

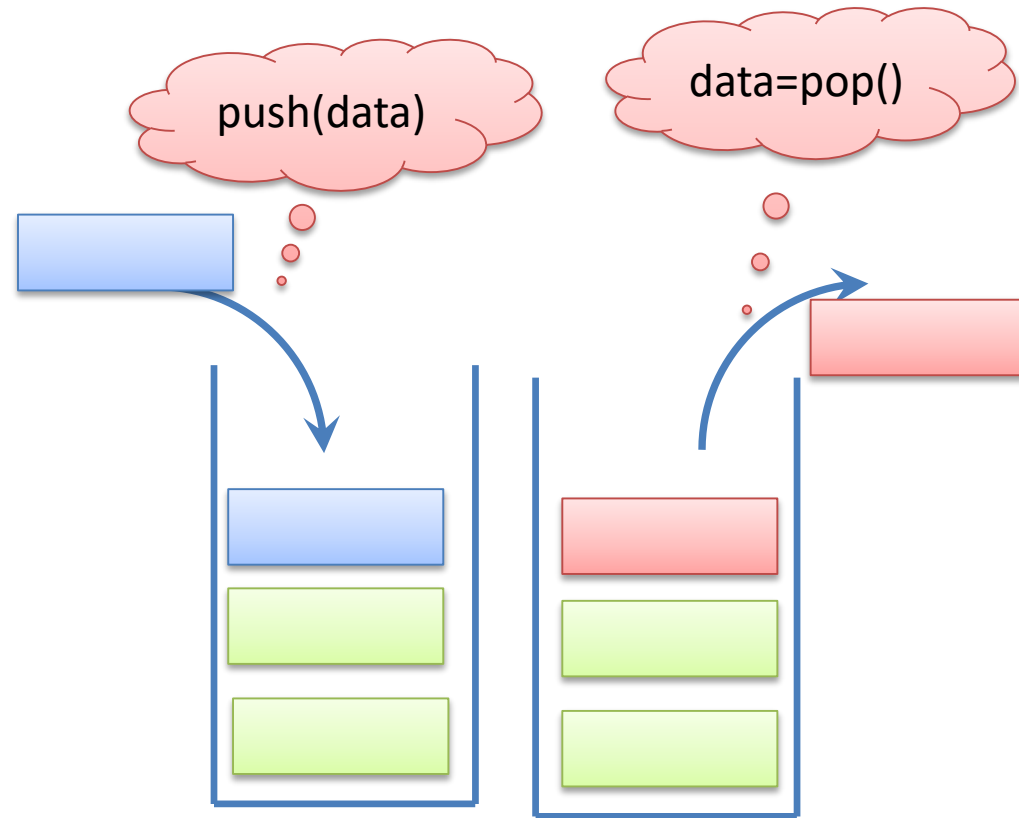
# **Yığıt Soyut Veri Tipi (Stack ADT) Yığıt Veri Yapısı**

# Yığıt

- Yığıt, elemanları sadece bir uçtan eklenip çıkarılabilen liste benzeri bir yapıdır.
- Son giren ilk çıkar mantığına göre çalışır.
- Last In First Out (LIFO)
- Yığıta eklenen son elemanın konumu en üsttedir.
- Yığıttan bir eleman çıkarılacağı zaman en üstteki eleman çıkartılır

# Yığıt nasıl çalışır?

- Stack ADT temel işlemleri:
- yığıtın üstüne veri gönderme
- yığıtın üstünden veri silme
- yığıtın üstünden veri okuma
- Yığıtı boşaltma
- **void push(data)**
- **void pop()**
- eleman **top()** veya **peek()**
- **void clear()**



# Yığıt nasıl çalışır?

- stack sınıfından tanımlanmış stack1 isimli bir yığıt olsun. Başlangıçta yığıt boş.



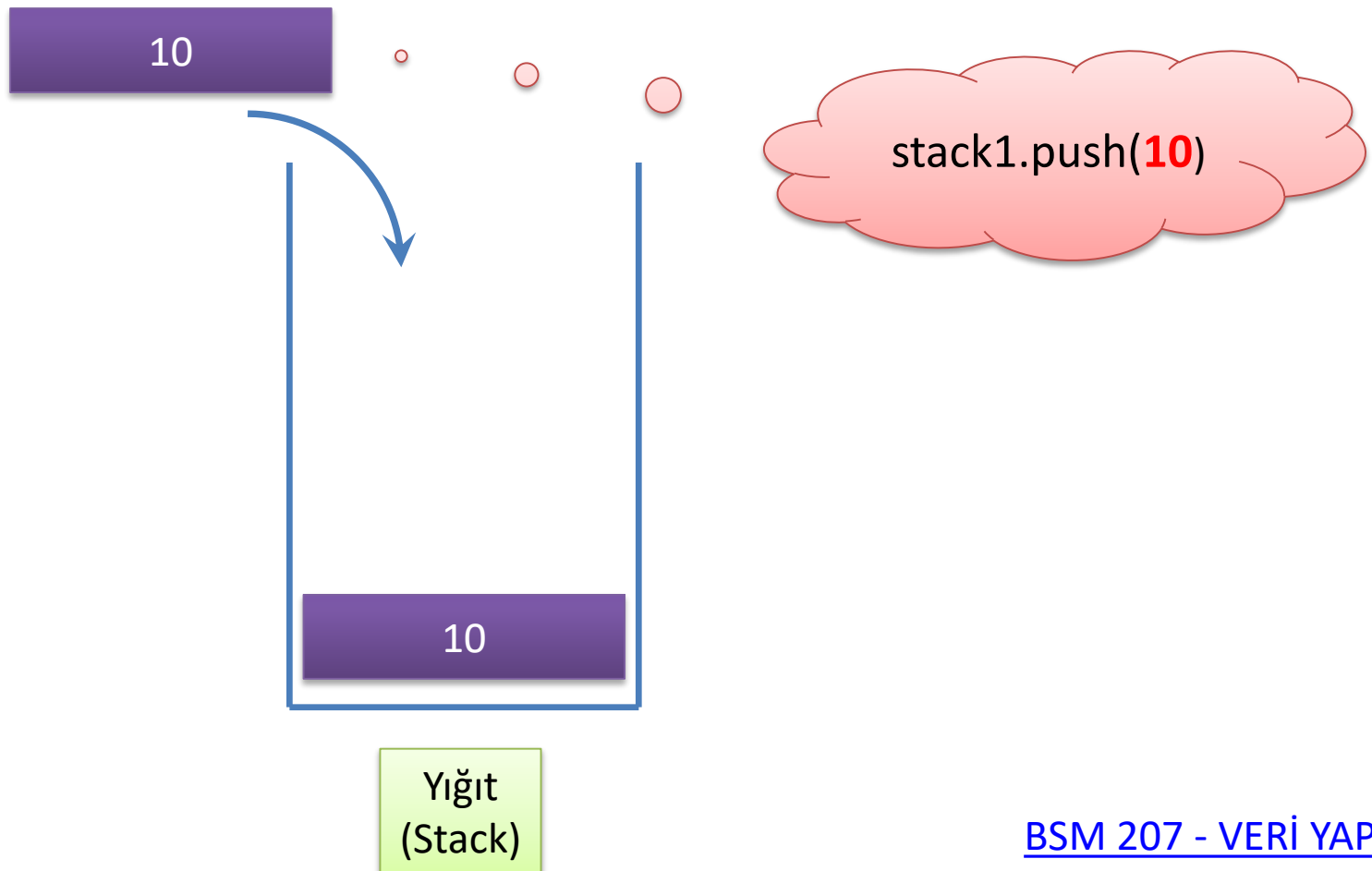
Yığıt  
(Stack)

A thought bubble with a pink fill and a red outline, containing the C++ code `stack<int> stack1;`. Three small red circles lead from the bubble towards the stack diagram.

```
stack<int> stack1;
```

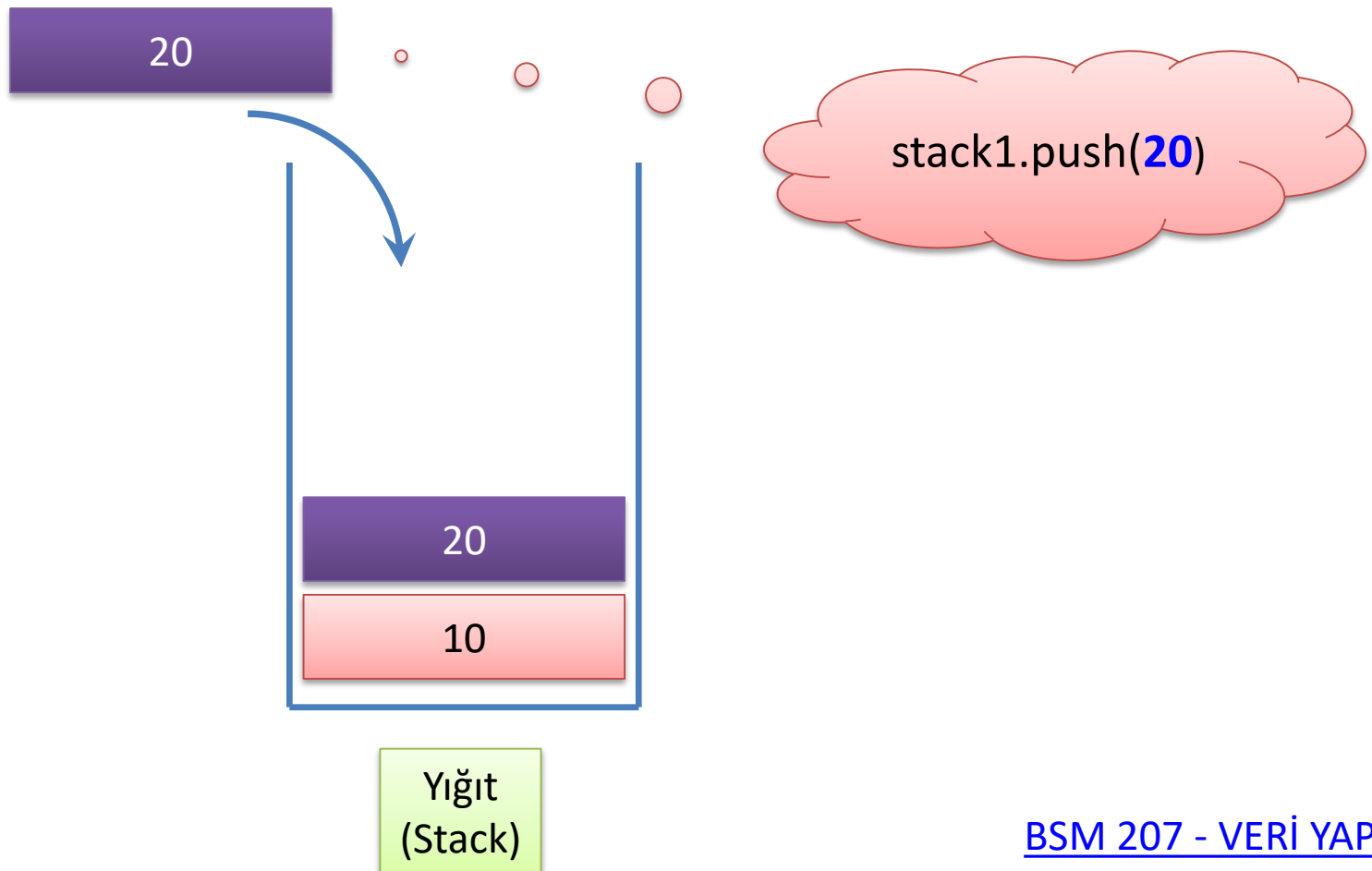
# Yığıt nasıl çalışır?

- Push ile yığıtın üstüne yeni bir eleman ekliyoruz.



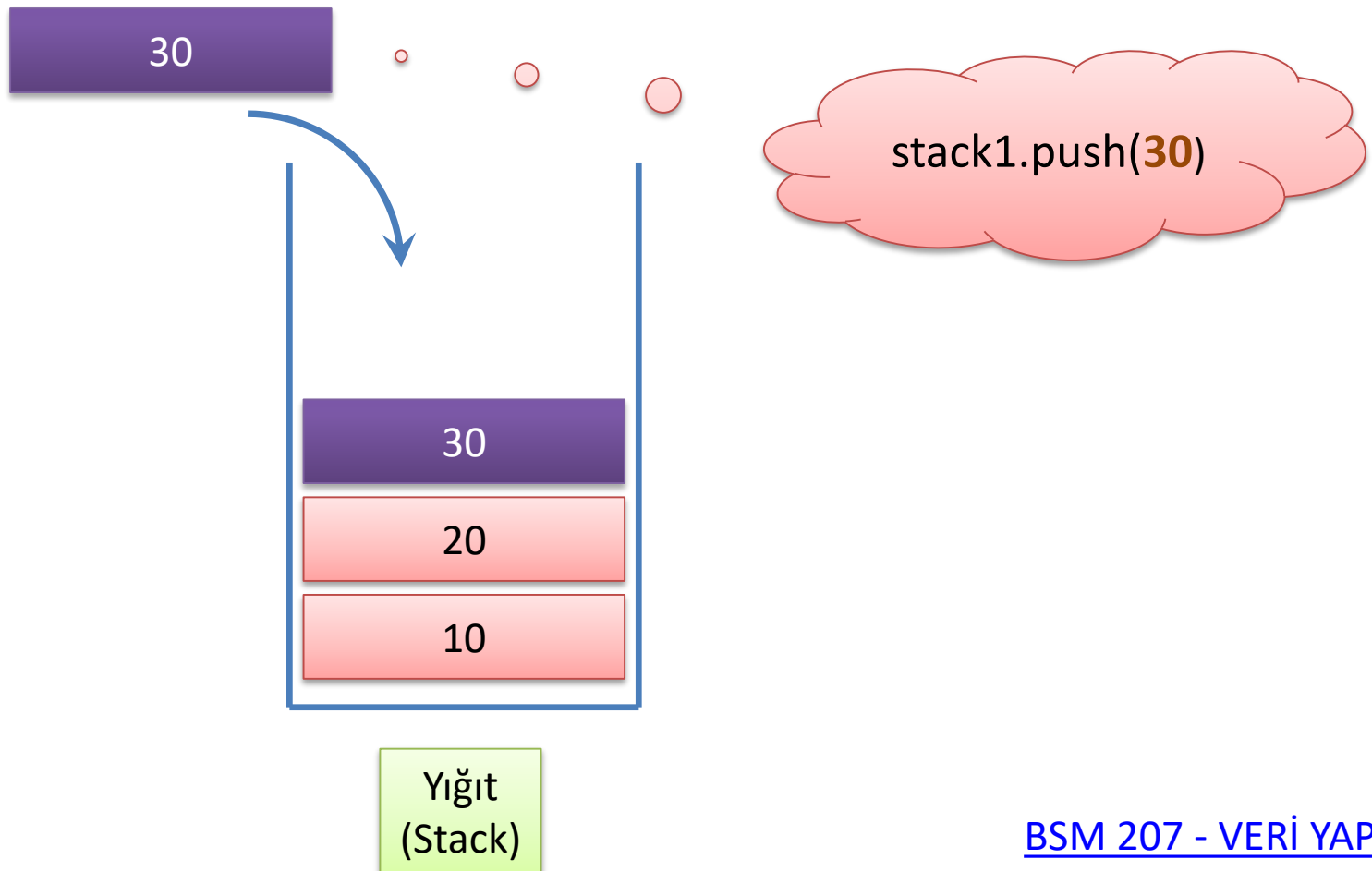
# Yığıt nasıl çalışır?

- Push ile yığita yeni bir eleman ekliyoruz.



