BIL207 VERİ YAPILARI
3. Hafta
Yığıt/Yığın/(Stacks)

Doç. Dr. Sercan YALÇIN

Yığıt/Yığın (Stack)

- 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.

Yığıt/Yığın (Stack)

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

- **empty stack:** Boş yığıt
- **push (koy):**Yığıta eleman ekleme.
- **pop (al):**Yığıttan eleman çıkarma

Yığın İşlemleri

- **Ana yığın işlemleri:**
  - **push(nesne):** yeni bir nesne ekler
    - **Girdi**: Nesne **Çıktı**: Yok
  - **pop():** en son eklenen nesneyi çıkarıp geri döndürür.
    - **Girdi**: Yok **Çıktı**: Nesne

- **Yardımcı yığın işlemleri:**
  - **top():** en son eklenen nesneyi çıkarmadan geri döndürür.
    - **Girdi**: Yok **Çıktı**: Nesne
  - **size():** depolanan nesne sayısını geri döndürür.
    - **Girdi**: Yok **Çıktı**: Tamsayı
  - **isEmpty**(): yığında nesne bulunup bulunmadığı bilgisi geri döner.
    - **Girdi**: Yok **Çıktı**: Boolean

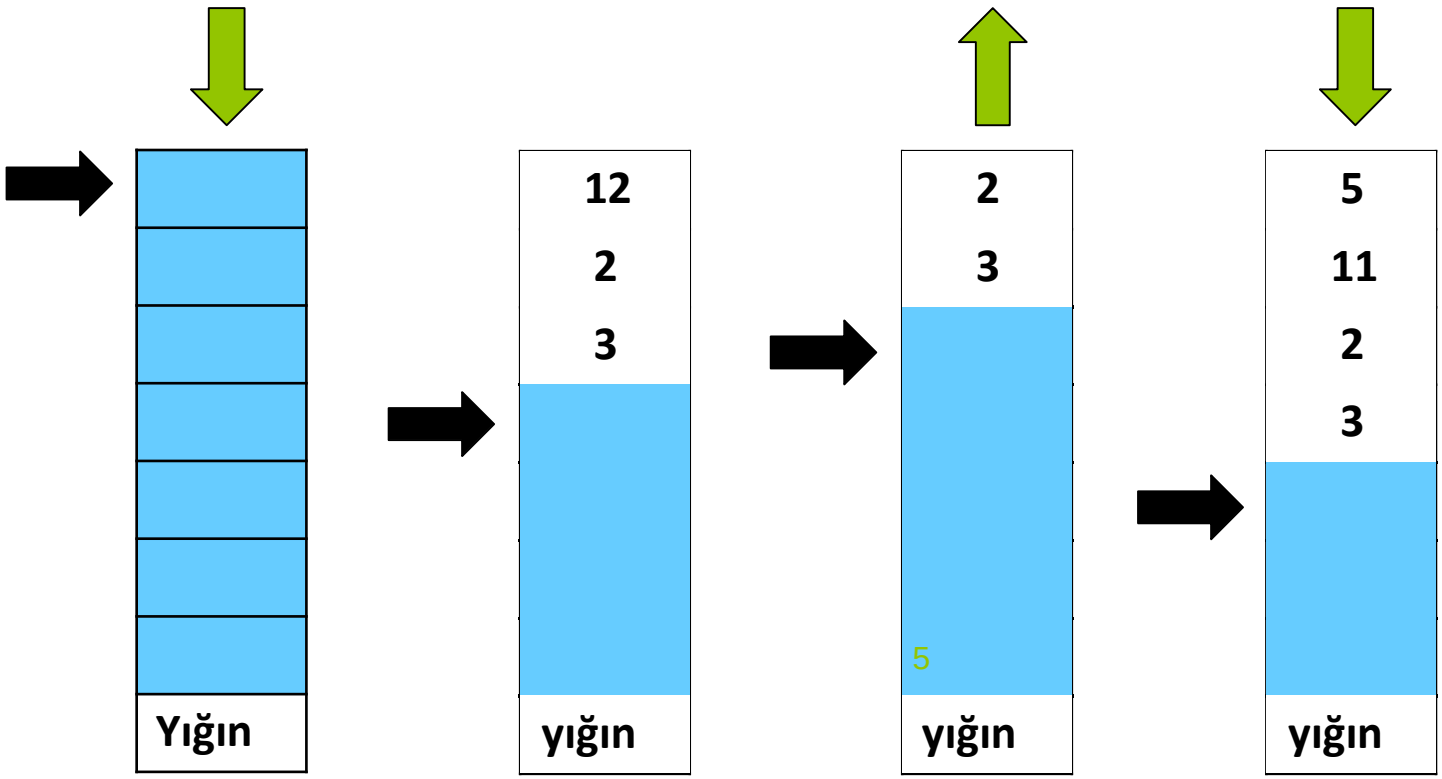
Yığın (stack) Yapısı

push(3);
push(2);
push(12);

pop();

