Özyineleme (Recursion)

- Özyineleme tanımlamaları
- Özyineleme çağırma
- Tail özyineleme
- Nontail özyineleme
- Dolaylı (Indirect) özyineleme
- İçiçe (Nested) özyineleme

Özyineleme (Recursion)

- **Kendi kendisini doğrudan veya dolaylı olarak çağıran fonksiyonlara özyineli (recursive) fonksiyonlar adı verilir.**
- **Özyineleme bir problemi benzer şekilde olan daha küçük parçalara bölünerek çözülmesini sağlayan bir tekniktir.**
- **Özyineleme, döngülere (iteration) alternatif olarak kullanılabilir.**

Özyineleme (Recursion)

- **Bir problem özyineli olmayan basit bir çözüme sahiptir (stopping case, base case).**
- **Problemin diğer durumları özyineleme ile durdurma durumuna (stopping case) indirgenebilir.**
- **Özyineleme işlemi durdurma durumu sağlanınca sonlandırılır.**

Genel yazımı:

if *(durdurma durumu sağlandıysa)*
    çözümü yap
else
    problemi özyineleme kullanarak indirge

Özyineleme (Recursion)

n! = 1 * 2 * 3 * … * (n – 1) * n

- **Faktoriyel işlemini özyineli tanımlamak için küçük sayıların faktöriyeli şeklinde tanımlamak gerekir.**

n! = n * (n – 1) !

- **Durdurma durumu 0 ! = 1 olarak alınır.**
- **Her çağırmada n değeri bir azaltılarak durdurma durumuna ulaşılır.**

Özyineli tanımlama:

n! = 1 if *n = 0*
*n! = n * (n – 1)!* if *n > 0*

Özyineleme (Recursion)

Aşağıdaki kod çalıştığında n sayısının faktöriyel değerini hesaplar.

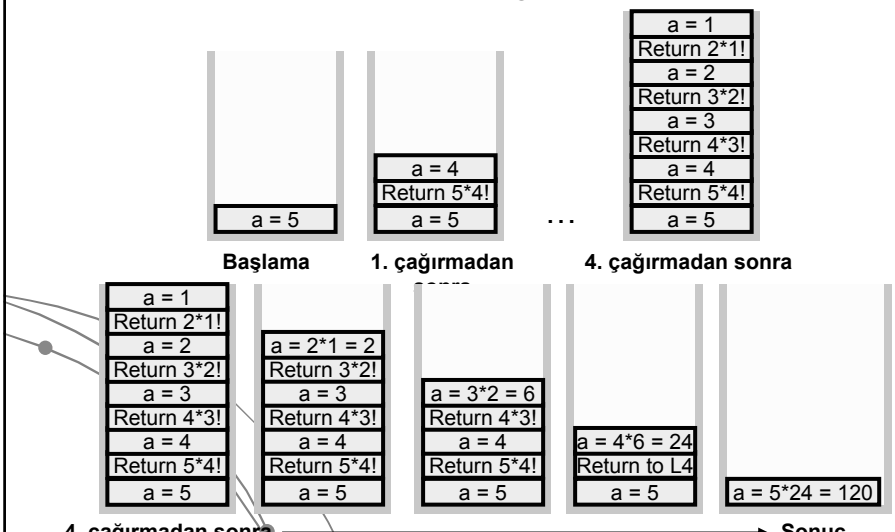
```
public int recFact(int n)
{
    if ( n == 1 )  return( 1 );
    else   return( h * recFact( n - 1 ));
}
```

$$
\begin{aligned}
5! &= 5(4!) \\
&= 5(4(3!)) \\
&= 5(4(3(2!))) \\
&= 5(4(3(2(1!)))) \\
&= 5(4(3(2(1)))) \\
&= 5(4(3(2))) \\
&= 5(4(6)) \\
&= 5(24) \\
&= 120
\end{aligned}
$$

n! değerini hesaplar ve bulduğu değeri return ile gönderir.

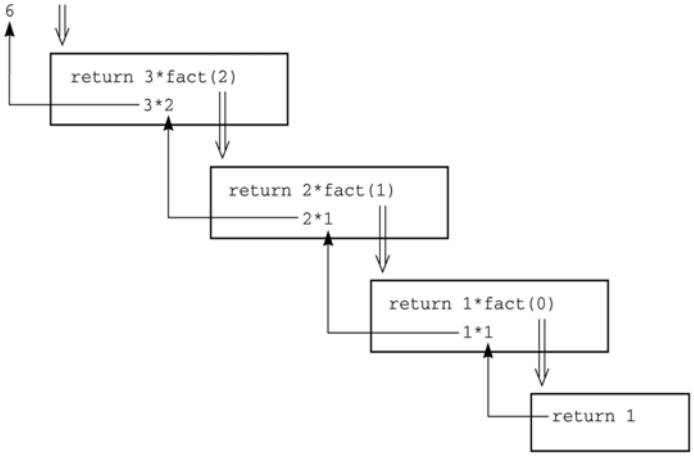
Özyineleme (Recursion)

recFact(5) için çalışmasının stack ile gösterimi



Özyineleme (Recursion)

recFact(3); için gerçekleşen işlemler aşağıdaki gibidir.



