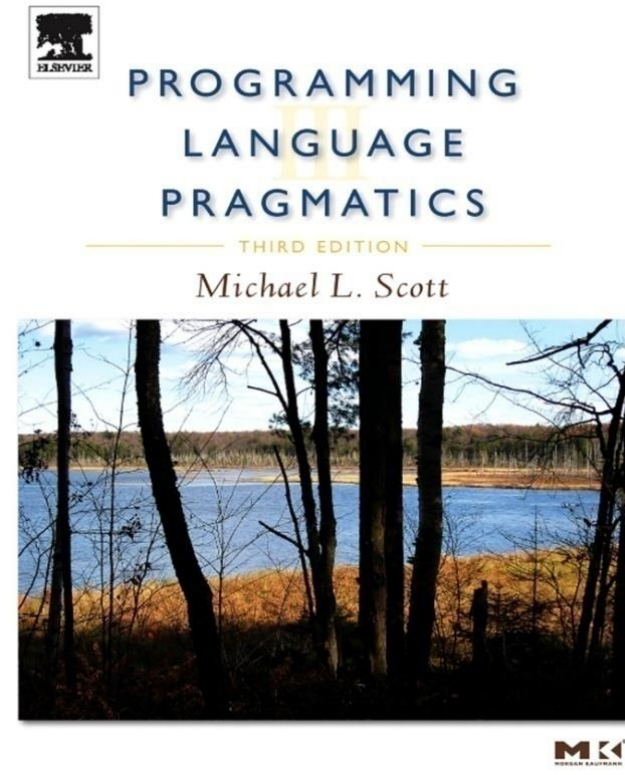
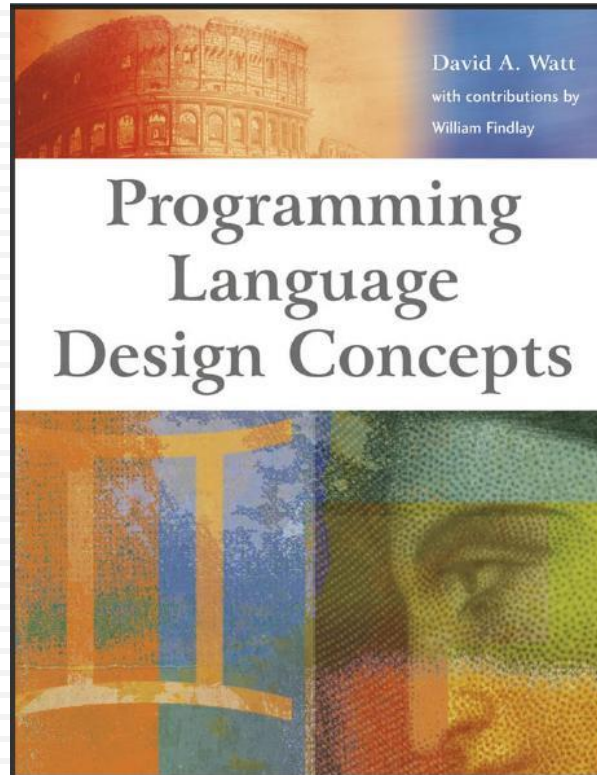
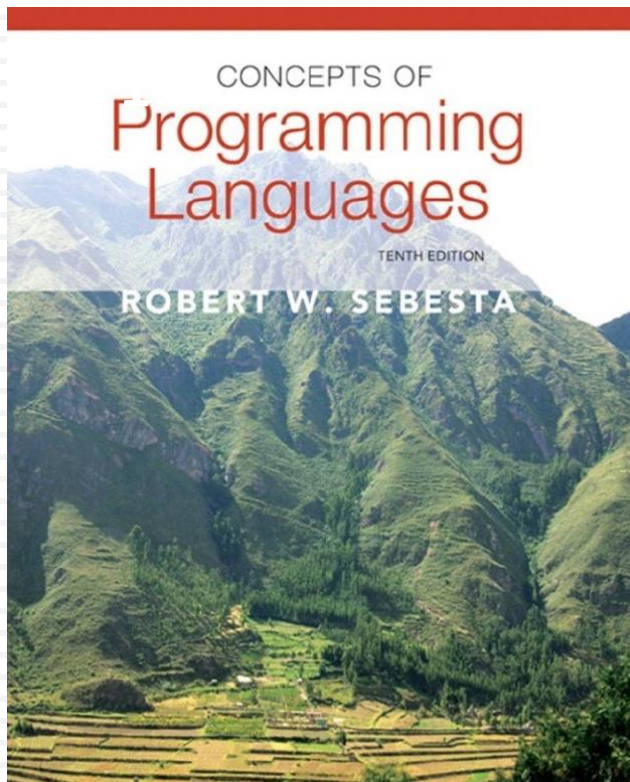


# Bölüm 8: Altprogramlar



# BÖLÜM 8- Konular

2

- Yazılım Mühendisliği Açısından Altprogramlar
- Altprogramların Özellikleri

# 8.1. YAZILIM MÜHENDİSLİĞİ AÇISINDAN ALTPROGRAMLAR

3

- Bir programda birden çok kez etkin duruma getirilebilecek bir dizi deyimın bir isim altında gruplanması ile **altprogramlar** oluşturulur.
- Böylece tek bir isim, bir dizi deyimi göstermekte ve bir soyutlama gerçekleştirmektedir.
- Programlama dillerinde, programlama dillerinin gelişiminin ilk dönemlerinde tanıtılan **işlev soyutlaması** (Process abstraction) ve 1980'li yıllardan başlayarak popülerlik kazanan **veri soyutlaması** (Data abstraction) olmak üzere iki türlü soyutlama gerçekleştirilebilir.
- Altprogramlar ile işlev soyutlaması amaçlanır. Altprogramların yazılım geliştirmeye önemli katkıları vardır.
- Veri soyutlaması (Data abstraction) – daha sonraki bölümlerde anlatılacaktır.

# 8.1. YAZILIM MÜHENDİSLİĞİ AÇISINDAN ALTPROGRAMLAR

4

- ❑ **Kaynak Tasarrufu** : Bir programda sıkça kullanılan işlemler için altprogramlar oluşturulması, programda tekrarları önleyerek, bellek ve zaman kazancı sağlar.
- ❑ **Soyutlama**: Bir dizi deyiminin gruplanması ile oluşturulan alt programları kullanan deyimler, altprogramlar ile sadece parametreler aracılığıyla iletişim kurarlar. Bu nedenle altprogramlar, bir işlevinin gerçekleştirimin, o işlevi kullanan program deyimlerinden gizlenmesini sağlarlar. Örneğin, bir sıralama altprogramının algoritması, o altprogramı kullanan deyimler etkilenmeden değiştirilebilirler.
- ❑ **Modüler Programlar Geliştirilmesi**: Altprogramlar, bir işin bir çok alt işe bölünmesini sağlayarak, program ayrıştırma için bir mekanizma sağlarlar. Bunun sonucu olarak bir yazılım üzerinde aynı anda birden fazla kişi çalışabilir.
- ❑ **Anlaşılabilirliğin Artması**: Bir programın bir çok alt işlere bölünmesi, programların anlaşılabilirliğini artırarak, programların sınanmasını ve okunabilirliğini kolaylaştırır.
- ❑ **Kütüphaneler Oluşturulması**: Bir çok dilde altprogram, birden çok uygulama arasında paylaşılabilir ve bir çok program tarafından kullanılabilir. Böylece birden fazla uygulama arasında paylaşılan altprogram kütüphaneleri oluşturulabilir.

# Altprogramların esasları

5

- Altprogramların genel özellikleri:
  1. Altprogramın bir tane giriş yeri olur.
  2. Altprogram çağırıldığında ve yürütülürken çağıran program bekler.
  3. Çağırılan program bitince kontrol her zaman çağıran programa geri döner.

# Temel tanımlamalar

6

- Altprogram tanımı: Altprogram soyutlamasının betimlenmesi.
- Altprogram çağırılması (subprogram call): altprogramın yürütülmesinin talep edilmesi.
- Altprogram başlığı (subprogram header): adı, parametreleri ve ne tip bir altprogram olduğuna dair bilgiler.
- Parametre profili: Altprogramın parametrelerinin sayısı, sırası ve tipleri.
- Altprogram protokolü: Parametre profili ve eğer fonksiyonsa döndüğü değer tipi.
- Altprogram bildirimi (subprogram declaration): protokol belirlenir ancak altprogramın gövdesi belirlenmez.
- Resmi parametre (formal parameter): Altprogram başlığında listelenip altprogram içinde kullanılan parametre.
- Gerçek parametre (actual parameter): Altprogram çağırılırken adres veya değeri için yazılan parametreler.

# Altprogramlarda tasarım hususları

7

1. Hangi parametre geçirme (parameter passing) metotları uygulanacak?
2. Parametrelerin tipleri kontrol edilecek mi?
3. Lokal değişkenler statik mi, dinamik mi olacak??
4. Altprogram tanımlamaları başka altprogramların tanımlamalarında yer alabilecek mi (iç içe tanımlama)?
5. Eğer altprogramlar parametre olarak başka altprogramlara geçirilebilirse ve altprogramlar iç içe tanımlanabilirse bu durumda parametre ile geçen altprogramın referans çevresi (referencing environment) (erişilebilir tüm isimler ve bunların kapsamlarıdır) ne olacak?
6. Altprogramlar fazla yüklü (overloaded) (aynı isimli başka altprogramlar) olabilecek mi?
7. Altprogramların cinsine özgü (generic) (farklı çağrılarda farklı veri tiplerini işleyebilen) olmasına izin verilecek mi?

# Lokal Referans Çevresi

8

- Eğer lokal değişkenler yığıt dinamikse (stack-dynamic):
  - ▣ Avantaj:
    - a. Özyinelemeye destek.
    - b. Lokaller için ayrılan alan altprogramlar tarafından paylaşılabilir.
  - ▣ Dezavantajları:
    - a. Tahsis/geri verme zamanı.
    - b. Dolaylı adresleme. Çoğu bilgisayarda yavaş.
    - c. Altprogramlar geçmişe hassas değil. Altprogram bitince bütün lokal değerler unutuluyor.
- Statik lokallerde avantaj ve dezavantajlar yer değiştirir.

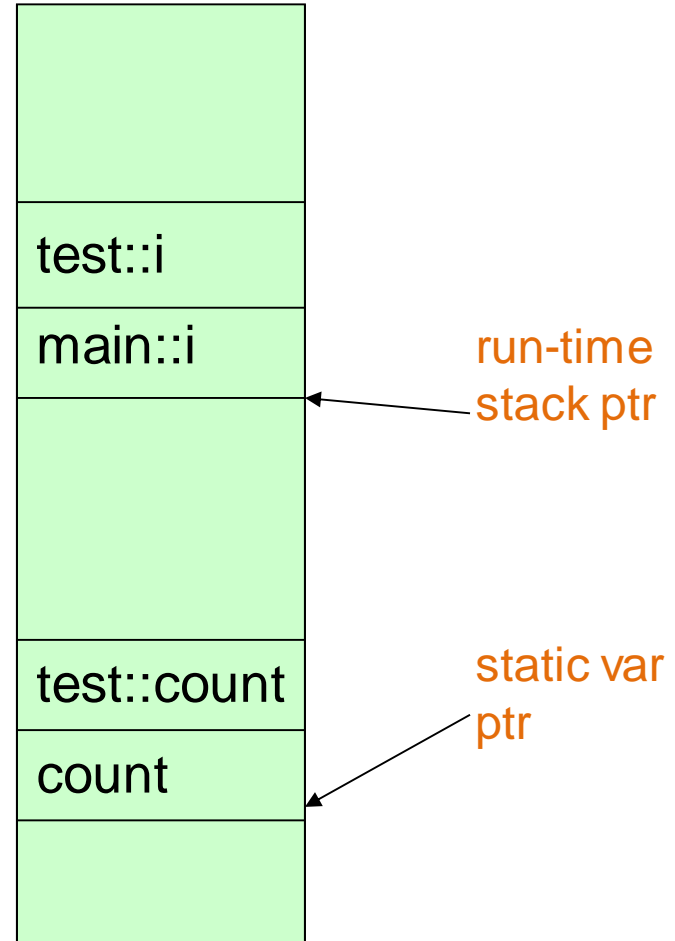


# Örnek 1

9

```
#include <stdio.h>
int count;
main( ) {
    int i;
    for (i=0; i<=10; i++)
    { test( ); }
}
test( ) {
    int i;
    static int count = 0;
    count = count + 1;
}
```