12

push(11);
push(5);



Yığıt/Yığın (Stack)

- Örnek kullanım yerleri
  - Yazılım uygulamalarındaki Undo işlemleri stack ile yapılır. Undo işlemi için LIFO yapısı kullanılır.
  - Web browser'lardaki Back butonu (önceki sayfaya) stack kullanır. Buradada LIFO yapısı kullanılır.
  - Matematiksel işlemlerdeki operatörler (+,*,/,- gibi) ve operandlar için stack kullanılabilir.
  - Yazım kontrolündeki parantezlerin kontrolünde stack kullanılabilir.

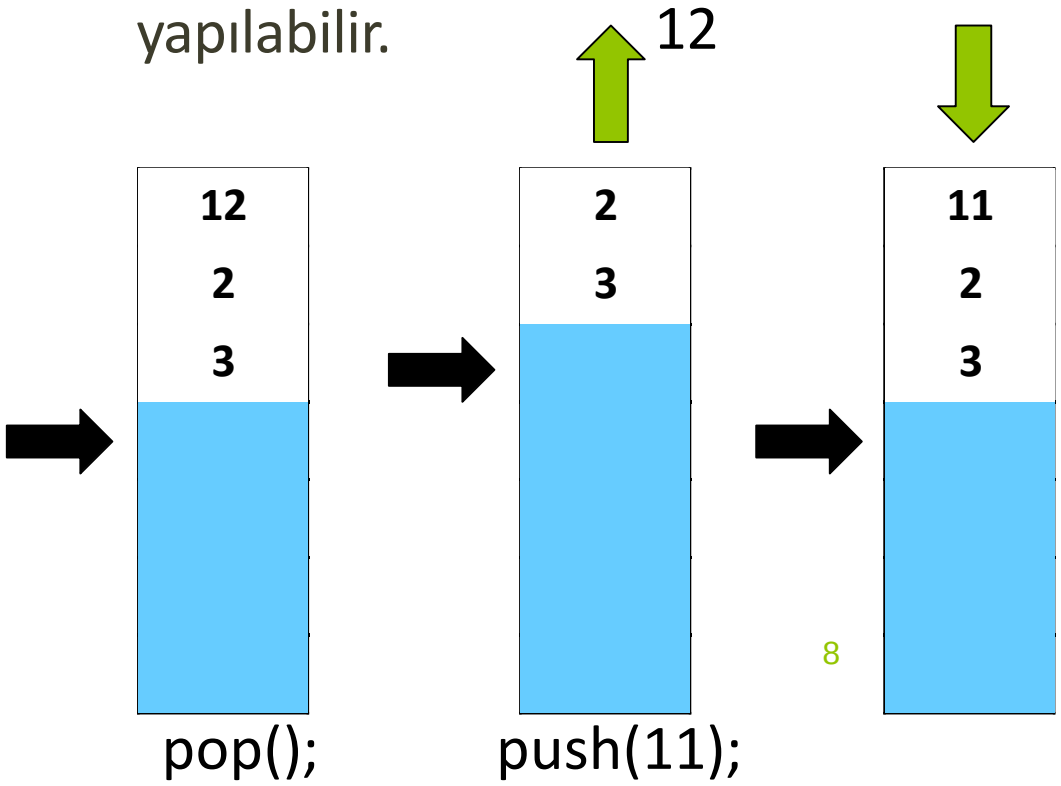
Yığıt/Yığın (Stack)

- **Örnek : Yığına ekleme ve çıkarma**

İşlem	Yığıt (tepe)	Çıktı
push(“M”);	M	
push(“A”);	MA	
push(“L”);	MAL	
push(“T”);	MALT	
pop();	MAL	T
push(“E”);	MALE	T
pop();	MAL	TE
push(“P”);	MALP	TE
pop();	MAL	TEP
push(“E”);	MALE	TEP
pop();	MAL	TEPE

Dizi Tabanlı Yığın

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



Dizi Tabanlı Yığın

- Nesneleri soldan sağa doğru ekleriz. Bir değişken en üstteki nesnenin index bilgisini izler. Eleman çıkarılırken bu index değeri alınır.

```
Algorithm size()
    return t + 1

Algorithm pop()
    if isEmpty() then
        throw EmptyStackException
    else
        t ← t − 1
        return S[t + 1]
```

Dizi Tabanlı Yığın

- Yığın nesnelerinin saklandığı dizi dolabilir. Bu durumda push işlemi aşağıdaki mesajı verir.
- FullStackException (DoluYığınİstinası)
  - Dizi tabanlı yaklaşımın sınırlamasıdır.

```
Algorithm push(o)
  if t = S.length − 1 then
    throw FullStackException
  else
    t ← t + 1
    S[t] ← o
```

Başarım ve Sınırlamalar

- Başarım
  - n yığındaki nesne sayısı olsun
  - Kullanılan alan O(n)
  - Her bir işlem O(1) zamanda gerçekleşir.
- Sınırlamalar
  - Yığının en üst sayısı önceden tanımlanmalıdır ve değiştirilemez.
  - Dolu bir yığına yeni bir nesne eklemeye çalışmak istisnai durumlara sebep olabilir.

