Özyineleme (Recursion)

- Özyinelemeler veya artık teknik Türkçeye girmiş olan rekürsiflik en çok duyulan fakat kullanımında zorluklar görülen tekniklerdendir.
- Bu bölümde algoritmalarda özyinelemeli tekniklerin kullanımına değinilecektir.

Özyineleme (Recursion)

- İlk önce özyinelemeliğin bir algoritma olmayıp sadece algoritma tasarımında kullanılan bir programlama tekniği olduğuna değinmek gerekir.
- Ancak birçok algoritmanın uygulanmasında özyinelemeli yaklaşımın uygulanabilirliğinin bilinmesi gerekir

Özyineleme (Recursion)

- Genellikle bu yöntem n boyutlu problemin daha küçük boyutları ele alınarak benzeri şekilde çözümünde kullanılmaktadır.
- Problem çözümlemede problemin daha küçük alt problemlere parçalanarak çözümlenmesine böl ve yönet (devide and conquer) yaklaşımı denir.

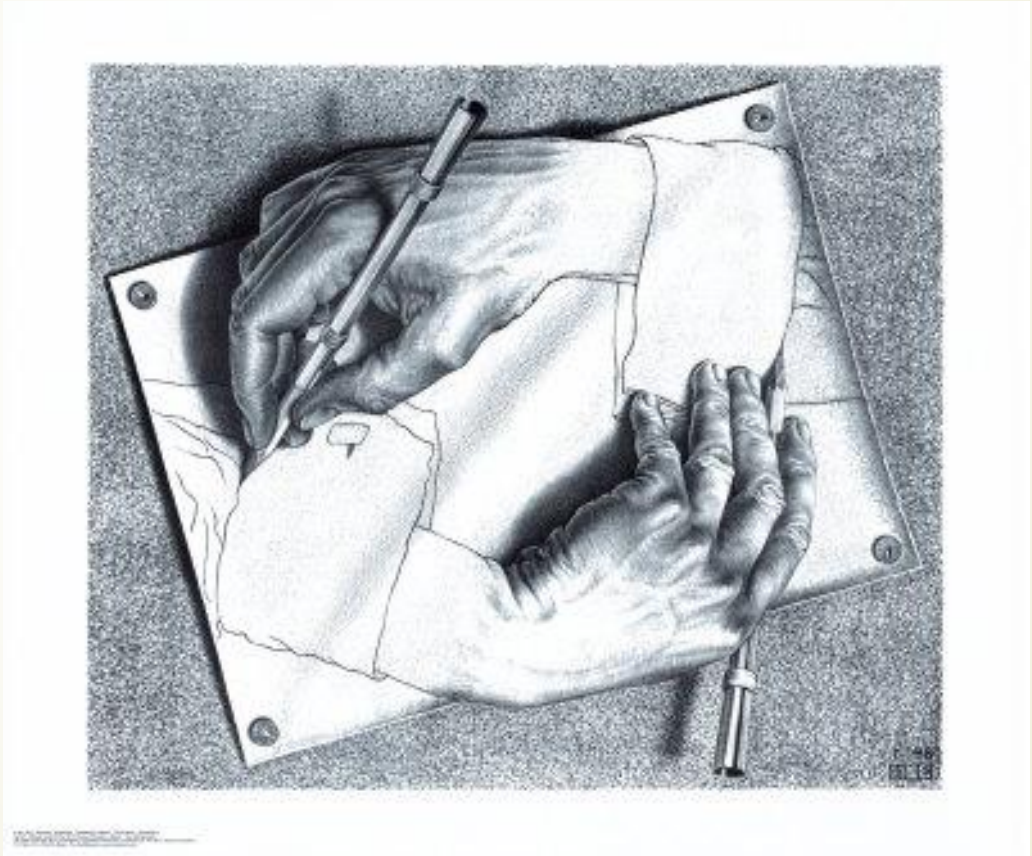
Özyineleme (Recursion)

- Divide-and-conquer metoduyla algoritma tasarımı:
- Problem kendisine benzer küçük boyutlu alt problemlere bölünür. Alt problemler çözülür ve bulunan çözümler birleştirilir.
  - **Divide:** Problem iki veya daha fazla alt probleme bölünür
  - **Conquer:** Divide-and-conquer özyinelemeli (recursively) olarak kullanılarak alt problemleri çözer
  - **Combine:** Alt problemlerin çözümleri alınır ve orijinal problemin çözümü olacak şekilde birleştirilir.