Sanal adres uzayı



# Örnek 1

10

```
#include <stdio.h>
int count;
main( ) {
    int i ;
    for (i=0; i<=10; i++)
    { test( ) ; }
}
test( ) {
    int i ;
    static int count = 0;
    count = count + 1 ;
}
```

	Tip- bağlama	Bellek- bağlama
count	statik	statik
main::i	statik	stack-dinamik
test::i	statik	stack-dinamik
test::count	statik	statik

# Lokal Referans Çevresi

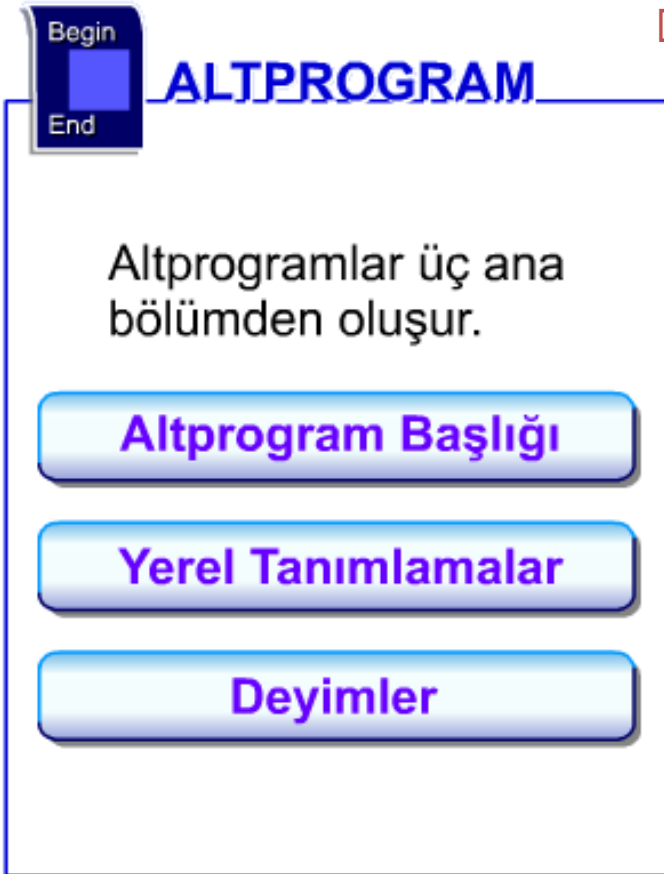
11

## □ Dil Örnekleri:

- 1. FORTRAN 77, 90, 95 – Özyineleme  
öngörülmediğinden lokal değişkenler statik. Dinamik için "recursive subroutine" tanımlaması gerekli. Bu durumda "save" tanımlanmış değişkenler yine de statik olur.
- 2. C – ikisi de: **static** tanımlanmış değişkenler statik; varsayılan yığıt dinamik (stack dynamic).
- 3. Pascal, Java, ve Ada – sadece dinamik.

## 8.2.1. Altprogramların Bölümleri

12



- Bir altprogram yandaki şekilde sıralanan bölümlerden oluşur.

## 8.2.1.1. Altprogram Başlığı

13

- *Altprogram başlığı*, izleyen program biriminin bir altprogram tanımı olduğunu, altprogram ismini, varsa parametreler listesini ve varsa altprogramın döndürdüğü değer tipini belirterek altprogramın arayüzünü oluşturur.
- Bir altprogram başlığındaki parametrelere **resmi (formal) parametreler** adı verilir. Bir altprogramın **parametre profili**, resmi parametrelerinin sayısı, sırası ve tipini gösterir.

### Altprogram Başlığı

- Altprogram İsmi
- Parametreler Listesi
- Altprogramın Döndürdüğü Değer

Örnek

procedure **ornek** (**a: integer, b: real**)

altprogram ismi

resmi parametreler

## 8.2.1.2. Yerel Tanımlamalar

14



- Altprogram başlığından sonra yer alan tanımlamalar altprograma yerel olup, bu değişkenlere erişim altprogram deyimleri ile sınırlı olduğu için, **yerel değişkenler** olarak adlandırılır.
- Bir altprogramda başvuru yapılan bir ismin bir tanımlama ile bağlanması, o programlama dilinin kapsam kuralları ile belirlenir.

## 8.2.1.3. Deyimler

15



- Altprogramın gerçekleştirdiği işlemler, bir dizi deyim olarak yer alır.
- Yerel tanımlamalar ve çalıştırılabilir deyimler, altprogram gövdesini oluştururlar.

## 8.2.2. Yordam ve Fonksiyonlar

16

- Bir altprogram, bir **yordam** veya bir **fonksiyon** olabilir.
- İki altprogram türü arasındaki fark, fonksiyonların çağıran program birimine bir sonuç değeri döndürmelerinin şart olmasıdır.
- Fonksiyon altprogramlar bir programlama dilinde var olan (*built-in*) işlemcilere benzer, yordam altprogramlar ise bir programlama dilinde var olan (*built-in*) deyimlere benzer olarak düşünülebilir.



## 8.2.2. Yordam ve Fonksiyonlar

17

### Pascal

Örnek

Ortalama isimli bir  
yordam başlığı

`procedure ortalama (parametreler);`

### C

Örnek

Ortalama isimli bir  
fonksiyon başlığı

`ortalama (parametreler);`

- **Pascal ve Ada'da Yordamlar:** Yordamlar, Pascal ve Ada'da `procedure` anahtar kelimesi ile belirtilirler.
- **C'de Yordamlar:** C'de yordamların belirtilmesi için özel bir anahtar kelime kullanılmaz.
- C ve C++'da sadece fonksiyonlar yer alırsa da, bu fonksiyonlar yordamların işlevlerini de gerçekleştirebilirler (void fonksiyonlar).
- Çoğu popüler *imperative* dilde hem fonksiyonlar hem de yordamlar bulunur. C ve C++'da sadece fonksiyonlar yer alırsa da, bu fonksiyonlar yordamların işlevlerini gerçekleştirebilirler.
- FORTRAN'da genel özellikleri diğer dillerdeki yordamlara benzer olmakla birlikte, yordamlara `SUBROUTINE` ismi verilir.

## 8.2.3. Parametreler

18

Gerçek Parametre  
Konumsal Parametre

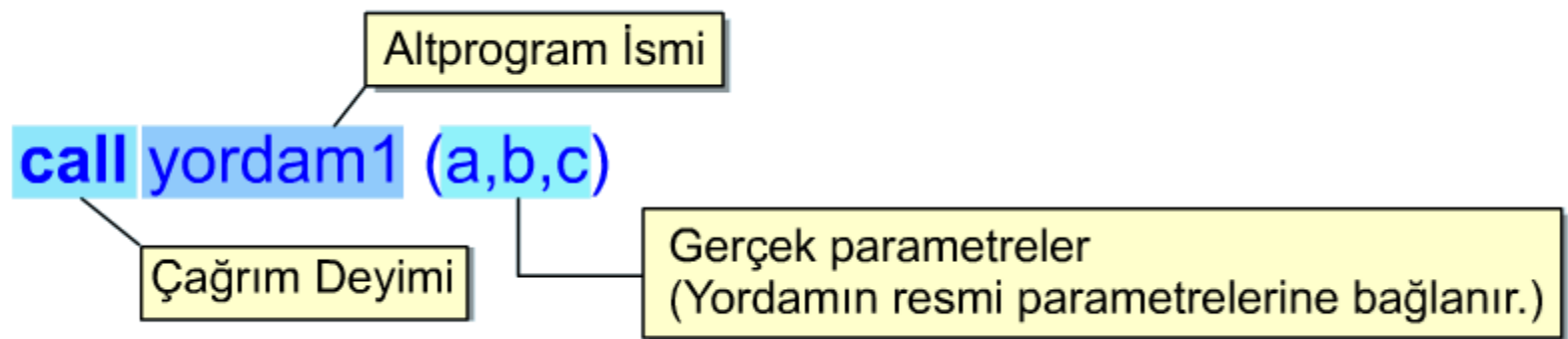
Anahtar Kelime  
Parametre Yöntemi

- **Resmi parametreler**, altprogram başlığında bulunan parametreleri, **parametre profili** ise altprogramın resmi parametrelerinin sayısı, sırası ve tipini gösterir.
- Bunlara ek olarak **gerçek parametrelerin** ve resmi parametrelerin ilişkilendirilmesi için kullanılan iki yöntem olan **konumsal** ve **anahtar kelime parametre** yöntemleri bulunmaktadır.

## 8.2.3.1. Yordam Çağırımı

19

- Bir yordamın etkin duruma gelmesi için, özel bir çağırım deyimi kullanılmalıdır. Bu çağırım deyiminde, altprogramın ismine ek olarak, yordamın resmi parametreleriyle eşleştirilecek **gerçek** (*actual*) **parametreler** de yer alır.



## 8.2.3.1. Yordam Çağırımı

20

- Diğer dillerden farklı olarak C'de özel bir çağrım deyimi yoktur ve sadece yordam isminin parametreleriyle birlikte yazılması, çağrım için yeterlidir.
- Örnek: ***yordam1 (a,b,c);***
- deyimi ile yordam1 isimli yordam etkin duruma getirilmektedir.

## 8.2.3.1. Yordam Çağırımı

21

- Bir yordam çağırımı gerçekleşince, etkin olan program birimindeki komutların çalışması durdurulur ve yordam etkin duruma geçer.
- Yordam gövdesindeki komutlar çalıştırıldıktan sonra etkinlik, yeniden çağırımın yapıldığı program birimine geçirilir ve yordam çağırım deyimini izleyen ilk deyim etkin olur.

## 8.2.3.2. Gerçek/Resmi parametrelerin eşleşmesi:

22

### 1. Konumsal (Positional)

- Eğer bir yordam çağrımında gerçek parametreler ile resmi parametreler arasındaki bağlama, parametrelerin çağrım deyimindeki ve yordam başlığındaki konumuna göre yapılıyorsa, bu parametrelere **konumsal** (*positional*) **parametre** adı verilir.
- Bir çok programlama dilinde uygulanan bu yöntem, parametre sayısı az olduğu zaman kullanışlıdır.
  - ▣ Dezavantaj: parametre sayısı artınca karıştırılabilir.
  - ▣ Birçok dilde resmi ve gerçek parametreler eşleşmelidir. Ancak, C, C++, Java ve Perl'de eksiklik durumunda eşleşme programcının sorumluluğundadır. Bu durum belirsiz parametrelili fonksiyonlara izin verir.

## 8.2.3.2. Gerçek/Resmi parametrelerin eşleşmesi:

23

### 2. Anahtar kelime

- Resmi parametreler anahtar kelime olarak kullanılarak gerçek parametrelerle eşleştirilirler.
  - ▣ ADA örneği, **`SORT(LIST => A, LENGTH => N)`**.
  - ▣ Avantaj: sıra önemli değil.
  - ▣ Dezavantaj: kullanıcı resmi parametre adını bilmelidir.
  - ▣ C++, Fortran 95, Ada ve PHP'de resmi parametrelerin varsayılan değeri olabilir:

```
procedure SORT (LIST : LIST_TYPE;  
    LENGTH : INTEGER := 100) ;  
  
    . . .  
  
SORT (LIST => A) ;
```

## 8.2.3.2. Gerçek/Resmi parametrelerin eşleşmesi:

24

### 3. Değişken sayılı parametreler

#### ▣ C#

```
public void DisplayList(params int[] list) {  
    ... }  
myObject.DisplayList(2, 4, 6, 8);
```

▣ Ruby, Python gibi dillerde de bu özellik var.



## 8.2.3.1.1. Konumsal Parametreler

25

- Aşağıdaki örnekte görüldüğü gibi bir çağrım deyimi ile etkin duruma geçen *ortalama* yordamında, **c** resmi parametresi, **a** gerçek parametresi ile, **d** resmi parametresi, **b** gerçek parametresi ile bağlanıyorsa, konumsal parametreler yaklaşımı uygulanmıştır.

```
call ortalama(a,b);  
Procedure ortalama(c: integer; d:integer);  
begin  
    ...  
end;
```

Resmi Parametre	Gerçek Parametre
<b>c</b>	<b>a</b>
<b>d</b>	<b>b</b>

## 8.2.3.1.2. Anahtar Kelime Parametre Yöntemi

26

- Gerçek ve resmi parametreler arasındaki bağlamayı konuma göre belirlemek yerine, her iki parametrenin de ismini belirterek gösterme, **anahtar kelime parametre** yöntemi olarak adlandırılır.
- Bu durumda çağırım deyiminde, hem resmi, hem de gerçek parametrelerin isimleri belirtilir.