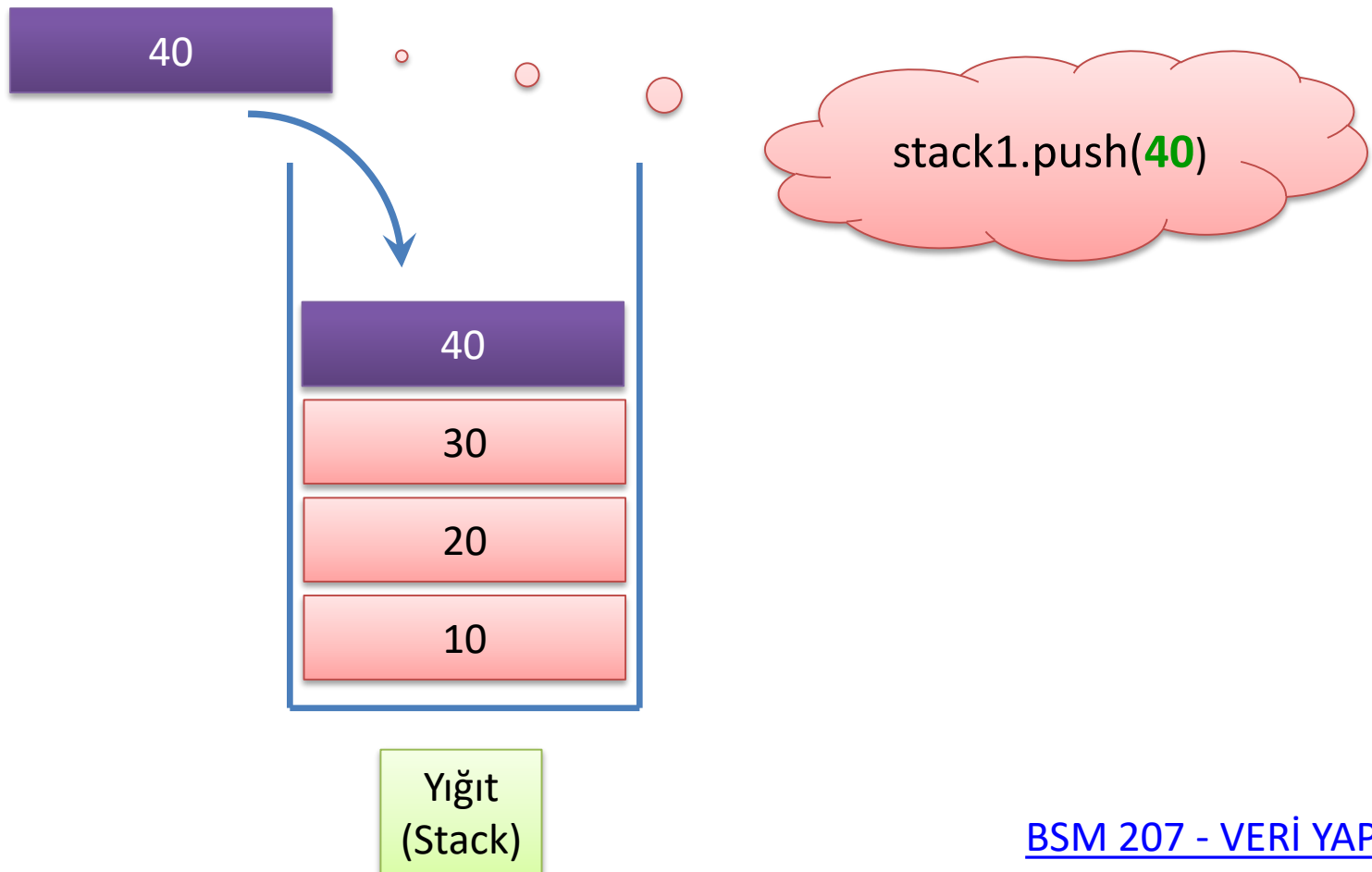
# Yığıt nasıl çalışır?

- Push ile yığıtın üstüne yeni bir eleman ekliyoruz.



# Yığıt nasıl çalışır?

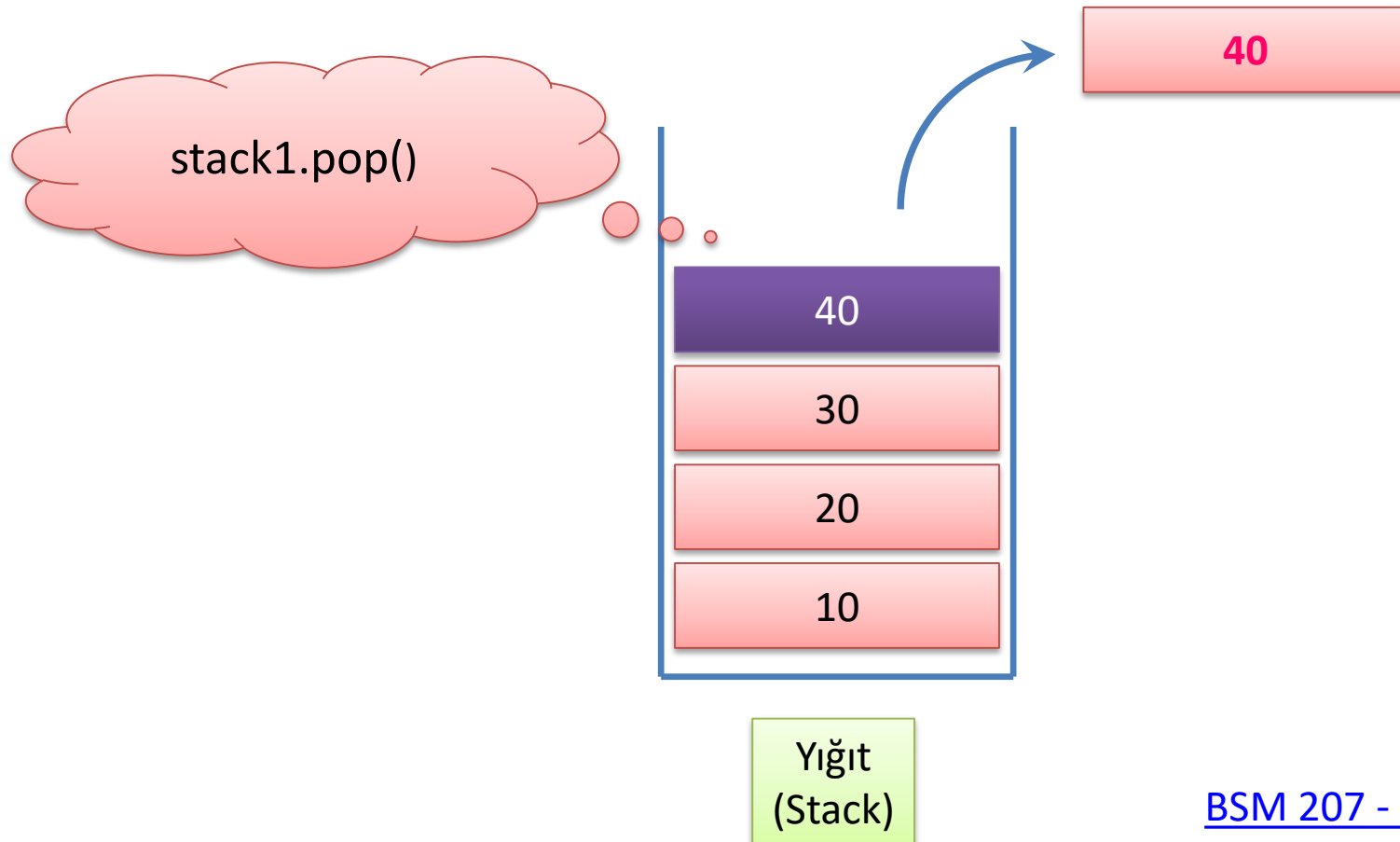
- Push ile yığıtın üstüne yeni bir eleman ekliyoruz.





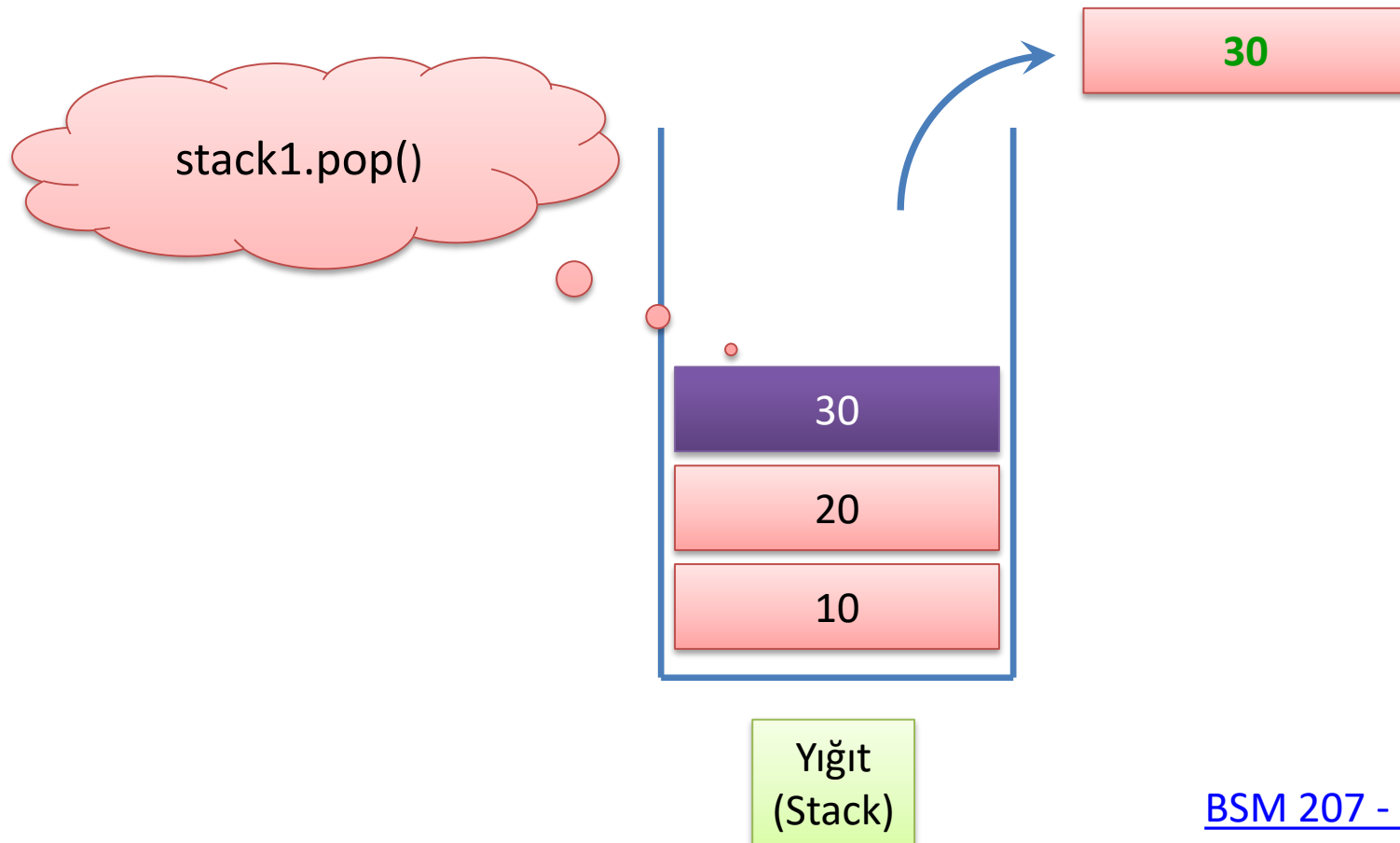
# Yığıt nasıl çalışır?

- Pop ile yığıtın üstünden bir eleman çıkarıyoruz.



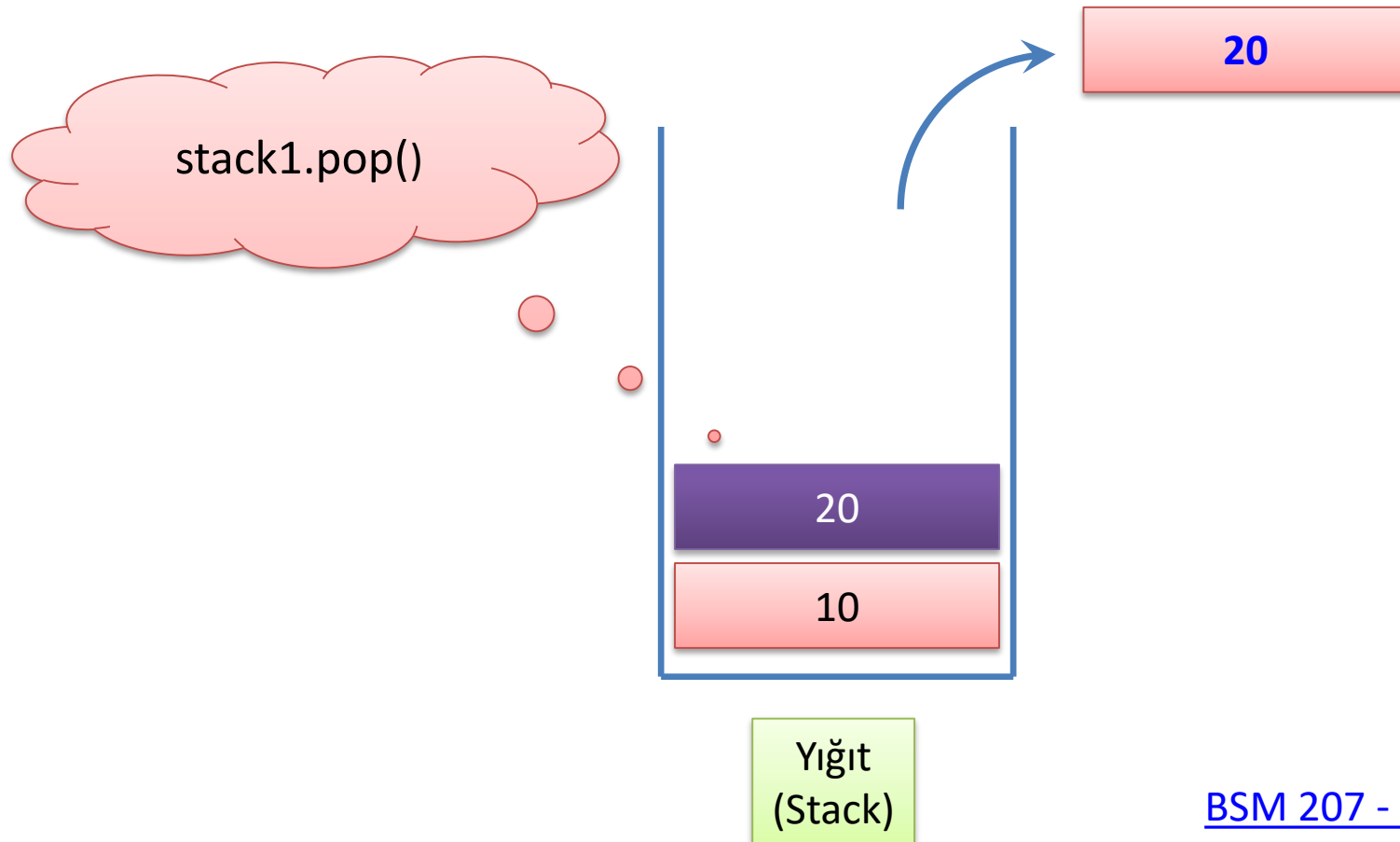
# Yığıt nasıl çalışır?

- Pop ile yığıtın üstünden bir eleman çıkarıyoruz.



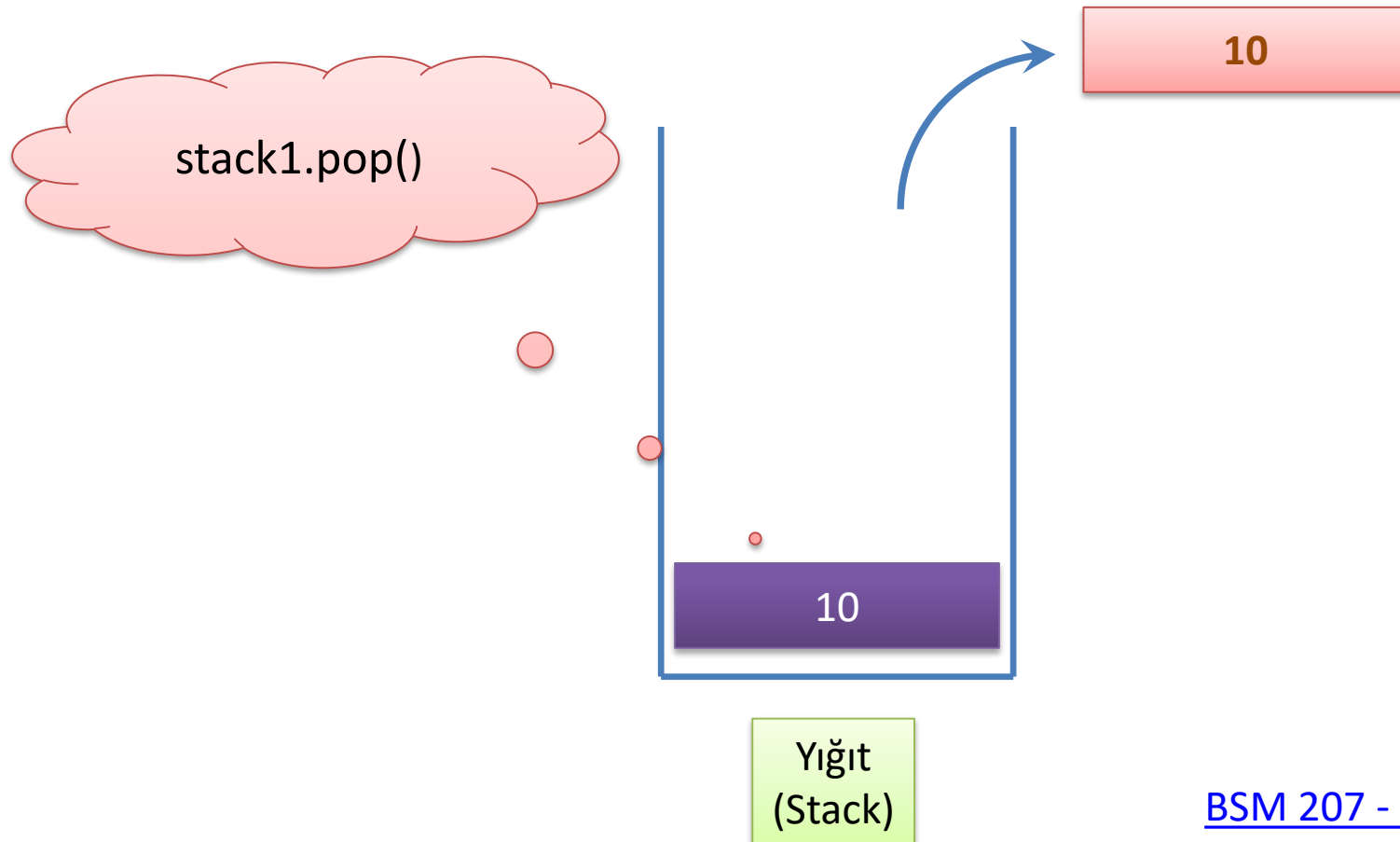
# Yığıt nasıl çalışır?

