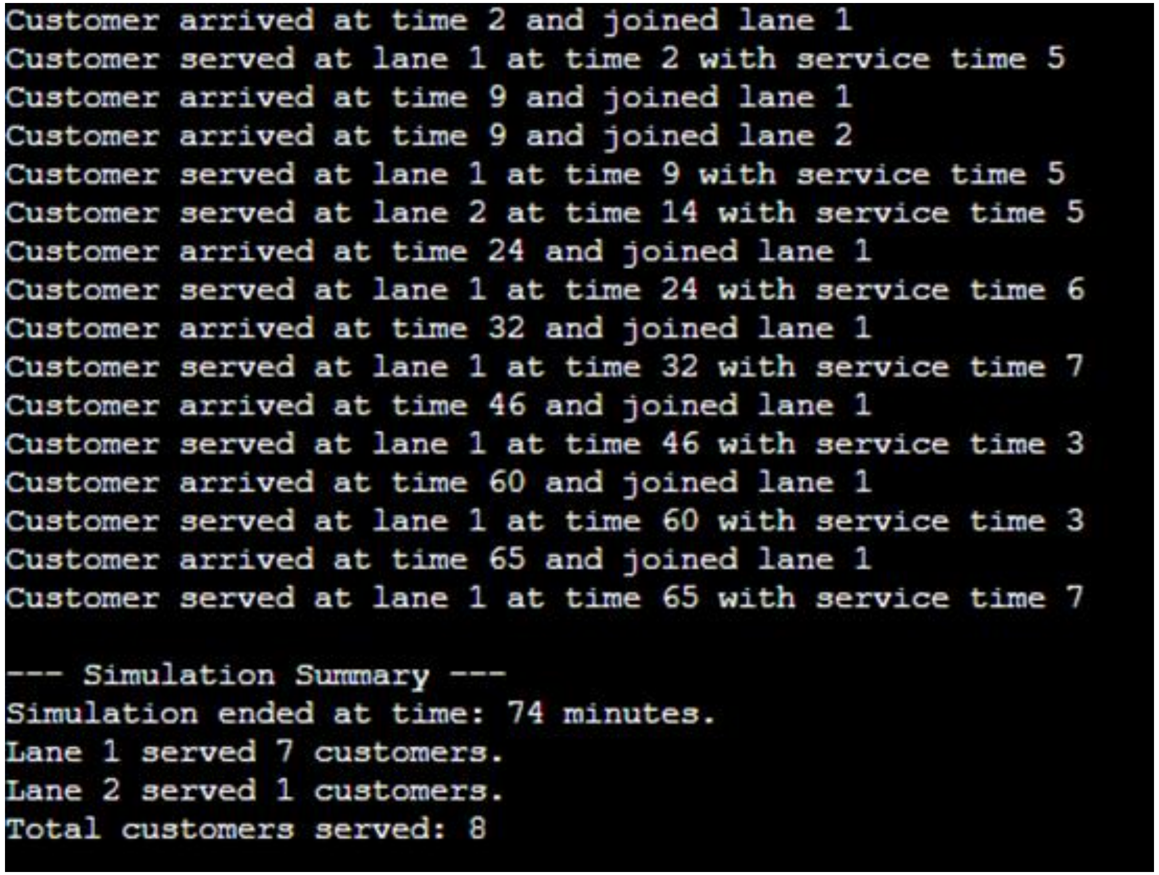
```
#include <queue>   using std::queue;  queue<int>myQueue; push(e); pop(); front(); back(); size(); empty;
```

C++ STL QUEUE (SUPERMARKET CHECKOUT)

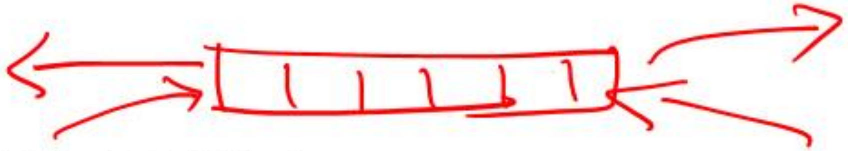
```
while (current_time < simulation_time) {
    //Check for new customer arrival
    if (random() < arrival_rate)
        new_customer = {arrival_time: current_time, service_time: random(1, 5)};
        shortest_lane = lane with the shortest queue;
        lanes[shortest_lane].enqueue(new_customer);

    //Process each checkout lane
    for each (lane ∈  lanes)
      if (lane is NOT empty && cashier_available[lane] <= current_time)
          customer = lane.dequeue();
          cashier_available[lane] = current_time + customer.service_time;
          waiting_time = current_time - customer.arrival_time;
          total_customers_served += 1;
          total_waiting_time += waiting_time;

    // Increment time
    current_time += 1;
}
```