## 8.2.3.1.2. Anahtar Kelime Parametre Yöntemi

27

- ❑ Örneğin aşağıdaki yordam çağrımında, **uzunluk** gerçek parametresi, **altuzunluk** resmi parametresine bağlanmaktadır.

```
CALL ORTALAMA(uzunluk=>altuzunluk, sayac=>altsayac);
```

- ❑ Anahtar kelime parametre yöntemi, parametre sayısının çok olduğu durumda yararlı bir yöntemdir.
- ❑ Ada ve FORTRAN 90'da hem anahtar kelime hem de konumsal parametreler yöntemi kullanılabilmektedir.

## 8.2.3.2. Fonksiyon Çağırımı

28

### C

C'de geri dönüş değerinin tipi, fonksiyon adından önce yazılmalıdır.

Örnek

```
int topla(int say1, int say2);
```

Fonksiyon Adı

Geri Dönüş Tipi

### Pascal

Pascal'da geri dönüş değerinin tipi tanımın en sonunda yazılmalıdır.

Örnek

```
function kare( say3: integer): integer ;
```

Geri Dönüş Tipi

## 8.2.3.2. Fonksiyon Çağırımı

29

- Fonksiyonlar gerekli parametrelerle birlikte yer aldıkları ifadenin çalışması ile etkin duruma geçerler. Bir fonksiyon etkin olunca, o fonksiyonun çalışması tamamlanıncaya kadar, etkin olan program birimi durdurulur.
- Bir fonksiyonun çalışması bitince, sonuç olarak tek bir değer üretirler ve bu değer, fonksiyonu çağıran ifadeye döndürülerek fonksiyon çağrısının yerini alır. Böylece etkinlik yeniden ilk program birimine geçer.
- **Geri Dönüş Tipi:**
- Fonksiyonlar bir değer geri döndürdükleri için, fonksiyon tanımında sonuç olarak geri döndürülen değerın tipi belirtilmelidir.

## 8.2.4. Parametre Aktarım Yöntemleri

30

- Bir yordam etkin duruma getirilirken, çağrım deyimindeki gerçek parametreler ile yordamın resmi parametrelerine farklı değerler aktarılabilir.
- Gerçek parametreler yordamlara aktarılacak değerleri gösterdikleri için, değişken, sabit veya ifade olabilir.
- Resmi parametreler ise bu değerleri tutacak bellek yerlerini gösterdikleri için değişken olmak durumundadır.

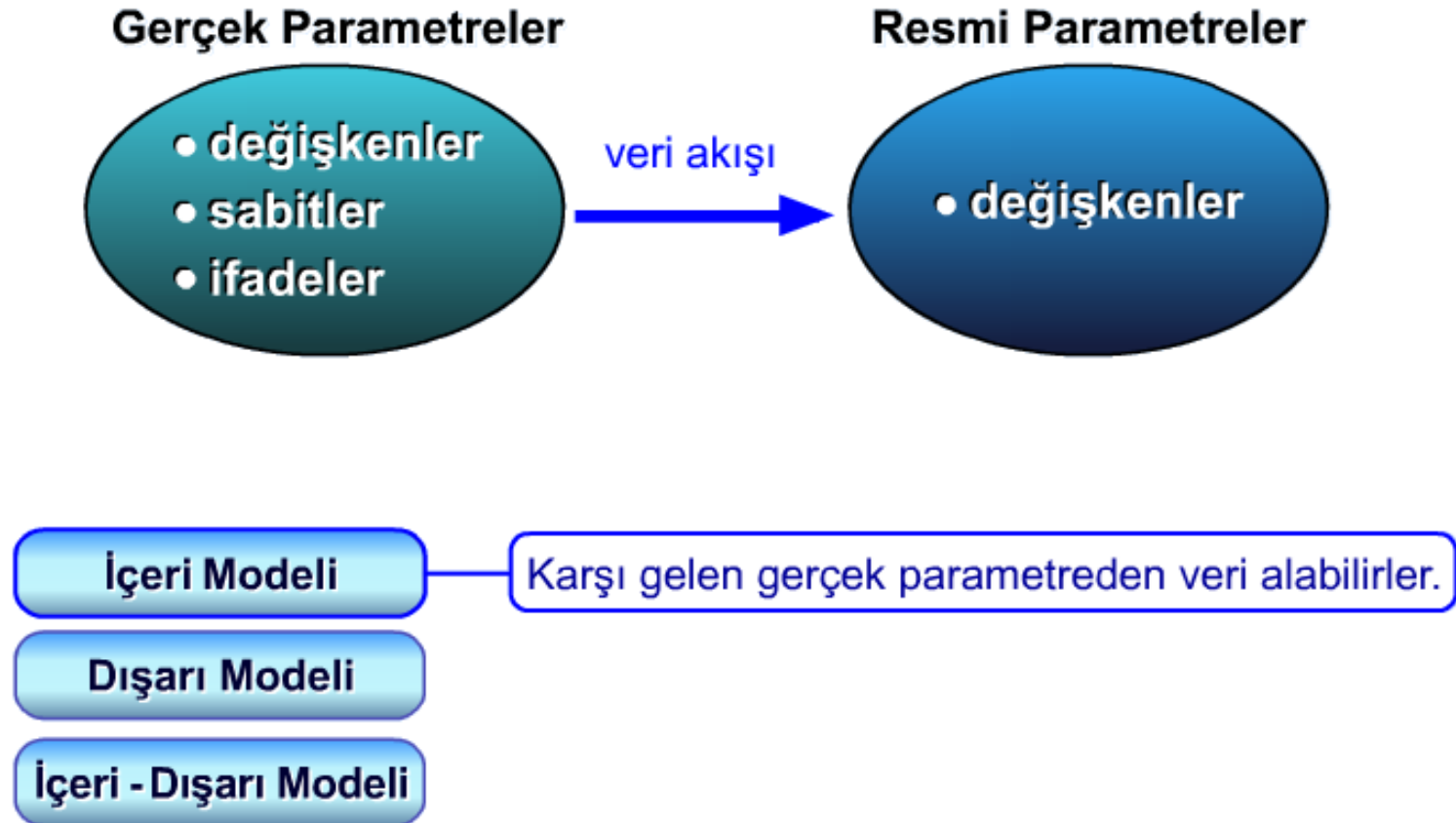
## 8.2.4. Parametre Aktarım Yöntemleri

31

- ❑ **Veri Akışı Modelleri:**
- ❑ Bir yordam, resmi parametrelerinin değerini veya resmi parametre olmayan ancak erişebildiği değişkenlerin değerlerini değiştirebilir.
- ❑ Resmi parametreler, içeri, dışarı ve hem içeri hem de dışarı modeli uygulayabilirler.