- Pop ile yığıtın üstünden bir eleman çıkarıyoruz.



# Yığıt nasıl çalışır?

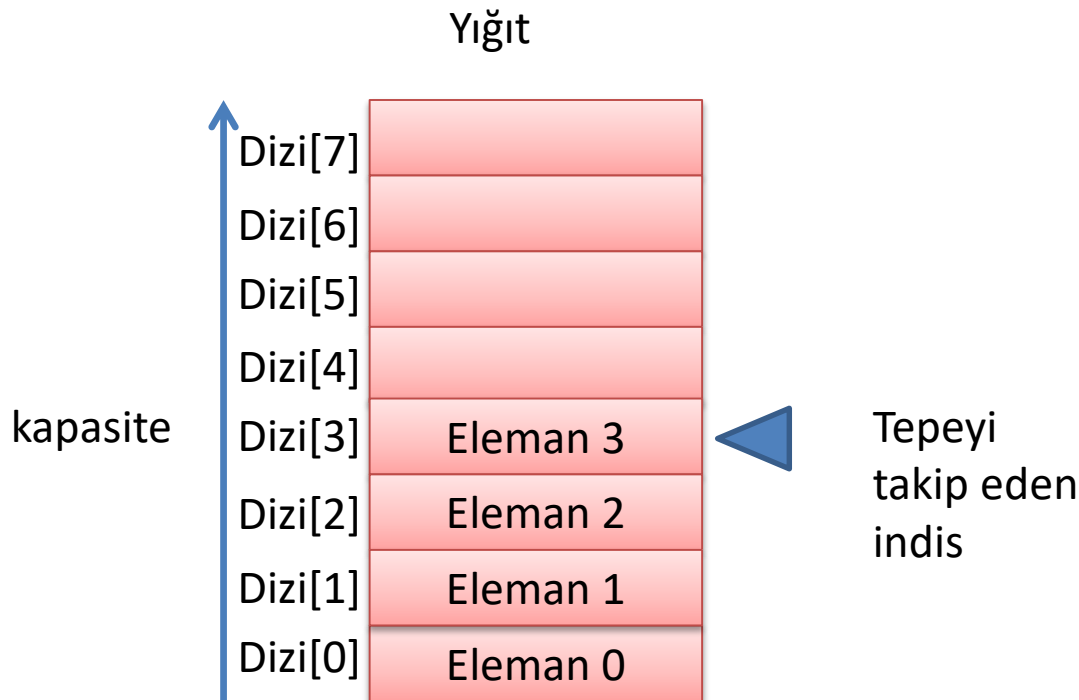
- Pop ile yığıtın üstünden bir eleman çıkarıyoruz.



# Yığıt üzerinde tanımlı temel işlemler

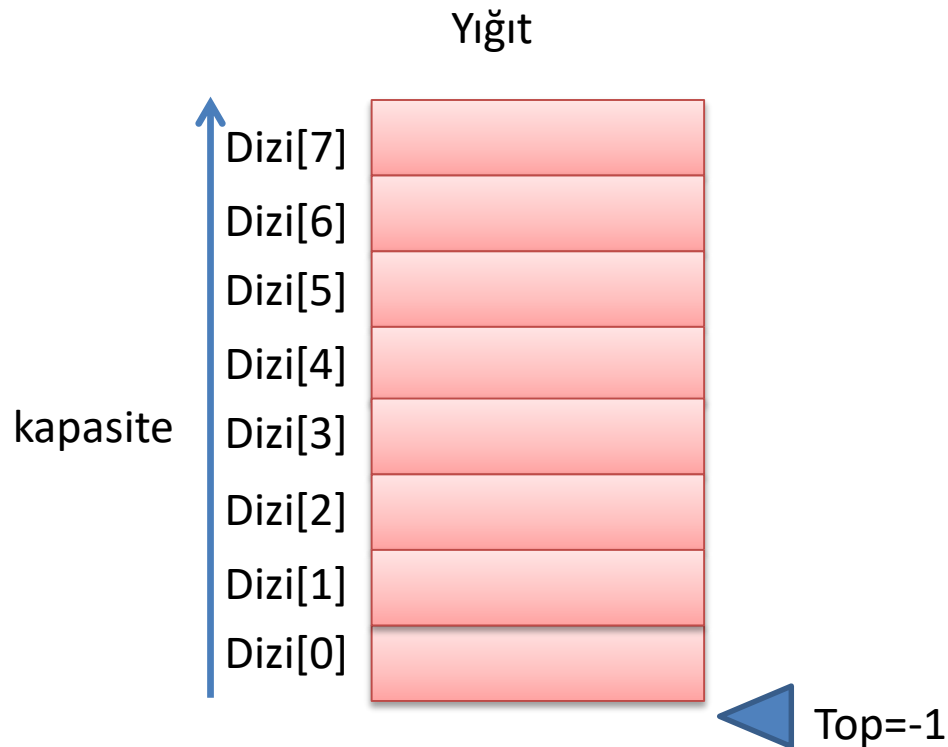
- **push** : veriyi yığıtın üstüne gönder
- **pop** : yığıtın üstündeki veriyi çıkart
- **top,peek**: en üstteki veriyi oku
- **isEmpty** : yığıt boş mu?
- **clear** : yığıtı boşalt

# Stack ADT – Dizi ile gerçekleştirme

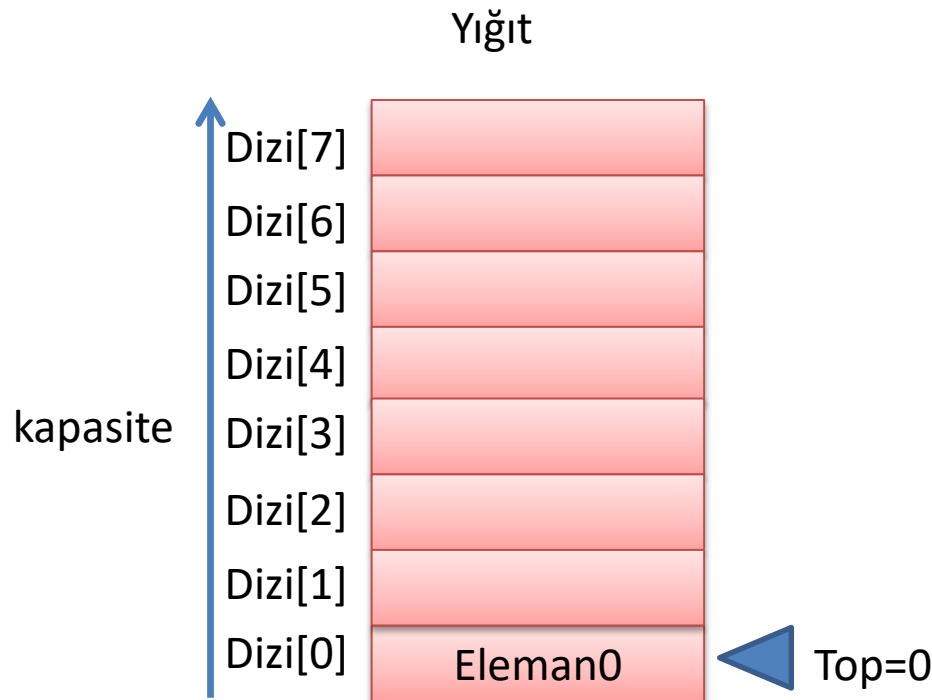


- Dizi ile gerçekleştirilmede stack elemanları dizi üzerinde tutulur.
- Bir tam sayı değişken yığının tepesindeki elemanın adresini takip etmek için kullanılır.

# Stack ADT – Dizi ile gerçekleştirme



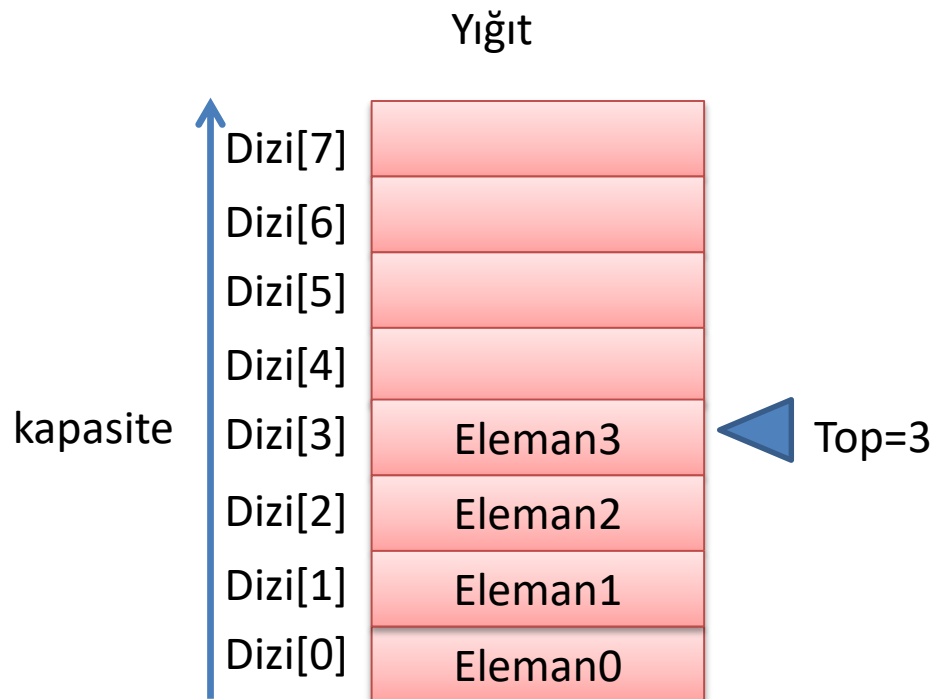
# Stack ADT – Dizi ile gerçekleştirme



`stack1.push(Eleman0)`

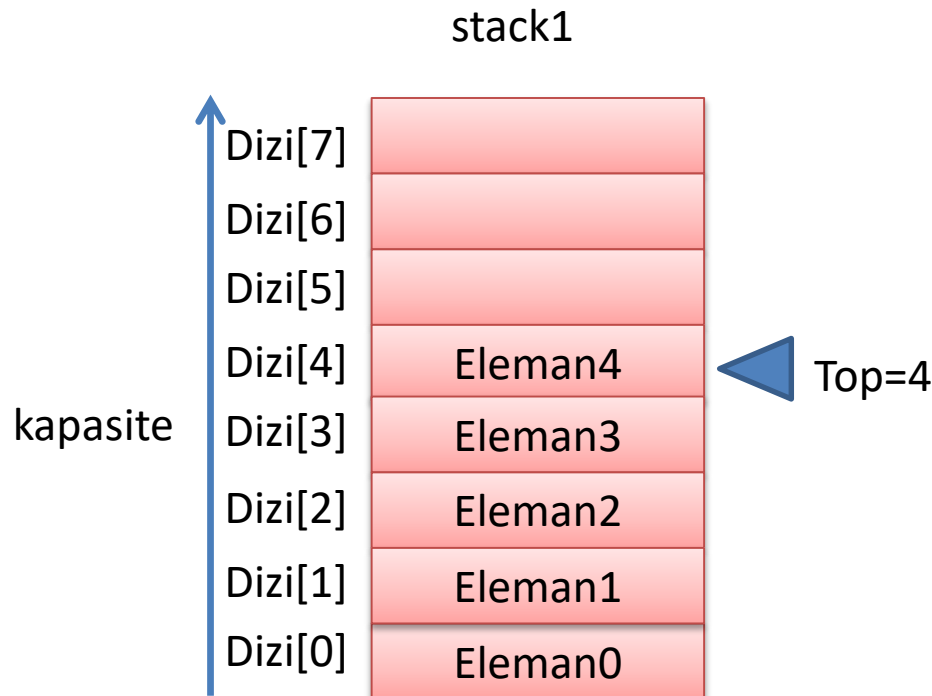


# Stack ADT – Dizi ile gerçekleştirme



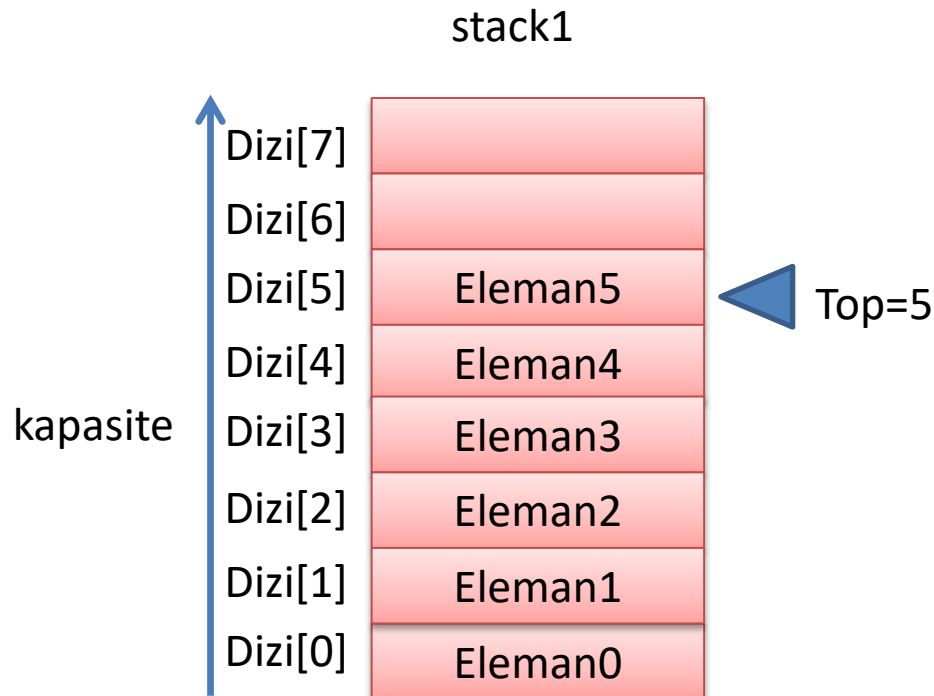
```
stack1.push(Eleman0)  
stack1.push(Eleman1)  
stack1.push(Eleman2)  
stack1.push(Eleman3)
```

# Stack ADT – Dizi ile gerçekleştirme



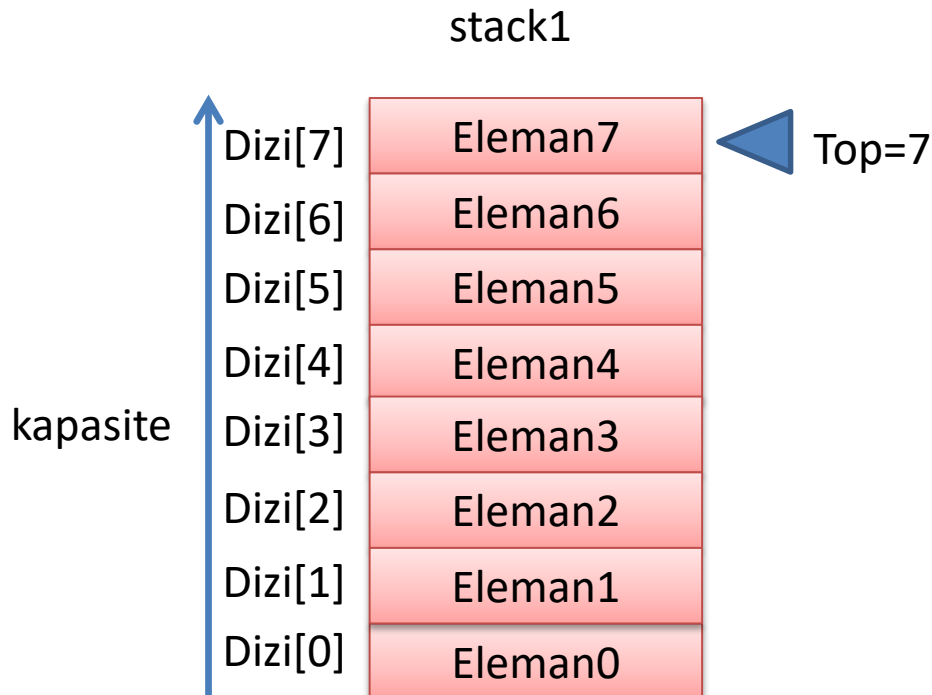
```
stack1.push(Eleman0)
stack1.push(Eleman1)
stack1.push(Eleman2)
stack1.push(Eleman3)
stack1.push(Eleman4)
```