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.

```
Algorithm push(o)
    if t = S.length – 1 then
        A ← new array of
                size …
        for i ← 0 to t do
            A[i] ← S[i]
        S ← A
    t ← t + 1
    S[t] ← o
```

Yığın ve Operasyonları

```
public class Yigin {

  int kapasite=100;  // maksimum eleman sayısı
  int S[]; //Yığın elemanları – pozitif tam sayı
  int p;  // eleman sayısı

  public Yigin(){    // yapıcı yordam
    s[] = new int[kapasite];
    p = 0;
  }

  int koy(int item);
  int al();
  int ust();
  boolean bosmu();
  boolean dolumu();
}
```

Yığın Operasyonları – bosmu, dolumu

```
// yığın boşsa true döndür
public boolean bosmu() {
   if (p < 1) return true;
   else return false;
} //bitti-bosmu


// Yığın doluysa true döndür
public boolean dolumu(){
   if (p == kapasite-1) return true;
   else return false;
} // bitti-dolumu
```

Yığın Operasyonları: koy

```
// Yığının üstüne yine bir eleman koy
// Başarılı ise 0 başarısız ise -1 döndürür.
int koy(int yeni){

  if (dolumu()){
    // Yığın dolu. Yeni eleman eklenemez.
    return -1;
  }

  S[p] = yeni;
  p++;

  return 0;
} /bitti-koy
```

Yığın Operasyonları: ust

```
// Yığının en üstündeki sayıyı döndürür
// Yığın boşsa, -1 döndürür
public int ust(){
  if (bosmu()){
    // Yığın başsa hata dönder
    System.out.println("Stack underflow");
    return -1;
  }

  return S[p-1];
}
```

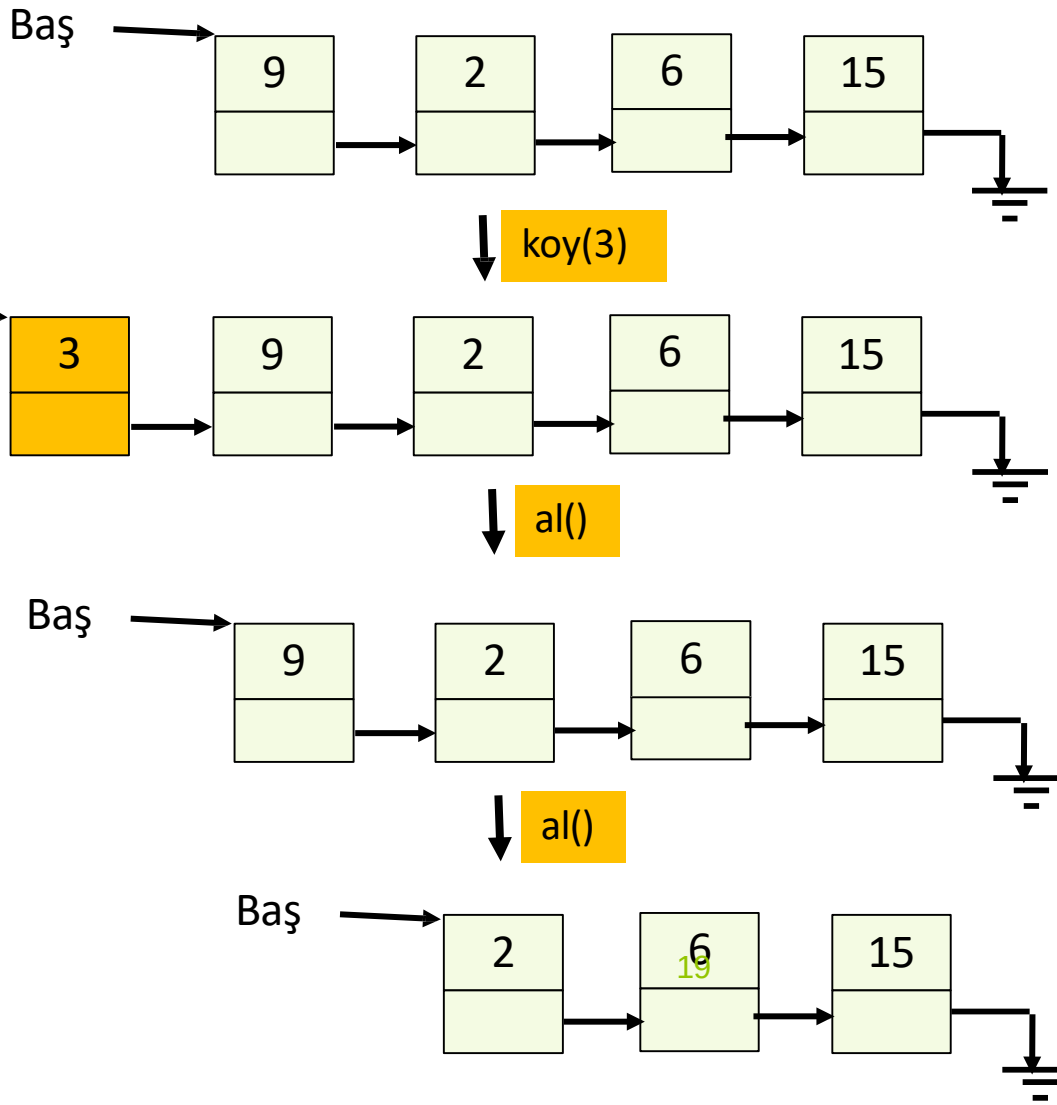
Stack Operations: al

```
// En üsteki elemanı dönder.
// Yığın boşsa -1 dönder.
public int al(){
  if (bosmu()){
    // Yığın boşsa hata dönder
    System.out.println("Stack underflow");
    return -1;
  }

  int id = p-1; // en üsteki elemanın yeri
  p--;          // elemanı sil

  return S[id];
}
```