## 8.2.4. Parametre Aktarım Yöntemleri

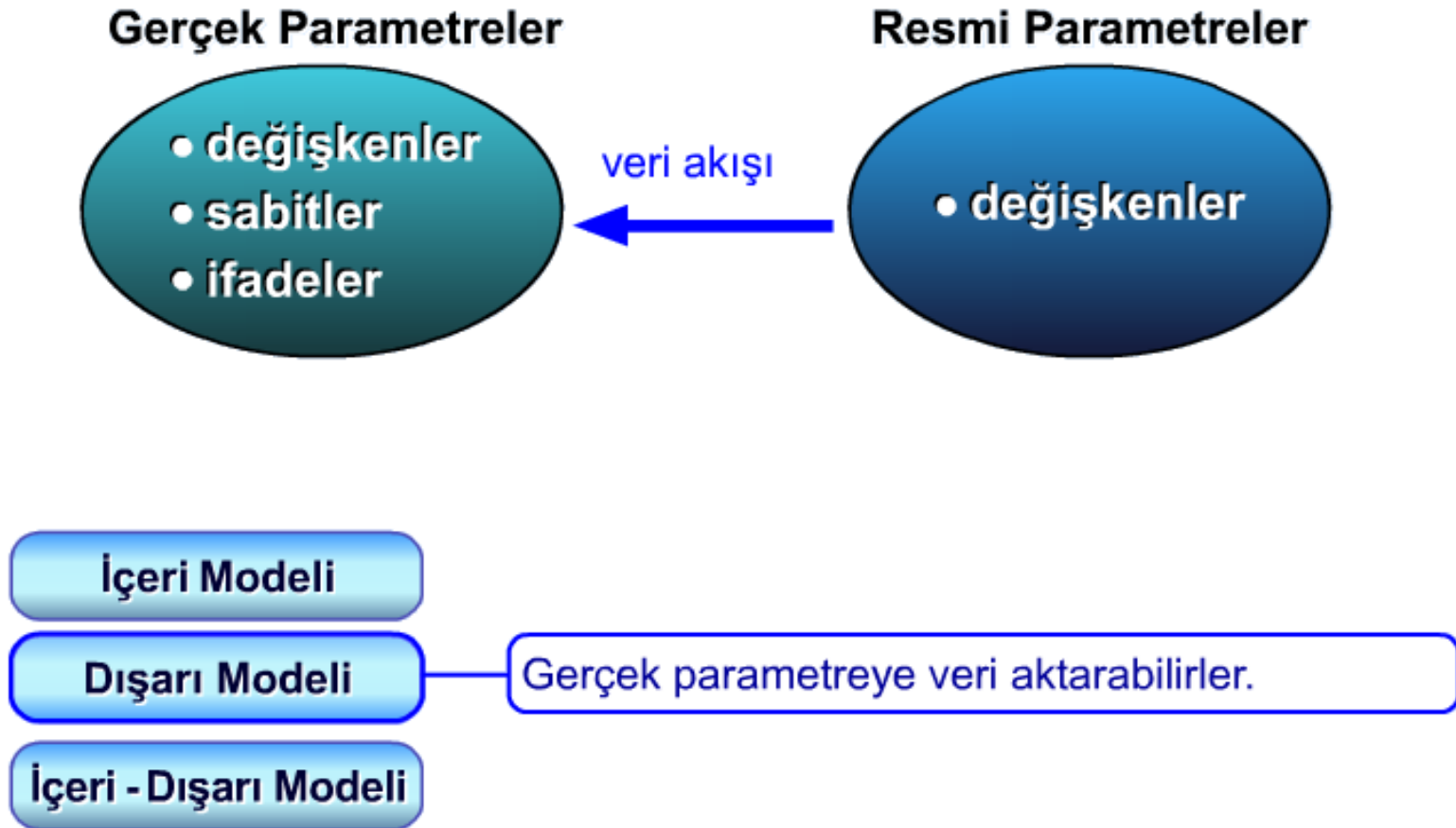
32





## 8.2.4. Parametre Aktarım Yöntemleri

33



## 8.2.4. Parametre Aktarım Yöntemleri

34

### Gerçek Parametreler



### Resmi Parametreler



İçeri Modeli

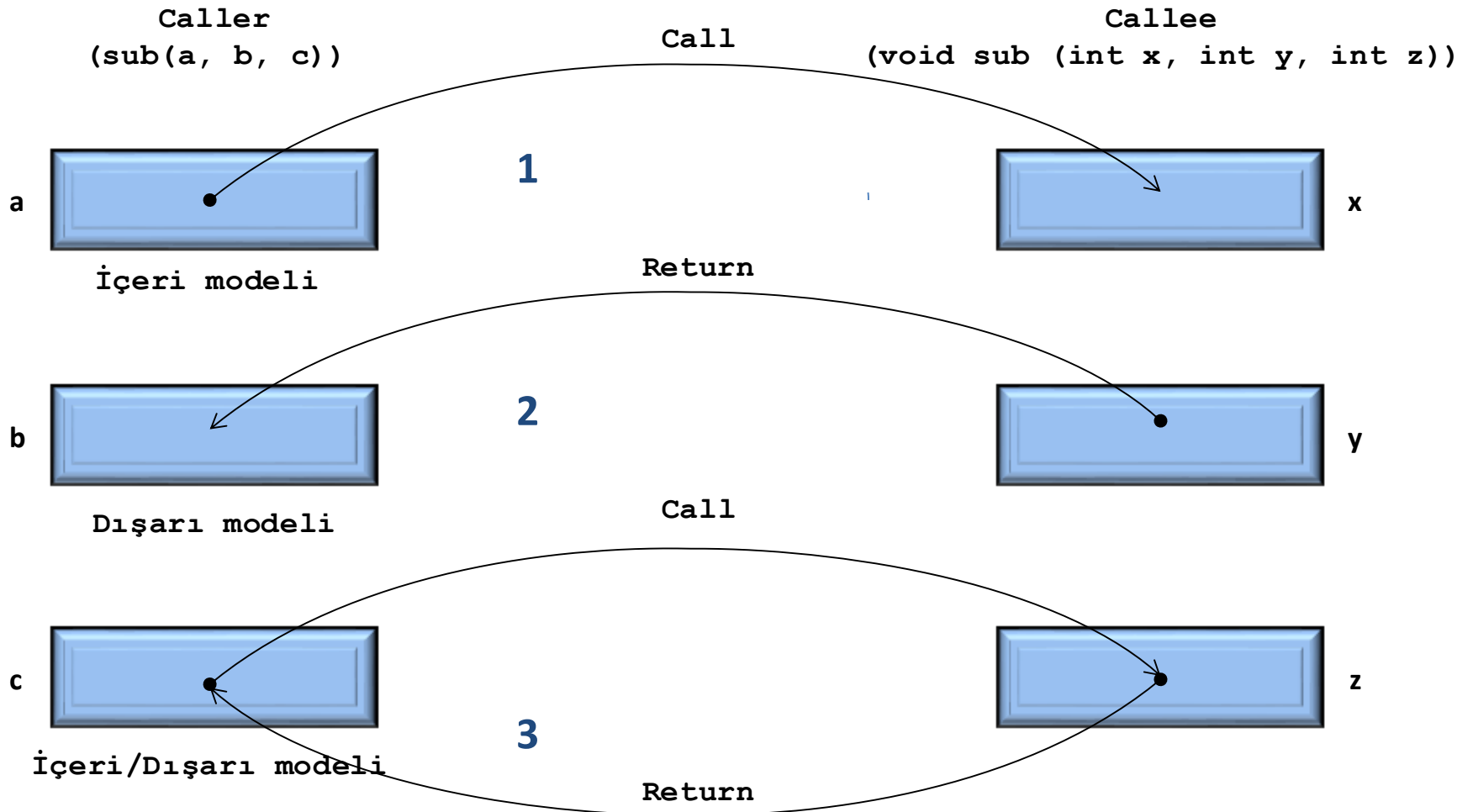
Dışarı Modeli

İçeri - Dışarı Modeli

Gerçek parametreden hem veri alabilir hem de gerçek parametreye veri aktarabilirler.

## 8.2.4. Parametre Aktarım Yöntemleri

35



## 8.2.4. Parametre Aktarım Yöntemleri

36

- ❑ Resmi parametreler ve gerçek parametreler arasındaki veri akışı, parametre aktarım yöntemlerine göre gerçekleştirilir.
- ❑ Değer ile çağırma, sonuç ile çağırma, değer ve sonuç ile çağırma yöntemlerinde, gerçek ve resmi parametreler arasındaki veri fiziksel olarak kopyalanarak aktarılmakta, başvuru ile çağırma yönteminde ise veri yerine verinin erişim yolu aktarılmaktadır.

ÇAĞIRMA YÖNTEMLERİ	
Değer ile Çağırma Sonuç ile Çağırma Değer ve Sonuç ile Çağırma	} Gerçek değer fiziksel olarak kopyalanarak aktarılır.
Başvuru ile Çağırma İsim ile Çağırma	
	} Verinin erişim yolu aktarılır.

## 8.2.4.1. Değer ile Çağırma (*Call by Value*)

37

- Değer ile çağırma yöntemi, içeri modelinin gerçekleştirmesidir. Bu yöntemde resmi parametre, karşı gelen gerçek parametrenin değeriyle ikklendikten sonra, altprograma yerel bir değişken olarak nitelendirilir.
- Gerçek parametre, sabit, değişken veya ifade olabilir.
- Bu yöntem, sadece gerçek parametreden resmi parametreye değer geçişi olduğu için en güvenilir parametre aktarım yöntemidir. Ancak, bu değer geçişi sırasında fiziksel olarak veri kopyalanması gerçekleşir . Yani, resmi parametre için de bellek ayrılması gerekir.

## 8.2.4.1. Değer ile Çağırma (*Call by Value*)

38

### DEĞER İLE ÇAĞIRMA İŞLEMİ

**call** ortalama (a, b)

Gerçek  
Parametreler

altprogram ortalama (c, d)

.....  
.....  
.....

Resmi  
Parametreler

altprogram sonu

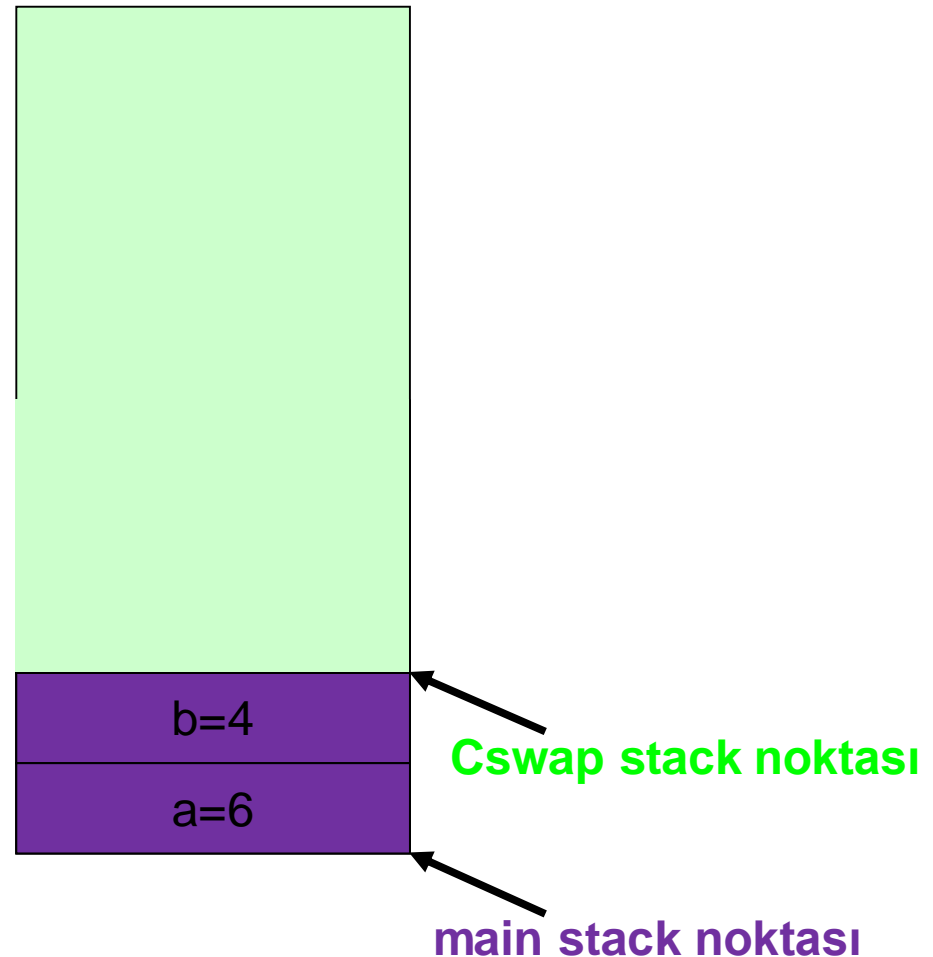
- $a \rightarrow c$  (a'nın değeri c'ye aktarılır.)
- c yerel bir değişken olarak nitelendirilir.

# Değer ile Çağırma (Örnek)

39

```
main( ) {  
    int a = 6;  
    int b = 4;  
    Cswap(a, b);  
    // a = 6  
    // b = 4  
}
```

```
Cswap(int c, int d) {  
    int temp = c;  
    c = d;  
    d = temp;  
}
```



## 8.2.4.1. Değer ile Çağırma (*Call by Value*)

40

```
void f(int x) {  
    x = 3;  
}  
  
int main() {  
    int x = 0;  
    f(x);  
    printf("%d\n", x);  
}
```

Sadece gerçek parametreden formal parametreye değer geçişi olduğu için en güvenilir parametre aktarım yöntemidir

Programın çıktısı= 0 olacaktır.

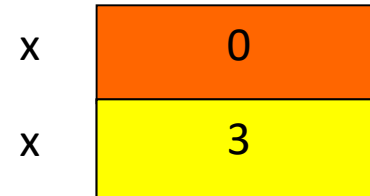


## 8.2.4.1. Değer ile Çağırma (*Call by Value*)

41

- Gerçek parametre, resmi parametrenin yığındaki yerine kopyalanır

```
void f(int x) {  
    x = 3;  
}  
  
int main() {  
    int x = 0;  
    f(x);  
    printf("%d\n", x);  
}
```



- Resmi parametrenin değişimi gerçek parametreye yansımadı!

## 8.2.4.2. Sonuç ile Çağırma (*Call by Result*)

42

- Sonuç ile çağırma yöntemi, dışarı modelinin gerçekleştirimidir.
- Bu yöntemde çağırım deyimi ile altprograma bir değer aktarılmazken, gerçek bir parametreye karşı gelen resmi parametrenin değeri, altprogram sonunda, denetim yeniden çağıran programa geçmeden önce, gerçek parametreyi gösteren değişkene aktarılır. Bu tanımlamadan anlaşıldığı gibi, gerçek parametrenin değişken olması zorunludur.
- Gerçek parametreye karşı gelen resmi parametre, altprogramın çalışması süresince yerel değişkendir.
- Bu yöntemin uygulanmasındaki güçlük, gerçek parametrenin değerinin resmi parametreye aktarılmasının önlenmesidir.

## 8.2.4.2. Sonuç ile Çağırma (*Call by Result*)

43

### SONUÇ İLE ÇAĞIRMA İŞLEMİ



- c'nin değeri altprogram sonunda a'ya aktarılır.  $c \rightarrow a$



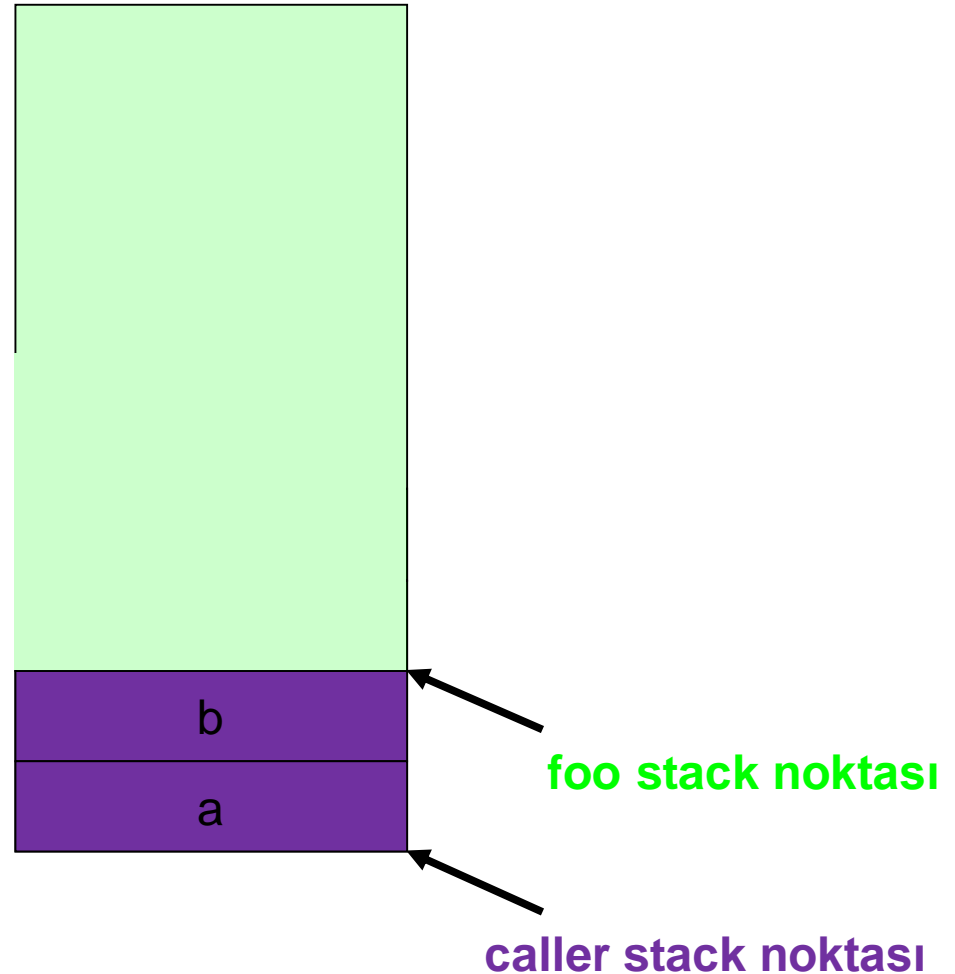
Gerçek parametrenin değişken olması zorunludur.

