

Stack ADT=Abstract Data type **LAST IN FIRST OUT**

Complexity	
Pushing	$O(1)$
Popping	$O(1)$
Peeking	$O(1)$
Searching	$O(n)$
Size	$O(1)$

Stack ADT

↳ features/operations ✓
implementation ✗

Push ekler pop siler top ise listenint en basindaki elemani dondurur is empty stacking bos olup olmadigini sorgular. False olarak True olarak geri doner. Buradaki butun operasyonlar constant timedir.

Stack ADT

A list with the restriction that insertion and deletion can be performed only from one end, called the top.

Operations

(1) Push(x)
(2) Pop()
(3) Top()
(4) IsEmpty()

} Constant

Stack ADT (LIFO)

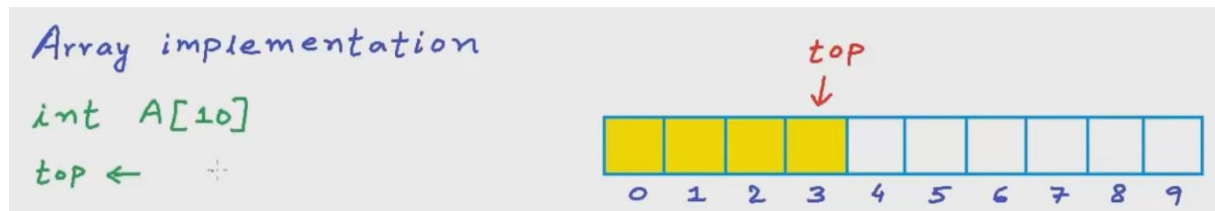
Applications

- Function Calls/ Recursion
- undo in an editor
- Balanced Parentheses

Array implementation of stacks

We can implement stacks using Arrays:

A=Arrays B=Linked lists



I am going to use this array to store stack. Any point some part of this array starting index zero till an index marked as 'top' will be my stack. We can create a variable named top to store index of top of stack.

For an empty stack top is set as -1 Burada push function ile top yerine top +1 ekliyoruz.

Push 2 yaptigimizi dusunelim; her push yaptiginda top bir sonraki sayiya geciyor mantik bu

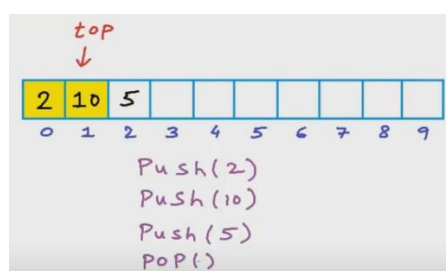
```
Push(x)
{
    top ← top + 1
    A[top] ← x
}
```

Push için

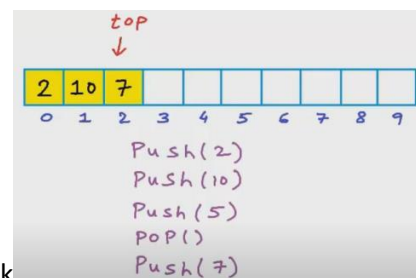
```
Pop()
{
    top ← top - 1
}
```

pop için

Pop kullandıktan sonra önceden 5 sayisinde olan topumuz bir önceki sayiya doner



Simdi yeniden push yaparsak



Pop ve push functionu constant time aliyor Time complexity O(1) Arraye yalnızca arrayda boş yer kaldıysa push yapabiliriz. Stacking tüm arrayi tükettiği durumlarda top arrayin en sonunda olacaktır.



A further push is not possible because it will result an overflow this is one limit based array to avoid an overflow we can always create an array for that we will have to be reasonably sure that stack will not grow Bu durumu onlemek icinde ornegin kod yazarken push fonctionun arrayin exhausted olup olmadigini ogrendini yazan bir kod yazabiliriz.

Bunun yerine dynamic arrayi kullanabiliriz. Overflow durumlarında yeni daha büyük bir array yaratabiliriz diger stackin icerigini bu yeni arraya aktararak bunu yapariz.