Yığın Kullanım Örneği

```
public static void main(String[] args){
  Yigin y = new Yigin();

  if (y.bosmu())
    System.out.println("Yığın boş");

  y.koy(49);    y.koy(23);

  System.out.println("Yığının ilk elemanı: "+ y.al());

  y.koy(44);    y.koy(22);

  System.out.println("Yığının ilk elemanı: "+ y.al());
  System.out.println("Yığının ilk elemanı: "+ y.al());
  System.out.println("Yığının ilk elemanı: "+ y.ust());
  System.out.println("Yığının ilk elemanı: "+ y.al());.

  if (y.bosmu()) System.out.println("Yığın boş");
}
```

Yığın- Bağlantılı Liste Gerçekleştirimi



Bağlantılı Liste Gerçekleştirimi

```
public class  YiginDugumu {
  int eleman;
  YiginDugumu sonraki;

  YiginDugumu(int e){
    eleman = e; sonraki = NULL;
  }
}
```

```
public class Yigin {
  private YiginDugumu ust;

  public Yigin() {ust = null;}

  void koy(int eleman);
  int al();
  int ust();
  boolean bosmu();
};
```

Yığın Operasyonları: koy, bosmu

```
// Yığına yeni eleman ekle
public void koy(int eleman){
  YiginDugumu x = new YiginDugumu(eleman);
  x.sonraki = ust;
  ust = x;
}

// Yığın boşsa true döndür
public boolean bosmu(){
  if (ust == NULL)
    return true;
  else
    return false;
}
```

Yığın Operasyonları: ust

```
// Yığının ilk elemanını döndür
public int ust(){
  if (bosmu()){
    System.out.println("Stack underflow"); // Boş yığın
    return -1; // Hata
  }

  return ust.eleman;
} //bitti-ust
```

Yığın Operasyonları: Al()

```
// Yığının en üst elemanın siler ve döndürür.
public int Al(){
  if (bosmu()){
    System.out.println("Stack underflow"); // Boş yığın.
    return -1; // Hata
  }

  YiginDugumu temp = ust;

  // Bir sonraki elemana geç
  ust = ust.sonraki;

  return temp.eleman;
} //bitti-al
```

Yığın Kullanım Örneği

```
public static void main(String[] args){
  Yigin y = new Yigin();

  if (y.bosmu())
    System.out.println("Yığın boş");

  y.koy(49);    y.koy(23);

  System.out.println("Yığının ilk elemanı: "+ y.al());

  y.koy(44);    y.koy(22);

  System.out.println("Yığının ilk elemanı: "+ y.al());
  System.out.println("Yığının ilk elemanı: "+ y.al());
  System.out.println("Yığının ilk elemanı: "+ y.ust());
  System.out.println("Yığının ilk elemanı: "+ y.al());.

  if (y.bosmu()) System.out.println("Yığın boş");
}
```

Yığın Kullanımı - 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.
- Örneğin:
- a + b * c = (a + b) * c ?
- = a + (b * c) ?