# Sonuç ile Çağırma (Örnek)

44

```
caller( ) {  
    int a ;  
    int b ;  
    foo(a, b);  
    // a = 6  
    // b = 4  
}
```

```
foo(int c, int d ) {  
    c = 6 ;  
    d = 4 ;  
}
```



```
void plus(by-value int a, by-value int b, by-  
    result int c)  
{  
    c = a+b;  
}  
  
void f()  
{  
    int x = 3;    int y = 4;    int z;  
    plus(x, y, z);  
    write z;  
}
```

```
m, n : integer;

    procedure r (k,j:integer)
    begin
        k := k+1;
        j := j+2;
    end r;

...

m := 5;
n := 3;
r(m,n);
write m,n;
```

- *r* prosedüründe hata:  
*başlatılmayan parametreler  
kullanamaz!*

## 8.2.4.3. Değer ve Sonuç ile Çağırma (*Call by Value Result*)

47

- Değer ve sonuç ile çağırma yöntemi, içeri-dışarı modelinin gerçekleştirimi olup, değer ile çağırma ve sonuç ile çağırma yöntemlerinin birleşimidir.
- Bu yöntemde, gerçek parametrenin değeri ile karşı gelen resmi parametrenin değeri ilklendikten sonra resmi parametre, altprogramın çalışması süresince yerel değişken gibi davranır ve altprogram sona erdiğinde resmi parametrenin değeri gerçek parametreye aktarılır. Bu yöntemde de gerçek parametrenin değişken olması zorunludur.
- Bu yöntemin dezavantajları, parametreler için birden çok bellek yeri gerekmesi ve değer kopyalama işlemlerinin zaman almasıdır.

# Değer ve Sonuç ile Çağırma (Örnek)

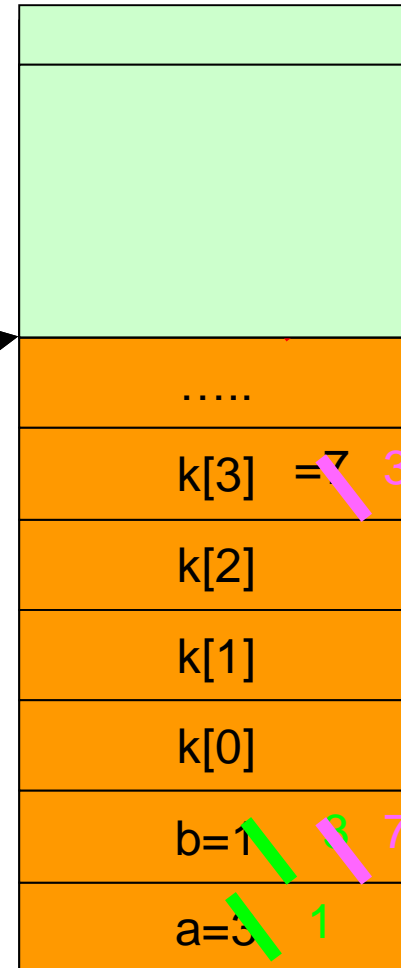
48

```
integer a = 3 ;  
integer b = 1 ;  
integer k[10] ;  
k[3] = 7 ;  
swap(a, b);  
swap(b, k[b]);
```

```
procedure swap(a : in out integer,  
               b : in out integer) is
```

```
temp : integer;  
begin  
    temp := a ;  
    a := b ;  
    b := temp ;  
end swap;
```

swap stack noktası



main stack noktası



```
int x=0;
int main()
{
    f(x);
    .....
}
void f(int a) {
    x=3;
    a++;
}
```

x'in son değeri değer-sonuç aktarımına göre 1 olacaktır.

Parametreler için birden çok bellek yeri gerekmesi ve değer kopyalama işlemlerinin zaman almaktadır.

```
{  
    m,n : integer;  
  
    procedure r (k,j :  
        integer)  
    begin  
        k := k+1;  
        j := j+2;  
    end r;  
  
    ...  
    m := 5;  
    n := 3;  
    r(m,n);  
    write m,n;  
}
```

Çıktı: 6 5

## 8.2.4.4. Başvuru ile Çağırma (*Call by Reference*)

51

- Başvuru ile çağırma yöntemi de gerçek ve resmi parametreler arasında iki yönlü veri aktarımı sağlar.
- Ancak önceki yöntemlerden en önemli farkı, altprograma verinin adresinin aktarılmasıdır.
- Bu adres aracılığıyla altprogram, çağıran program ile aynı bellek yerine erişebilir ve gerçek parametre, çağıran program ve altprogram arasında ortak olarak kullanılır.

## 8.2.4.4. Başvuru ile Çağırma (*Call by Reference*)

52

### BAŞVURU İLE ÇAĞIRMA İŞLEMİ



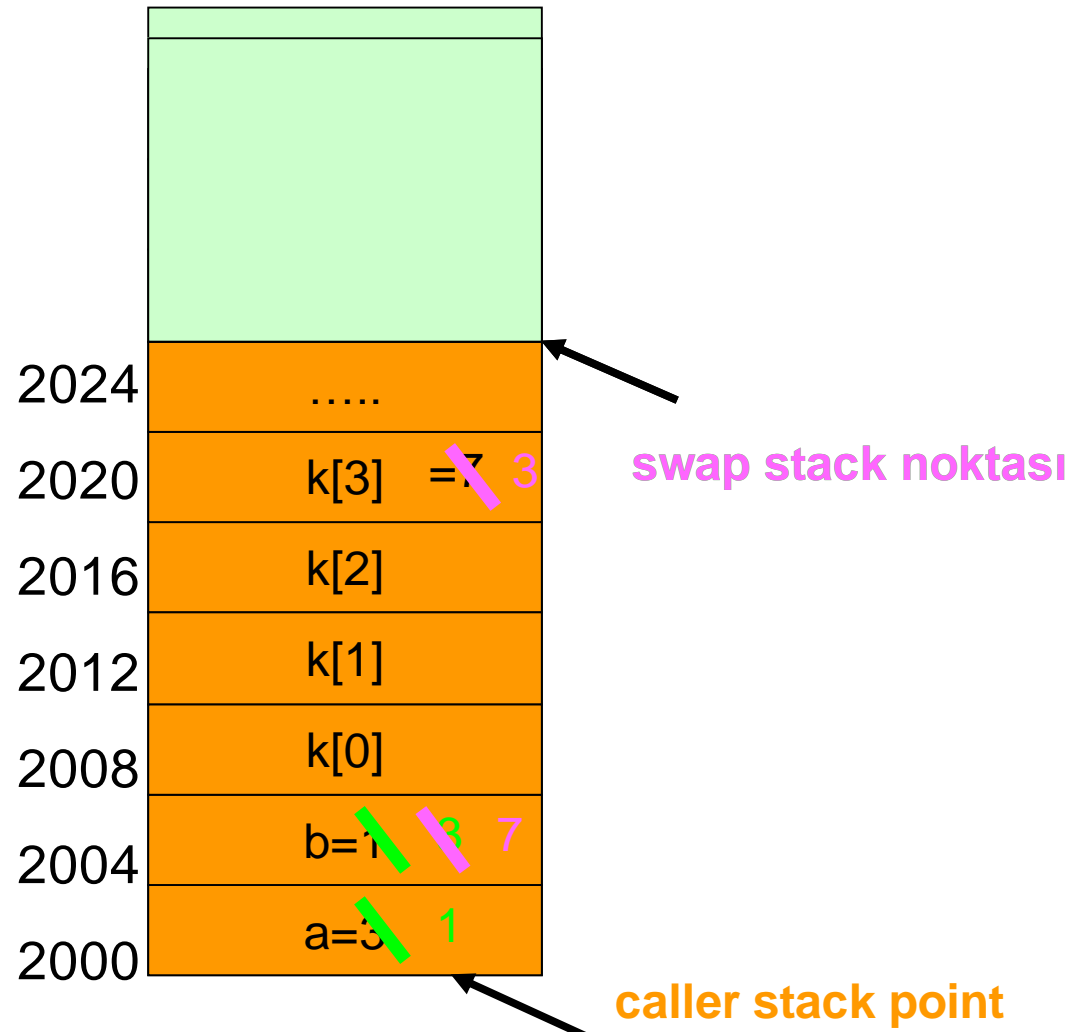
- Parametre olarak bellek adresi geçirilir.
- $c^*$ , a ile gösterilen bellek belgesini gösteren bir işaretçidir.

# Başvuru ile Çağırma (Örnek)

53

```
caller( ) {  
    int a = 3 ;  
    int b = 1 ;  
    int k[10] ;  
    k[3] = 7 ;  
    swap(&a, &b) ;  
    swap(&b, &k[b]) ;  
}
```

```
swap(int *c, int *d) {  
    temp = *c ;  
    *c = *d ;  
    *d = temp ;  
}
```



## □ (C++)

```

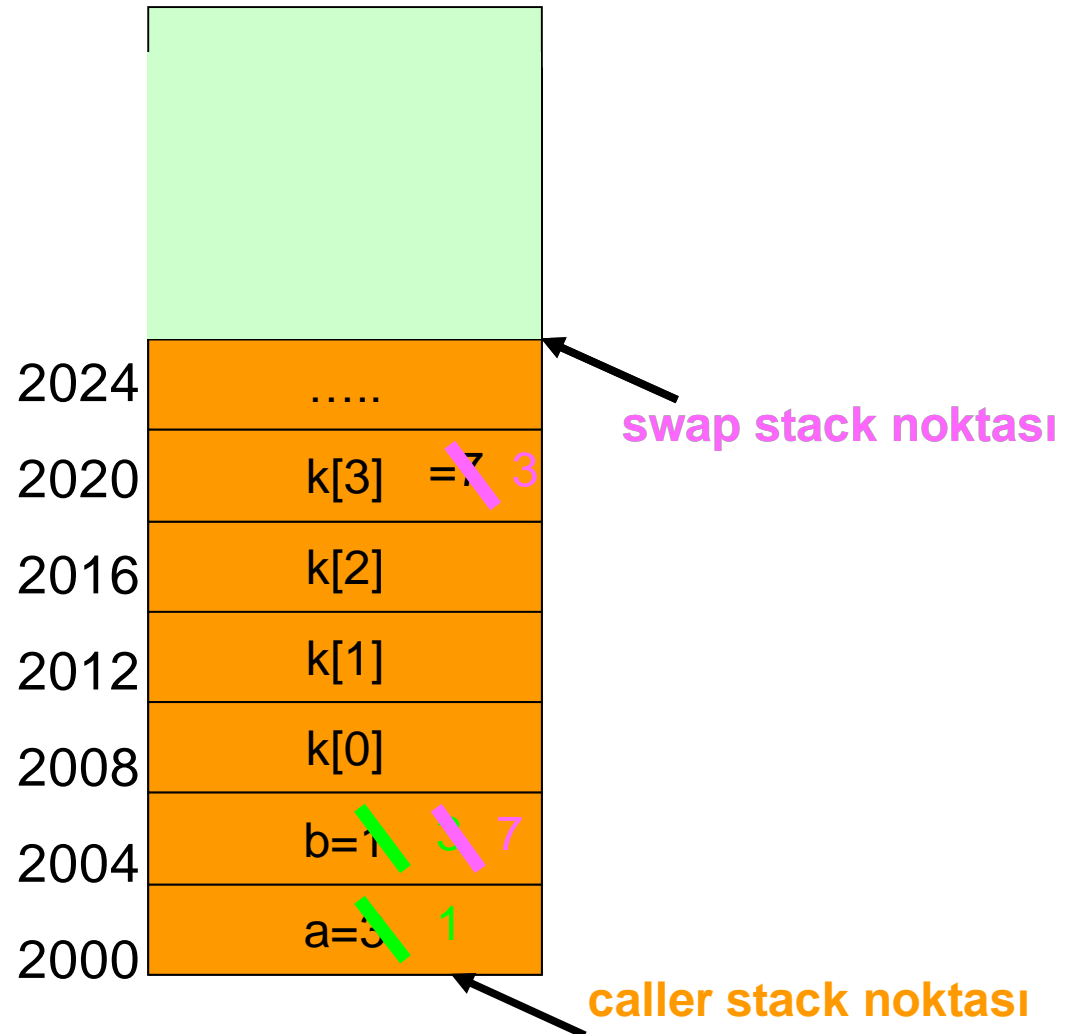
caller( ) {
    int a = 3; int b = 1;
    int k[10];
    k[3] = 7;
    swap(a, b);
    swap(b, k[b]);
}

