

# 315 Programlama Dilleri

Yrd. Doç. Dr. Ahmet Arif AYDIN

Altprogramların Gerçekleştirilmesi

(Implementation of Subprograms)

# Call ve Return

## Semantics of Call and Return

Altprogram çağrısının ve geri değer alma işlemlerinin hepsine birden altprogram bağlantısı (subprogram linkage) denmektedir.

# Call ve Return

## Semantics of Call and Return

Altprogram çağrısı aşağıdaki işlemler ile gerçekleştirilir:

- Parametre gönderme yöntemini içerir
- Altprogramda bulunan ve statik olmayan değişkenler için yer ayrılması gerekir
- Alt programın durum bilgisi ( kayıt değerleri, CPU durum biti, ortam işaretçileri)

# Call ve Return

## Semantics of Call and Return

Altprogram çağrısı aşağıdaki işlemler ile gerçekleştirilir:

- Parametre gönderme yöntemini içerir
- Altprogramda bulunan ve statik olmayan değişkenler için yer ayrılması gerekir
- Alt programın durum bilgisi ( kayıt değerleri, CPU durum biti, ortam işaretçileri)
- Geri gönderilecek olan değerler (varsa) gerçek parametrelere gönderilir
- Çağıran (ana) programa döndükten sonra local değişkenler için deallocation işlemi gerçekleştirilir.

# Altprogramların Gerçekleştirilmesi

Local değişkenler statik ve iç içe programlar desteklenmiyor (Fortran)

## Altprogramın Çağırılması (call)

1. Ana programın çalışma durumunu kaydet
2. Parametreleri hesapla ve gönder
3. Altprograma dönüş adresini gönder
4. Kontrolü alt programa gönder

# Altprogramların Gerçekleştirilmesi

Local değişkenler statik ve iç içe programlar desteklenmiyor (Fortran)

## Altprogramın Değer göndermesi (return)

1. Pass-by-value (result) veya out mode parametreleri varsa hazırlanır
2. Altprogram fonksiyon ise fonksiyonun göndereceği değer ana program için hazırlanır
3. Ana programın çalışma durumunu güncelle
4. Kontrolü ana programa devret

# Altprogramların Gerçekleştirilmesi

Call ve return işlemleri aşağıdakiler için bellek alanı ayrılması gerektirmektedir:

- Ana programın durumu (status)
- Parametreler
- Geri dönüş adresi
- Geri gönderilecek olan değerler
- Alt program tarafından kullanılan geçici alanlar



# Altprogramların Gerçekleştirilmesi

Bir alt program iki temel alt kısımdan oluşmaktadır:

1. Altprogramın kod kısmı (sabit)
2. Local değişkenler ve önceden mevcut olan veri (program çalışınca değişebilir)

# Altprogramların Gerçekleştirilmesi

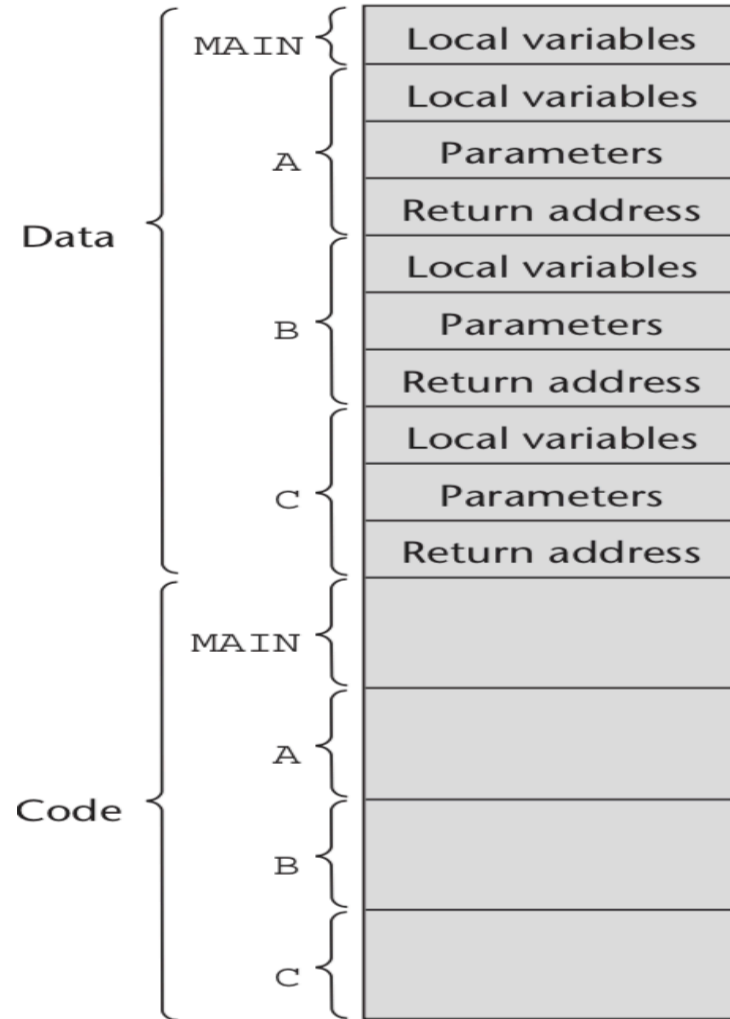
Bir alt program iki temel alt kısımdan oluşmaktadır:

