4) Lecture 5-6 (Abstract Data Type, Stack, Queue)

07 August 2024 11:35

Elementry data Structure

Abstract Data type: is a data type that focues on what it does by ignoring how it does.

" Facilates some functionalities while abstracting out the details how actual implemantation is done".

Stack, Queue, linked

[{ (

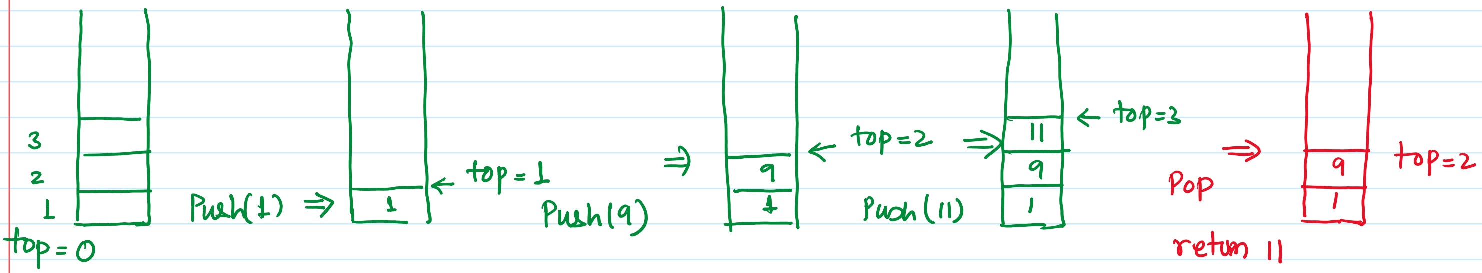
Checking if an arithematic expression has balanced parantheis

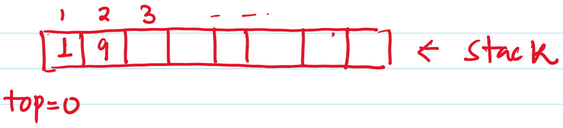
$[(a+b)*c]$ – ✓ ( )

$[(a+b)-d)\}$ ✗

Stack Data structure: Last In First Out (LIFO)

Push Pop top





(i) top = 0 ⇒ the stack is empty

(ii) If we attempt to 'pop' on empty stack, get error - "underflow"

(iii) If we attempt to push an element in full stack, get error - "overflow"

```
Stack_empty(stack, top)
{
    if (top ==0)
        return TRUE
    else
        return FALSE
}
```

Time complexity of push/pop = $\Theta(1)$

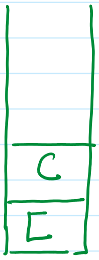
```
Push(stack, top, item)
{
    if stack_full(stack, top)
        return overflow

    top ← top+1
    Stack[top] ← Item
}
```

```
Pop(Stack, top)
{ if stack_empty(Stack, top)
        retun "error Underflow"
    top ← top-1
    return Stack[top+1]
}
```

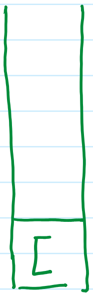
Solution of Balanced parantheses Using Stack

}

str = [(a+b) * c]   length of string n



Check '(' is same as)



- Traverse the arithemetic expression
- If the current character in the expression is an opening bracket
  (or { or [
  Push it on the stack
- If the current character is closing bracket
) or } or]
  Pop a character from stack

Pop a character from stack

return False if popped char doesnt match with current char

- If the stack is empty and entire expression is traversed

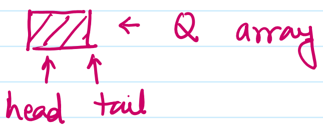
return TRUE.

Time complexity = $\theta(n)$

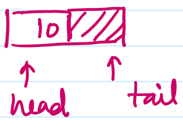
Queue

Enquene - inserting an element in Queue (at tail position)

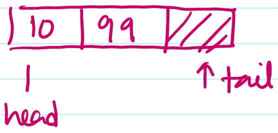
Dequene - deleting an element from Queue (at head position)



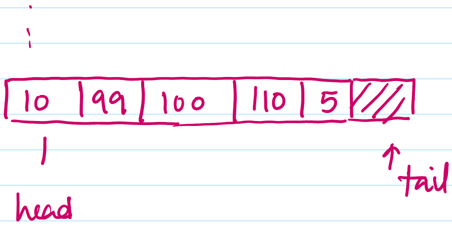
Enquene (Q, 10)



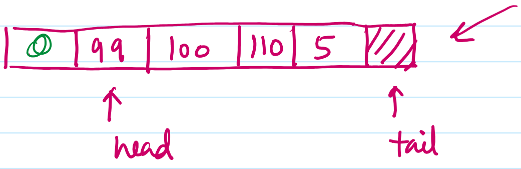
Enqueu (Q, 99)



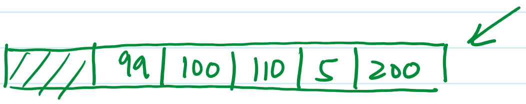
Enquene (Q, 5)