Overflow
↳ create a larger array. copy all elements in new array.
Cost - $O(n)$
where n = no. of elements in stack

Burada onemli olan bize bunun time complexity maliyeti $O(n)$ olacadir. Means time taken to copy element from last array to new array would be proportional to number of arrays in stack . or the size of the smaller array.because stack will occupy the stack of whole array. There must be strategy to decide

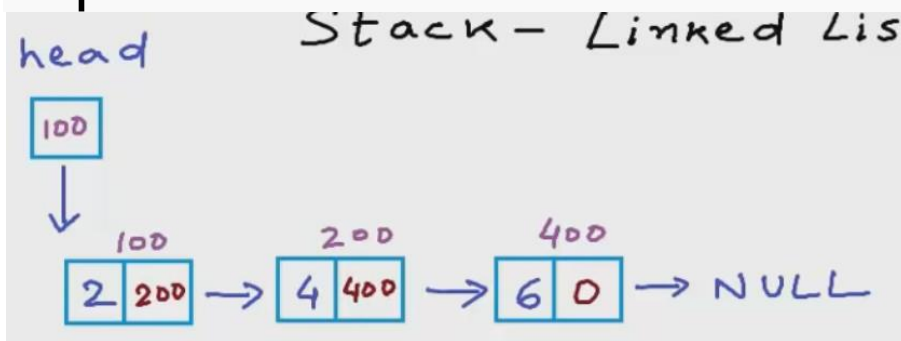
size of array. Bunun oluru onceki arrayimizin 2 kati bir array yaratmaktir. Ornegin onceki array 4 ise yeni olan 8 o da yetmediyse 16 tane gibi. Belli bir senaryoda push constant time iken yeni bir array eklenmesi ve bunlarin hepsinin kopyalanmasi gibi $O(n)$ yani lineer timedir.

Overflow twice the
↓ smaller
↳ create a larger array. copy all elements.
Push - $O(1)$ - best case
 $O(n)$ - Worst case
 $O(1)$ - Average case
c.n
↑
 $O(n)$ for n pushes

En iyi durumda constant time kotu durumda n time. Fakat yine de $O(1)$ avarage case ile ile bulabilriiz If we will calculate time taken for n pushes then it will be proportional N remember n is the number of elements in the stack. Burada sari isaretli $O(n)$ su anlama geliyor time taken will be very close to some constant times simple words time taken will be proportional to n

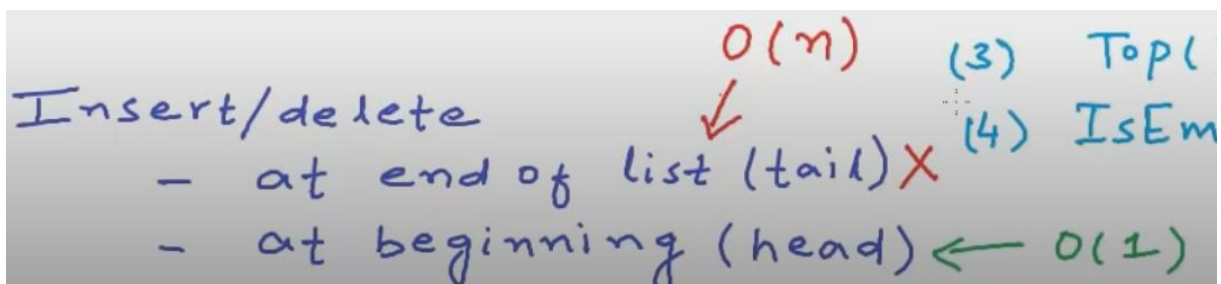
Data Structures: Linked List

implementation of stacks

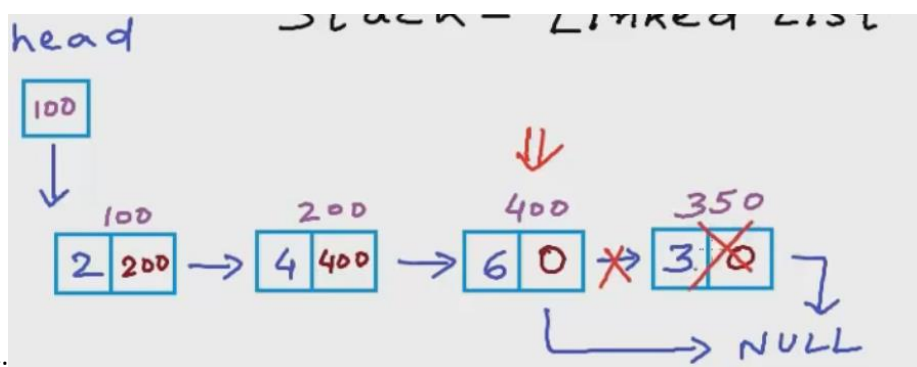


Linked listlerde 1 bolge data 1 bolge diger node'un adresini iceriyor. unlike arrays linked lists are not if fixed size and element in linked list are not stored in one contiguous block of memory