1. Altprogramın kod kısmı (sabit)
2. Local değişkenler ve önceden mevcut olan veri (program çalışınca değişebilir)

- Bir altprogramın kod içermeyen formatına aktivasyon kaydı (activation record) denir.
- Formatı statikdir.
- Altprogram çalıştığında kullanılır.
- Activation record instance: alt program çağrıldığında elde edilen bir örnektir.

Local variables
Parameters
Return address

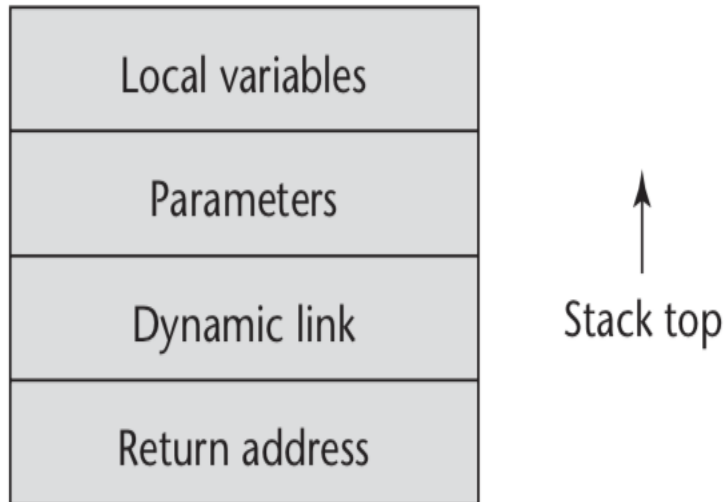
# Altprogramların Gerçekleştirilmesi



Şekilde verilen program linker sayesinde ana program ile alt programları (A,B,C) çağrılmış ve çalıştırılmıştır.

# Altprogramların Gerçekleştirilmesi

Yığıt Dinamik yerel değişkenler kullanılan altprogramların gerçekleştirilmesi



**Activation Record**

- Activation record yığıt dinamik değişken kullanılan dillerde çalışma anında dinamik olarak belirlenir.
- Dinamik link (dynamic link) ana programa olan bağlantı işaretçisidir.
  - Statik dillerde run-time hatası oluştuğunda hatanın oluştuğu yeri belirlemede kullanılır.
  - Dinamik olmayan dillerde local olmayan değişkenlere bağlantı için kullanılır.

# Altprogramların Gerçekleştirilmesi

```
void fun1(float r) {  
    int s, t;  
    ...           ←————— 1  
    fun2(s);  
    ...  
}
```

```
void fun2(int x) {  
    int y;  
    ...           ←————— 2  
    fun3(y);  
    ...  
}
```

```
void fun3(int q) {  
    ...           ←————— 3  
}
```

```
void main() {  
    float p;  
    ...  
    fun1(p);  
    ...  
}
```

Programın çalışması  
hangi sıya göre olur?