Infix, Postfix, & Prefix Gösterimleri

- Herhangi bir yere operatör koymamamızın önünde bir engel yoktur.
- **Operatör Önde (Prefix) : + a b**
  - Biçim: işlem işlenen işlenen (operator operand operand) şeklindedir: + 2 7
  - İşlem sağdan sola doğru ilerler. Öncelik (parantez) yoktur.
- **Operatör Arada (Infix) : a + b**
  - Biçim: işlenen işlem işlenen (operand operator operand) şeklindedir: 2 + 7
  - İşlem öncelik sırasına göre ve soldan sağa doğru ilerler.
- **Operatör Sonda (Postfix) : a b +**
  - Biçim: işlenen işlenen işlem (operand operand operator) şeklindedir: 2 7 +
  - İşlem soldan sağa doğru ilerler. Öncelik (parantez) yoktur.

Prefix, Postfix : Diğer İsimleri

- Prefix gösterimi Polanyalı (Polish) bir mantıkçı olan Lukasiewicz, tarafından tanıtıldığı için “Polish gösterim” olarak da isimlendirilir.

- Postfix gösterim ise ters Polish gösterim “reverse Polish notation” veya RPN olarak da isimlendirilebilir.

Neden Infix, Prefix, Postfix ?

- Soru: infix gösterimde çalışmayla herşey yolunda iken neden böyle “aykırı”, “doğal olmayan ” bir gösterim şekli tercih edilsin.?

- Cevap: postfix and prefix gösterimler ile parantez kullanılmasına gerek yoktur !

Infix, Prefix, Postfix İşlemleri

- **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)

Parantez -Infix

- 2+3*5 işlemini gerçekleştiriniz.
- **+** önce ise:
  - (2+3)*5 = 5*5 = 25
- ***** önce ise:
  - 2+(3*5) = 2+15 = 17
- Infix gösterim paranteze ihtiyaç duyar.

Prefix Gösterim

- + 2 * 3 5
- = = + 2 * 3 5
- = + 2 15 = 17
- * + 2 3 5 =
- = * + 2 3 5
- = * 5 5 = 25
- Paranteze ihtiyaç yok!

Postfix Gösterim

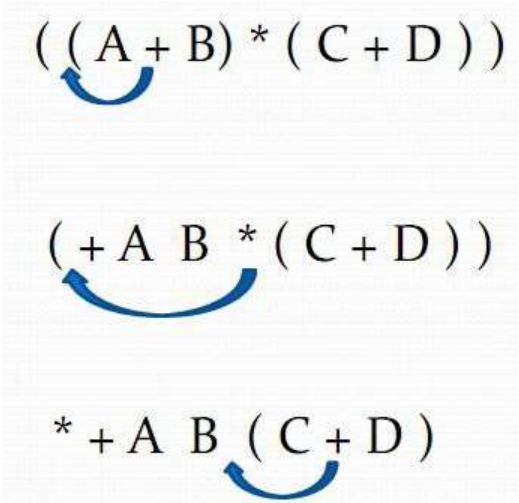
- 2 3 5 * +
- = 2 3 5 * +
- = 2 15 + = 17
- 2 3 + 5 *
- = 2 3 + 5 *
- = 5 5 * = 25
- Paranteze ihtiyaç yok!
- **Sonuç**:
- **Infix işlem sıralarınındüzenlenmesi için paranteze ihtiyaç duyan tek gösterim şeklidir**

Tamamen Parantezli Anlatım

- TPA gösterimde her operatör ve işlenenini çevreleyen parantezlerden oluşan tam bir set vardır.
- Hangisi tam parantezli gösterim?
  - (A + B) * C
  - **((A + B) * C)**
  - ((A + B) * (C)) ((A + B) (C))

Infix'ten Prefix'e Dönüşüm

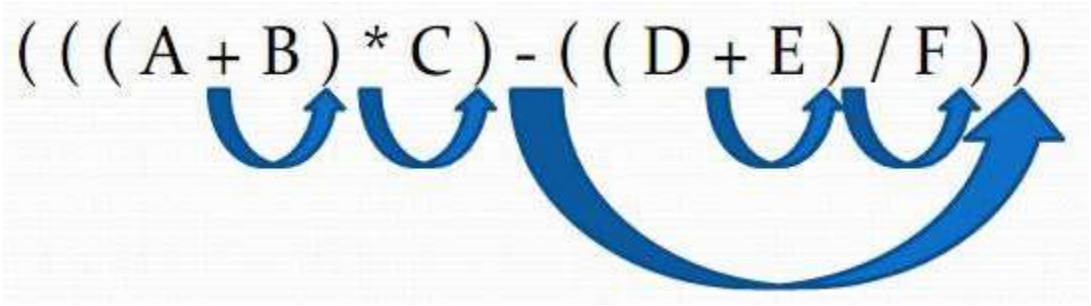
- Her bir operatörü kendi işlenenlerinin soluna taşı ve parantezleri kaldır. :



* + A B + C D

- İşlenenlerin sırasında bir değişiklik olmadı!