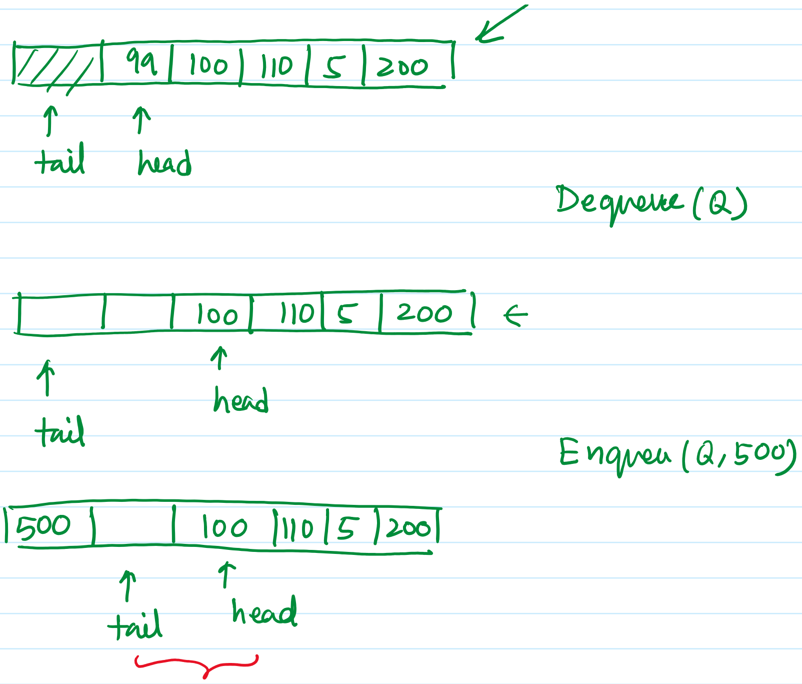


Dequene (Q)



Enquene (Q, 200)





Q.head ← index location of head

Q.tail ← " " tail

Q.length ← Space assigned for whole array

```
QUEUE-Full (Q)
{
    if ((Q.head ==1) and (Q.tail == Q.length)) OR ( Q.tail+1 == Q.head)
        {
            return TRUE
        }
}
```

```
QUEUE-EMPTY (Q)
{
    if (Q.head == Q.tail)
        {
            return TRUE
        }
}
```

↓ item that needs to be enqueued

```
ENQUEUE (Q, x)
{
```

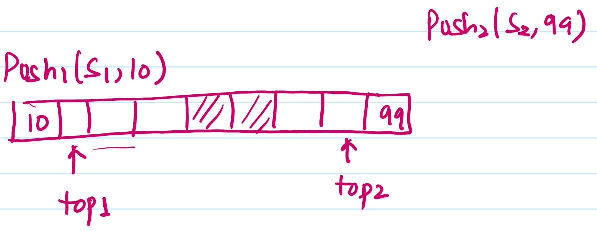
```
ENQUEUE (Q, x)
{
        if ( QUEUE-Full(Q))
            retorn "Overflow"
        }

        Q[Q.tail] ← x

        Q.tail ← (Q.tail+1) mod (Q.length) ⇐  [ if (Q.length == Q.tail)
                                                      Q.tail ← 1
                                                  }
                                                  else
                                                      Q.tail ++
    }
```

```
DEQUEUE (Q)
    {
        if { (QUEUE-EMPTY(Q))
            retorn "Underflow"
        }
        x ← Q[Q.head]

        Q.head ← (Q.head +1) mod (Q.length)

        retorn x
    }
```

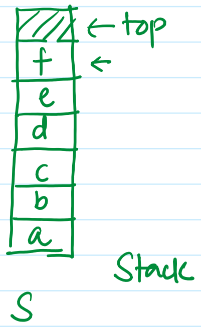
Implementing two stack in one array

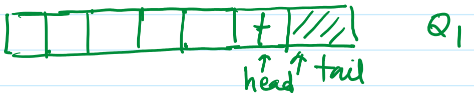
Implement two stack S_1 & S_2

$Push1(S_1, x)$	$Pop_1(S_1)$
$Push2(S_2, x)$	$Pop2(S_2)$



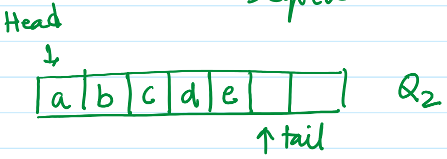
Question   Implementing stack using two Queue





Enqueue(Q, f)

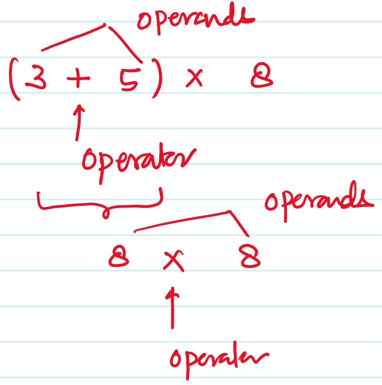
Dequeue



```
Push(S, x)
{
    Enqueue(Q1, x)
}
```

```
Pop(S)
{
    while Q1 is not empty
        if (head+1 == tail)
        {
            x = dequeue(Q1)
            return x
        }
        else
        {
            x = dequeue(Q1)
            Enqueue(Q2, x)
        }
}
```

Evaluating Arithmetic Expression



infix representation

operand 1 operator operand 2

Postfix notation operand 1 operand 2 operator

$\underbrace{(3+5)}\times 8 \Rightarrow 3\ 5 + 8 \times$

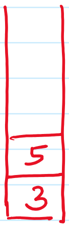
$3 + (5 \times 8) \Rightarrow 3\ 5\ 8 \times +$

$\underline{5 + ((\underline{1+2}) \times 4)} - 3 \Rightarrow 5\ 1\ 2 + 4 \times + 3 -$

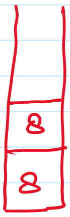
Given arithmetic expression in postfix form compute its value.

$3\ 5 + 8 \times$

↑



$3 + 5 = 8$



$8 \times 8 = 64$

```
while there are input token left to read
    if token is number push it on stack
    if token is operator
        val2 ← pop(stack)
        Val1 ← pop (stack)
        Val ← val1 operator val2
        Push (stack, val)

return Pop (stack)
```