# Altprogramların Gerçekleştirilmesi

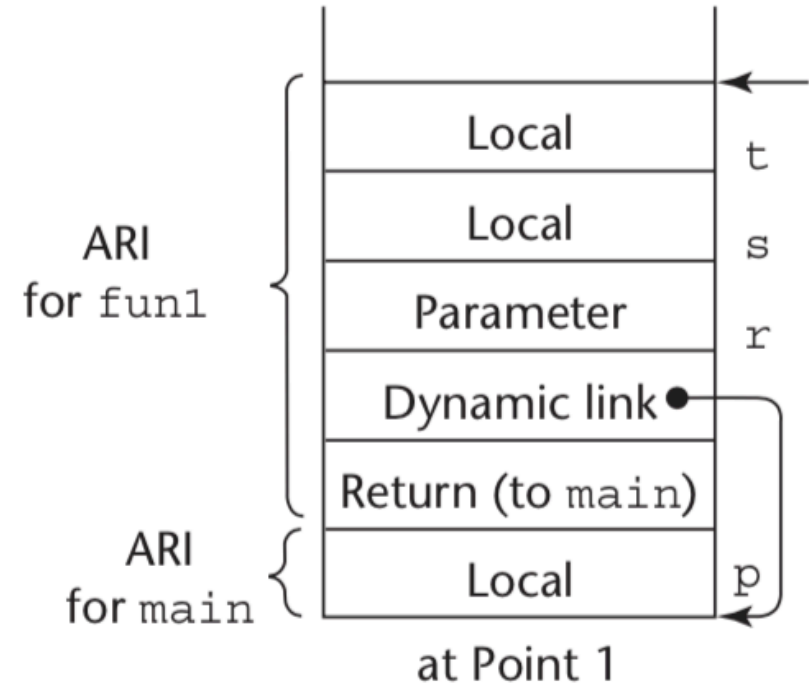
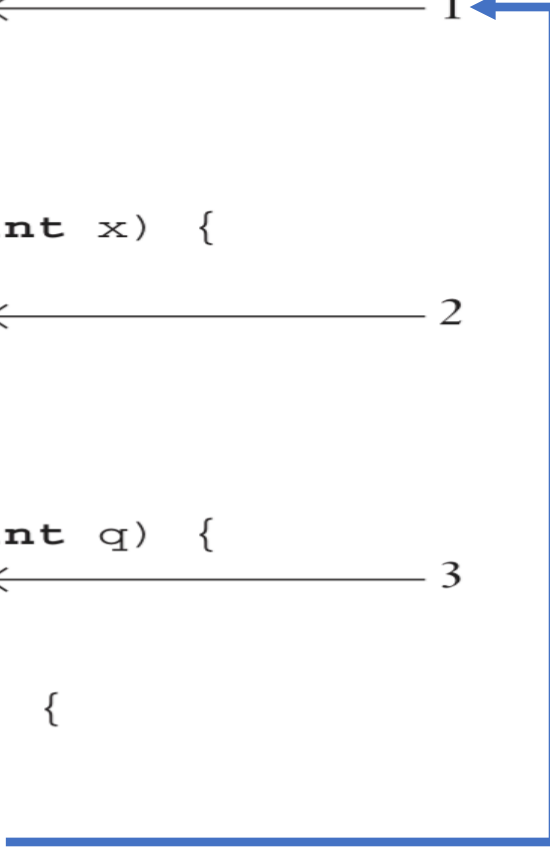
```
void fun1(float r) {  
    int s, t;  
    ...  
    fun2(s);  
    ...  
}  
  
void fun2(int x) {  
    int y;  
    ...  
    fun3(y);  
    ...  
}  
  
void fun3(int q) {  
    ...  
}  
  
void main() {  
    float p;  
    ...  
    fun1(p);  
    ...  
}
```

The diagram illustrates the execution flow of the provided code. Blue arrows show the sequence of function calls and returns. The numbers 1 through 6 are placed in red boxes to mark specific points in the execution:

- 1: The call to `fun1` from `main`.
- 2: The call to `fun2` from `fun1`.
- 3: The call to `fun3` from `fun2`.
- 4: The return from `fun3` to `fun2`.
- 5: The return from `fun2` to `fun1`.
- 6: The return from `fun1` to `main`.