# Stack ADT – Dizi ile gerçekleştirme



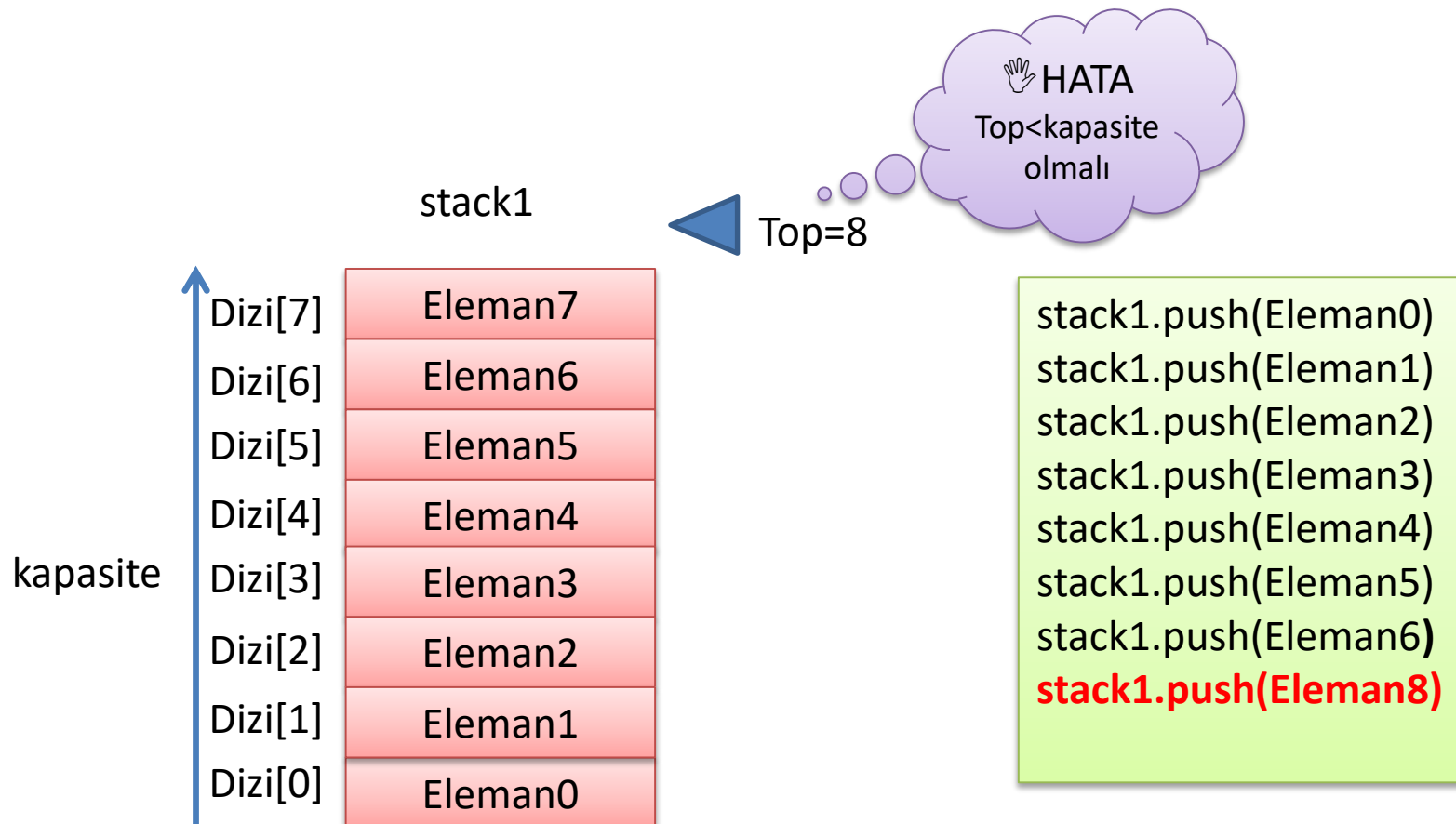
```
stack1.push(Eleman0)
stack1.push(Eleman1)
stack1.push(Eleman2)
stack1.push(Eleman3)
stack1.push(Eleman4)
stack1.push(Eleman5)
```

# Stack ADT – Dizi ile gerçekleştirme

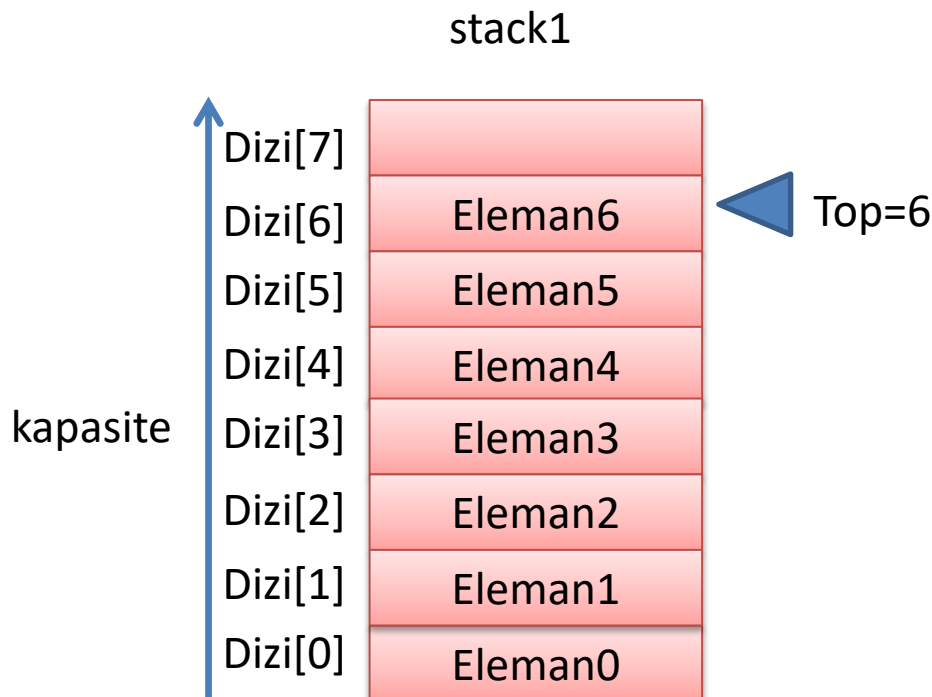


```
stack1.push(Eleman0)
stack1.push(Eleman1)
stack1.push(Eleman2)
stack1.push(Eleman3)
stack1.push(Eleman4)
stack1.push(Eleman5)
stack1.push(Eleman6)
stack1.push(Eleman7)
```

# Stack ADT – Dizi ile gerçekleştirme

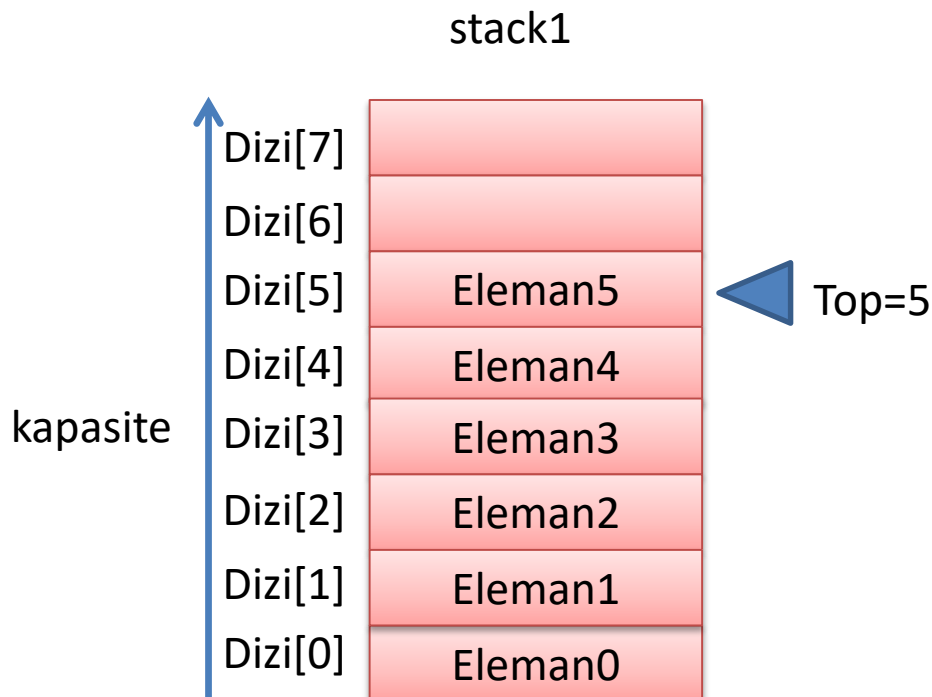


# Stack ADT – Dizi ile gerçekleştirme



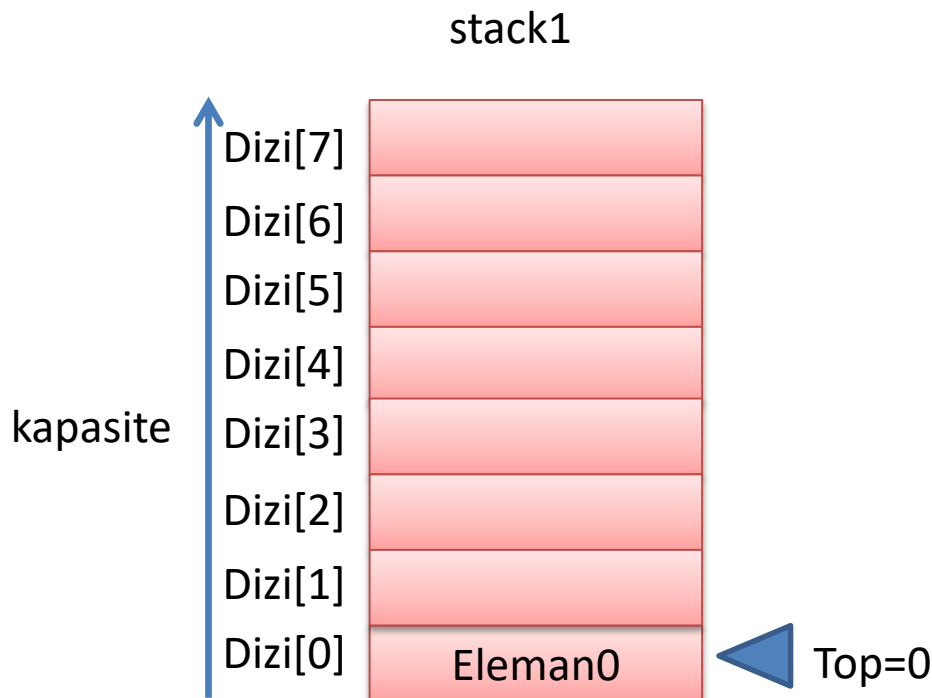
```
stack1.push(Eleman0)
stack1.push(Eleman1)
stack1.push(Eleman2)
stack1.push(Eleman3)
stack1.push(Eleman4)
stack1.push(Eleman5)
stack1.push(Eleman6)
stack1.push(Eleman7)
stack1.pop()
```

# Stack ADT – Dizi ile gerçekleştirme



```
stack1.push(Eleman0)
stack1.push(Eleman1)
stack1.push(Eleman2)
stack1.push(Eleman3)
stack1.push(Eleman4)
stack1.push(Eleman5)
stack1.push(Eleman6)
stack1.push(Eleman7)
stack1.pop()
stack1.pop()
```

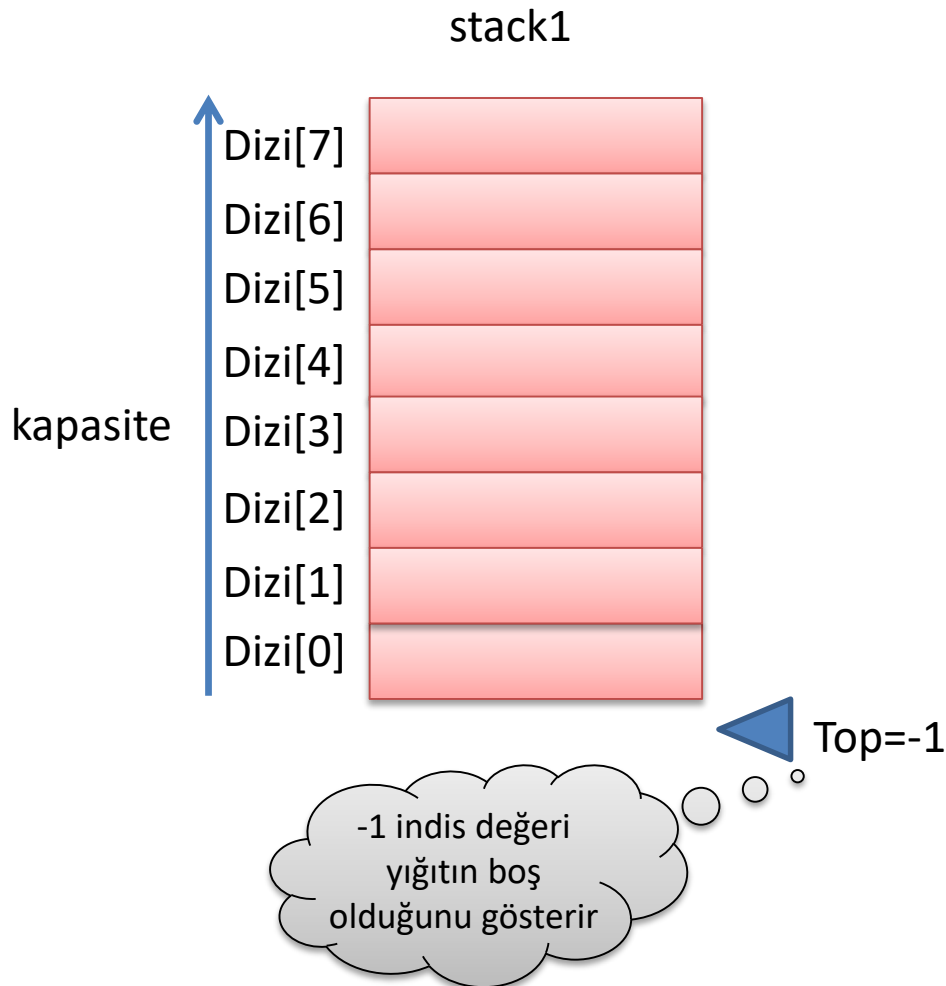
# Stack ADT – Dizi ile gerçekleştirme



```
stack1.push(Eleman0)
stack1.push(Eleman1)
stack1.push(Eleman2)
stack1.push(Eleman3)
stack1.push(Eleman4)
stack1.push(Eleman5)
stack1.push(Eleman6)
stack1.push(Eleman7)
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
```

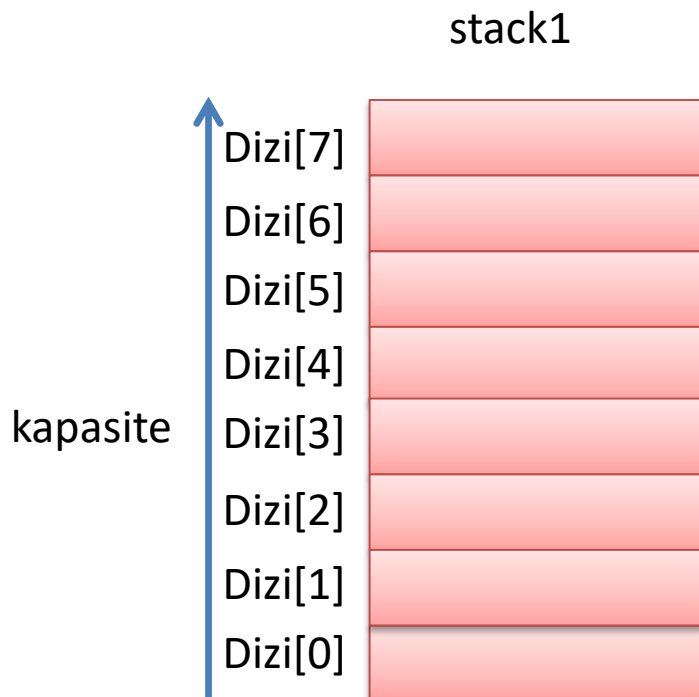


# Stack ADT – Dizi ile gerçekleştirme



```
stack1.push(Eleman0)
stack1.push(Eleman1)
stack1.push(Eleman2)
stack1.push(Eleman3)
stack1.push(Eleman4)
stack1.push(Eleman5)
stack1.push(Eleman6)
stack1.push(Eleman7)
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
```