Infix'ten Postfix'e Dönüşüm



- ((AB+* C) - ((D + E) / F))
- (AB+C* - ((D + E) / F))
- AB+C* ((D + E) / F)-
- AB+C* (DE+ / F)-
- A B + C * D E + F / -
- İşlenenlerin sırası değişmedi!
- Operatörler değerlendirme sırasına göre!

Infix, Prefix, Postfix

- Aşağıda verilen işlemlerde işleyişe bakınız

Infix	Postfix	Prefix
A+B-C	AB+C-	-+ABC
(A+B)*(C-D)	AB+CD-*	*+AB-CD
A^B*C-D+E/F/(G+H)	AB^C*D-EF/GH+/+	+-*^ABCD//EF+GH
((A+B)*C-(D-E))^(F+G)	AB+C*DE—FG+^	^-*+ABC-DE+FG
A-B/(C*D^E)	ABCDE^*/-	-A/B*C^DE

Infix, Prefix, Postfix İşlemleri

- **Örnek:** Parantezsiz operatör arada ifadenin operatör sonda hale çevrilmesi :a + b * c – d

Okunan	Yığıt	Çıktı /Operatör sonda ifade
a		a
+	+	a
b	+	a b
*	+ *	a b
c	+ *	a b c
-	+ *	a b c
	+	a b c *
	-	a b c * +
d	-	a b c * + d -

Operatör Sonda (Postfix) İfadenin İşlenişi:

- **Örnek:** a b c * + d - ifadesini a=2 b=3 c=5 d=10 --> 2 3 5 * + 10 -

Okunan	Yığıt	Hesaplanan
2	2	
3	2 3	
5	2 3 5	
*	2	islem=* pop1=5 pop2=3
	2 15	3 * 5=15
+	17	islem=+ pop1=15 pop2=2
		2 + 15 =17
10	17 10	
-	7	islem=- pop1=10 pop2=17
		17 – 10= 7
		a b c * + d -

Infix, Prefix, Postfix İşlemleri

- **Örnek:** Parantezli operatör arada ifadenin operatör sonda hale çevrilmesi. Infix ifade: (2 + 8) / (4 – 3)

Okunan	Yığıt	Hesaplanan
(	(	
2	(	2
+	(+	2
8	(+	2 8
)		2 8 +
/	/	2 8 +
(	/ (	2 8 +
4	/ (	2 8 + 4
-	/ (-	2 8 + 4
3	/ (-	2 8 + 4 3
)	/	2 8 + 4 3 –
		2 8 + 4 3 –
		2 8 + 4 3 – /

Örnek: Java

- postfix olarak yazılmış ifadeyi hesaplayarak sonucu bulan bir program yazınız. Örnek olarak 5 3 2 - / 4 7 * + ifadesinin sonucunu bulunuz. Sonuç = 33 olarak bulunacak.

Örnek: Java

```
import java.awt.*;
import java.io.*;
import java.util.*;

public class yigin {
public int top=0;

public void push( int x, int a[], int top)
{   a[top] = x;      ++top;  this.top=top;  }

 public int pop( int a[], int top)
{   --top;          this.top=top;  return a[top]; }
```

Örnek:

```
public static void main(String[] args)  {
yigin metot =new yigin();
String str="";   int x = 0 , a=0,b=0;   int stk[]=new int [120];
Scanner  oku = new Scanner(System.in);
System.out.println("\n Postfix ifadeyi giriniz\n");
str=oku.nextLine();
for(int i=0;i<str.length();i++)
{   if(str.charAt(i) == '+')
     {  b = metot.pop( stk, metot.top);
        a = metot.pop( stk, metot.top);
        metot.push( a + b, stk, metot.top);
        System.out.println("\n a+b ="+ (a+b));
        }
```

Örnek:

```
else if(str.charAt(i) == '-')
    {  b = metot.pop( stk,metot.top);        a = metot.pop( stk, metot.top);
    metot.push( a - b, stk,metot.top);
    System.out.println("\n a-b ="+ (a-b));
    }
   else if(str.charAt(i) == '/')
   { b = metot.pop( stk, metot.top);    a = metot.pop( stk, metot.top);
   metot.push( a / b, stk, metot.top);
   System.out.println("\na/b= "+ a/b);
   }
   else if(str.charAt(i) == '*')
   {  b = metot.pop( stk, metot.top);   a = metot.pop( stk, metot.top);
  metot.push( a * b, stk, metot.top);
  System.out.println("\n a*b= "+ a*b);
  }
```