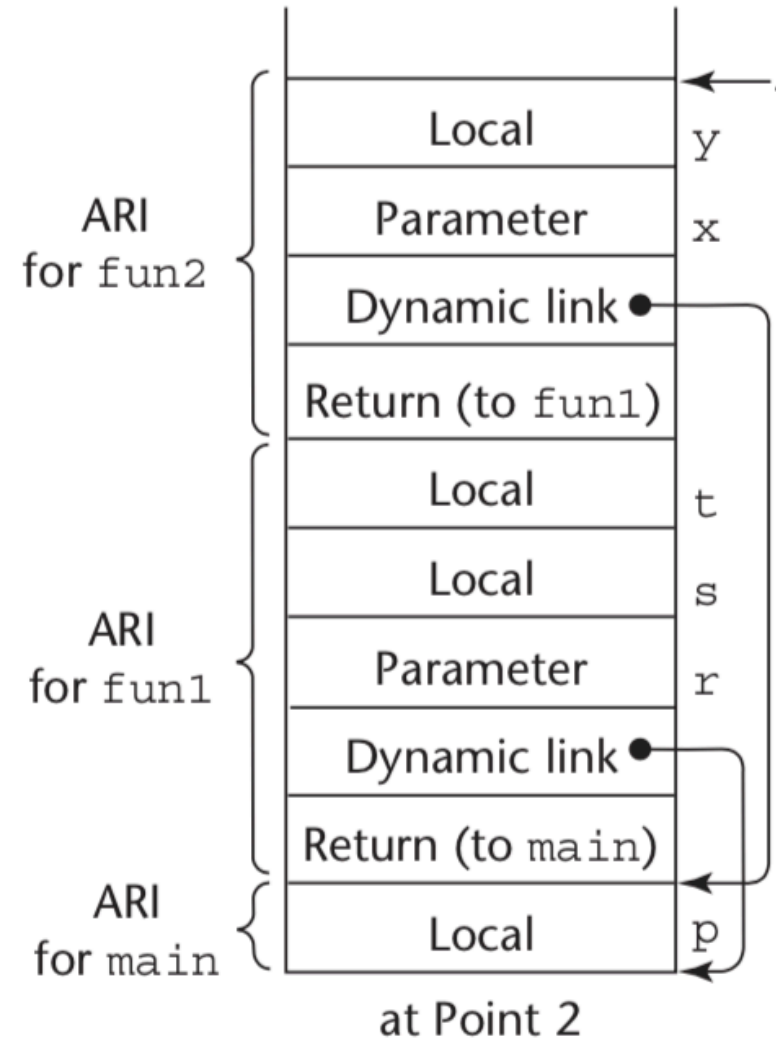
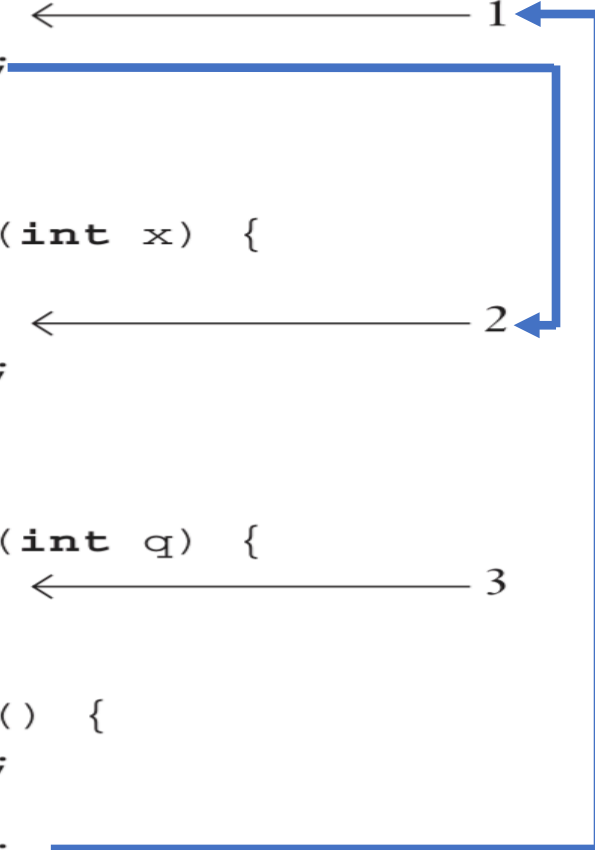
# Altprogramların Gerçekleştirilmesi

```
void fun1(float r) {  
    int s, t;  
    ... ← 1  
    fun2(s);  
    ...  
}  
  
void fun2(int x) {  
    int y;  
    ... ← 2  
    fun3(y);  
    ...  
}  
  
void fun3(int q) {  
    ... ← 3  
}  
  
void main() {  
    float p;  
    ...  
    fun1(p);  
    ...  
}
```



