BMB2006
VERİ YAPILARI

Doç. Dr. Murtaza CİCİOĞLU

Bursa Uludağ Üniversitesi

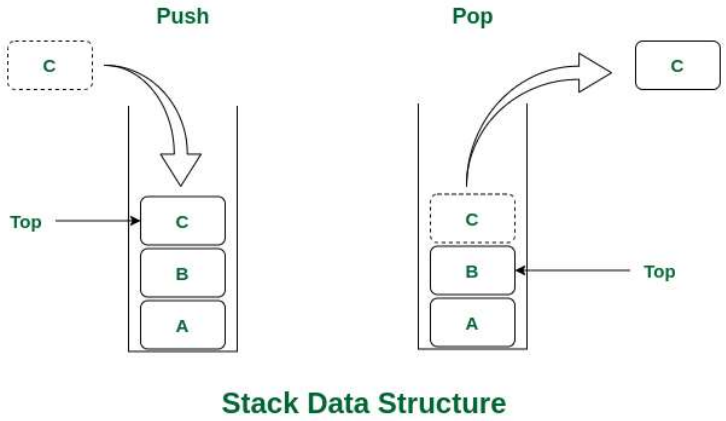
Bilgisayar Mühendisliği Bölümü

Bu ders notları Olcay Taner Yıldız ile Erkan TANYILDIZI hocaların notları kullanılarak hazırlanmıştır.

Hafta 5: Stack

Amaç:

- Stack çalışma yapısı
- Stack kullanım alanları



Stack Data Structure

Yol haritası:

- Giriş
- Dizi ile Stack Tanımı
- Baglı Liste ile Stack Tanımı
- Infix, Postfix, & Prefix Gösterimleri

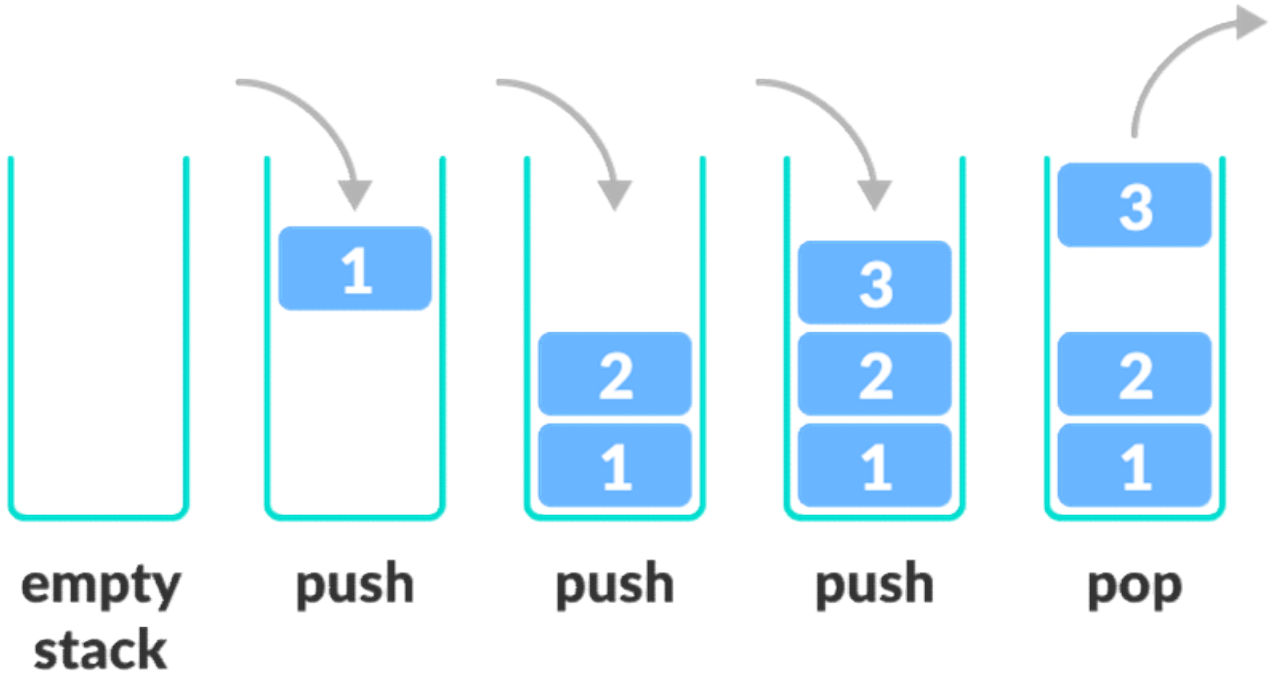
Giriş

- Son giren ilk çıkar (Last In First Out-LIFO) veya İlk giren son çıkar (First-in-Last-out FILO) mantığıyla çalışır.
- Eleman ekleme çıkarmaların en üstten (top) yapıldığı veri yapısına yığıt (stack) adı verilir.
- Bir eleman ekleneceğinde yığıtın en üstüne konulur. Bir eleman çıkarılacağı zaman yığıtın en üstündeki eleman çıkarılır.
- Bu eleman da yığıttaki elemanlar içindeki en son eklenen elemandır. Bu nedenle yığıtlara LIFO (Last In First Out Son giren ilk çıkar) listesi de denilir.