Örnek:

```
if(str.charAt(i) >= '0' && str.charAt(i) <= '9')
  {  x =str.charAt(i)- '0';     metot.push( x, stk, metot.top);     }
}
System.out.println("\n Postfix ifadesinin işlem sonucu="+ metot.pop( stk, metot.top));
  }
```

Çıktı:

Postfix ifadeyi giriniz

532-/47*+

a-b =1

a/b= 5

a*b= 28

a+b =33

Postfix ifadesinin işlem sonucu=33

Infix, Prefix, Postfix İşlemleri

- **Örnek:** Aşağıda boş bırakılan alanları tamamlayınız

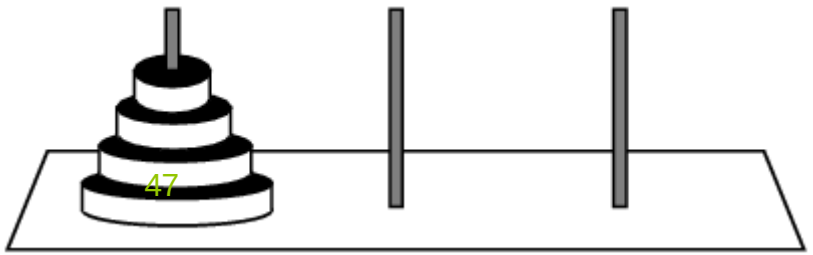
Infix	Prefix	Postfix
2x(3+5)-7^2(2+1)		
	++x23^-5721	
		235+7-^2x1+
2x3+5-7^2+1		

Infix, Prefix, Postfix İşlemleri

Cevap Infix	Prefix	Postfix
2x(3+5)-7^2(2+1)	**-x2+35^7+21**	**235+x721+^-**
(2x3)+(5-7)^2+1	++x23^-5721	**23x57-2^+1+**
2x(3+5-7)^2+1	**+x2^-+35721**	235+7-^2x1+
2x3+5-7^2+1	**+-+x235^721**	**23x5+72^-1+**

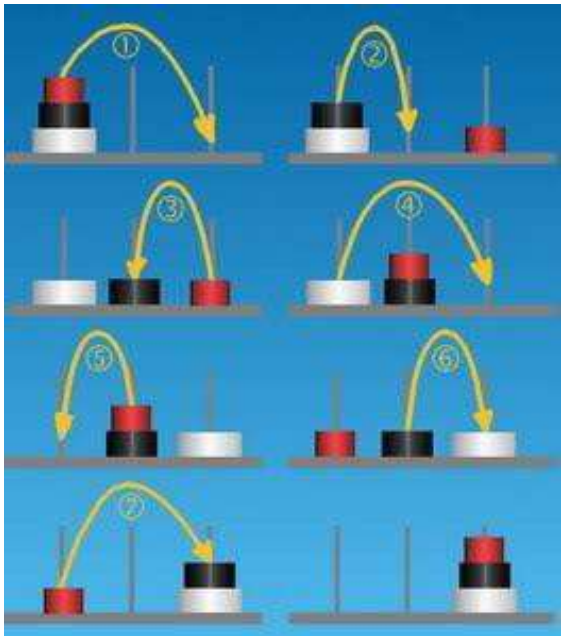
Hanoi Kuleleri Yığın temelli Çözüm

- **Verilen:** üç iğne
  - İlk iğnede en küçük disk en üstte olacak şekilde yerleştirilmiş farklı büyüklükte disk kümesi.
- **Amaç:** diskleri en soldan en sağa taşımak.
- **Şartlar:** aynı anda sadece tek disk taşınabilir.
- Bir disk boş bir iğneye veya daha büyük bir diskin üzerine taşınabilir.



Hanoi Kuleleri Yığın temelli Çözüm

- Problem karmaşıklığı 2^n
- 64 altın disk
- 1 taşıma işlemi 1 saniye sürsün:
- 18446744073709551616 sn
- 593.066.617.596,15 yıl
- Dünyanın sonuna 600.000.000.000 yıl var ☺



Hanoi Kuleleri– Özyinelemeli Çözüm-Java

```
package hanoikuleleri;
import java.util.*;
public class Hanoikuleleri  {

public static void main(String[] args)
{
 System.out.print("n değerini giriniz : ");
 Scanner klavye = new Scanner(System.in);   int n = klavye.nextInt();
 tasi(n, 'A', 'B', 'C');
}
 public static void tasi(int n, char A, char B, char C)
{if(n==1) System.out.println(A + " --> " + B);
 else
 {
 tasi(n-1, A, C, B);       tasi(1, A, B, C);     tasi(n-1, C, B, A); }
 return;
}
```