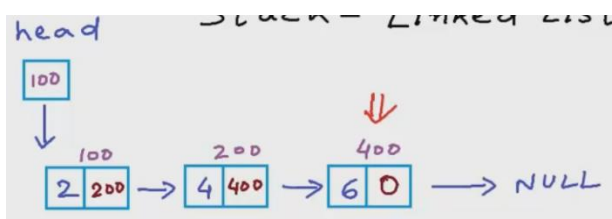
To insert element in linked list for a stack we want that insertion and deletion must always happen from the same end we can use our linked list as stack if we always insert and delete at same end. we have two options we can delete or add at the end of the list ya da basindan ekleyip cikarabiliruz.



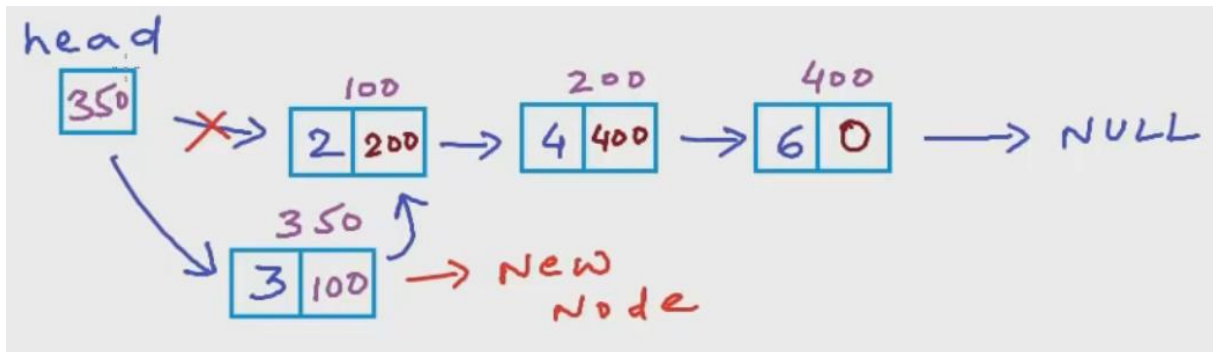
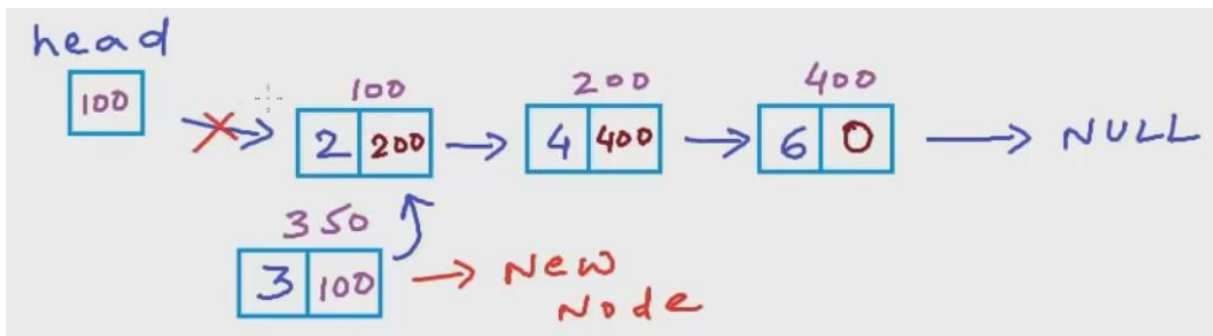
***Inserting a node at end of linked list is not constant time. Hem insertion hem deletion to at end of list is $O(n)$ yani linear time. Stack icin elbette constant time ama mevzu linked listlere geldiginde bunun linear time $O(n)$



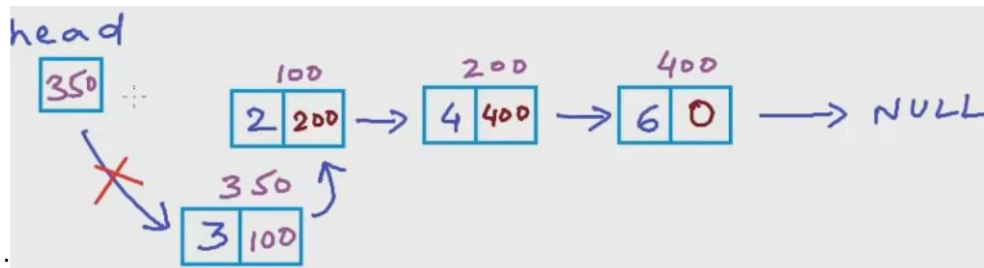
Deleting;



