

Data Structures and Algorithms

Mohammad GANJTABESH

*Department of Computer Science,
School of Mathematics, Statistics and Computer Science,
University of Tehran,
Tehran, Iran.*

mgtabesh@ut.ac.ir



*Dept. of Computer Science
University of Tehran*

Stack

Last In First Out
(LIFO)



Data Structures and Algorithms
Undergraduate course



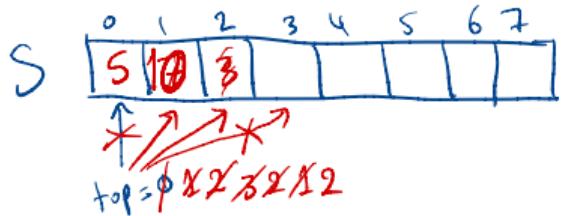
Mohammad GANJTABESH
mgtabesh@ut.ac.ir



Dept. of Computer Science
University of Tehran

Basics

Using Array for stack

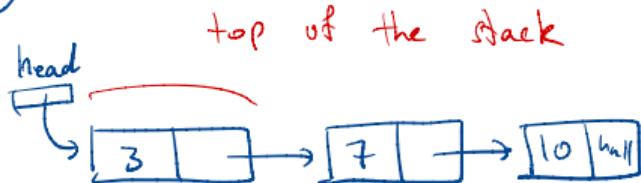


- push(5) \rightarrow initial
push(7)
push(3)
 $3 \leftarrow pop()$ \rightarrow after
 $7 \leftarrow pop()$ \rightarrow after
push(10)
 $10 \leftarrow Top()$ \rightarrow after

AddLast $\rightarrow O(1)$

RemoveLast $\rightarrow O(1)$

Using Linked List :



push(5) \equiv AddFirst(5)

pop() \equiv RemoveFirst()

Stack implementation: Array vs. Linked List

Array

```
class Stack {  
    int *S;  
    int capacity;  
    int top;  
  
    stack (int n){  
        capacity = n  
        S = new int [n];  
    }  
    top = 0  
}
```

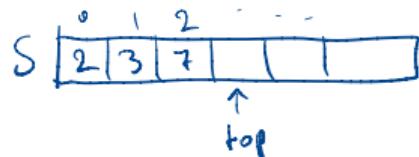
- Fixed length
- Unlimited
- constant time operations
- constant time operations

Linked List

```
class Stack {  
    LinkedList L;  
  
    stack (){  
        L = new LinkedList();  
    }  
    } // methods
```

Array :

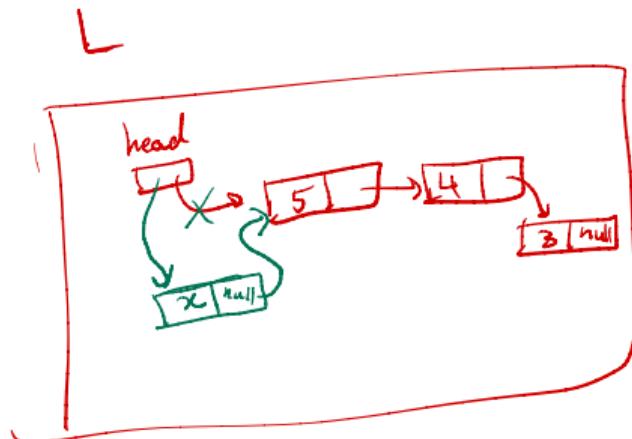
```
void push (int x){  
    if (top < capacity)  
        S[top] = x;  
    top++;  
}  
else  
    throw exception ("overflow");  
}
```



Stack: Push operation

Linked List:

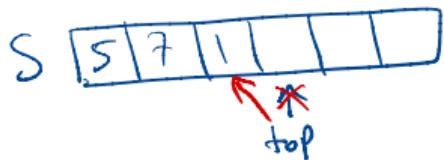
```
void push (int x){  
    L.AddFirst (x);  
}
```



Stack: Pop operation

Array:

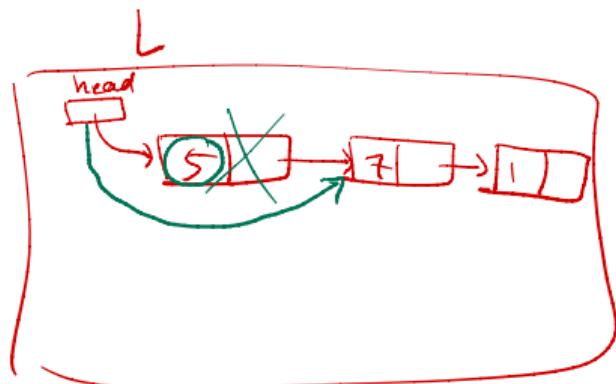
```
int pop () {  
    if (top >= 0){  
        top --;  
        return (S[top]);  
    }  
    else  
        throw exception ("Underflow");  
}
```



Linked List

```
int pop () {  
    return (L.RemoveFirst());  
}
```

3



Stack: Top operation

Array :

```
int Top () {  
    if (top > 0) {  
        return (S[top - 1]);  
    }  
    else  
        //exception;  
}
```

Linked List :

```
int Top () {  
    return (PeekFirst());  
}
```



Applications

- ذخیره سازی ادرس هر یکت همچنین فراخوان را باعث می کند
- ذخیره اندامان را باعث می کند
- BackTracking - 3
- (DFS) باعث می کند که راوند را بازگرداند
- (BFS) اندامان را بازگرداند

infix

Evaluating mathematical expressions

$$"2+3" = 5$$

prefix

$$\begin{array}{r} + 23 \\ \hline 5 \end{array}$$

postfix

$$\begin{array}{r} 23 + \\ \hline 5 \end{array}$$

$$"(2+3)*4" = 20$$

$$\begin{array}{r} * + 23 4 \\ \hline 5 \\ 20 \end{array}$$

$$\begin{array}{r} 23 + 4 * \\ \hline 5 \\ 20 \end{array}$$

infix :



()

* ÷

+ -

prefix :



postfix :



Evaluating mathematical expressions

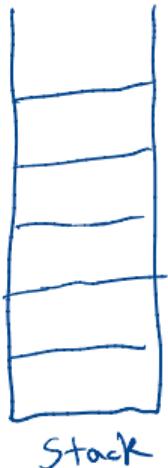
infix postfix

$$4 * (2+3) / (6-4) + 1$$

$$4 * (2+3) / (6-4) + 1$$

⇒ post fix

$$4 \underline{2} \underline{3} + * \underline{6} \underline{4} - / \underline{1} \underline{1} +$$

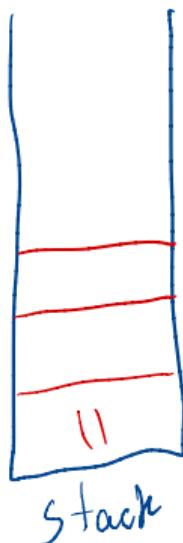


- مداخل تبدیل به حروف پسنه:
- عبارت infix را از پایه راست بخوانیم.
 - آن عدد دیده شد، آن را پسنه.
 - آن عملگر دیده شد، اینها را داخل پسنه کردند و اولویت بزرگتر را باساوی کردند.
 - آن از پسنه خارج شد، و نیویمه سپس آن عملگر را در پسنه و کرد.
 - آن باتر بود دیده شد، آن را در پسنه و کرد.
 - آن باتر بوده دیده شد، تا اوین باتر بود داشت پسنه، عملگری موجود را خارج و نیویمه.
 - از انتها عبارت ساده شد، پسنه را خالی کرد.

Evaluating mathematical expressions

نحوه ارزیابی عبارت های پس پردازی : 5

postfix : $4 \boxed{2} 3 + * 6 4 - / 1 +$
 $4 2 3 + * 6 \boxed{4} - / 1 +$



اعداد postfix داریم که را از پس پردازی بگیریم.

- آن عدد دیگر نیست، آن را در پیش وارد کنیم.
- آن عمل دیگر نیست، دو عنصر بالا را با هم جمع کنید.
- و مبتدا را رسماً اعمال کرد و نتیجه را دوباره در پیش قرار دهیم.
- آخر، انتها عبارت رسماً است. نتیجه را پیش وارد کرد.

$$6 - 4 \rightsquigarrow 6 4 -$$