Özyineleme (Recursion)

- Bazı problemlerin çözümünde özyinelemeli yaklaşım mecburi olsa da uygulamalarda algoritmaların daha derinden incelenmesi ve sonra özyinelemeli şekilde çözüleceğine karar verilmelidir.
- Özyineleme (Latince geri gelme anlamında olan recursio kelimesinden) "böl ve yönet" yaklaşımlı olup programın icrası sırasında bir fonksiyonun veya sürecin kendisini direkt veya dolaylı yolla diğer fonksiyon ve süreçler yardımıyla tekrar çağırmasıdır.

Özyineleme (Recursion)



Özyineleme (Recursion)

- Özyinelemelerin sonsuz olarak devam etmesinin önlenmesi için bitiş koşulunun olması gerekmektedir.
- Dolayısıyla özyinelemeli programlarda bu sürecin ne zaman veya hangi koşullarda durması gerektiği önceden bilinmelidir.
- Fakat program devamlı kendi kendini çağırıyorsa, o nasıl durur.

Özyineleme (Recursion)

- Tanım gereği fonksiyon kendi kendini çağırdığında genellikle her seferinde daha küçük boyutlu problem ele alınacağından sürecin sonsuz devam edemeyeceği anlaşılmaktadır.
- Özyineleme işlemi durdurma durumu sağlanınca sonlandırılır.
- Genel yazımı:

if (durdurma durumu sağlandıysa)

çözümü yap

else

problemi özyineleme kullanarak indirge

Özyineleme (Recursion)

- Özyineleme programlamada bazı algoritmaların anlaşılmasında kolaylıklar sağlasa da birçok durumlarda olumsuz yönlerini de göstermektedir
- Devamlı olarak fonksiyonun kendi kendini çağırması belirli işlem yüküne de sebep olacaktır.
- Çünkü her fonksiyonun tekrar çağırılması birçok atama işleminin gerçekleştirilmesinin ve yeni değişkenlere uygun bölge ayırımını gerektirmektedir.
- Bu nedenle aynı özyinelemede süreç kendisini çok sayıda çağırdığından bellek ve zaman problemi ortaya çıkacaktır.
- Dolayısıyla eğer programın özyinelemesiz çözümü mümkünse bu çözüm biçimi tercih edilmedir.

Özyineleme (Recursion)

- Şimdi kısaca problemler üzerinde özyinelemeli yaklaşımın olumu ve olumsuz yönlerini ele alalım.
- Çarpımı toplamla ifade ettiğimizde veya toplamı ardışık olarak 1 ler şeklinde gösterdiğimizde rekursifliğe başvurmaktayız:
  - **6*5**=(6*4)+6=((6*3)+6)+6=(((6*2)+6)+6)+6=((((6*1)+6)+6)+6)+6=6+6+6+6+6k=30

  Veya

  - **6+5**=(6+4)+1=((6+3)+1)+1)=(((6+2)+1)+1)+1=((((6+1)+1)+1)+1)+1=6+1+1+1+1+1**=11**

Özyineleme (Recursion)

- Benzeri şekilde n sayısının toplamını da rekürsif olarak bulabiliriz.
- Gauss'un ünlü $S_n = \frac{a_1+a_n}{2} * n$ formülünde faklı olarak yaptığımız işlem bu sayıların ardışık toplanması şeklinde olmuştur.
- İşte eğer n sayısının toplamının bulunması isteniyorsa (*n-1*) sayının toplamını bularak üzerine n eklemekle sonuca ulaşmak mümkündür.

Özyineleme (Recursion)

- Benzeri şekilde problem boyutunu küçülterek istenen çözüm elde edilmektedir.
- Fakat bu sonsuz hesaplamaların sonlandırılması için belirli bir gerçeğe örneğin "*n=1 ise sonuç 1'dir*" sonlandırılmasına ihtiyaç vardır.

Özyineleme (Recursion)

- Aşağıda n tamsayının toplamını bulan rekürsif program gösterilmiştir.

```
1   int toplam (int n)
2   {
3      İf (n==1)
4         return 1;
5      else
6         Return toplam(n-1)+n;
7   }
```

Özyineleme (Recursion)

- Özyinelemeli algoritmalar anlaşıldığı gibi iki temel kısımdan oluşmaktadır. Bunlar;
  - Algoritmanın sonlandırılmasını sağlayan ***bitiş kısmı*** ve,
  - Fonksiyonun belirtilen sayıda tekrarlanmasını sağlayan ***gövde kısmı***dır.