Giriş

- Stack yapısını gerçekleştirmek için 2 yol vardır.
  - Dizi kullanmak
  - Bağlantılı liste kullanmak

- Stack İşlemleri
  - push(nesne)
  - pop()
  - top()
  - size()
  - isEmpty()

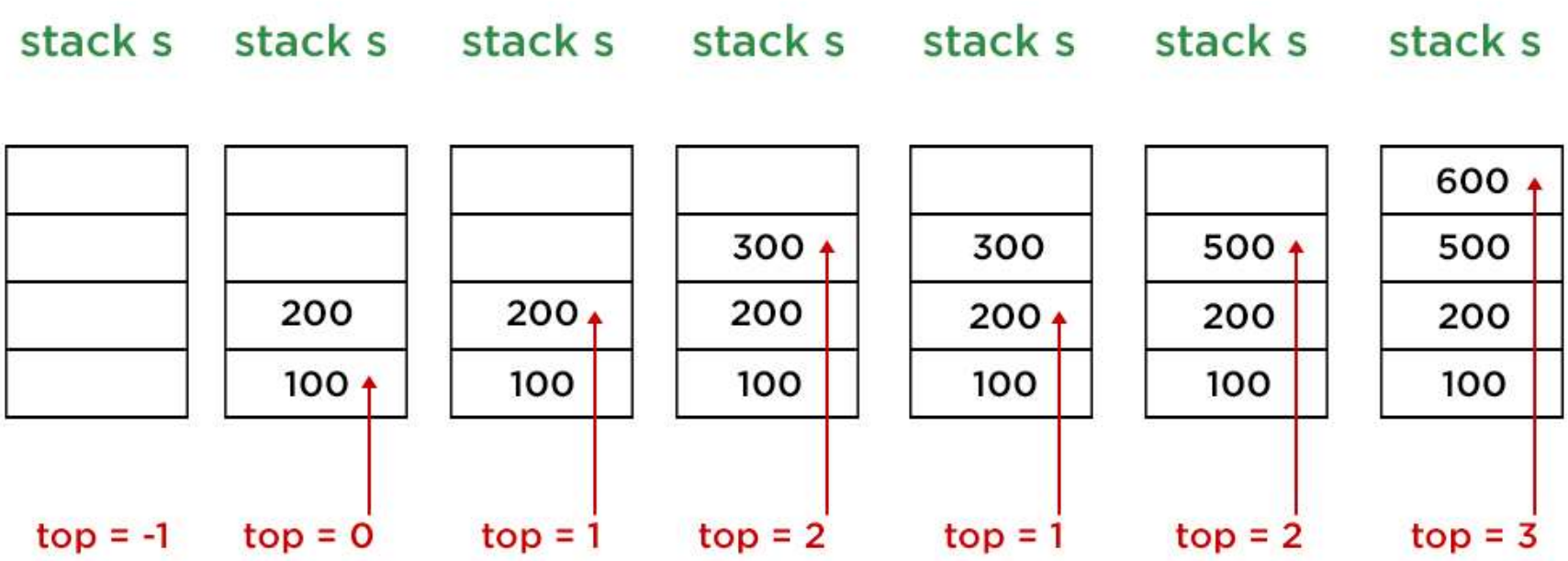


Kullanım alanları

- Yazılım uygulamalarındaki Undo işlemleri stack ile yapılır. Undo işlemi için LIFO yapısı kullanılır.
- Donanım uygulamaları için mikroişlemcilerde stack pointer
- Web browser'lardaki Back butonu (önceki sayfaya) stack kullanır. Burada da LIFO yapısı kullanılır.
- Matematiksel işlemlerdeki operatörler (+,*,/,- gibi) ve operandlar için stack kullanılabilir.
- Yazım kontrolündeki parantezlerin kontrolünde stack kullanılabilir
- (A+B)-C*(D+1)) → parantez kontrolü

Dizi tabanlı

- Bir stack gerçeklemenin en kolay yolu dizi kullanmaktır. Stack yapısı dizi üzerinde en fazla N tane eleman tutacak şekilde yapılabilir.