# Altprogramların Gerçekleştirilmesi

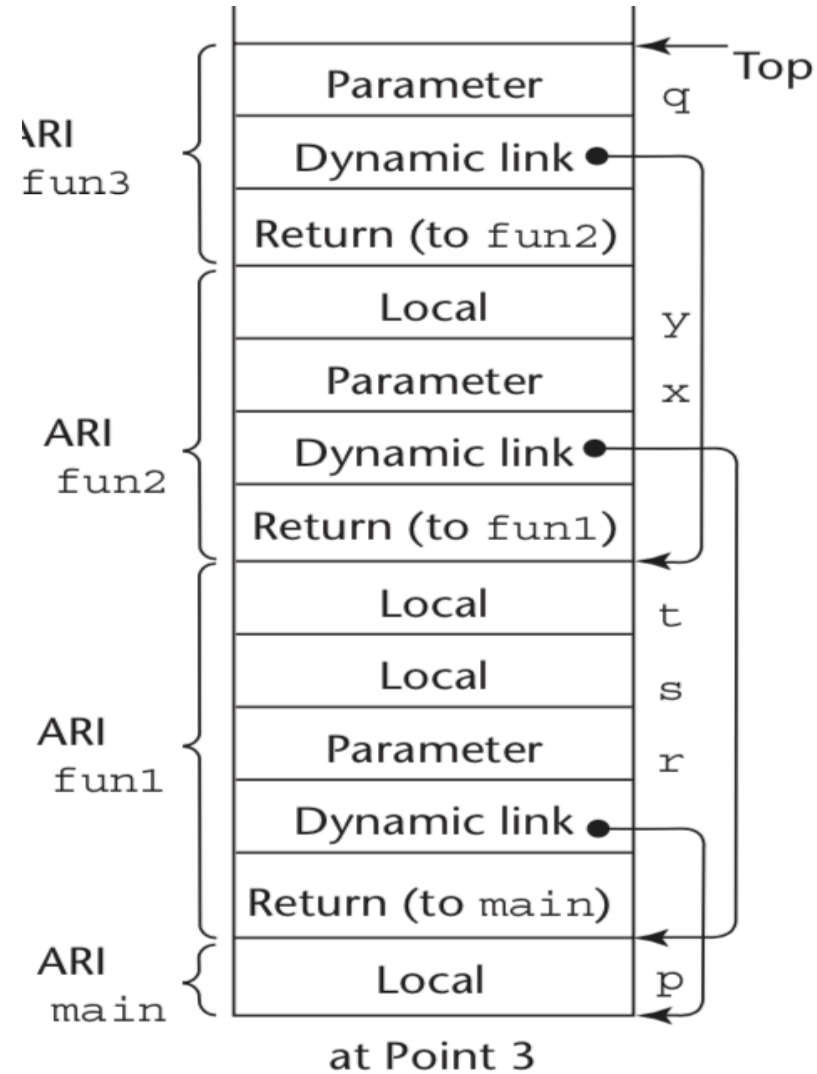
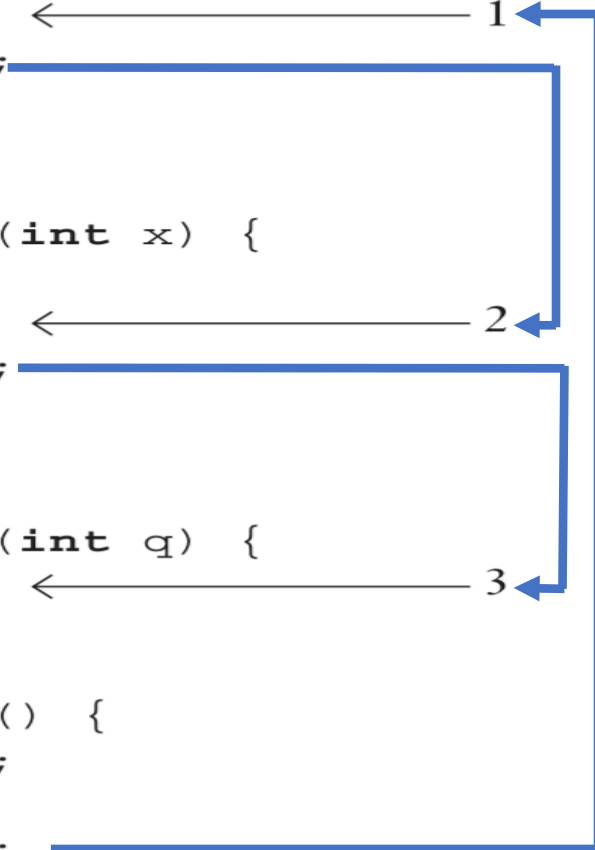
```
void fun1(float r) {  
    int s, t;  
    ...  
    fun2(s);  
    ...  
}  
  
void fun2(int x) {  
    int y;  
    ...  
    fun3(y);  
    ...  
}  
  
void fun3(int q) {  
    ...  
}  
  
void main() {  
    float p;  
    ...  
    fun1(p);  
    ...  
}
```