```

```

swap(int &c, int &d) {
    temp = c;
    c = d;
    d = temp;
}

```



# Değer modeli – başvuru modeli

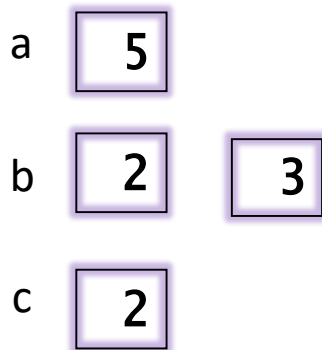
55

`b = 2;`

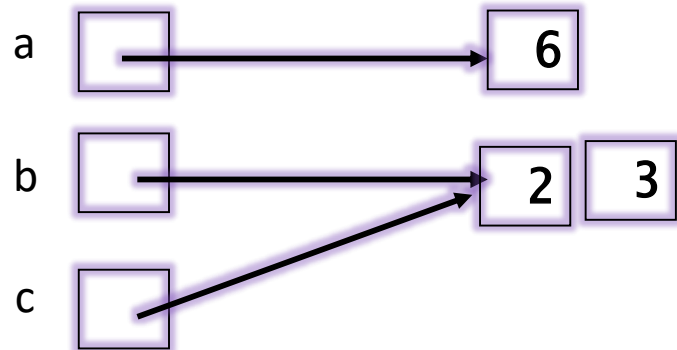
`c = b; b = b + 1;`

`a = b + c;`

Değer modeli

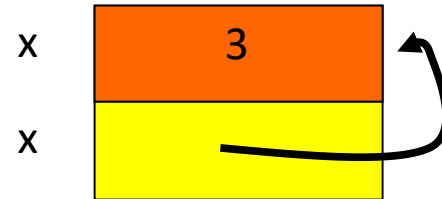


Başvuru modeli



- Gerçek parametreye kapalı olarak gösterge ya da referans geçir
  - Eğer fonksiyon ona yazarsa gerçek parametre değişir

```
void f(int x) {  
    x = 3;  
}  
  
int main() {  
    int x = 0;  
    f(x);  
    printf("%d\n", x);  
}
```





```
x : integer;  
  procedure foo(y : out integer)  
    y := 3;  
    print x;  
    . . .  
x := 2;  
foo(x);  
print x;
```

Eğer y başvuru ile geçirilmişse program iki kere 3 yazar.  
Eğer y değer/sonuç ile geçirilmişse önce 2 sonra 3 yazar.

# Başvuru ile çağırma-Değer sonuç ile çağırma

58

```
{ y: integer;
  procedure p(x: integer)
  { x := x + 1;
    x := x + y;
  }
```

...

```
y := 2;
p(y);
write y;
}
```

Sonuç (başvuru ile çağırma): 6

```
{ y: integer;
  procedure p(x: integer)
  { x := x + 1;
    x := x + y;
  }
```

...

```
y := 2;
p(y);
write y;
}
```

Sonuç (değer-sonuç ile çağırma): 5

## 8.2.4.4. Başvuru ile Çağırma (*Call by Reference*)

59

### □ **Yöntemin Avantajı:**

- Başvuru ile çağırma yönteminin en önemli üstünlüğü, aktarımın hem yer hem de zaman açısından etkin olmasıdır.
- Değer ve sonuç ile çağırma yönteminde olduğu gibi bellek yeri veya kopyalama zamanı gerekli değildir.

### □ **Yöntemin Dezavantajı:**

- Bu yöntemin dezavantajı, resmi parametrelere erişim için verilerin aktarıldığı yöntemlere göre fazladan bir dolaylı erişim gerektirmesidir.
- Ayrıca eğer sadece çağıran programdan altprograma değer aktarılması isteniyorsa, bu yöntemde gerçek parametrenin bellekteki yerine altprogram tarafından ulaşılabilirdiği için, değerinde istenmeyen değişiklikler yapılabilir.

## 8.2.4.5. İsim ile Çağırma (*Call by Name*)

60

- İsim ile çağırma yöntemi de bir içeri-dışarı modeli için gerçekleştirimdir.
- Bir gerçek parametre isim ile çağırma yöntemi ile aktarıldığında, altprogramda gerçek parametreye karşı gelen resmi parametrenin bulunduğu her yere metinsel olarak gerçek parametre yerleştirilir.
- Eğer gerçek parametre bir sabit değerse, isim ile çağırma yöntemi, değer ile çağırma yöntemi ile aynı şekilde gerçekleşir.
- Eğer gerçek parametre bir değişkense, isim ile çağırma yöntemi başvuru ile çağırma yöntemi ile aynı şekilde gerçekleşir.

## 8.2.4.5. İsim ile Çağırma (*Call by Name*)

61

- İsim ile çağırma yönteminin gerçekleştirilmesi güçtür ve kullanıldığı programların hem yazılmasını hem de okunmasını karmaşıktırabilir.
- Bu nedenle ALGOL 60 ile tanıtılan isim ile çağırma yöntemi, günümüzde popüler olan programlama dillerinde uygulanmamaktadır.

```
let add x y = x + y  
let z = add (add 3 1) (add 4 1)
```

OCaml; değer ile çağırma; parametreler burada değerlendirilir

Haskell; isim ile çağırma; parametreler burada değerlendirilir

```
add x y = x + y  
z = add (add 3 1) (add 4 1)
```

```
void f(by-name int a, by-name int b)
{
    b=5;
    b=a;
}

int g()
{
    int i = 3;
    f(i+1,i);
    write i;
}
```

Çıktı: 6

- Değerlendir

$$y = \sum_{1 \leq x \leq 10} 3x^2 - 5x + 2$$

- İsim ile çağırma:

$$y := \text{sum}(3 \cdot x \cdot x - 5 \cdot x + 2, x, 1, 10)$$

```
real proc sum(expr, i, low, high);  
  value low, high;  
  real expr;  
  integer i, low, high;  
begin  
  real rtn;  
  rtn := 0;  
  for i := low step 1 until high do  
    rtn := rtn + expr;  
  sum:=rtn  
end sum;
```

```

{ c: array [1..10] of
  integer;
  m, n : integer;
  procedure r (k, j :
    integer)
  begin
    k := k+1;
    j := j+2;
  end r;
  /* set c[n] to n */
  m := 2;
  r(m, c[m]);
  write m, c[m];
}

```

m:= m+1

c[m] := c[m] + 2

m	c[ ]
2	1 2 3 4 5 6 7 8 9 10
3	1 2 5 4 5 6 7 8 9 10



## 8.2.4.6. Çağırma Yöntemlerinin Karşılaştırılması

65

- Değer ile çağırma yöntemi güvenli ama çoğu durumda yetersiz bir parametre aktarım yöntemi olmakta, değer ve sonuç ile çağırma ve başvuru ile çağırma yöntemleri ise, yeterince güvenli olmayan ama gerek duyulan aktarım yöntemleri olmaktadır.

Programlama Dili	Değer	Sonuç	Değer-Sonuç	Başvuru (Referans)	İsim
<b>FORTRAN IV</b>				X	
<b>Fortran 77</b>			X		
<b>ALGOL 60</b>	seçimlik				X
<b>ALGOL W</b>			X		
<b>C++</b>	X			X	
<b>PASCAL, Modula2</b>	X			seçimlik	
<b>ADA</b>	X	X	X	X	
<b>Java</b>	X			X	

# Parametre Geçirme Metotları (1)

67

Çağırın program

```
...  
int result = myFunc (10, "val");  
...
```

Altprogram

```
int myFunc (int a, char *s) {  
...  
}
```

Değer

Ulaşım Yolu

**Call-by-Value**

Duplicated uzaya ihtiyaç duyar

**Pass-by-Value (In Mode)**

**Pass-by-Reference  
(Call-by-Reference)**

Dolaylı adreslemeye ihtiyaç duyar

# Parametre Geirme Metotları (2)

68

ağırان program

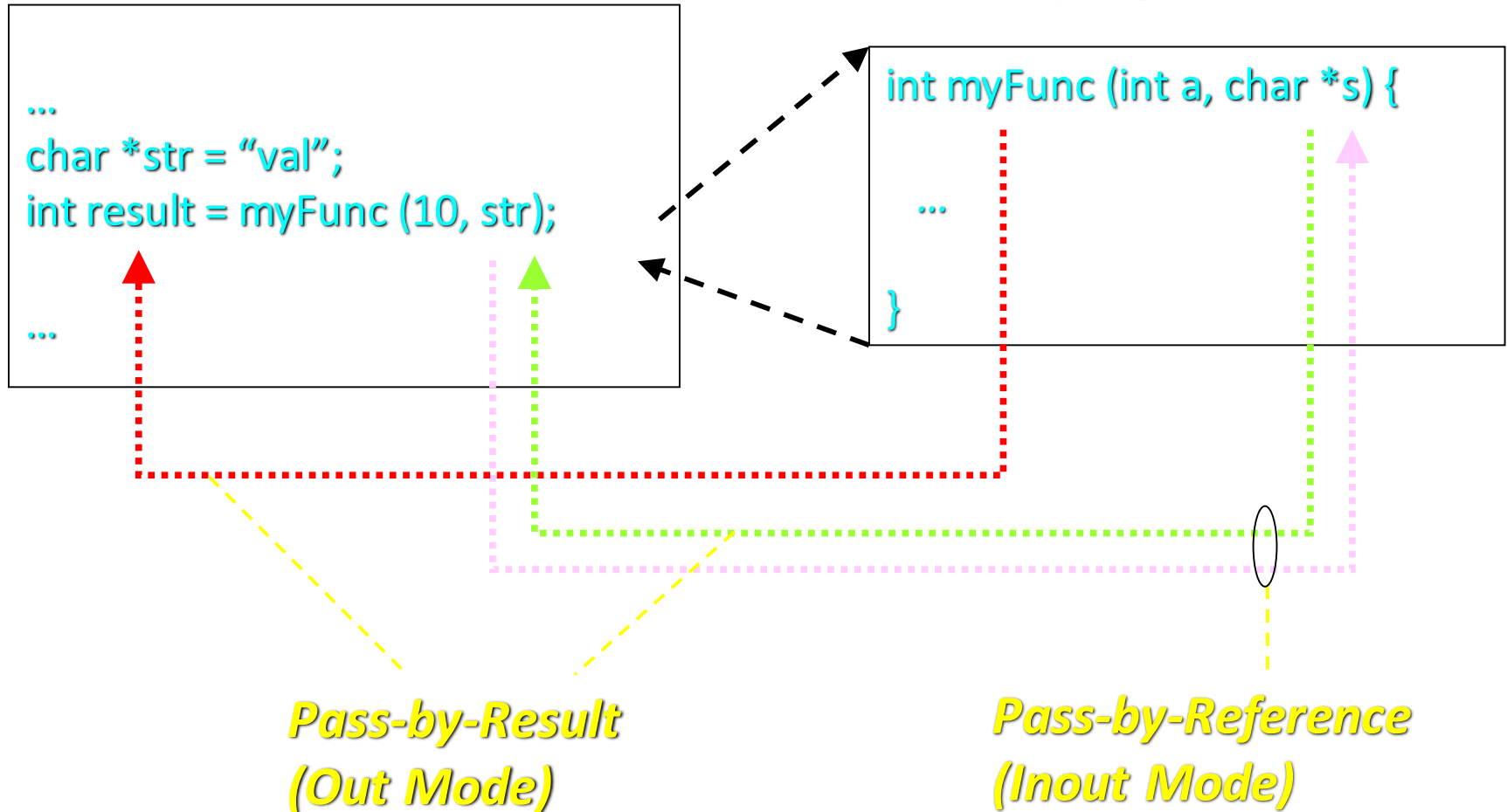
```
...  
char *str = "val";  
int result = myFunc (10, str);  
...
```