Stack Kod Uygulaması

- Stack bağlı liste uygulaması (C dilinde)
- Stack dizi uygulaması (C dilinde)

Stack ile problem çözümü

- Matematiksel ifadelerde iç içe yuvalanmış parantezlerin doğruluğunun kontrolü
- Kural 1: Eşit sayıda sağ ve sol parantez
- Kural 2: Bir sağ parantezden önce mutlaka bir sol parantez olmalı
- *(c+d)-(k-3/(c+x)*
- *(c+x)+d)*(a+b)/(b+c)-2)*
- *a-(3+(b*(c+d)-(k-3)/(c+x)+d)*(a+(b+c)-2)+5*c)*
- Bir değişken tanımlayıp parantez sayısına göre kontrol edebiliriz. Son değer 0 olmalı

(c + d) - (k – 3 / (c + x)

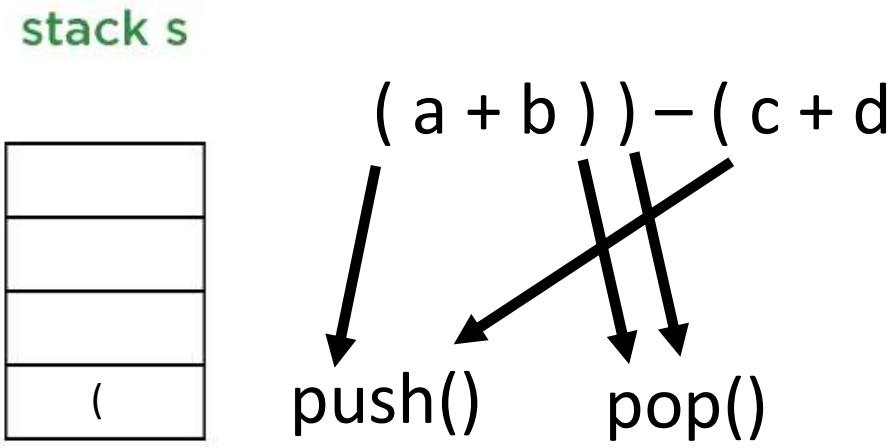
11 1 10 011 1 11 2 2 2 2 **1**

Stack ile problem çözümü

- Değişken son eleman olarak 0 değerini aldı. Kurala uyuyor!

(a + b)) – (c + d

11 1 10-1-1000 **0**



Infix Gösterimi

- Genellikle cebirsel işlemleri şu şekilde ifade ederiz: a + b
- Buna infix gösterim adı verilir, çünkü operatör (“+”) ifade içindedir.
- Problem: Daha karmaşık cebirsel ifadelerde parantezlere ve öncelik kurallarına ihtiyaç duyulması.
- a + b * c = (a + b) * c ?
- = a + (b * c) ?

Infix, Postfix, & Prefix Gösterimleri

- Operatör Önde (Prefix) : + a b
  - İşlem sağdan sola doğru ilerler. Öncelik (parantez) yoktur.
- Operatör Arada (Infix) : a + b
  - İşlem öncelik sırasına göre ve soldan sağa doğru ilerler.
- Operatör Sonda (Postfix) : a b +
  - İşlem soldan sağa doğru ilerler. Öncelik (parantez) yoktur.

- *i++*
- *++i*
- *&p*

Infix, Postfix, & Prefix Gösterimleri

- İşlem önceliği (büyükten küçüğe)
  - Parantez
  - Üs Alma
  - Çarpma /Bölme
  - Toplama/Çıkarma

- Parantezsiz ve aynı önceliğe sahip işlemcilerde soldan sağa doğu yapılır (üs hariç).
- Üs almada sağdan sola doğrudur. A-B+C'de öncelik (A-B)+C şeklindedir.
- A^B^C'de ise A^(B^C) şeklindedir. (parantezler öncelik belirtmek için konulmuştur)

Infix, Postfix, & Prefix Gösterimleri

- A+B*C → postfix gösterimi ??
- (A+B)*C → postfix gösterimi ??
- A*B/C → postfix gösterimi ??

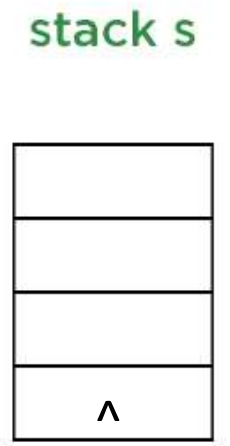
Postfix

$ab/c - de * + ac * -$ arka gösteriminin değerinin hesaplanması aşağıda gösterilmiştir.