Lab 7: next week

Output:

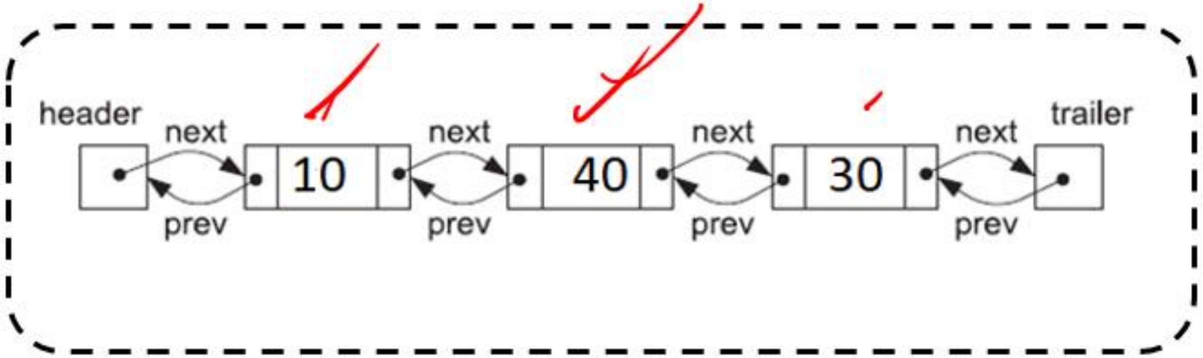


```
Customer arrived at time 2 and joined lane 1
Customer served at lane 1 at time 2 with service time 5
Customer arrived at time 9 and joined lane 1
Customer arrived at time 9 and joined lane 2
Customer served at lane 1 at time 9 with service time 5
Customer served at lane 2 at time 14 with service time 5
Customer arrived at time 24 and joined lane 1
Customer served at lane 1 at time 24 with service time 6
Customer arrived at time 32 and joined lane 1
Customer served at lane 1 at time 32 with service time 7
Customer arrived at time 46 and joined lane 1
Customer served at lane 1 at time 46 with service time 3
Customer arrived at time 60 and joined lane 1
Customer served at lane 1 at time 60 with service time 3
Customer arrived at time 65 and joined lane 1
Customer served at lane 1 at time 65 with service time 7

--- Simulation Summary ---
Simulation ended at time: 74 minutes.
Lane 1 served 7 customers.
Lane 2 served 1 customers.
Total customers served: 8
```

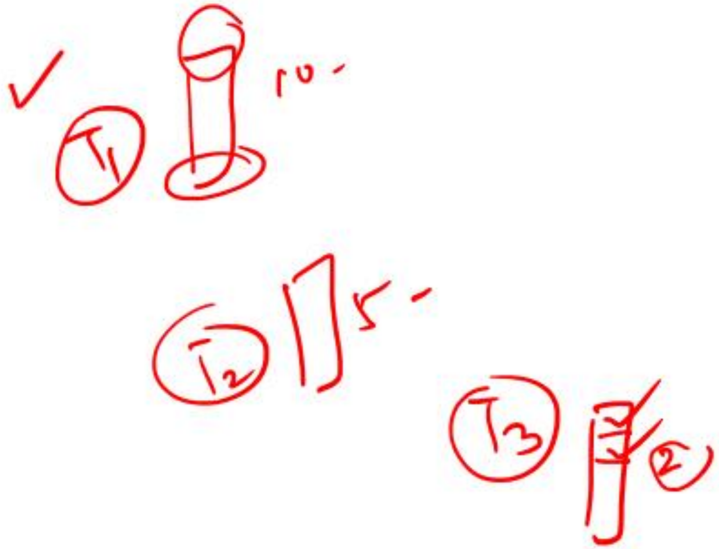


DOUBLE-ENDED QUEUE ADT: DEQUE

list



```
#include <deque>

deque<int>myDeque;

myDeque.push_front(10);

myDeque.push_back(40);

myDeque.push_back(30);

myDeque.pop_front();
```