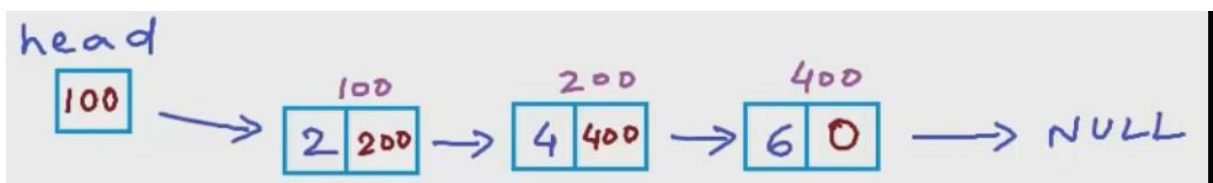
Bunun yanında baslangicta insert ve delete $O(1)$ yani constant time.



Yeni node ekliyoruz birinci resimde head ile ikinci node baglantisini kes alta ekledigimiz yeni node' a eski head sayimiz 100 yaz sonrasinda yen inode sayimiz 350'yi 100' un yeyirne yaz.



Delete a node ;



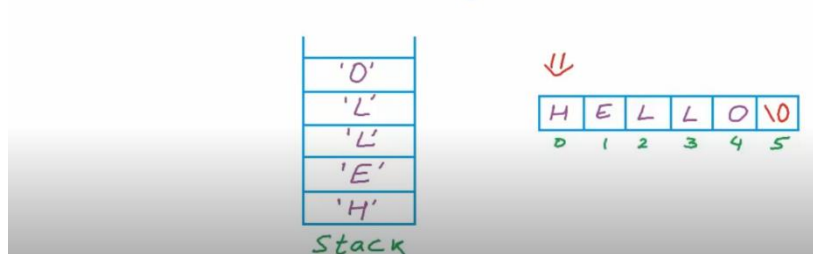
Linked list implementation of stack is pretty straightforward all we need is insert and delete beginning so head of linked list is top of stack.

Reverse a string or linked list using stack.

A stack can be used to reverse a list or collection. Or simply traverse a list or collection.

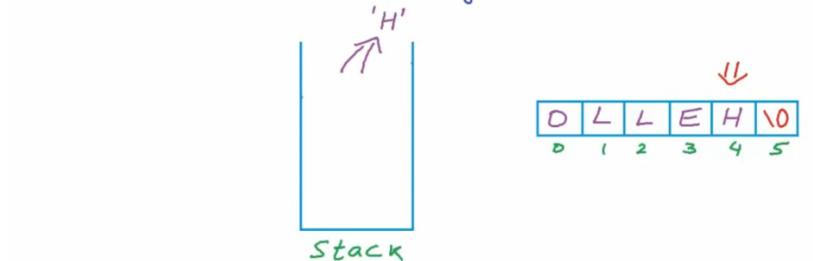
Problem; Reverse a string

Problem: Reverse a string



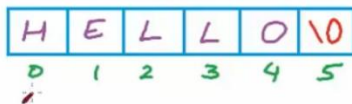
Popu kullanarak simdi her harfi alalim

Problem: Reverse a string



Using stack to reverse

Problem: Reverse a string



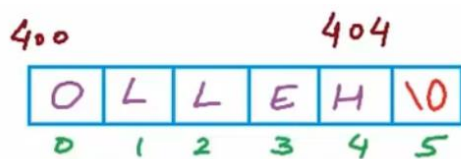
Time-Complexity
 $= O(n)$

Space-Complexity
 $= O(n)$

```
void Reverse(char *C,int n)
{
    stack<char> S;
    //loop for push
    for(int i=0;i<n;i++){
        S.push(C[i]);
    }
    //loop for pop
    for(int i =0;i<n;i++){
        C[i] = S.top();
        S.pop();
    }
}
```

Sous-titres

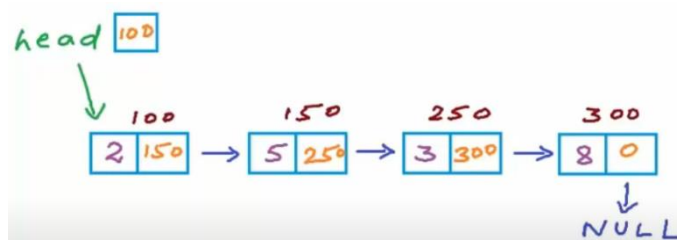
Problem: Reverse a Linked List



Bu yandaki Array ornegimiz.