T_1 = a / b	T_1 c - d e * + a c * -
T_2 = T_1 - c	T_2 d e * + a c * -
T_3 = d * e	T_2 T_3 + a c * -
T_4 = T_2 + T_3	T_4 a c * -
T_5 = a * c	T_4 T_5 -
T_6 = T_4 - T_5	T_6

Infix, Postfix, & Prefix Gösterimleri

- A^B*C-D+E/F/(I-J) → postfix gösterimi ??

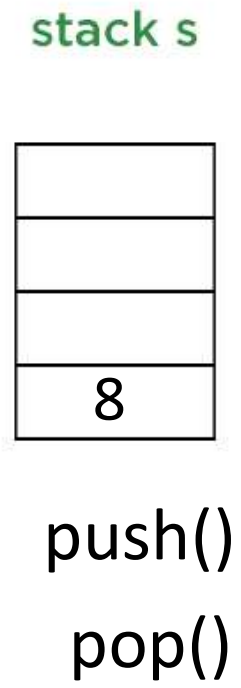


push()

pop()

Infix, Postfix, & Prefix Gösterimleri

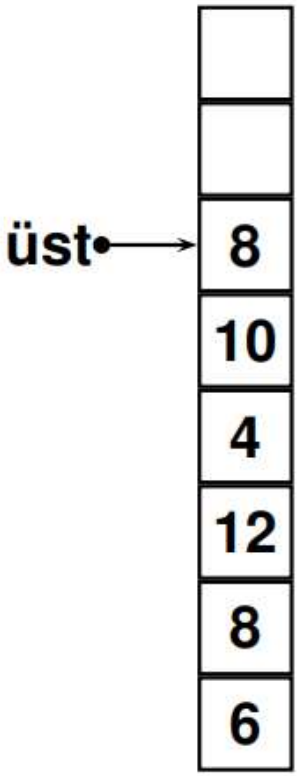
- 845+-493/+*2^3+ → postfix gösteriminin sonucu ??



Dizi tabanlı

Sabit dizi ile tanımlı bir stack ait elemanların tanımı

```
public class node{
        public int veri;
        public node(int veri) {
            this. veri = veri;
        }}
```



Dizi tabanlı

Tam sayılar içeren bir çıkının sabit dizi ile tanımı

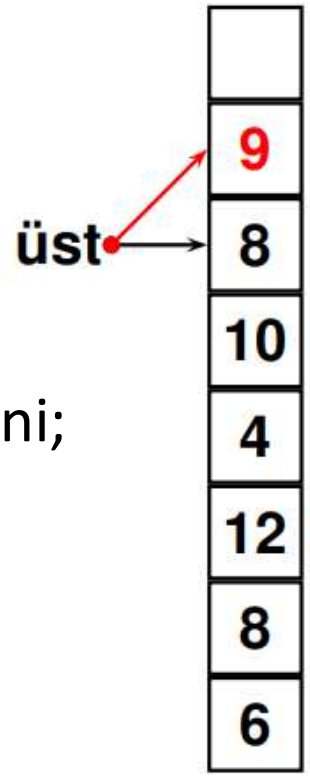
```
public class stack{
        public int ust;
        public int boyut;
        public node[] dizi;

        public stack(int boyut) {
            this.boyut = boyut;
            dizi=new node[boyut];
            ust=-1;
        }
}
```

```
node peek(){
        return dizi[ust];}

boolean dolu(){
if(ust == boyut-1)
            return true;
else
            return false;}

boolean bos(){
if(ust == -1)
            return true;
else
            return false;}
```

Dizi tabanlı

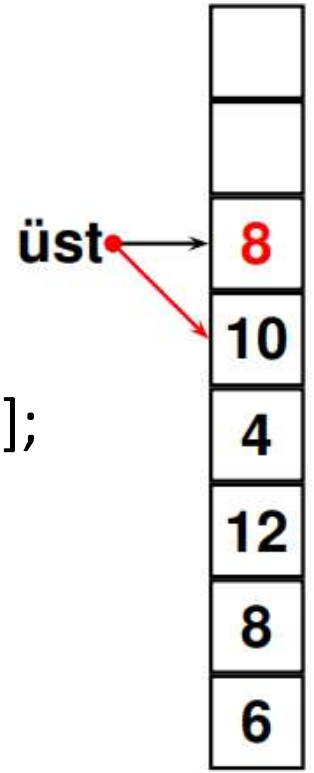
yeni bir eleman ekleme

```
void push(node yeni){
        if(!dolu()){
                ust++;
                dizi[ust]=yeni;
        }
}
```



eleman silme

```
node pop(){
        if(!bos()){
                ust--;
                return dizi[ust+1];
        }
        return null;}
```



Dizi tabanlı

- Ekleme: O(1)
- Silme: O(1)
- Sınırlamalar
  - Yığının en üst sayısı önceden tanımlanmalıdır ve değiştirilemez.
  - Dolu bir yığına yeni bir nesne eklemeye çalışmak istisnai durumlara sebep olabilir.