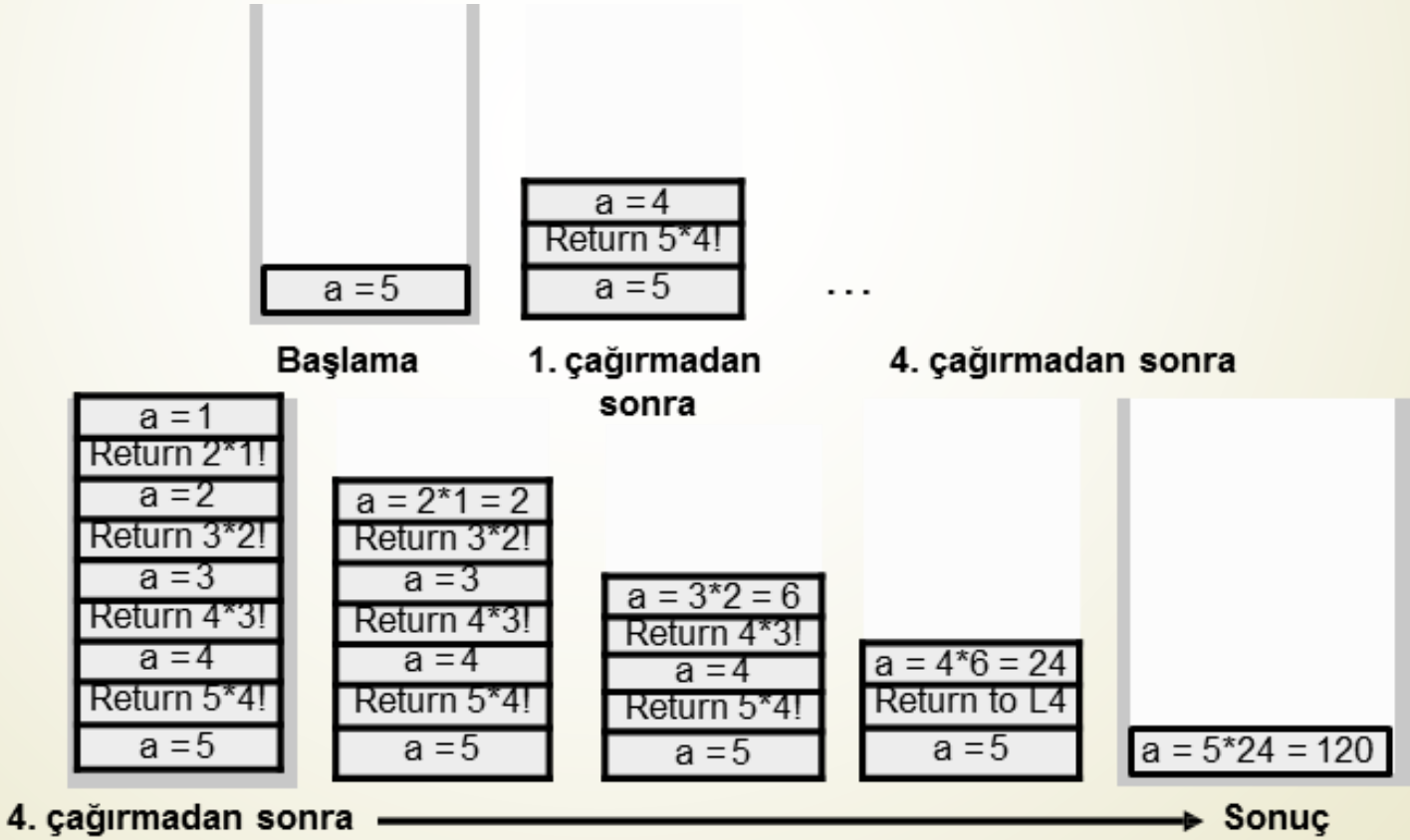
Özyineleme (Recursion)

- **Faktöriyelin özyineli olarak hesaplanması**
- 0!=1 olduğundan n>=1 ise n!=n*(n-1)!
- Burada 0!=1 bitiş koşulu olmaktadır. Fakat bu koşul programcıya bağlı olarak örneğin 2!=2 şeklinde de değiştirilebilir.

```
int faktoriyel (int n)
{
if (n==0)
    return 1;
return n*faktoriyel(n-1);
}
```