# Stack ADT – Dizi ile gerçekleştirme

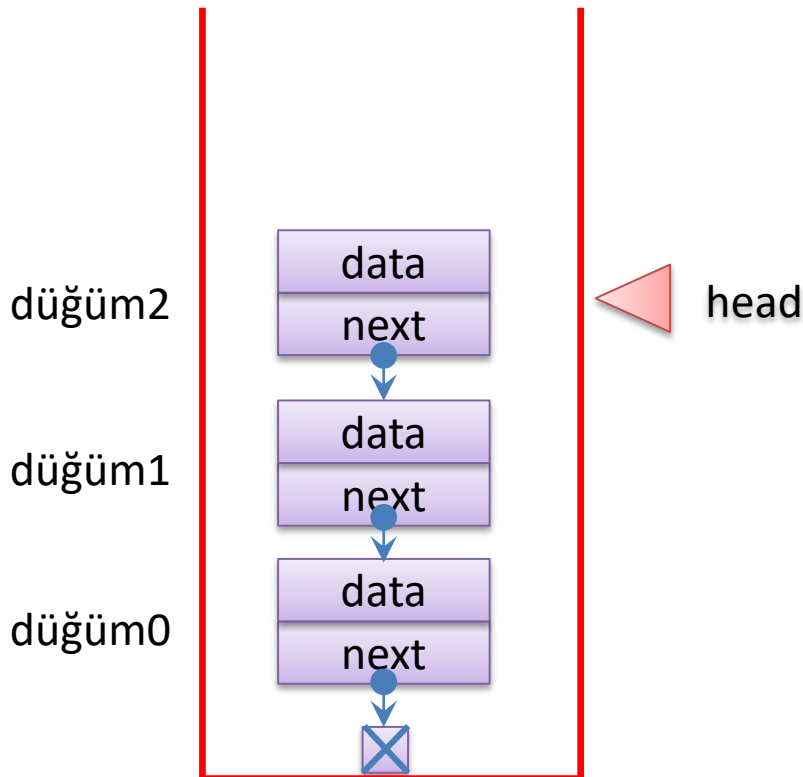


Top=-2

HATA  
fırlatılmalı  
-1'den küçük  
olamaz

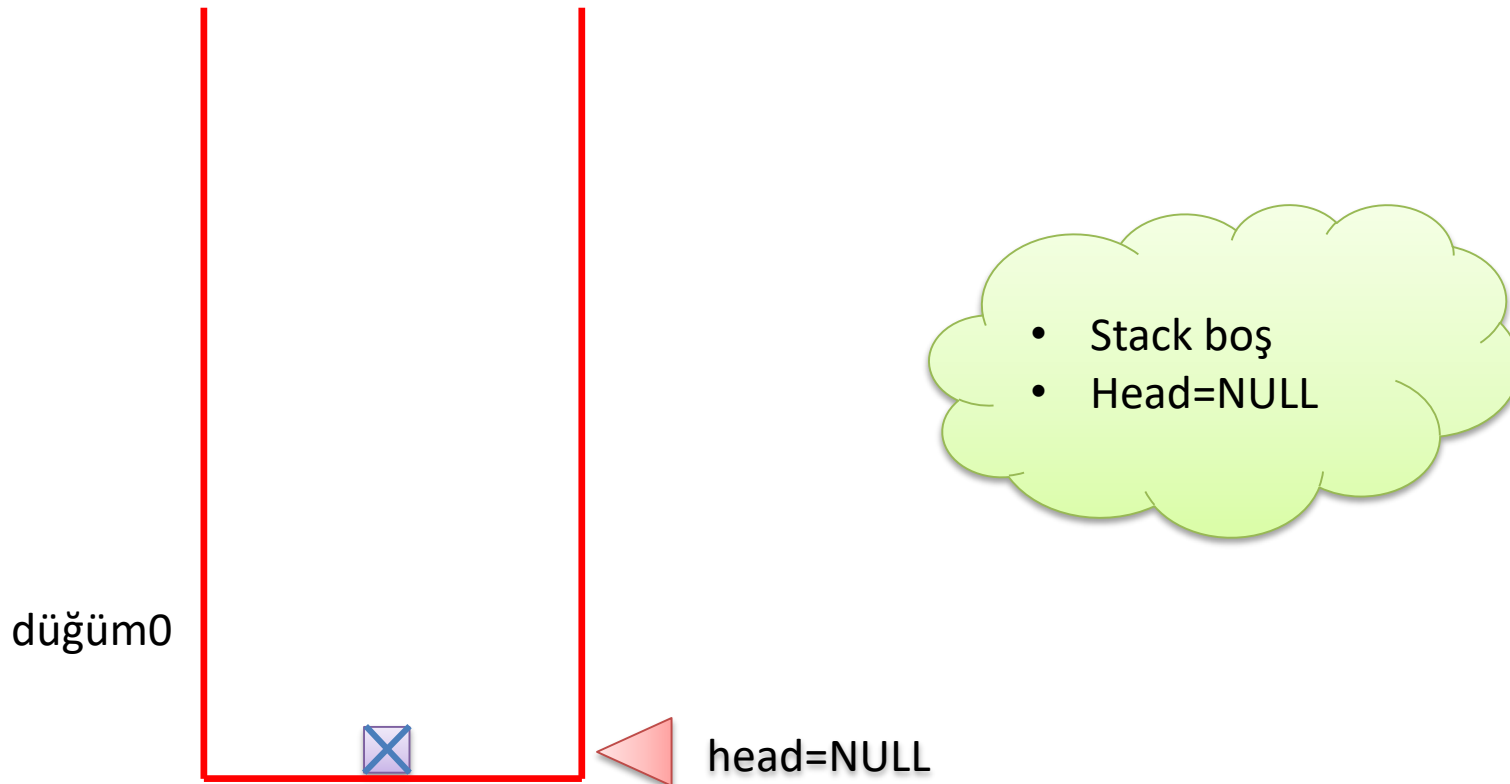
```
stack1.push(Eleman0)
stack1.push(Eleman1)
stack1.push(Eleman2)
stack1.push(Eleman3)
stack1.push(Eleman4)
stack1.push(Eleman5)
stack1.push(Eleman6)
stack1.push(Eleman7)
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
stack1.pop()
```

# Bağlı Yığıt Veri Yapısı

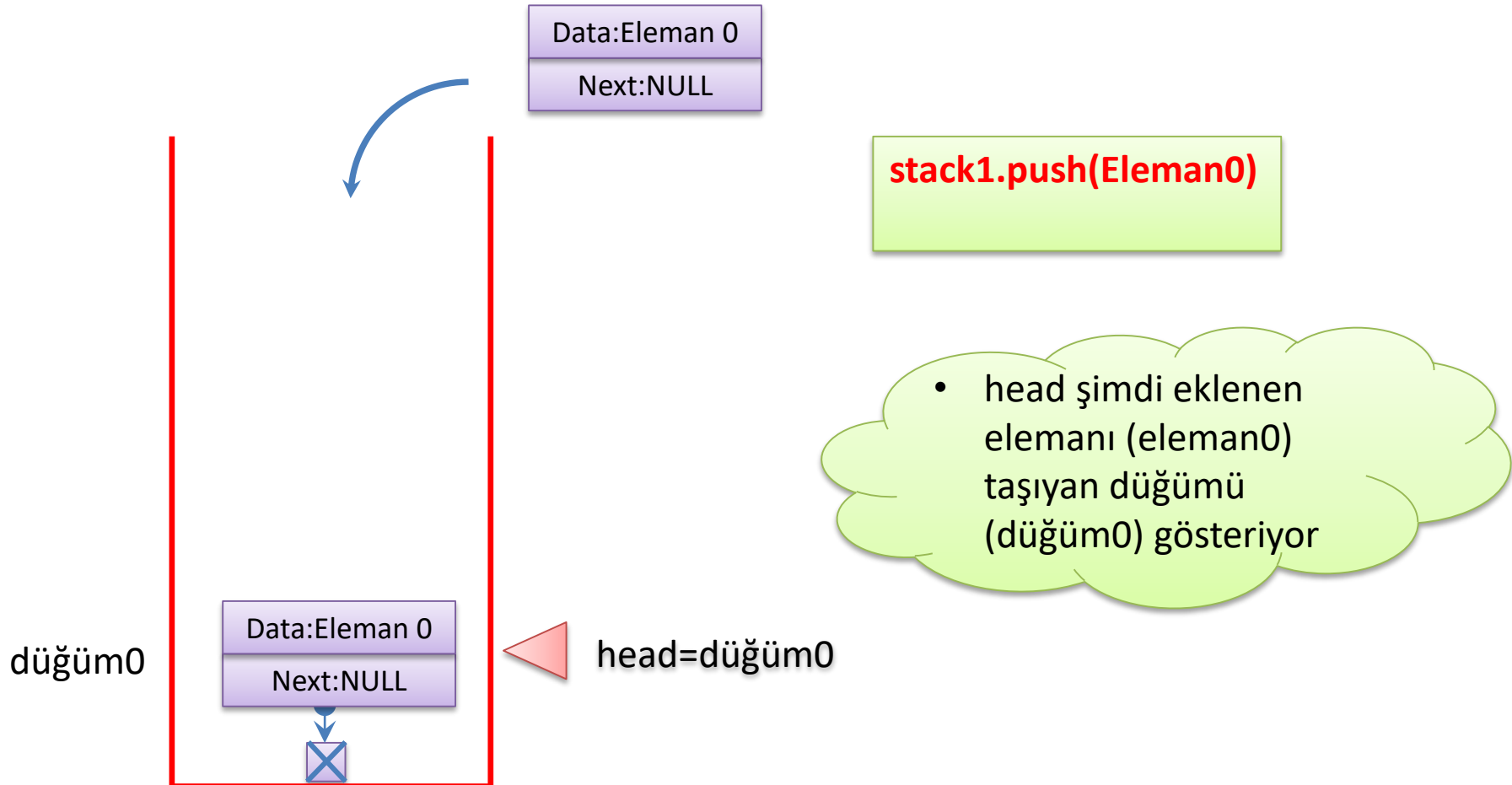


- Bağlı listeyi oluşturan düğümler, yığıt veri yapısını oluşturmak için kullanılabilir.
- Yığita eleman itmek bir bağlı listenin başına eleman eklemek gibi düşünülebilir. (`push_front(eleman)`)
- Yığıttan eleman silmek ise yine bağlı listenin ilk elemanını silmek gibidir. (`pop_front()`)
- Liste başı (`head`) yığıtın tepesini gösterir.