Dizi tabanlı

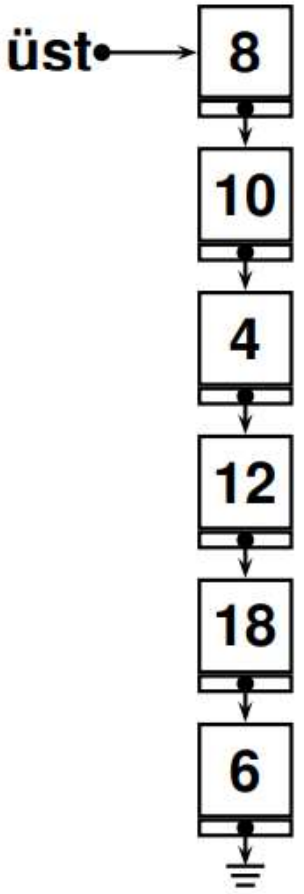
- Büyüyebilir Dizi Tabanlı Yığın Yaklaşımı
- push işlemi esnasında dizi dolu ise bir istisnai durum bildirimi geri dönmektense yığının tutulduğu dizi daha büyük bir dizi ile yer değiştirilir.
- Yeni dizi ne kadar büyüklükte olmalı?
  - Artımlı strateji: yığın büyüklüğü sabit bir c değeri kadar arttırılır.
  - İkiye katlama stratejisi: önceki dizi boyutu iki kat arttırılır
- Bağlı liste yapılarını kullanarak yığın işlemini gerçekleştirmek bu tür problemlerin önüne geçmede yararlı olur.

Baglı Liste ile Stack Tanımı

- Tam sayılar içeren bir stack için bağlı liste ile tanımı

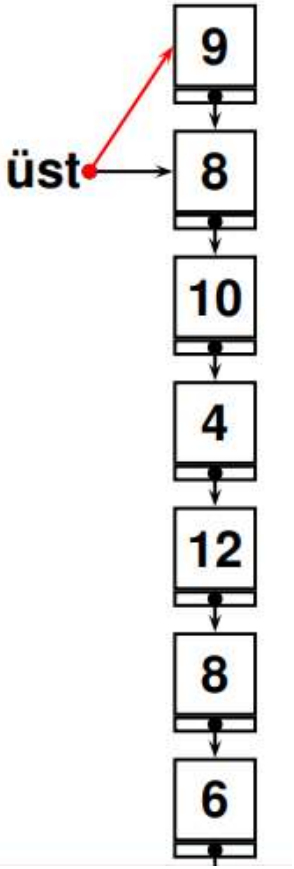
```
public class stack{
        node ust;

        public stack() {
            ust=null;
        }
}
```

```
node peek(){
        return ust.veri;}

boolean bos(){
     if(ust == null)
          return true;
     else
          return false;}
```



Baglı Liste ile Stack Tanımı

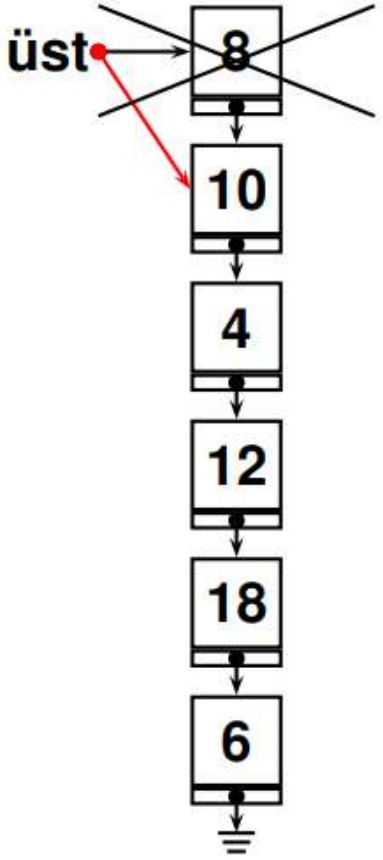
- eleman ekleme




```
void push(node yeni){
        yeni.ileri=ust;
        ust=yeni;
 }
```

Ekleme: O(1)
Silme: O(1)

eleman silme




```
void pop(){
    node temp=ust;
    if(!bos())
        ust=ust.ileri;
    return temp;}
```