# Altprogramların Gerçekleştirilmesi

```
void fun1(float r) {  
    int s, t;  
    ...  
    fun2(s);  
    ...  
}  
  
void fun2(int x) {  
    int y;  
    ...  
    fun3(y);  
    ...  
}  
  
void fun3(int q) {  
    ...  
}  
  
void main() {  
    float p;  
    ...  
    fun1(p);  
    ...  
}
```



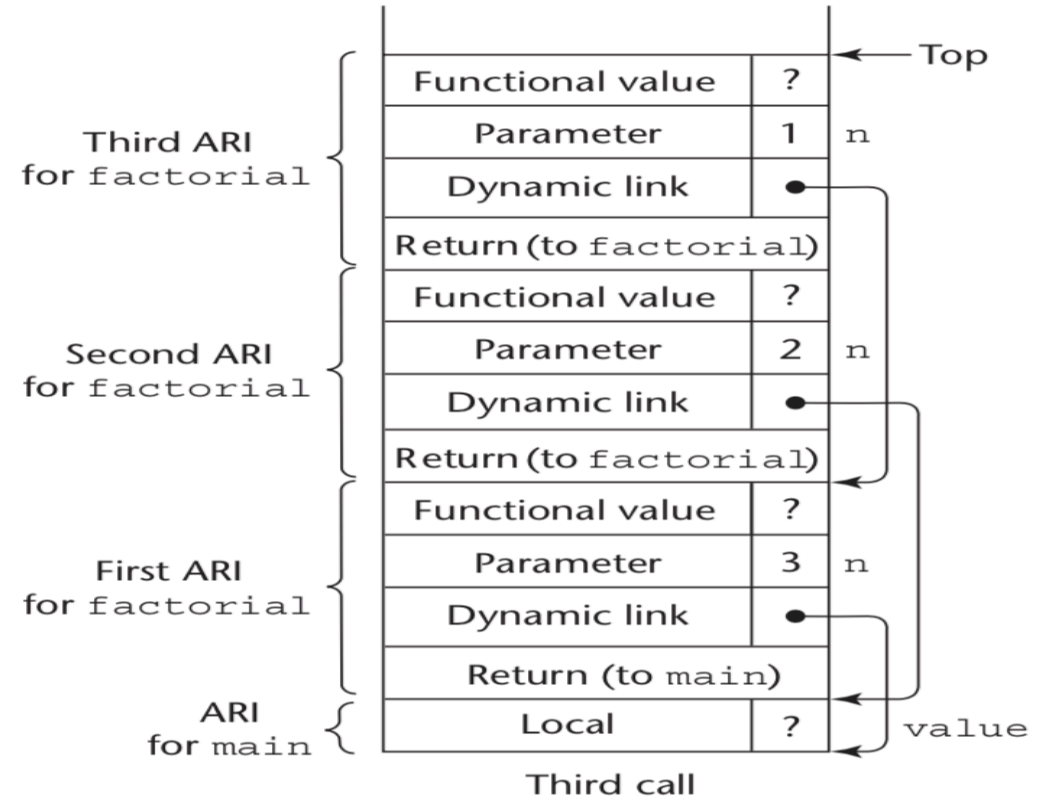
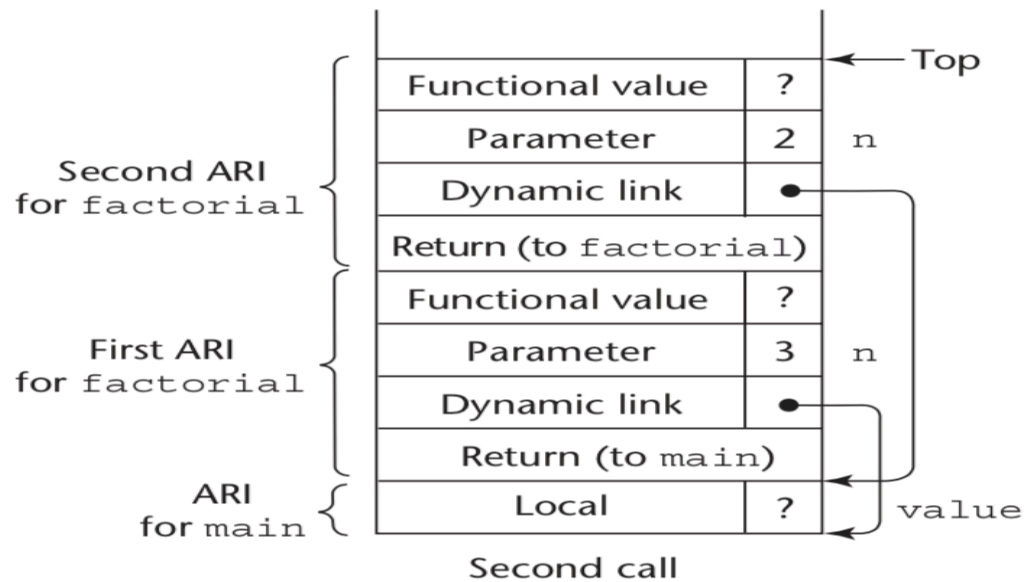
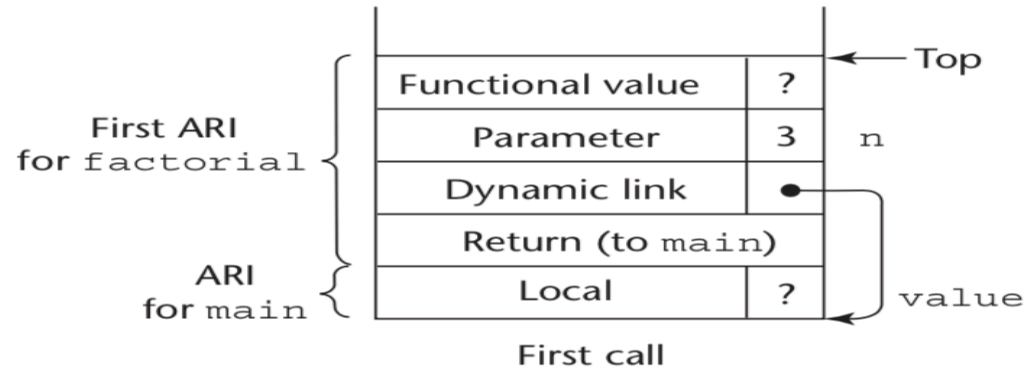
# Altprogramların Gerçekleştirilmesi