Özyineleme (Recursion)

Recursive

```
public int recFact(int n)
{
  if ( n == 1 )  return( 1 );
  else   return( h * recFonk( n - 1 ));
}
```

Iterative

```
public int iteFonk(int n)
{
    int araDeger = 1;
    for (int i = n; i > 0; i- -)
        araDeger * = i;
    return araDeger;
}
```

Özyineleme (Recursion)

- Genellikle iterative fonksiyonlar zaman ve yer bakımından daha etkindirler.
  - Iterative algoritma döngü yapısını kullanır.
  - Özyineleme algoritması dallanma (branching) algoritmasını kullanır.
  - Fonksiyon özyineli olarak her çağrılışında yerel değişkenler ve parametreler için bellekte yer ayrılır.
- Özyineleme problemin çözümünü basitleştirebilir, sonuç genellikle kısadır ve kod kolayca anlaşılabilir.
- Her özyinelemeli olarak tanımlanmış problemin iterative çözümüne geçiş yapılabilir.

Özyineleme (Recursion)

Örnek : Fibonacci dizisinin özyineleme ile bulunması:

Fibonacci dizisinde her eleman kendinden önceki iki elemanın toplamı şeklinde hesaplanır.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

0.eleman : 0
1.eleman : 1
2.eleman : 0+1 = 1
3.eleman : 1+1 = 2
4.eleman : 1+2 = 3
5.eleman : 2+3 = 5

Özyineleme (Recursion)

Örnek : Fibonacci dizisinin özyineleme ile bulunması (devam):

Fibonacci dizisinin tanımı:

fib(n) = n if ((n==0) || (n==1))
fib(n) = fib(n-2) + fib(n-1) if (n>=2)

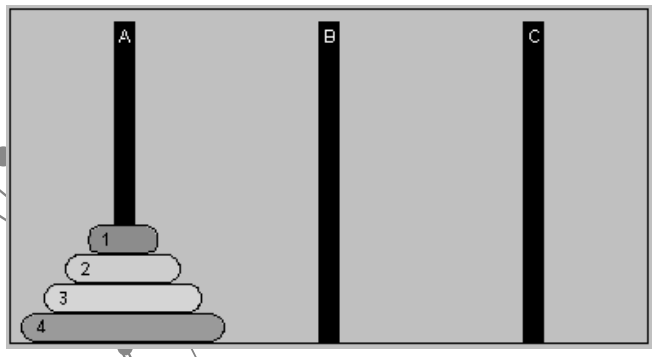
Fibonacci dizisi program:

```
public int recFib(int n)
{
if (n<=1) return n;
else return recFib(n-1)+recFib(n-2);
}
```


Özyineleme (Recursion)

Örnek : Hanoi Kuleleri Problemi :

- **N adet disk bir kulede iken diğer kulelerden birisine hepsi aynı sırada yerleştirilecek**
- **Hiçbir zaman büyük disk küçük diskin üzerine gelmeyecek.**
- **A kaynak kule, C hedef kule ve B ise ara geçişlerde kullanılacak geçici kuledir.**



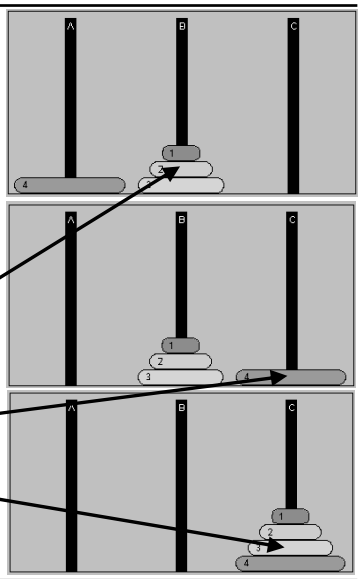
Özyineleme (Recursion)

Örnek : Hanoi Kuleleri Problemi (devam):

- **(n-1) disk C kulesi geçici kullanılarak B'ye aktarılabilirse A'daki n. disk doğrudan C'ye aktarılabilir.**
- **A'daki 1 diski doğrudan A'dan C'ye aktarabiliriz.**
- **B'deki tüm diskler A kulesi kullanılarak C'ye aktarılır.**

```
public void han(int n, char s, char d, char t)
{
if (n == 1) listBox1.Items.Add(s + " -> " + d);
else
{
    han(n-1, s, t, d);
    han(1, s, d, t);
    han(n-1, t, d, s);
}
}
```

```
han(4, A, C, B);
```