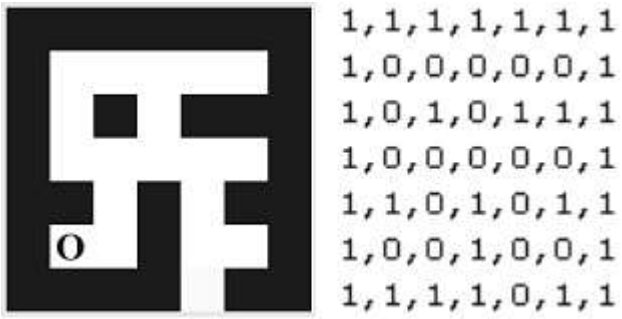
Ödev

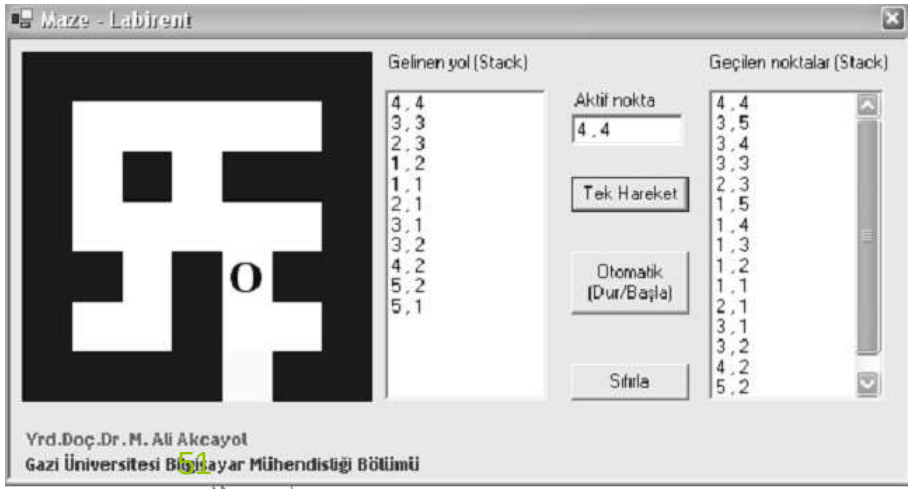
- 1-Infix'ten Prefix ve Postfix'e Çevirin
  - x
  - x + y
  - (x + y) - z
  - w * ((x + y) - z)
  - (2 * a) / ((a + b) * (a - c))
- 2-Postfix'ten Infix'e Çevirin
  - 3 r -
  - 1 3 r - +
  - s t * 1 3 r - + +
  - v w x y z * - + *

Ödev

- 3- postfix olarak yazılmış ifadeyi hesaplayarak sonucu bulan programı Java/C# bağlı liste yapısı ile yazınız.

- 4-Verilen Maze (Labirent) uygulamasını -başlangıç olarak istediğimiz noktadan başlayarak çıkışa ulaşmasını sağlayınız.





Örnekler –Java

- Java'da hazır Stack (yığıt) sınıfı da bulunmaktadır. Aşağıdaki örnekte String'ler, oluşturulan ***s*** yığıtına yerleştirilerek ters sırada listelenmektedir.

```
import java.util.*;
public class StackTest
{
  public static void main(String args[])
  {  String str[] = { "Bilgisayar", "Dolap", "Masa",        "Sandalye", "Sıra" };
     Stack s = new Stack();
     for(int i=0; i < str.length; ++)    s.push(str[i]);
     while(!s.empty() ) System.out.println(s.pop());
   }
}
```

Uygulama Ödevi

- Derleyici/kelime işlemciler
  - Derleyicileri düşünecek olursak yazdığımız ifadede ki parantezlerin aynı olup olmadığını kontrol ederler.
  - Örneğin: 2*(i + 5*(7 – j / (4 * k)) ifadesinde parantez eksikliği var. ")"
  - Yığın kullanarak ifadedeki parantezlerin eşit sayıda olup olmadığını kontrol eden programı yazınız.

Uygulama Ödevi

- Yığın kullanarak parantez kontrol:
  1) Boş bir yığın oluştur ve sembolleri okumaya başla
  2) Eğer sembol başlangıç sembolü ise (‘(‘, ’[’, ’{’) Yığına koy
  3) Eğer sembol kapanış sembolü ise (‘)’, ’]’, ’}’)
     I. Eğer yığın boşsa hata raporu döndür
     II. Değilse
         Yığından al
         Eğer alınan sembol ile başlangıç sembolü aynı değilse hata gönder
  4) İfade bitti ve yığın dolu ise hata döndür.