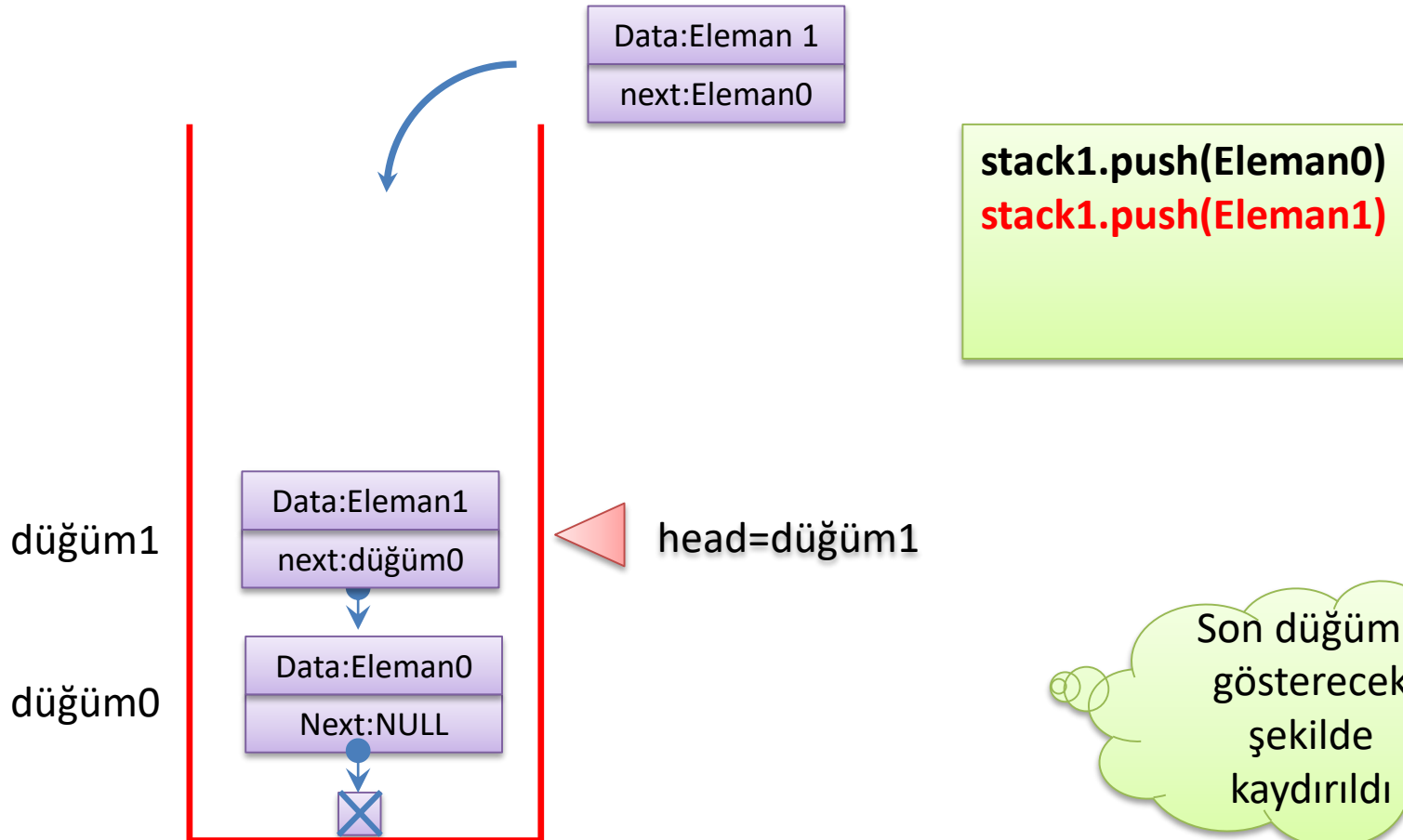
# Bağlı Yığıt Veri Yapısı



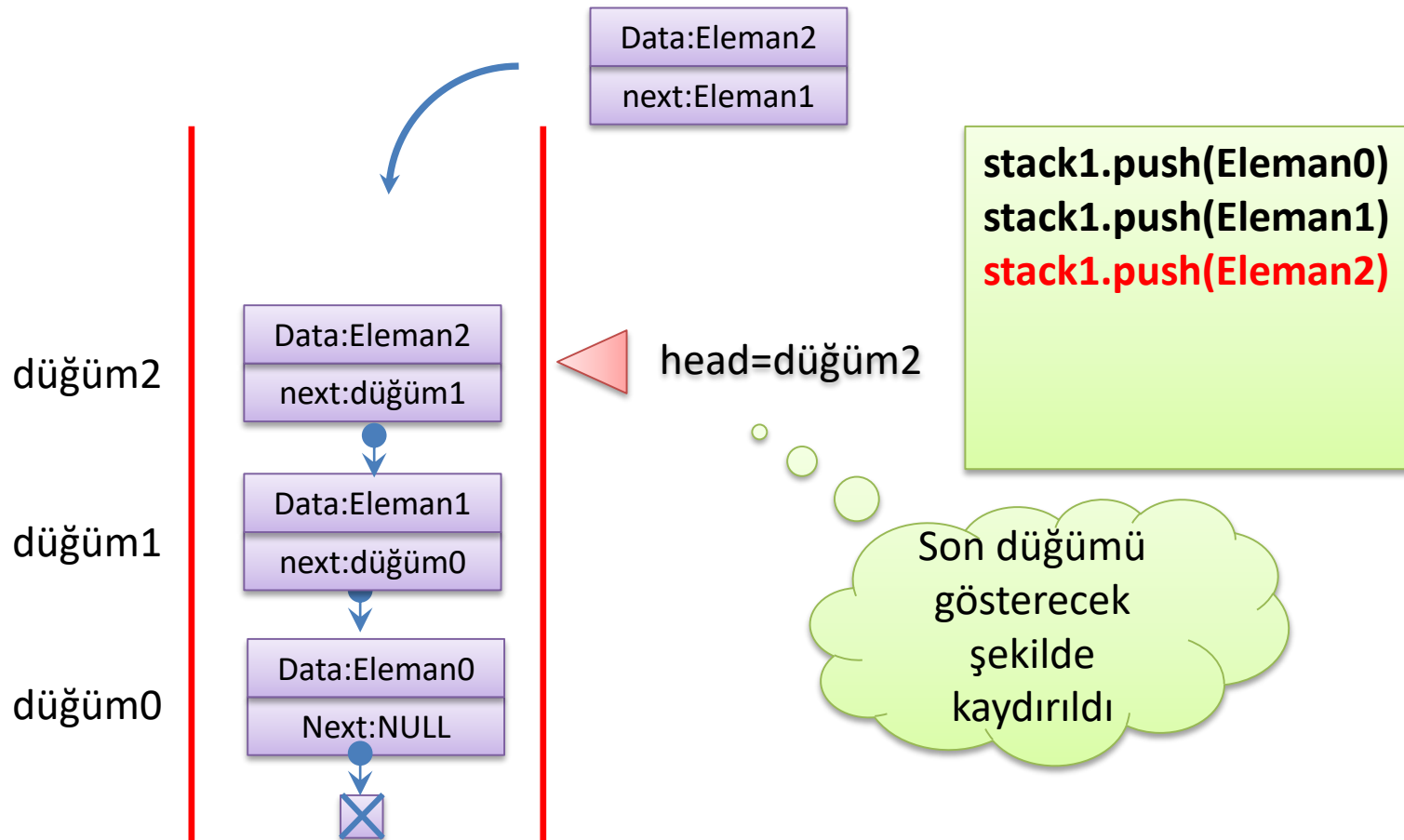
# Bağlı Yığıt Veri Yapısı



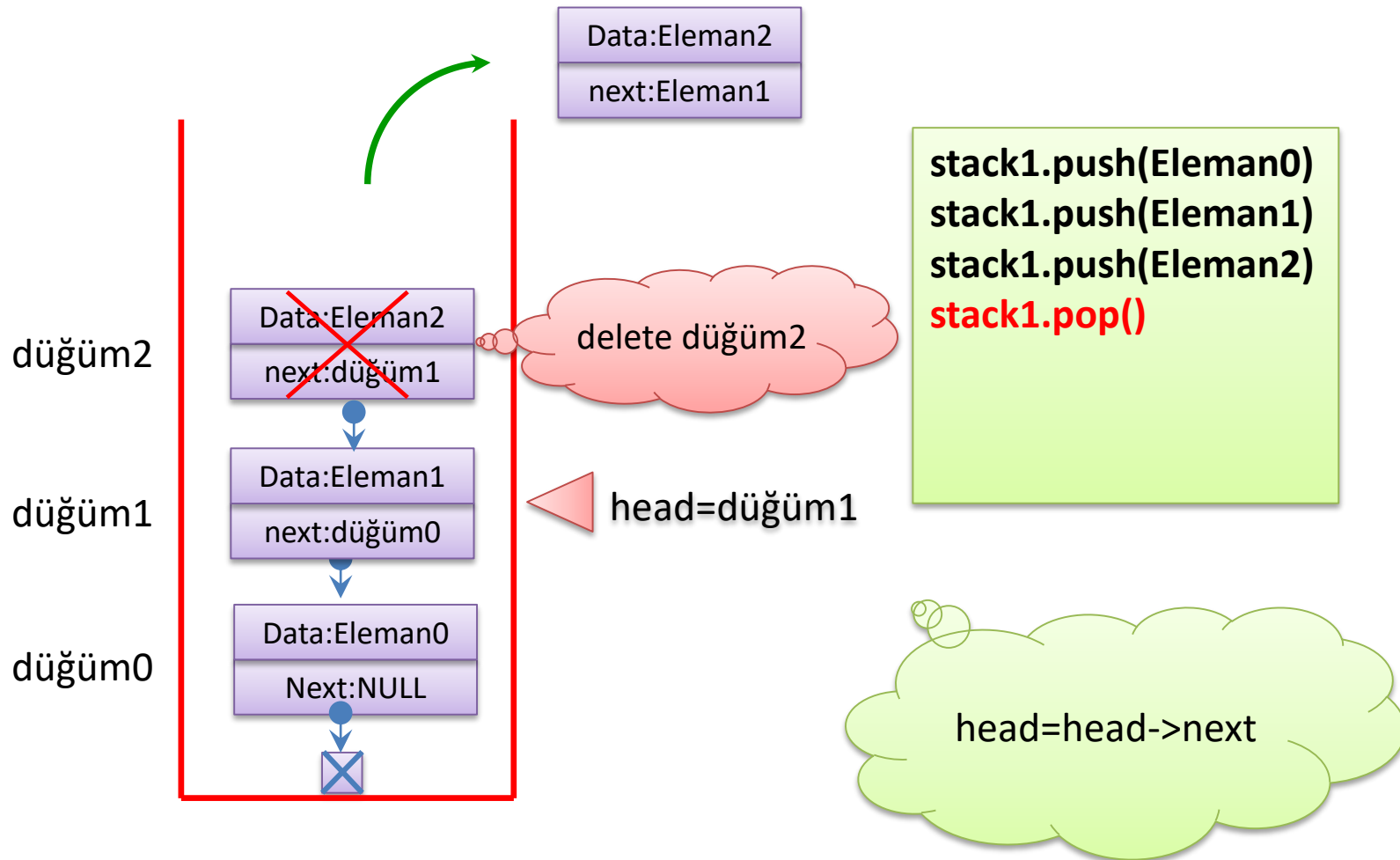
# Bağlı Yığıt Veri Yapısı



# Bağlı Yığıt Veri Yapısı

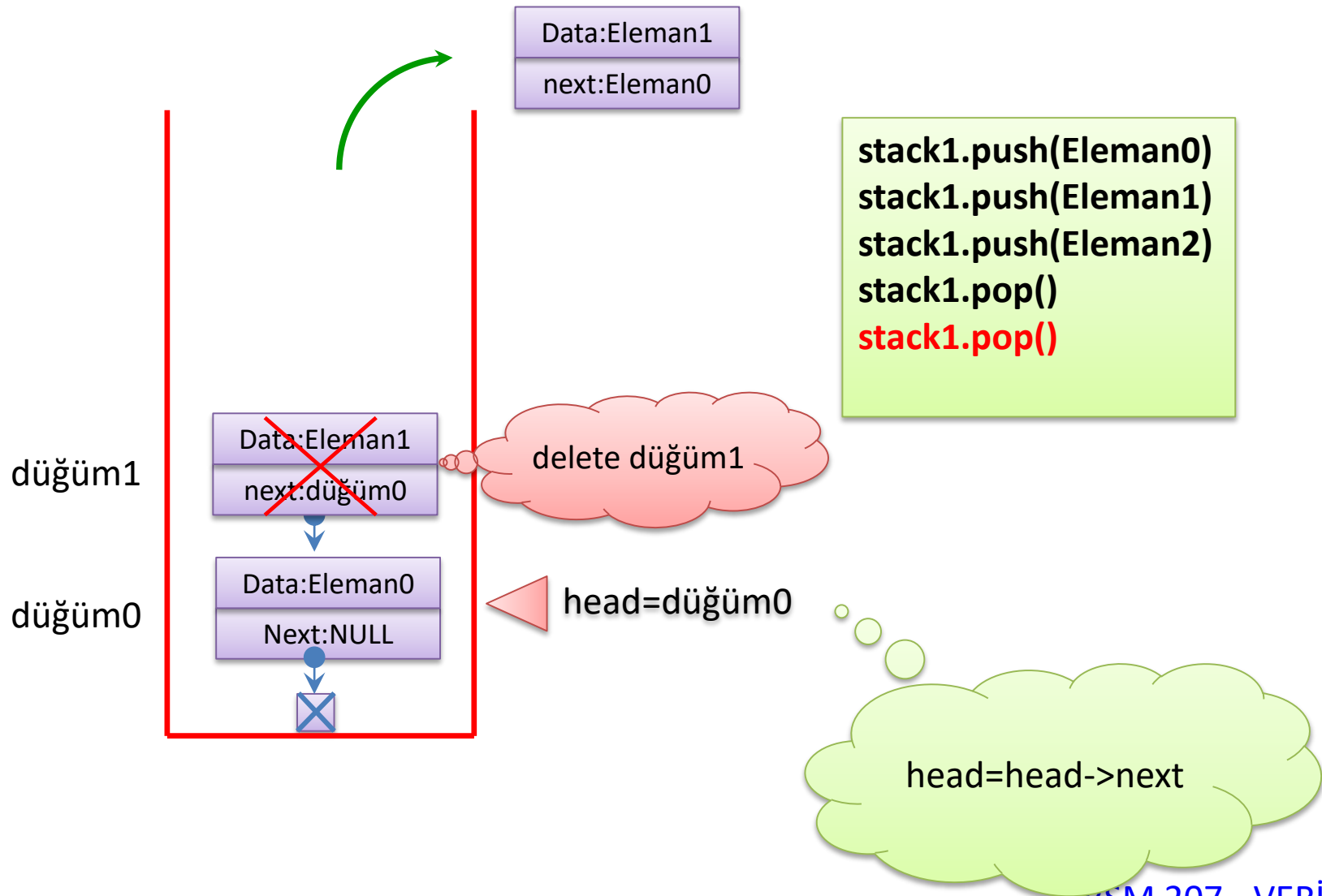


# Bağlı Yığıt Veri Yapısı

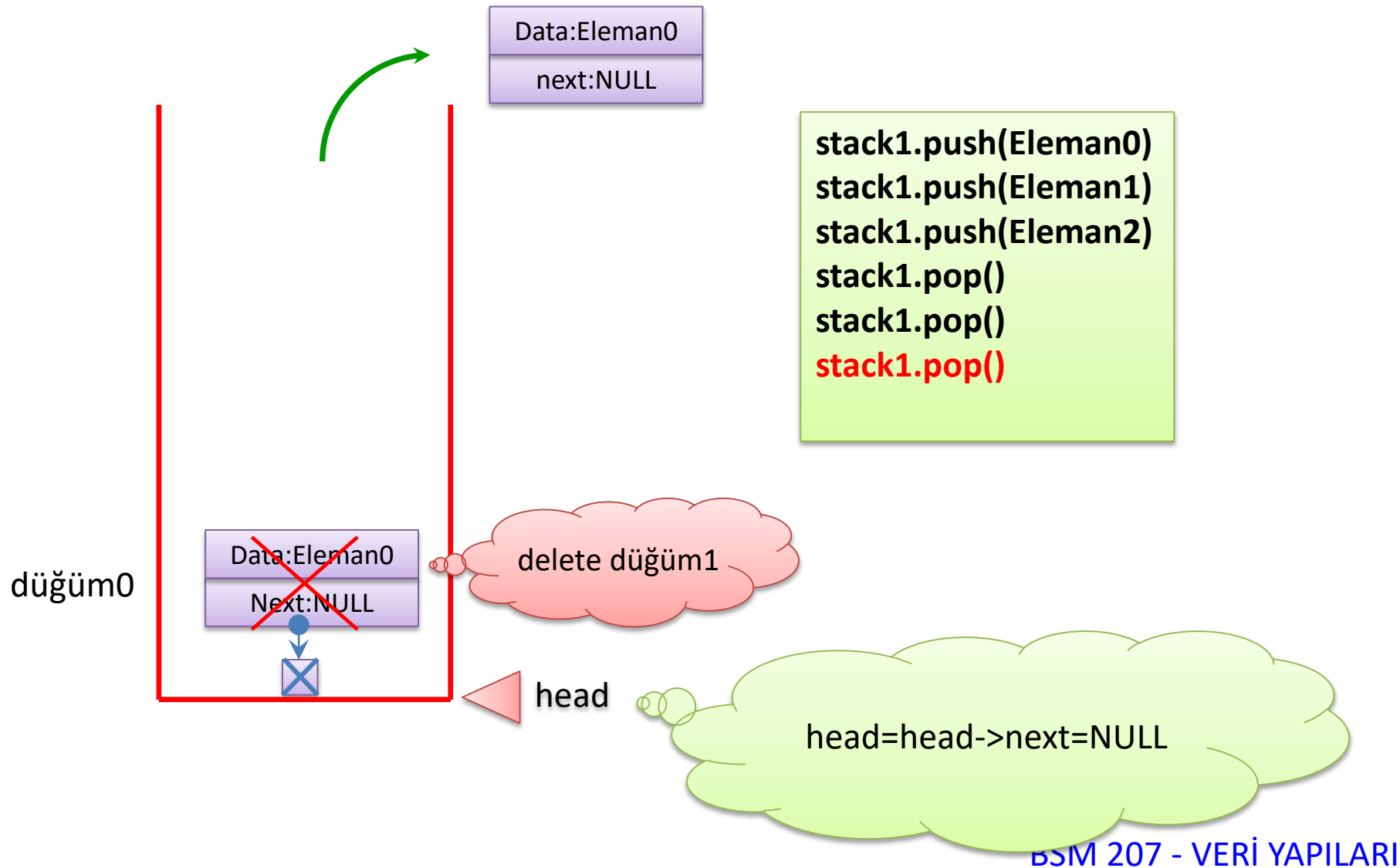




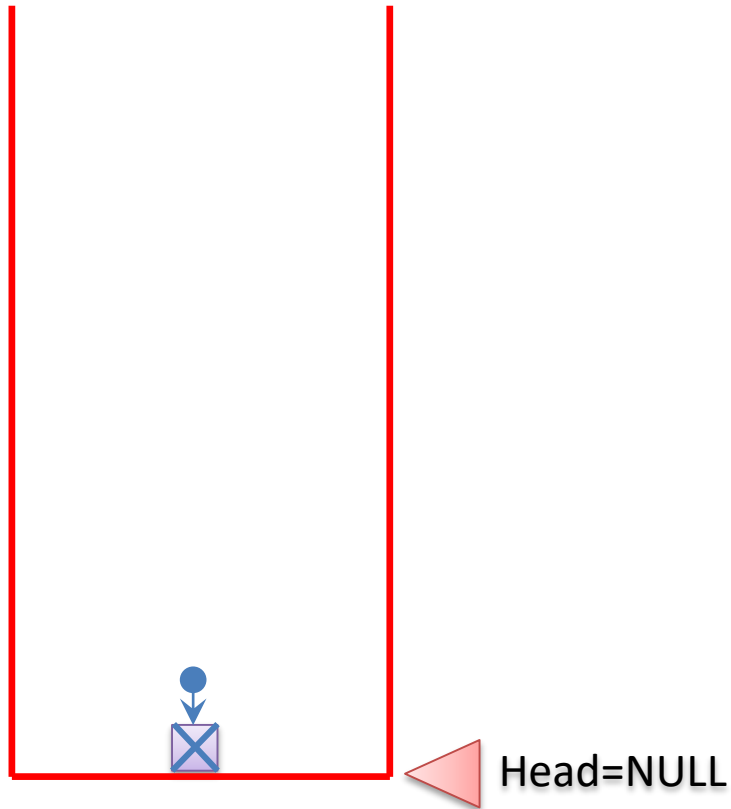
# Bağlı Yığıt Veri Yapısı



# Bağlı Yığıt Veri Yapısı



# Bağlı Yığıt Veri Yapısı



```
stack1.push(Eleman0)
stack1.push(Eleman1)
stack1.push(Eleman2)
stack1.pop()
stack1.pop()
stack1.pop()
```

# Infix, Prefix, Postfix

- Parantezler
- $5+6*7$  ifadesinin hesaplanması
- Önce toplama:  
$$(5+6)*7 = 11*7 = 77$$
- Önce çarpma:  
$$5+(6*7) = 5+42 = 47$$
- Parantezlerin kullanımı infix notasyonunda önemlidir.