A[0] or A[4]
↓
constant time

Linked liste bakalim.



Iterative Solution

Time - $O(n)$

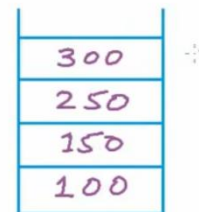
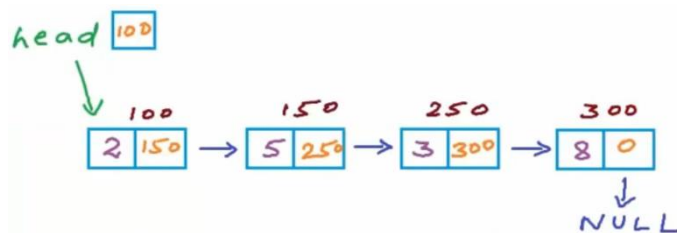
Space - $O(1)$

Recursive Solution
(Implicit Stack)

Time - $O(n)$

Space - $O(n)$

Now let's see how the can use an explicit stack to solve this problem.



`stack<struct Node*> S;`

Check for balanced parentheses using stack

Expression	Balanced? () or {} or []
$(A+B)$	Yes
$\{(A+B)+(C+D)\}$	Yes
$\{(x+y)*(z)$	NO
$[2+3]+(A)]$	NO
$\{a+z\}$	NO

myc

Bu expressionlar balanced mi evet cunku ornegin sozluk isaretiyle acilan sozluk isqretiyle kapatilmis yine tuple isaretiyle olan tuple ile kapatilmis Fakat ucuncu sirada sozluk kapatma isareti olmadigi icin balanced degildir.

Compiler burada parentez isaretlerini vs kontrol ediyor.

Expression	Balanced? () or {} or []
$) ($	NO
$[()]$	Yes
$[()]$	NO
$[() ()]$	
<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;"> \uparrow \Rightarrow Last opened first closed </div> </div>	

Solution:

- \Rightarrow scan from left to right
- \Rightarrow if opening symbol,
add it to a list
- \Rightarrow if closing symbol,
remove last opening
symbol in list

sed

Infix, Prefix and Postfix

Infix notation

Infix, Postfix, Prefix

Evaluation of expressions

$$\left. \begin{array}{l} 2 + 3 \\ A - B \\ (P * 2) \end{array} \right\} \text{ <operand> <operator> <operand> }$$

Operand \rightarrow object on which operation is performed

Burada operator yani toplama olur cikarma vs olur ortada oluyor sayilara veya variableslar ise saginda veya solunda.

Infix

$$\begin{array}{l} 2 + 3 = 5 \\ 4 + 6 * 2 = 4 + 12 = 16 \\ \{(2 * 6) / 2\} - (3 + 7) = -4 \end{array}$$

Order of operation

- 1) Parentheses $() \{ \} []$
- 2) Exponents (right to left)
- 3) Multiplication and division (left to right)
- 4) Addition and Subtraction (left to right)

Prefix notation(polish notation)

Infix, Postfix, Prefix

Prefix
(Polish notation)

$$\text{<operator> <operand> <operand>}$$

Infix

Prefix

$$\begin{array}{ll} 2 + 3 & + 2 3 \\ P - Q & - P Q \\ A + B * C & + A * B C \end{array}$$

Bu da polisj notation operatorler toplama carpma gibi onunde yer aliyur.

Postfix notation: Bu 1950'li yıllarda computer scientistler taraifndan önerildi. Zamandan ve yerden kazandigi için icat edilmistir.

<p>Postfix (Reverse polish notation)</p> <p><operand> <operand> <operator></p>			
Infix	Prefix	Postfix	$a + (b * c)$
$2 + 3$	$+ 2 3$	$2 3 +$	\Downarrow
$p - q$	$- p q$	$p q -$	$a + (b c *)$
$a + b * c$	$+ a * b c$	$a b c * +$	\Downarrow
			$+$