Altprogram

```
int myFunc (int a, char *s) {  
    ...  
}
```

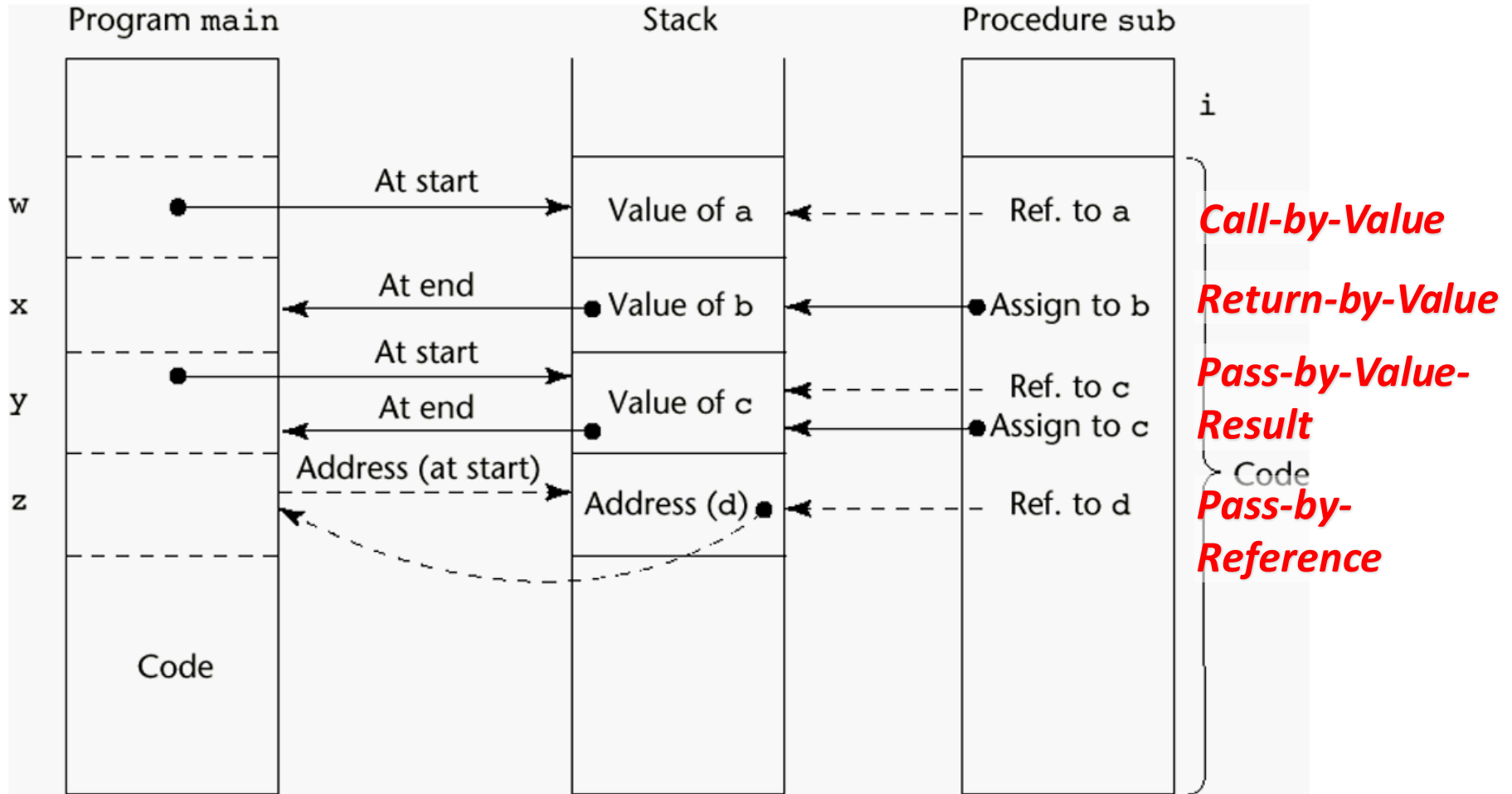
*Pass-by-Result  
(Out Mode)*

*Pass-by-Reference  
(Inout Mode)*



# Parametre geçirmenin stack gerçekeştirmesi

69



# ÖRNEKLER

70

- Üç çağırma yöntemini altında aşağıdaki programı göz önüne alalım
  - Her biri için, *i*'nin değerini ve hangi *a[i]* (eğer varsa) değiştirildiğini bulalım

```
int i = 1;

void p(int f, int g) {
    g++;
    f = 5 * i;
}

int main() {
    int a[] = {0, 1, 2};
    p(a[i], i);
    printf("%d %d %d %d\n",
        i, a[0], a[1], a[2]);
}
```

# Örnek: Değer ile çağırma

71

```
int i = 1;

void p(int f, int g) {
    g++;
    f = 5 * i;
}

int main() {
    int a[] = {0, 1, 2};
    p(a[i], i);
    printf("%d %d %d %d\n",
        i, a[0], a[1], a[2]);
}
```

i	a[0]	a[1]	a[2]	f	g
1	0	1	2		
				1	1
				5	2

# Örnek: Başvuru ile çağırma

72

```
int i = 1;

void p(int f, int g) {
    g++;
    f = 5 * i;
}

int main() {
    int a[] = {0, 1, 2};
    p(a[i], i);
    printf("%d %d %d %d\n",
        i, a[0], a[1], a[2]);
}
```

i/g	a[0]	a[1]/f	a[2]
1	0	1	2
2		10	
2		10	



# Örnek: isim ile çağırma

73

```
int i = 1;

void p(int f, int g) {
    g++;
    f = 5 * i;
}

int main() {
    int a[] = {0, 1, 2};
    p(a[i], i);
    printf("%d %d %d %d\n",
        i, a[0], a[1], a[2]);
}
```

i	a[0]	a[1]	a[2]
1	0	1	2
2			10
2			10

`a[i]` ifadesi ihtiyaç duyulmayıncaya kadar değerlendirilmez, bu durumda `i` değiştirildikten sonraya kadar

# ÖRNEKLER: Çağırma Yöntemleri

74

## Değer ile çağırma

Çıktı:

(5,6,5)

```
var y:integer;
procedure
  A(x:integer);
begin
  write(x);
  x := 1;
  write(y+x);
end;
begin
  y := 5;
  A(y);
  write(y);
end;
```

# ÖRNEKLER: Çağırma Yöntemleri

75

## Başvuru ile çağırma

Çıktı:

(5,2,1)

```
var y:integer;
procedure
  A(x:integer);
begin
  write(x);
  x := 1;
  write(y+x);
end;
begin
  y := 5;
  A(y);
  write(y);
end;
```

# ÖRNEKLER: Çağırma Yöntemleri

76

## Sonuç ile çağırma

Çıktı:

(??,6,1)

```
var y:integer;
procedure
  A(x:integer);
begin
  write(x);
  x := 1;
  write(y+x);
end;
begin
  y := 5;
  A(y);
  write(y);
end;
```

# ÖRNEKLER: Çağırma Yöntemleri

77

**Değer/Sonuç ile çağırma**

Çıktı:

(5,6,1)

```
var y:integer;
procedure
  A(x:integer);
begin
  write(x);
  x := 1;
  write(y+x);
end;
begin
  y := 5;
  A(y);
  write(y);
end;
```

# Parametre olarak çok boyutlu diziler

78

- Eğer çok boyutlu bir dizilim bir altprograma parametre olarak geçirilirse, derleyici doğru adresleme yapabilmek için dizilimin tanımlama boyutlarını bilmek ister.

# Parametre olarak çok boyutlu diziler

79

## ■ C ve C++

- Programcı birinci boyu hariç diğer boyutların boylarını resmi parametrede vermek zorundadır.

□  $\text{adres}(\text{mat}[i,j]) = \text{adres}(\text{mat}[0,0]) + i * \text{sutun\_sayisi} + j$

```
void fun(int matrix[][10]) { . . . }
```

```
void main() {  
    int mat[5][10];  
    . . .  
    fun(mat);  
    . . .  
}
```

- Bu durum esnek program yazmayı engeller. Her boyutta matrisle çalışabilecek bir altprogram yazılamaz.
- Çözüm: matrisin işaretçisini parametre olarak geç ve bilgi olarak da boyutları ayrı parametreler olarak geç. Programcı adres hesaplayan fonksiyonu da sağlamalıdır.

```
void fun(float *mat_ptr, int num_rows, int num_cols);  
#define mat_ptr(i,j) = (*(mat_ptr + ((i) * num_cols) + (j)))
```

- Pascal
  - ▣ Problem değil, dizilimin boyu dizilimin parçası.
- Ada
  - ▣ Kısıtlanmış dizilimler – Pascal gibi.
  - ▣ Kısıtlanmamış dizilimler – boyutlar nesnenin bir parçası (Java da benzer).
- Java, C#
  - ▣ Ada gibi
- Pre-90 FORTRAN
  - ▣ Resmi parametrelerde dizilimle birlikte boyutları da geçilebilir.

```
SUBPROGRAM SUB (MATRIX, ROWS, COLS, RESULT)
INTEGER ROWS, COLS
REAL MATRIX (ROWS, COLS), RESULT
...
END.
```



## 8.2.5. C'de Parametre Aktarımı

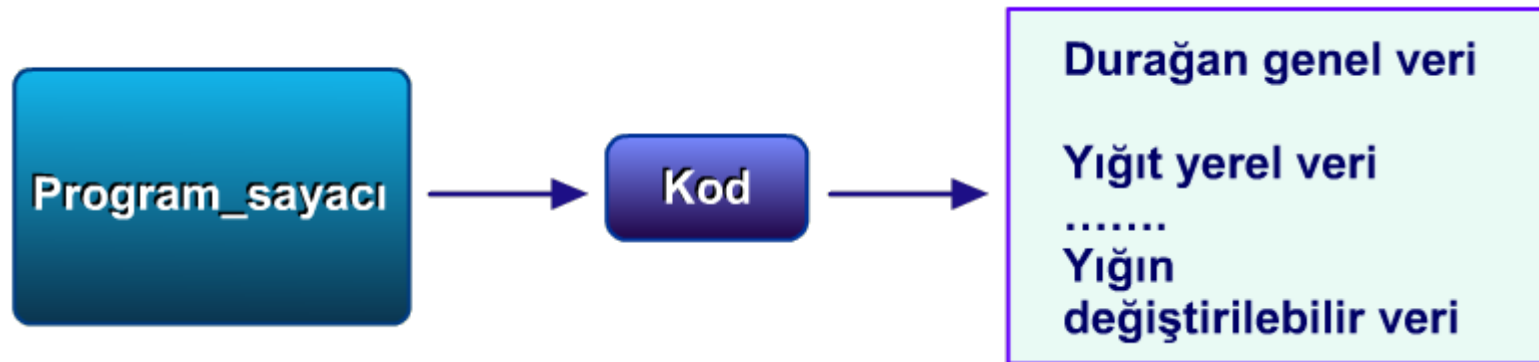
81

- ❑ **C Dili'nin Özellikleri:**
- ❑ C'de parametre aktarımını incelemeden önce, C dilinin özelliklerini hatırlamakta yarar vardır.
- ❑ Bir C programı, bir dizi fonksiyon, tip ve değişken tanımından oluşur. Tipler ve değişkenler, bir fonksiyona yerel olarak tanımlanabilir ancak bir fonksiyon, bir diğerine yerel tanımlanamaz. C'de bir tanımlamanın kapsamı ya bir fonksiyon içinde, ya da birlikte derlenen tüm fonksiyonlara genel özelliktedir.

## 8.2.5. C'de Parametre Aktarımı

82

- Çalışan bir C programının elemanları aşağıdaki şekilde bellek düzeninde görülmektedir.



- C'de değişken tanımlamaları,  $\{..\}$  içinde tanımlanan bloklar içinde bulunabilir.
- C'de bir blok, bir dizi tanımlama ve deyimden oluşur, ancak bir blok bir fonksiyon tanımını içermez.

## 8.2.5. C'de Parametre Aktarımı

83

- ❑ **C'de Değer ile Çağırma Yöntemi:**
- ❑ C'de fonksiyonlara değer geçirmek için sadece değer ile çağırma yöntemi uygulanmaktadır.
- ❑ Örneğin, aşağıdaki animasyonda görülen örnekte *altpro1* isimli fonksiyonda değeri değiştirilen "*a*" değişkeni genel (*global*) "*a*" değişkeni değil, yerel değişken "*a*" olmaktadır.

## 8.2.5. C'de Parametre Aktarımı

84

### C'de Değer ile Çağırma

Aşağıdaki örnekte *altpro1* fonksiyonu, global değişken "a" değişkeninin değerini değiştirmektedir.

```
int    a = 5;
```

```
void altpro1(int a, float b)
```

```
{a = a + int(b);}
```

```
altpro1(a, 3.0);
```

global değişken "a"

yerel değişken olarak "a"

fonksiyon çağırımı

## 8.2.5. C'de Parametre Aktarımı

85

- **C'de Başvuru ile Çağırma Yöntemi:**
- Eğer global değişken "a" nın değeri değiştirilmek isteniyorsa, yani başvuru ile çağırma yönteminin uygulanması isteniyorsa, göstergeler kullanılarak aktarım yapılmalıdır.
- Aşağıdaki animasyonda görülen örnekte *altpro1* fonksiyonu, global değişken "a" değişkeninin değerini değiştirmektedir. Burada görüldüğü gibi, C'de başvuru ile çağırma yöntemini sağlamak için göstergeler kullanılmalıdır.