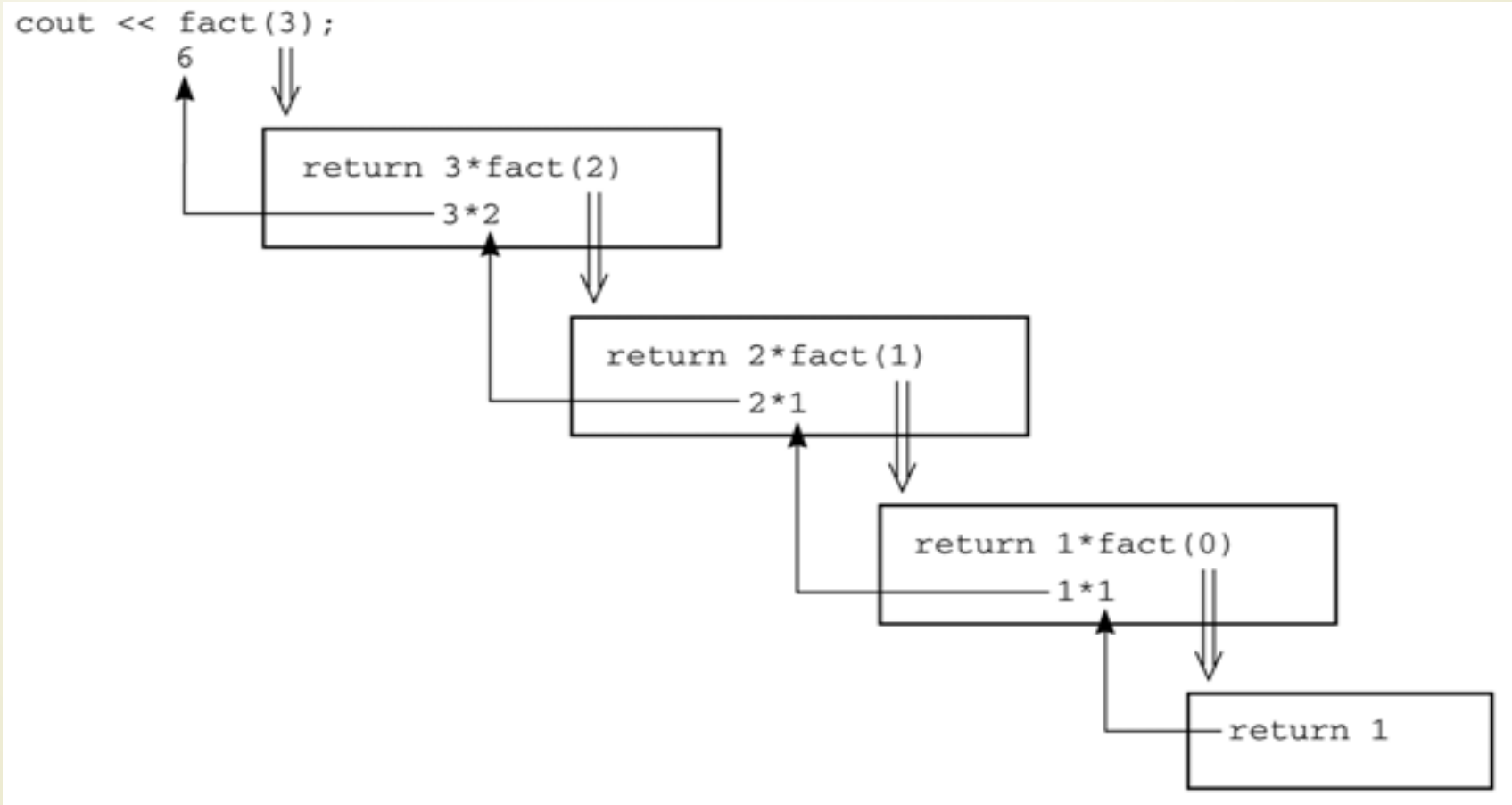
Özyineleme (Recursion)

- faktoriyel(5) için çalışmasının stack ile gösterimi.



Özyineleme (Recursion)

- fact(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 (Recursion)

- 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, ...

  1.eleman : 0

  2.eleman : 1

  3.eleman : 0+1 = 1

  4.eleman : 1+1 = 2

  5.eleman : 1+2 = 3

  6.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)

- Fibonacci Serisi problemini iteratif yöntemle de kodlayın.
- Fibonacci(45) ve sonrası değerler için recursive ve iteratif sonuçları inceleyerek sonuçları tartışın.