"aba" ✓ a, b, a

C++ STL DEQUE

b, a

a

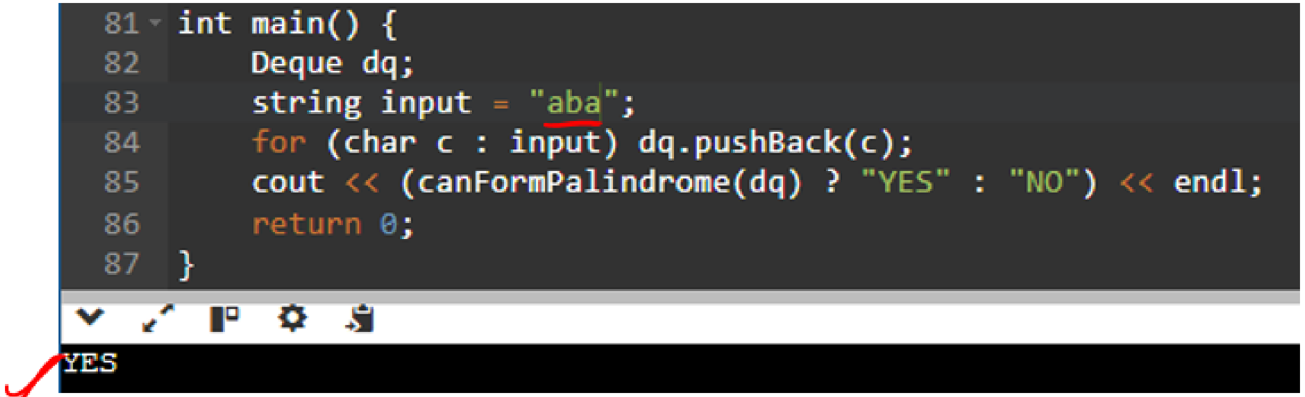
"a"

"ab"

madam c ✓

a a c d m m

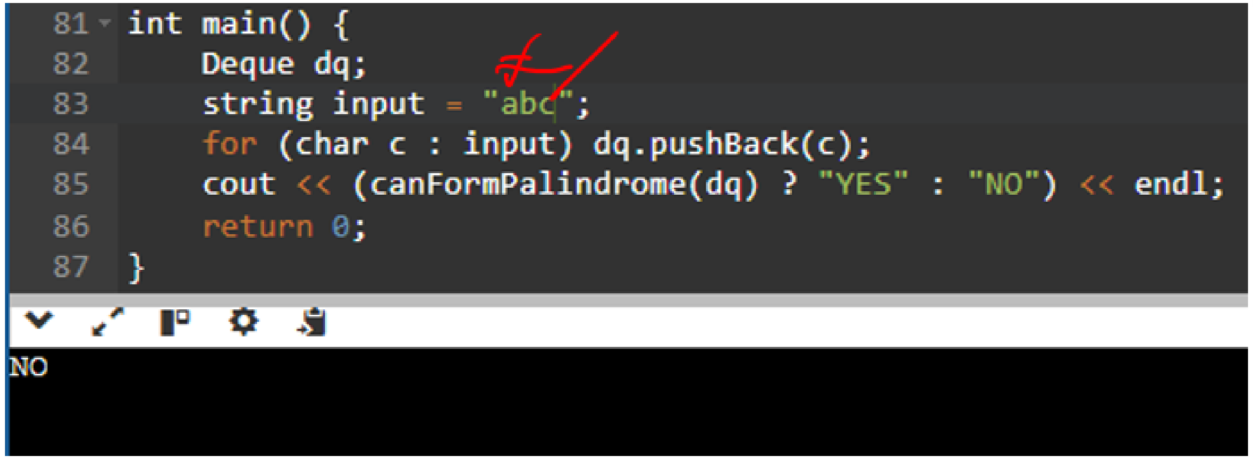
bool canFormPalindrome(Deque& dq, string s="")



```
int main() {
    Deque dq;
    string input = "aba";
    for (char c : input) dq.pushBack(c);
    cout << (canFormPalindrome(dq) ? "YES" : "NO") << endl;
    return 0;
}
```