# Stack uygulaması: Infix, Prefix, Postfix

- **Infix notasyonunda**

- A ve B yi toplamak için

$$A+B$$

- A ve B'yi çarpmak için

$$A*B$$

- Operatörler (\*,+, -, /) operandların (A,B) arasına gelir

# Infix, Prefix, Postfix

- **Prefix notasyonunda**
- iki operand arasındaki operatör önce belirtilir.
- A ve B yi toplamak için, topla A B  
**+ A B**
- A ve B'yi çarpmak için, çarp A B  
**\* A B**

# Infix, Prefix, Postfix

- **Postfix notasyonunda**
- iki operand arasındaki operatör sonra belirtilir.
- A ve B yi toplamak için, A B topla

**A B +**

- A ve B'yi çarpmak için, A B çarp

**A B \***

# Infix, Prefix, Postfix

- **Prefix notasyonu**

- $+ 5 * 6 7 =$   
 $= + 5 42$   
 $= 47$

- $* + 5 6 7 =$   
 $= * 11 7$   
 $= 77$

- Parantez kullanmadan işlem önceliği tanımlanabilir



# Infix, Prefix, Postfix

- **Postfix notasyonu**

- $5 \ 6 \ 7 \ * \ + \ =$   
 $\quad \quad \quad = 5 \ 42 \ +$   
 $\quad \quad \quad = 47$

- $5 \ 6 \ + \ 7 \ * \ =$   
 $\quad \quad \quad = 11 \ 7 \ *$   
 $\quad \quad \quad = 77$

- Parantez kullanmadan işlem önceliği tanımlanabilir

# Postfix dönüşümünün değerlendirilmesi

- Genelde bir compiler bir infix ifadeyi hesaplayacağı zaman önce postfix formuna dönüştürür.
- Böylece ortaya çıkabilecek olan belirsizlikler ortadan kaldırılır.
- $5*6+7*8 \rightarrow 5\ 6\ *\ 7\ 8\ *\ +$

# Algoritma- Infix'den postfix'e dönüşüm

1. Create an empty stack
2. Convert the input infix string to a list
3. Scan the token list from left to right.
  - If the token is an operand, append it to the end of the output list.
  - If the token is a **left parenthesis**, push it on the stack
  - If the token is a **right parenthesis**, pop the stack
  - If the token is an operator, \*, /, +, or -, push it on the stack. However, first remove any operators already on the stack that have **higher or equal precedence (sadece yüksek de olabilir)** and append them to the output list.
  - When the input expression has been completely processed, check the stack

<http://interactivepython.org/runestone/static/pythonds/BasicDS/InfixPrefixandPostfixExpressions.html>

---

**Algorithm InfixToPostFix ( $I$ )**

Transform an infix expression  $I$  to a postfix expression  $P$

---

```
create an empty stack  $S$ ;  
 $P \leftarrow$  empty expression;  
 $index \leftarrow 1$ ;  
while we have not reached the end of  $I$  do  
   $ch \leftarrow I[index]$ ; {store in  $ch$  the next character in  $I$ }  
  if  $ch$  is an operand then  
    append  $ch$  to the end of  $P$ ;  
  else if  $ch$  is a '(' then  
    push  $ch$  onto  $S$ ;  
  else if  $ch$  is a ')' then  
    repeat  
      pop operators from  $S$  and append them to  $P$ ;  
    until a '(' is popped;  
  else { $ch$  is an operator}  
    while  $S$  is not empty and top of  $S$  is not '(' and top of  $S$  is not a lower precedence operator do  
      pop operators from  $S$  and append them to  $P$ ;  
    end while  
    push  $ch$  onto  $S$ ;  
  end if  
   $index \leftarrow index + 1$ ;  
end while
```

# Operator öncelikleri

- operator precedence (and associativity) is
- -lowest:  $+$ ,  $-$
- (left to right, e.g.,  $1-2-3 = (1-2)-3$  )
- -middle:  $*$ ,  $/$  (left to right, e.g.,  $1/2/3 = (1/2)/3$  )
- - highest:  $^$  (right to left, e.g.,  $1^2^3 = 1^(2^3)$  )

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$

- Çıkış:

- Stack:

# Örnek: Infix'den postfix'e dönüşüm

- **(10+5\*3-16/2^3)\*(5+7)**



- Çıkış:

- Stack: (

- Stack geçmişi:
- Stack: (

# Örnek: Infix'den postfix'e dönüşüm

- $(\mathbf{10}+5*3-16/2^3)*(5+7)$



- Çıkış: **10**

- Stack: (

- Stack geçmişi:
- Stack: (



# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10

- Stack: ( +

- Stack geçmişi:
- Stack: (
- Stack: (+

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5

- Stack: (+

- Stack geçmişi:
- Stack: (
- Stack: (+

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5

- Stack: (+ \*

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3

- Stack: (+ \*

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3

- Stack: (+ \*

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*

- operatörü yığta itilmeden önce daha yüksek öncelikli \* operatörü çıkışa yazılmalı. Eşit veya daha düşük öncelikliler yığta kalıyor. Bazı algoritmalar eşit önceliklileri de çıkartıyor. Bu sonucu değiştirmiyor.

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \*

- Stack: (+

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+

- operatörü yığta itilmeden önce daha yüksek öncelikli \* operatörü çıkışa yazılmalı. Eşit veya daha düşük öncelikliler yığta kalıyor. Bazı algoritmalar eşit önceliklileri de çıkartıyor. Bu sonucu değiştirmiyor.

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \*

- Stack: (+ -

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+
- Stack: (+ -

- operatörü yığita itilmeden önce daha yüksek öncelikli \* operatörü çıkışa yazılmalı. Eşit veya daha düşük öncelikliler yığitta kalıyor. Bazı algoritmalar eşit önceliklileri de çıkartıyor. Bu sonucu değiştirmiyor.

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16

- Stack: (+ -

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+
- Stack: (+ -



# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16

- Stack: (+ - /

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+
- Stack: (+ -
- Stack: (+ - /

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2

- Stack: (+ - /

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+
- Stack: (+ -
- Stack: (+ - /

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2

- Stack: (+ - / ^

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3

- Stack: (+ - / ^

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3

- Stack: (+ - / ^

Sağ parantez okunduğu zaman, sol parantez yığıttan silinene kadar aradaki operatörler çıkışa yazdırılır.

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^

- Stack: (+ - /

Sağ parantez okunduğu zaman, sol parantez yığıttan silinene kadar aradaki operatörler çıkışa yazdırılır.

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^ /

- Stack: (+ -

Sağ parantez okunduğu zaman, sol parantez yığıttan silinene kadar aradaki operatörler çıkışa yazdırılır.

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^ / -

- Stack: (+

Sağ parantez okunduğu zaman, sol parantez yığıttan silinene kadar aradaki operatörler çıkışa yazdırılır.

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack: (+



# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^ / - +

- Stack: (

Sağ parantez okunduğu zaman, sol parantez yığıttan silinene kadar aradaki operatörler çıkışa yazdırılır.

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack: (

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^ / - +

- Stack:

Sağ parantez okunduğu zaman, sol parantez yığıttan silinene kadar aradaki operatörler çıkışa yazdırılır.

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^ / - +

- Stack:

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: \*

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^ / - +

- Stack: \* (

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: \*
- Stack: \*(

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^ / - + 5

- Stack: \* (

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: \*
- Stack: \*(

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^ / - + 5

- Stack: 

* ( +
-------

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: \*
- Stack: \*(
- Stack: \*(+

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^ / - + 5 7

- Stack: \* ( +

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: \*
- Stack: \*(
- Stack: \*(+

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^ / - + 5 7

- Stack: \* ( +

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: \*
- Stack: \*(
- Stack: \*(+



# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^ / - + 5 7 +

- Stack: \* (

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: \*
- Stack: \*(
- Stack: \*(+
- Stack: \*(

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$



- Çıkış: 10 5 3 \* 16 2 3 ^ / - + 5 7 +

- Stack: \*

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: \*
- Stack: \*(
- Stack: \*(+
- Stack: \*(
- Stack: \*

# Örnek: Infix'den postfix'e dönüşüm

- $(10+5*3-16/2^3)*(5+7)$

- Çıkış: **10 5 3 \* 16 2 3 ^ / - + 5 7 + \***

- Stack:

- Stack geçmişi:
- Stack: (
- Stack: (+
- Stack: (+ \*
- Stack: (+
- Stack: (+ -
- Stack: (+ - /
- Stack: (+ - / ^
- Stack: (+ - /
- Stack: (+ -
- Stack:
- Stack: \*
- Stack: \*(
- Stack: \*(+
- Stack: \*(
- Stack: \*
- Stack:

✓ Dönüşüm  
tamamlandı

# Örnek:

Infix:  $A * (B + C * D) + E$

Postfix:  $A B C D * + * E +$

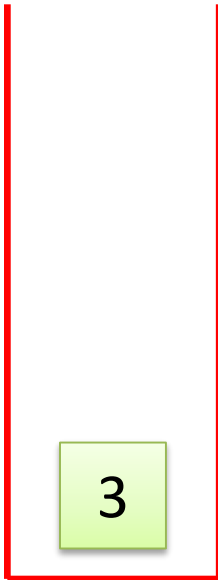
	current symbol	operator stack	postfix string
1	A		A
2	*	*	A
3	(	* (	A
4	B	* (	A B
5	+	* ( +	A B
6	C	* ( +	A B C
7	*	* ( + *	A B C
8	D	* ( + *	A B C D
9	)	*	A B C D * +
10	+	+	A B C D * + *
11	E	+	A B C D * + * E
12			A B C D * + * E +

# Örnekler

- Infix:  $(5+4-6)*(8+10)/((1+5)*(5-2))$
- Postfix:  $5\ 4\ 6\ -\ +\ 8\ 10\ +\ 1\ 5\ +\ 5\ 2\ -\ *\ / *$
- Infix:  $5-16/(4*2^2)$
- Postfix:  $5\ 16\ 4\ 2\ 2\ \wedge\ *\ /\ -$
- Infix:  $5-16/4*2^2$
- Postfix:  $5\ 16\ 4\ /\ 2\ 2\ \wedge\ *\ -$
- Infix:  $(5-16)/4*2^2$
- Postfix:  $5\ 16-\ 4\ /\ 2\ 2\ \wedge\ *$

# Bir postfix ifadenin hesaplanması

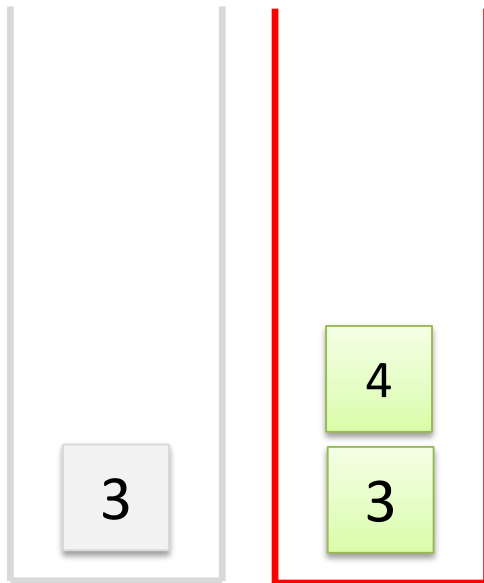
- Örnek: 3 4 + 5 6 \* 9 2 - + \*



Elemanlar yığta itilirken, ifade içerisinde bir operatöre sıra geldiği zaman yığt içerisindeki son iki ifade üzerinde işlem gerçekleştirilir. Son iki eleman çıkartılır ve sonuç tekrar yığta yazılır.