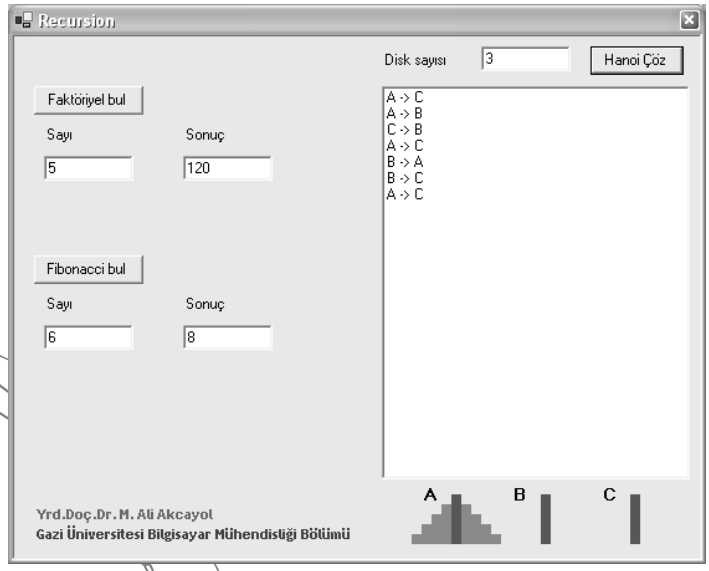


Özyineleme (Recursion)

n = 3 s=A, d=C, t=B	K	n = 2 s=A, d=B, t=C	K	n = 1 s=A, d=C, t=B	
n = 3 s=A, d=C, t=B	K	n = 2 s=A, d=B, t=C	K	n = 1 A -> C	
n = 3 s=A, d=C, t=B	K	n = 2 s=A, d=B, t=C	L	n = 1 A -> B	
n = 3 s=A, d=C, t=B	K	n = 2 s=A, d=B, t=C	M	n = 1 C -> B	
n = 3 s=A, d=C, t=B	L	n = 2 A -> C			

```
public void han(int n, char s, char d, char t) {
if (n == 1) listBox1.Items.Add(s + " -> " + d);
else {
han(n-1, s, t, d);          // K
han(1, s, d, t);            // L
han(n-1, t, d, s);          // M
}
```

Özyineleme (Recursion)



Özyineleme (Recursion)

Tail yineleme

• Yineleme çağrısı metodun en sonunda yapılır.

```
public int recFact(int n)
{
  if (n<=1) return 1;
  else return n * recFact(n-1);
}
```

```
public void tail()
{
  …..
  …..
  …..
  tail();
}
```

Özyineleme (Recursion)

Nontail yineleme

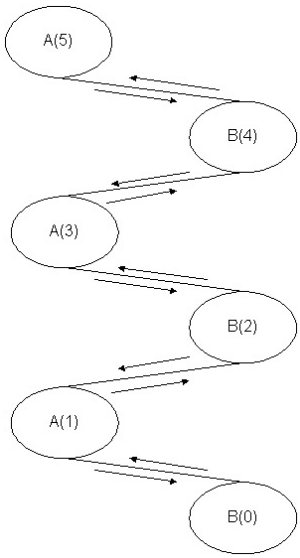
• Yineleme çağrısından sonra başka işlemler yapılır (yazdırma v.b.).

```
public int nontail(int n)
{
  if (n > 0)
  {
    .......
    .......
    nontail(n-1);
    Console.WriteLine(n);
    .......
    .......
    .......
    nontail(n-1);
  }
}
```


Özyineleme (Recursion)

Dolaylı (Indirect) yineleme

• Yineleme çağrısı başka bir fonksiyonun içinden yapılır.



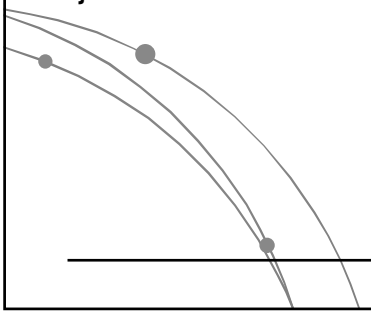

```
void A(int n)
{
  if (n <= 0) return 1;
  n- -;
  B(n);
}

void B(int n)
{
  if (n <= 0) return 1;
  n- -;
  A(n);
}
```

Özyineleme (Recursion)

İçiçe (Nested) yineleme

- **Yineleme çağrısı içindende yineleme çağrısı yapılır.**

```
public int A(int n, int m)
{
  if (n <= 0) return 1;
  return A(n-1 A(n-1, m-1));
}
```



Özyineleme (Recursion)

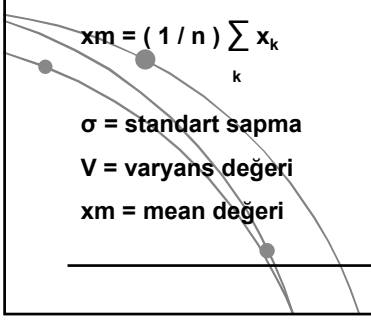
Haftalık Ödev:

n adet x değeri için standart sapmayı (σ) bulan programı iterasyon ve recursive ile yapınız.

$\sigma = \sqrt{V}$

$V = (1 / (n - 1)) \sum_{k} (x_k - xm)^2$

$xm = (1 / n) \sum_{k} x_k$



σ = standart sapma

V = varyans değeri

xm = mean değeri