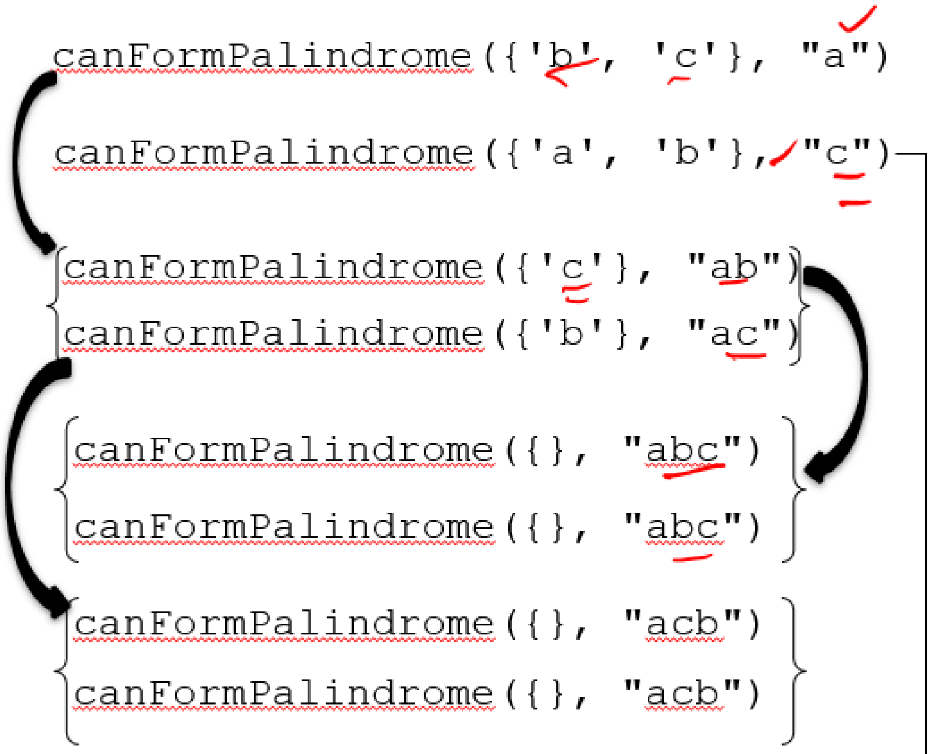
```
YES
```

✓



```
int main() {
    Deque dq;
    string input = "abc";
    for (char c : input) dq.pushBack(c);
    cout << (canFormPalindrome(dq) ? "YES" : "NO") << endl;
    return 0;
}
```

```
NO
```



```
canFormPalindrome({'b', 'c'}, "a")

canFormPalindrome({'a', 'b'}, "c")

canFormPalindrome({'c'}, "ab")
canFormPalindrome({'b'}, "ac")

canFormPalindrome({}, "abc")
canFormPalindrome({}, "abc")

canFormPalindrome({}, "acb")
canFormPalindrome({}, "acb")
```

As all paths fail, backtrack to Step 1 and try the second option: ‘c’ first.

…

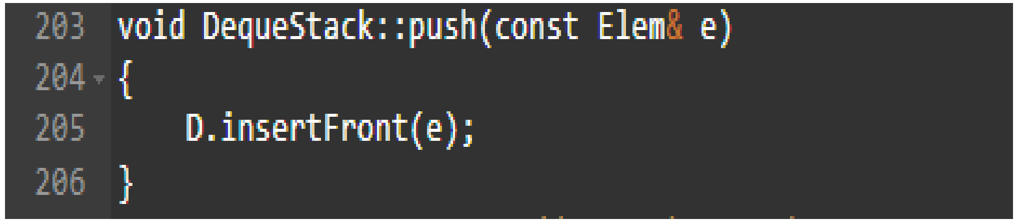
Lab 7: next week

ADAPTER DESIGN PATTERN

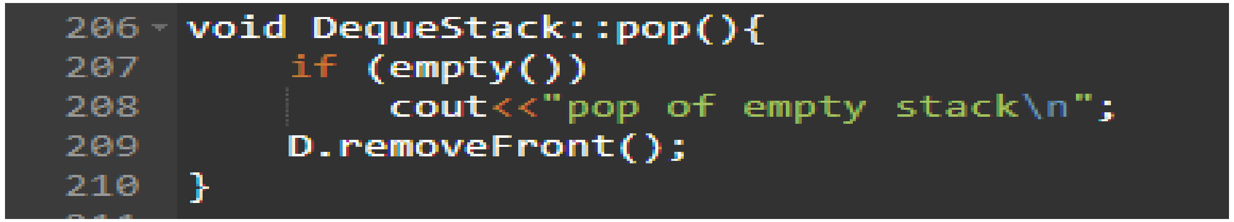
Deque	Stack	Queue
insertFront()	-	-
insertBack()	Push()	Enqueue()
removeFront()	-	Dequeue()
removeBack()	Pop()	-
Size()	Size()	Size()
Empty()	Empty()	Empty()

```
typedef string Elem;
DequeStack {// stack as deq
  public:
       DequeStack();
       int size() const;
       bool empty() const;
       const Elem& top();
       void push(const Elem& e);
       void pop();
  private:
       LinkedDeque D;
};
```

```
// A queue using a deque:
template<typename E>
void Queue<E>::enqueue(E elem)
{
    dq.insertBack(elem);
}
template<typename E>
void Queue<E>::dequeue() {
  if(dq.empty())
     throw “Queue Underflows!”;
  dq.removeFront();
}
```



```
void DequeStack::push(const Elem& e)
{
    D.insertFront(e);
}
```



```
void DequeStack::pop(){
    if (empty())
        cout<<"pop of empty stack\n";
    D.removeFront();
}
```

THANK YOU!

Next Class: Vectors, Amortization, and Iterators …