## Recursion

```
int factorial(int n) {  
    ←————— 1  
    if (n <= 1)  
        return 1;  
    else return (n * factorial(n - 1));  
    ←————— 2  
}  
void main() {  
    int value;  
    value = factorial(3);  
    ←————— 3  
}
```

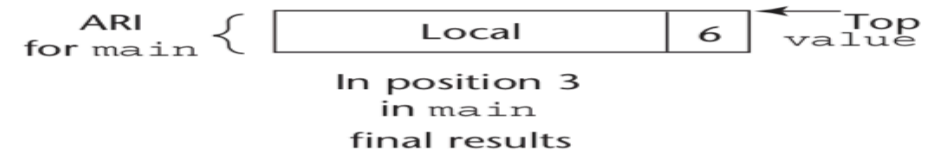
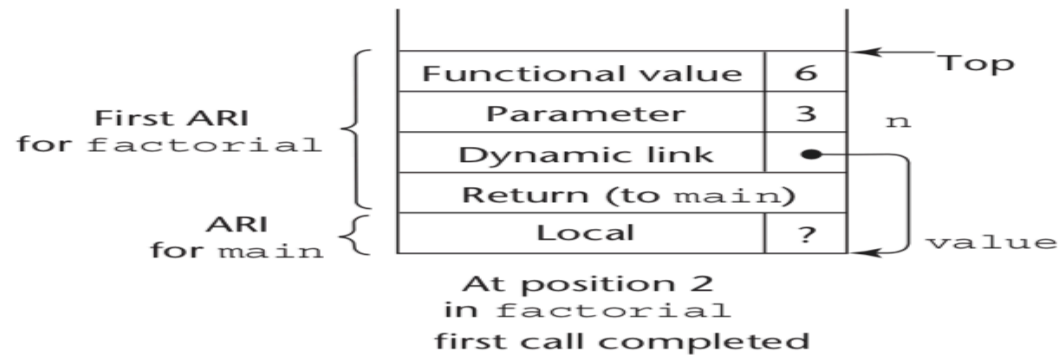
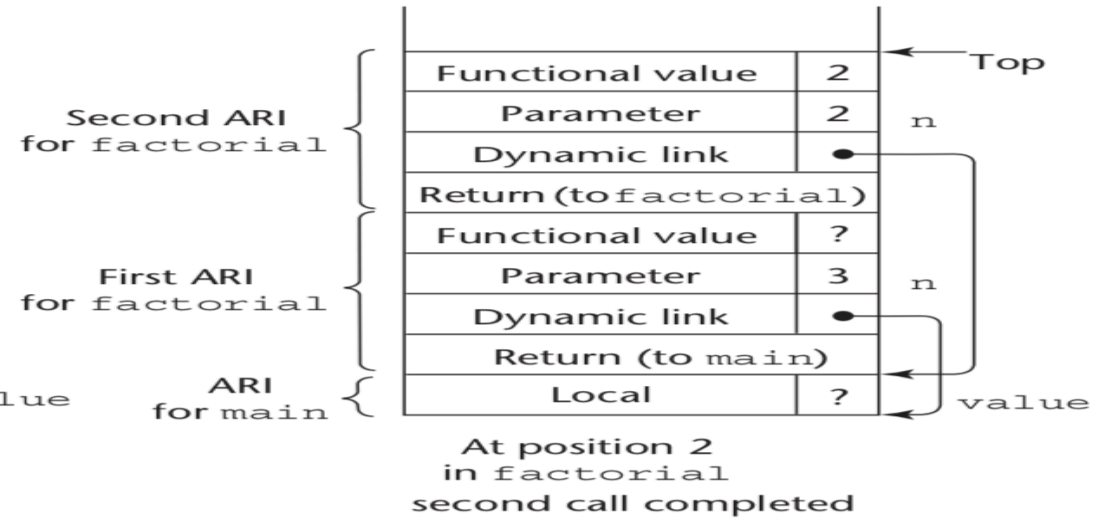
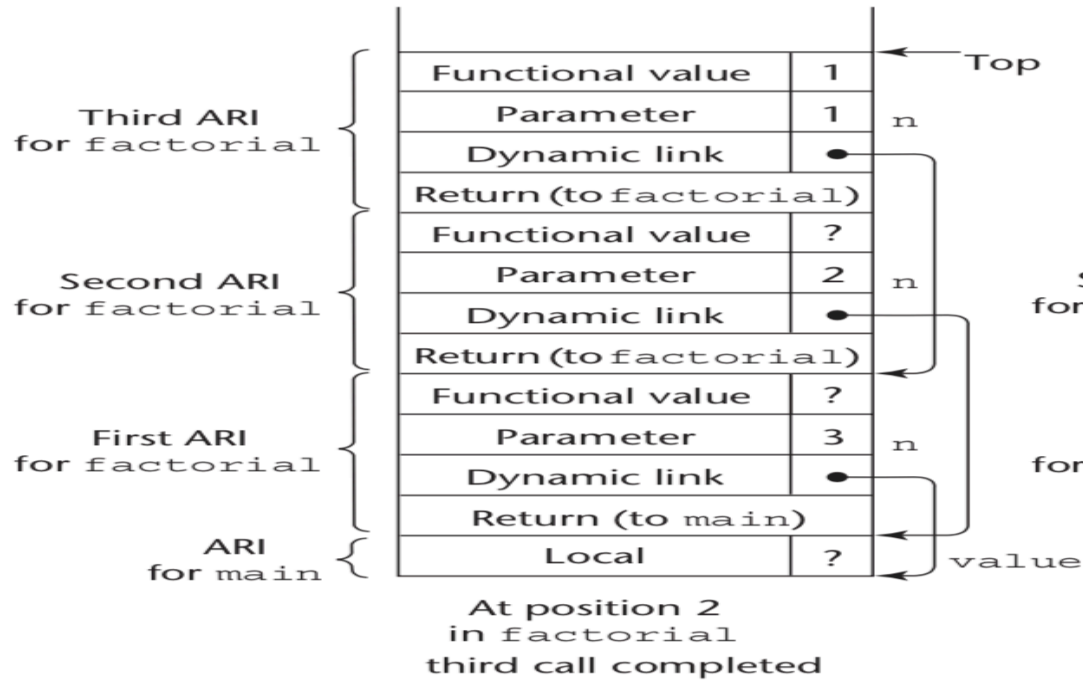
Functional value	n
Parameter	
Dynamic link	
Return address	

# Faktöriyel-1



ARI = activation record instance

# Faktöriyel-2



# Blocks

- İki veya daha fazla kod ifadesinin bir araya gelerek oluşturduğu yapıya blok denir.
- Her bir bloğun activation record' ı bulunmaktadır.
- Blok çalıştırıldığında bir aktivasyon kayıt örneği (activation record instance) oluşturulur.
- Parametresiz altprogram gibi işlem görür.

# Blocks

```
void main() {  
  int x, y, z;  
  
  while ( ... ) {  
    int a, b, c;  
    ...  
    while ( ... ) {  
      int d, e;  
      ...  
    }  
  }  
  while ( ... ) {  
    int f, g;  
    ...  
  }  
  ...  
}
```