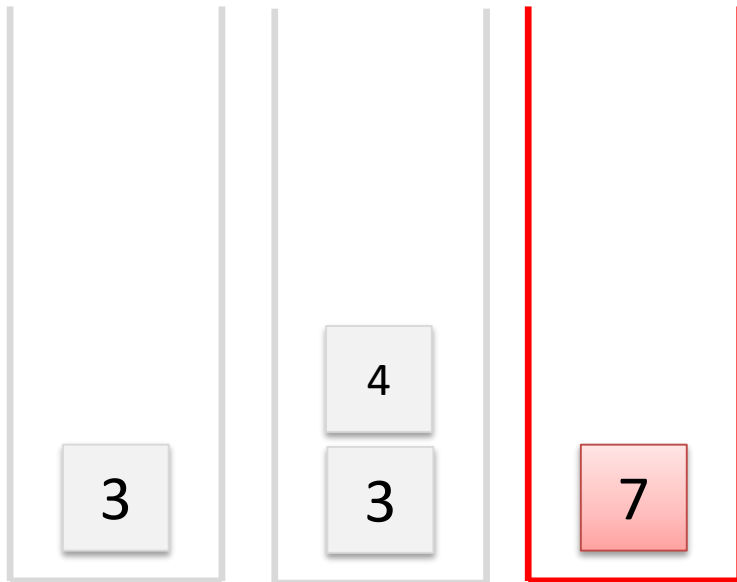
# Bir postfix ifadenin hesaplanması

- Örnek: 3 4 + 5 6 \* 9 2 - + \*



# Bir postfix ifadenin hesaplanması

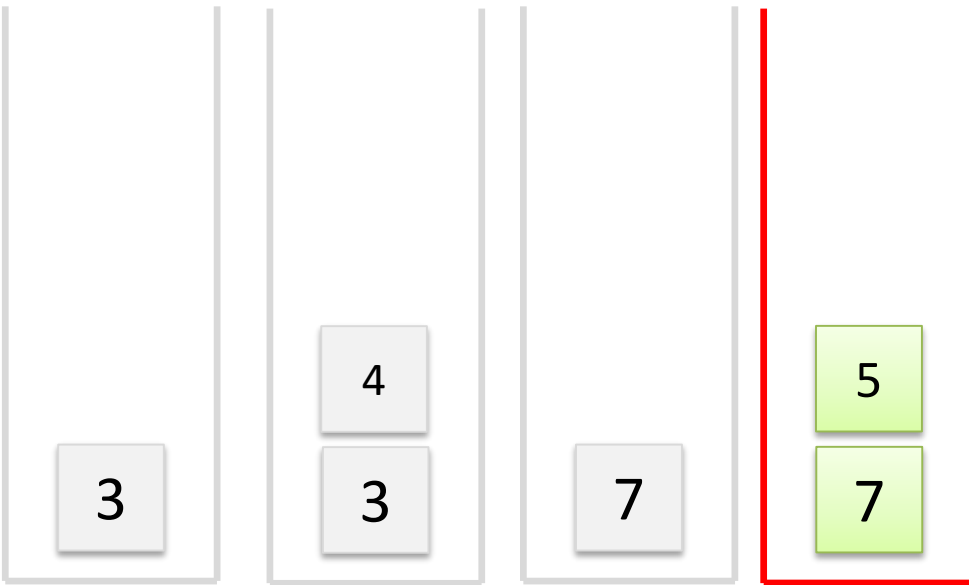
- Örnek: 3 4 + 5 6 \* 9 2 - + \*





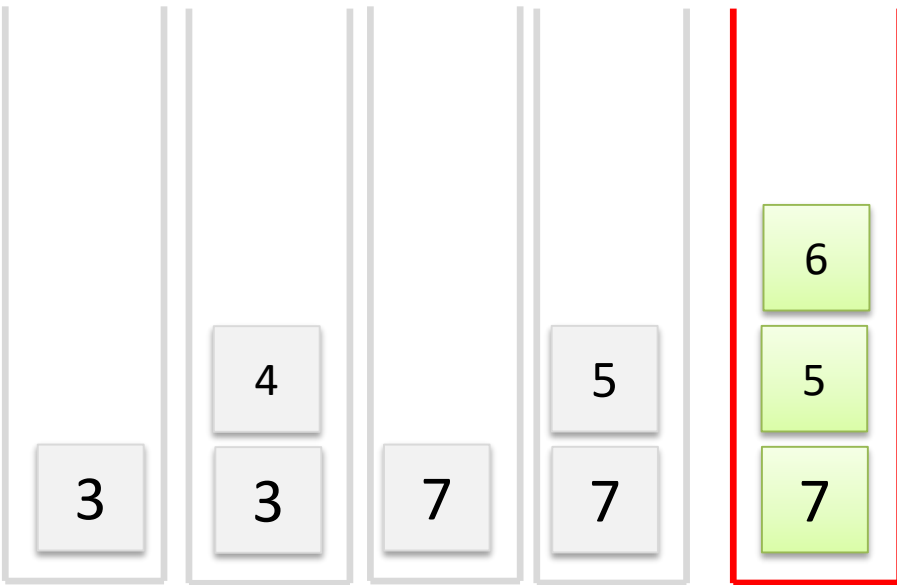
# Bir postfix ifadenin hesaplanması

- Örnek: 3 4 + 5 6 \* 9 2 - + \*



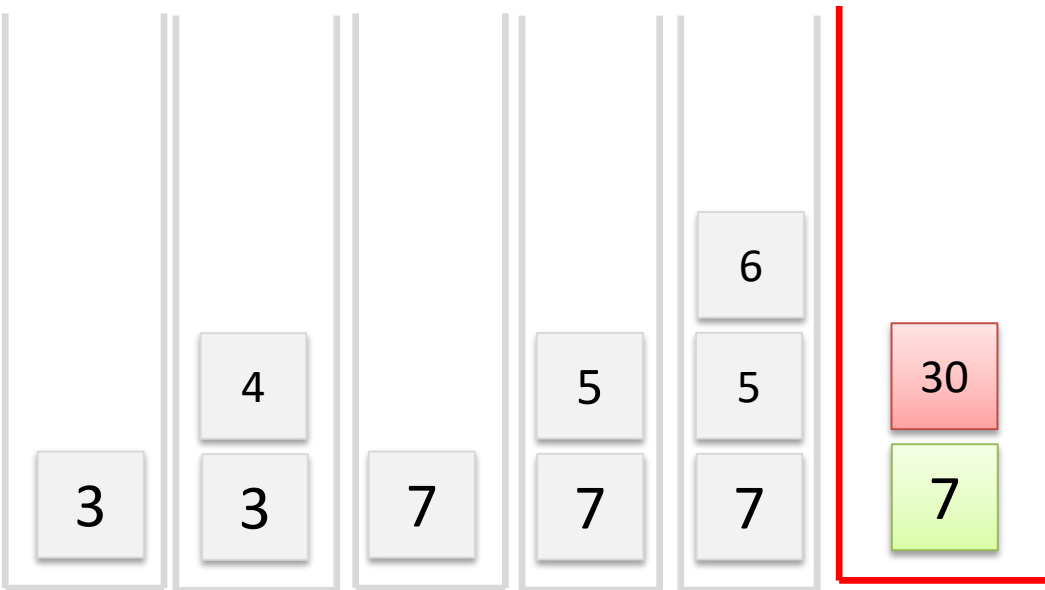
# Bir postfix ifadenin hesaplanması

- Örnek: 3 4 + 5 6 \* 9 2 - + \*



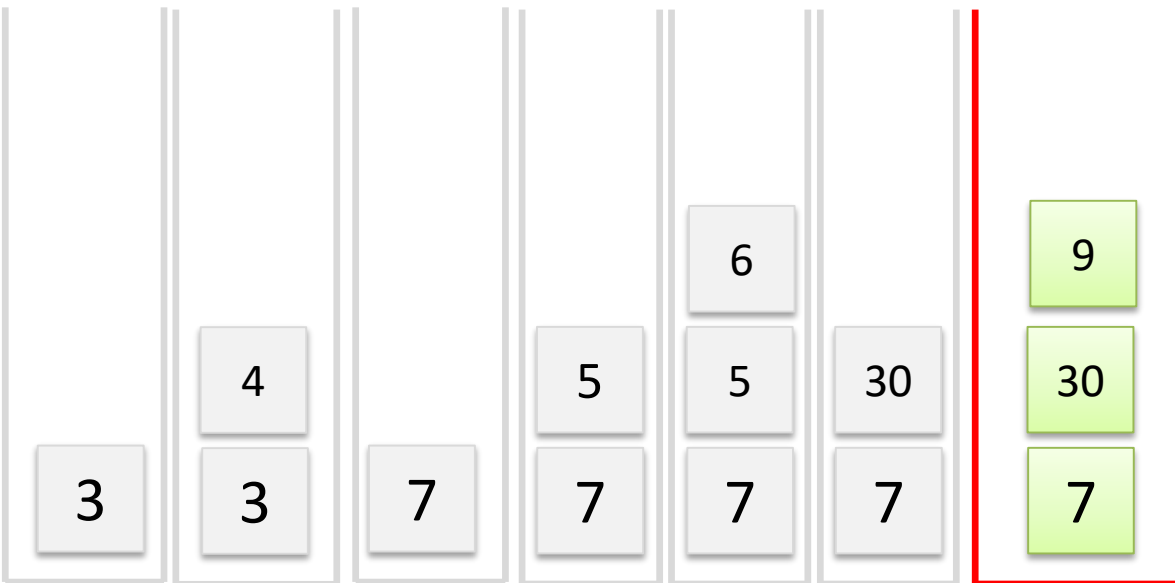
# Bir postfix ifadenin hesaplanması

- Örnek: 3 4 + 5 6 \* 9 2 - + \*



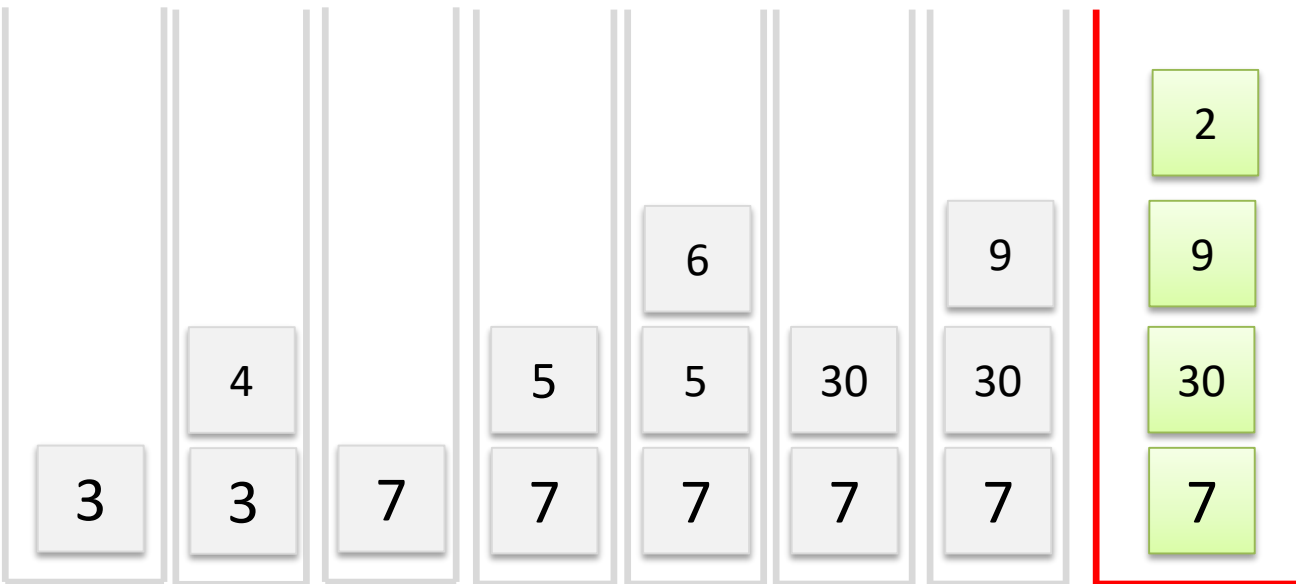
# Bir postfix ifadenin hesaplanması

- Örnek: 3 4 + 5 6 \* 9 2 - + \*



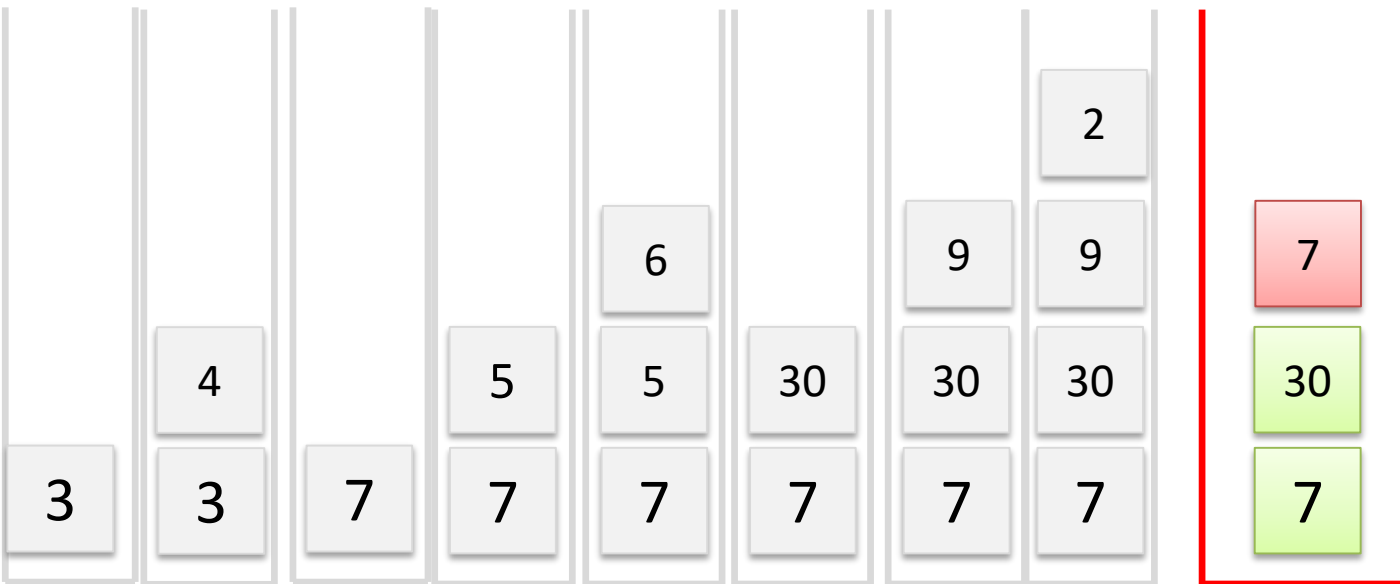
# Bir postfix ifadenin hesaplanması

- Örnek: 3 4 + 5 6 \* 9 2 - + \*



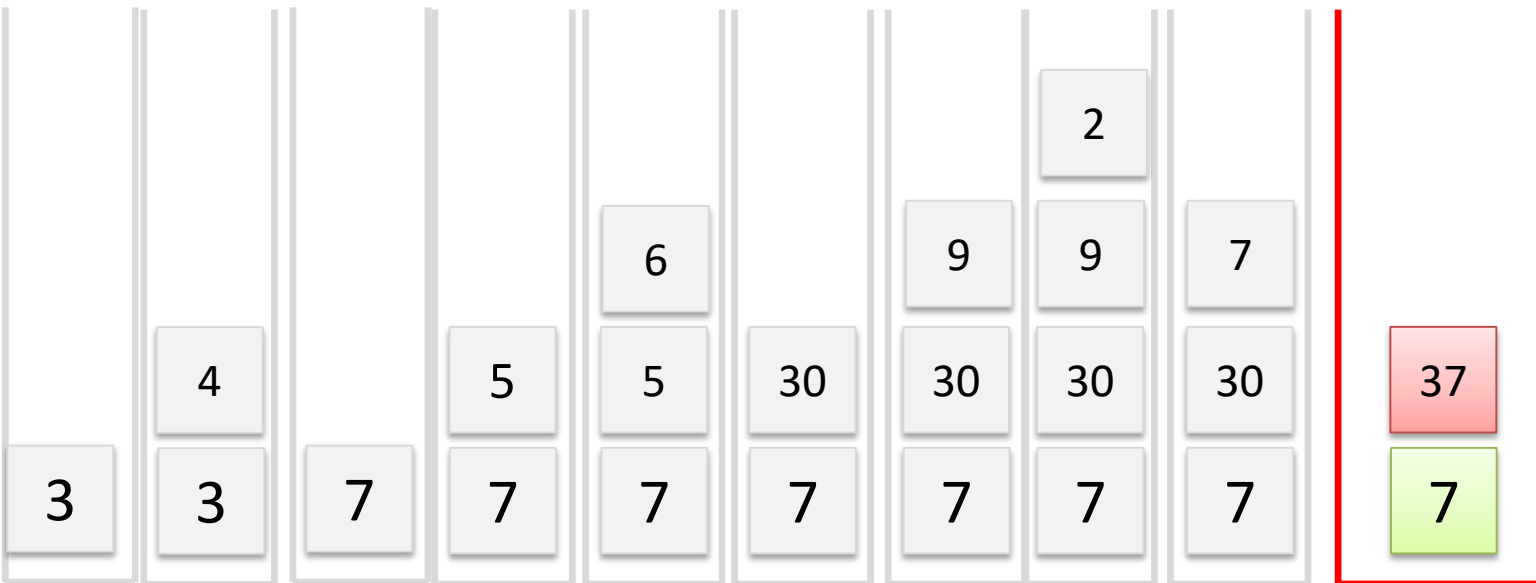
# Bir postfix ifadenin hesaplanması

- Örnek: 3 4 + 5 6 \* 9 2 - + \*



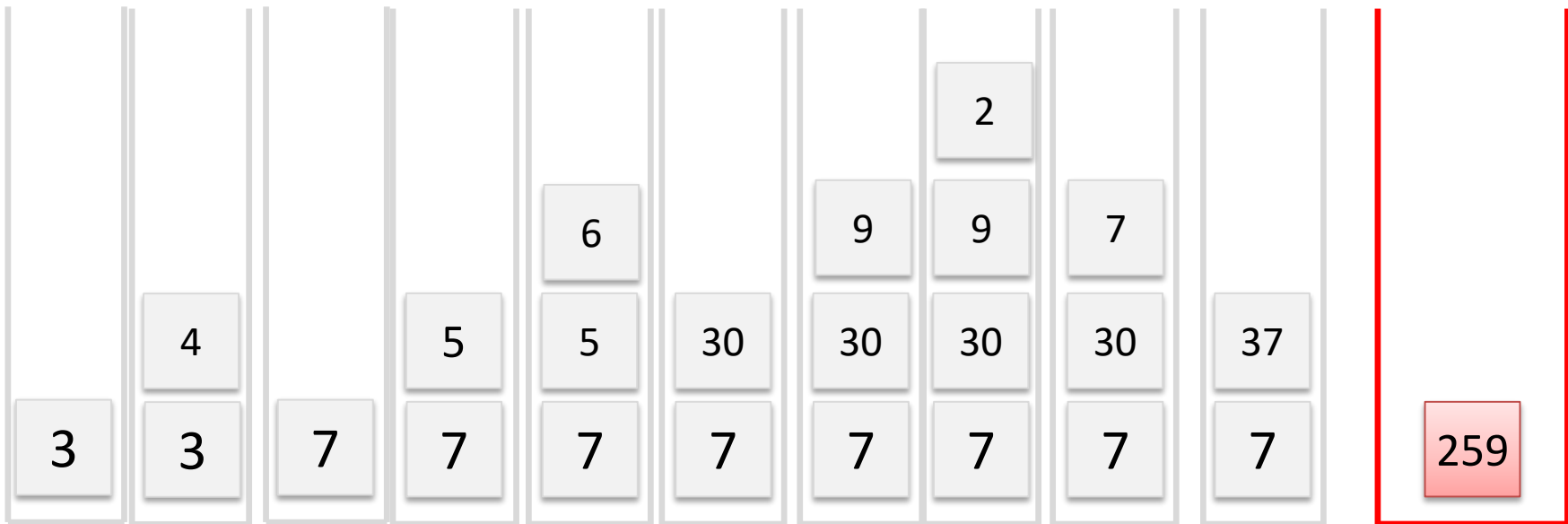
# Bir postfix ifadenin hesaplanması

- Örnek:  $3\ 4\ +\ 5\ 6\ *\ 9\ 2\ -\ +\ *$



# Bir postfix ifadenin hesaplanması

- Örnek: 3 4 + 5 6 \* 9 2 - + \*





# Algoritma- Bir postfix ifadenin hesaplanması

- Suppose P is an arithmetic expression in postfix notation. We will evaluate it using a stack to hold the operands.

Start with an empty stack. We scan P from left to right.

While (we have not reached the end of P)

    If an operand is found

        push it onto the stack

    End-If

    If an operator is found

        Pop the stack and call the value A

        Pop the stack and call the value B

        Evaluate B op A using the operator just found.

        Push the resulting value onto the stack

    End-If

End-While

Pop the stack (this is the final value)

- Notes:
- At the end, there should be only one element left on the stack.
- This assumes the postfix expression is valid.
- <http://faculty.cs.niu.edu/~hutchins/csci241/eval.htm>

# C++'de Stack ADT



## Example

```
1 // stack::push/pop
2 #include <iostream>           // std::cout
3 #include <stack>              // std::stack
4
5 int main ()
6 {
7     std::stack<int> mystack;
8
9     for (int i=0; i<5; ++i) mystack.push(i);
10
11     std::cout << "Popping out elements...";
12     while (!mystack.empty())
13     {
14         std::cout << ' ' << mystack.top();
15         mystack.pop();
16     }
17     std::cout << '\n';
18
19     return 0;
20 }
```

# Java ve C#'da stack

java.util

## Class Stack<E>

java.lang.Object  
  java.util.AbstractCollection<E>  
    java.util.AbstractList<E>  
      java.util.Vector<E>  
        java.util.Stack<E>

```
import java.util.Stack;

public class Program {
    public static void main(String[] args) {

        Stack<String> stack = new Stack<>();
        stack.push("fly");
        stack.push("worm");
        stack.push("butterfly");

        // Peek at the top of the stack.
        String peekResult = stack.peek();
        System.out.println(peekResult);

        // Pop the stack and display the result.
        String popResult = stack.pop();
        System.out.println(popResult);

        // Pop again.
        popResult = stack.pop();
        System.out.println(popResult);
    }
}
```

### Output

butterfly  
butterfly  
worm

<http://www.dotnetperls.com/stack-java>

Namespace: System.Collections

Assembly: mscorlib (in mscorlib.dll)

## Inheritance Hierarchy

```
using System;
using System.Collections;
public class SamplesStack {

    public static void Main() {

        // Creates and initializes a new Stack.
        Stack myStack = new Stack();
        myStack.Push("Hello");
        myStack.Push("World");
        myStack.Push("!");

        // Displays the properties and values of the Stack.
        Console.WriteLine( "myStack" );
        Console.WriteLine( "\tCount:    {0}", myStack.Count );
        Console.Write( "\tValues:" );
        PrintValues( myStack );
    }

    public static void PrintValues( IEnumerable myCollection ) {
        foreach ( Object obj in myCollection )
            Console.Write( "    {0}", obj );
        Console.WriteLine();
    }
}
```

[https://msdn.microsoft.com/en-us/library/system.collections.stack\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.collections.stack(v=vs.110).aspx)


BSM 207 - VERİ YAPILARI

# Uygulama

```
int main(int argc, char** argv) {
    stack<int> stack1;
    for (int i=1;i<=5;i++){
        stack1.push(i*10);
    }

    //stack1'i stack'ye gönder
    stack<int> stack2;
    while (!stack1.empty()) {
        stack2.push(stack1.top());
        stack1.pop();
    }

    cout<<"Stack içeriği"<<endl;
    while (!stack2.empty()) {
        cout<<"stack1.top()="<<stack2.top()<<endl;
        stack2.pop();
    }
}
```



Yanda verilen  
programın  
ekran çıktısı  
ne olur?