## 8.2.5. C'de Parametre Aktarımı

86

### C'de Başvuru ile Çağırma

Aşağıdaki örnekte *altpro1* fonksiyonu, global değişken "a" değişkeninin değerini değiştirmektedir.

```
int    a = 5;  
void altpro1(int * a, float b)  
{*a =*a + int(b);}  
altpro1(&a, 3.0);
```

genel değişken "a"

dolaylı erişim

adres operatörü & kullanılmalıdır.

# Parametre aktarım örnekleri

## C: değer ile çağırma

```
void swap1(int a, int b){
    int temp = a;
    a = b;
    b = temp;
}

swap1(c, d); // c ve d'nin içerikleri yer
değiştirmez.
```

## C: referans ile çağırma

```
void swap2(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

swap2(&c, &d); // c ve d'nin içerikleri yer
değiştirir.
```

## C++: referans ile çağırma

```
void swap3(int &a, int &b){
    int temp = a;
    a = b;
    b = temp;
}

swap3(c, d); // c ve d'nin içerikleri
yer değiştirir.
```

## Ada: değer ile çağırma

```
procedure swap4(a : in out Integer,
                b: in out Integer)is
    temp : Integer;
    temp := a;
    a := b;
    b := temp;
end swap4;

swap4(c, d); // c ve d'nin içerikleri yer değiştirir.
```

# Parametre aktarım örnekleri

**C gibi** sözdizim: referans ile veya değer sonuç ile çağırma karşılaştırması

```
int i = 3; /* global değişken */
void fun(int a, int b){
    i = b;
}
void main() {
    int list[10];
    list[i] = 5;
    fun(i, list[i]);
}
```

## değer-sonuç ile çağırma

```
addr_i = &i           //adresleri al
addr_listi = &list[i]
a = *addr_i           // a = 3
b = *addr_listi       // b = 5
i = b                 // i = 5
*addr_i = a           // i = 3
*addr_listi = b       // list[3]=5
```

## referans ile çağırma

```
a = &i
b = &list[i]
i = *b    // i = 5
```



# Çok boyutlu dizi parametresi

89

```
fun(int a[ ][ ][ ]) {  
    a[3][4][5] = 4321;  
}  
main( ) {  
    int a[10][20][25];  
    a[3][4][5] = 1234;  
    fun(a);  
}  
// derleyici hatası
```

```
fun(int a[ ][ ][25]) {  
    a[3][4][5] = 4321;  
}  
main( ) {  
    int a[10][20][25];  
    a[3][4][5] = 1234;  
    fun(a);  
}  
// derleyici hatası
```

```
fun(int a[ ][20][25]) {  
    a[3][4][5] = 4321;  
}  
main( ) {  
    int a[10][20][25];  
    a[3][4][5] = 1234;  
    fun(a);  
    // a[3][4][5] = 4321  
}
```

1. Dizi için heap-dynamic değişkenler kullan

- *int \*a; a = (int \*) malloc(sizeof(int) \* m \* n);*

2. Dizi değişkenini m ve n ile yalnız başına geçir

- *fun(int \*a, int num\_rows, int num\_cols)*
- *a[i, j] == \*(a + i \* num\_cols + j)*

## 8.2.6. Pascal'da Parametre Aktarımı

90

```
Program Bir;  
  Procedure Iki;  
    Procedure Uc;  
      Procedure Dort;  
        Procedure Bes;
```

- Altprogramlar, Pascal'da, C'dekinden farklı olarak içiçe yuvalanabilir.
- Yandaki şekilde verilen yapıdaki bir Pascal programında *Program Bir*'de tanımlı bir değişken, hem *Bir*, hem *İki*, hem *Uc*, hem *Dort*, hem de *Bes* yordamlarında görünür.
- İki yordamında tanımlı değişken, *İki* ve *Uc* de görünür. *Uc* yordamında tanımlanan değişken ise sadece *Uc* de görülür.

## 8.2.6. Pascal'da Parametre Aktarımı

91

- Başvuru ile çağırma gerçekleştirmek için **var** anahtar kelimesi kullanılır.
- **var** ile, çıktı: (5,2,1).
- **var**, olmaksızın çıktı: (5,6,5)

```
var y:integer;  
procedure A(var x:integer) ;  
begin  
    write(x) ;  
    x := 1;  
    write(y+x) ;  
end;  
begin  
    y := 5;  
    A(y) ;  
    write(y) ;  
end;
```

# Parametre olan altprogram isimleri

92

Etmenler:

- 1. Altprogramın parametre tipleri kontrol edilecek mi?
  - ▣ Erken Pascal ve FORTRAN 77 kontrol etmez.
  - ▣ Sonraki Pascal versiyonları ve FORTRAN 90 kontrol eder.
  - ▣ Ada altprogramların parametre olarak geçilmesine izin vermez.
  - ▣ Java metod isimlerinin parametre olarak geçilmesine izin vermez.
  - ▣ C ve C++ parametre olarak fonksiyon geçer (fonksiyon göstericileri ile) ve parametrelerinin tipleri kontrol edilebilir.

# Parametre olan altprogram isimleri

93

- 2. Parametre olarak geçilen bir altprogramın doğru referans çevresi
  - (referencing environment)?
  - Olasılıklar:
    - ▣ a. Çağırان altprogramın referans çevresi:
      - yüzeysel bağlama (shallow binding)
    - ▣ b. Parametre olarak çağrılan altprogramın referans çevresi:
      - derin bağlama (Deep binding)
    - ▣ c. Çağırان altprogramın, çağrılan altprograma parametre olarak geçtiği referans çevresi
      - Özel bağlama (Ad hoc binding) (Hiç kullanılmamış).
  - Statik kapsamlı (static-scoped) diller için derin bağlama en doğalıdır.
  - Dinamik kapsamlı (dynamic-scoped) diller için yüzeysel bağlama en doğalıdır.

# Parametre olan altprogram isimleri

94

Örnek:

```
sub1
  sub2
    sub3
      call sub4(sub2)
    sub4(subx)
      call subx
  call sub3
```

- sub2 sub4'ün parametresi olarak çağırıldığında referans çevresi nedir?
  - ▣ Yüzeysel bağlama (Shallow binding) => sub2, sub4, sub3, sub1
  - ▣ Derin bağlama (Deep binding) => sub2, sub1

# Parametre olarak fonksiyonlar

95

```
int Plus (int num) {  
    return num + num ;  
}
```

```
int Square (int num) {  
    return num * num;  
}
```

```
void Execute(int seed,  
              int (*pF)(int)) {  
    return pF(seed) ;  
}
```

```
int main( ) {  
    int Result ;  
    int (*pF)(int) ;
```

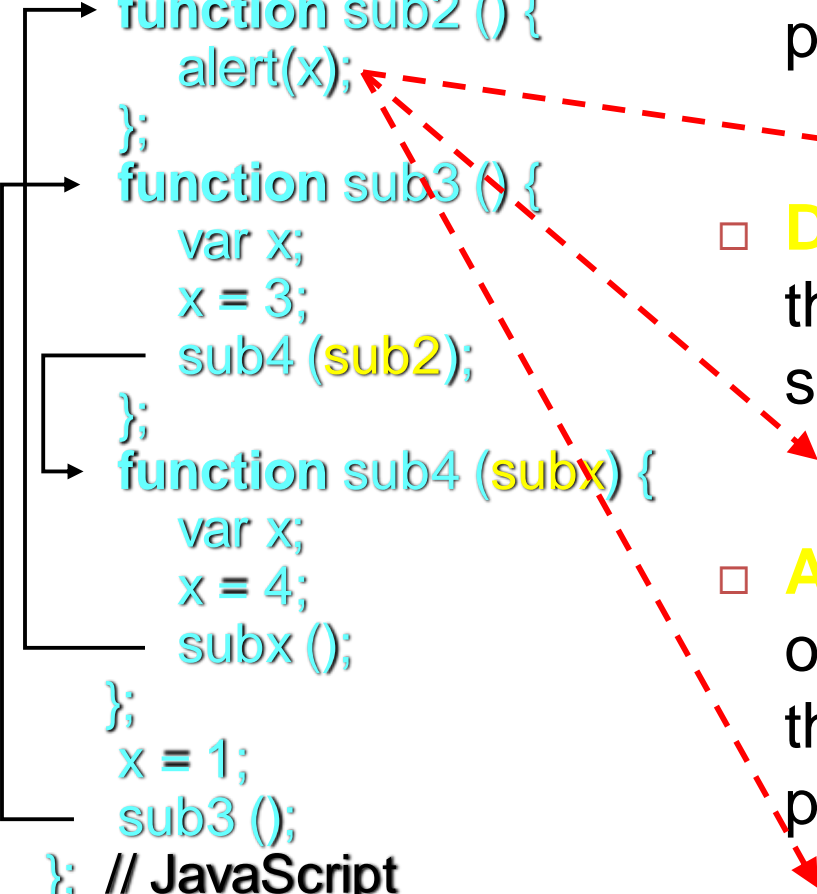
```
    pF = Plus;  
    Result = Execute(3, pF);  
    // Sonuç = 6
```

```
    pF = Square;  
    Result = Execute(3, pF);  
    // Sonuç = 9  
}
```

# Parametre olarak fonksiyonlar

96

```
function sub1 () {  
  var x;  
  function sub2 () {  
    alert(x);  
  };  
  function sub3 () {  
    var x;  
    x = 3;  
    sub4 (sub2);  
  };  
  function sub4 (subx) {  
    var x;  
    x = 4;  
    subx ();  
  };  
  x = 1;  
  sub3 ();  
}; // JavaScript
```



□ **Shallow Binding** – the *reference environment* of the call statement is passed to the subprogram

**X = 4**

□ **Deep Binding** – the environment of the definition of the passed subprogram

**X = 1**

□ **Ad Hoc Binding** – the environment of the call statement that passes the subprogram as an actual parameter

**X = 3**



## 8.3. Fazla yüklü altprogramlar

# Overloaded Subprograms

97

- Aynı referans çevresinde başka bir altprogramla aynı isimde olan altprograma "Fazla yüklü altprogram" (overloaded subprogram) denir.
- C++, Java, C# ve Ada yerleşik fazla yüklü altprogramlara sahiptirler ve yazılımcılar isterlerse kendi fazla yüklü altprogramlarını yazabilirler.
- C++, Java ve C#'da parametrelerinin farklılığından hangi altprogramın kullanılacağına karar verilir.
- Ada'da altprogramın döneceği değer de seçimde rol oynayabilir. Ada karışık modlu ifadelere izin vermediğinden bu özellik faydalıdır.
- Örneğin iki parametreleri aynı ama "integer" ve "float" dönen iki fun fonksiyonu olsun.

```
A, B : Integer;  
.  
.  
.  
.  
A := B + Fun(7);
```

Bu durumda derleyici tam sayı dönen fonksiyonu kullanır.

## 8.4. Cinsine özgü (jenerik) altprogramlar (Generic Subprograms)

98

- Farklı etkinleştirmelerde farklı tiplerde parametre alabilen altprogramlara cinsine özgü (generic) veya çok biçimli (polymorphic) altprogram denir. Fazla yüklü altprogramlar özel çok biçimlilik oluştururlar.
- Hazırlamış bir altprogramın değişik veri yapıları için kullanılması, farklı veri yapıları için farklı altprogramlar hazırlanmaması ana motivasyon kaynağıdır.
- Olası parametre tipleri bir tip ifadesinde tanımlanmış, cinsine özgü (generic parameter) parametre alan altprogramlara parametrik çok biçimliliği olan altprogram denir.

# Parametrik çok biçimliliğe (parametric polymorphism) örnekler

99

## 1. Ada

- Ada altprogramları ve paketlerinde tipler (Types), altsimge kapsamı (subscript range), sabit değerler gibi özellikleri cinsine özgü olabilir.

```
generic
```

```
    type element is private;
```

```
    type vector is array (INTEGER range <>) of element;
```

```
    procedure Generic_Sort (list: in out vector);
```

# Cinsine özgü altprogramlar (örnekler)

100

```
procedure Generic_Sort (list : in out vector)
    is temp: element;
begin
    for i in list'FIRST.. i'PRED(list'LAST) loop
        for j in i'SUCC(i)..list'LAST loop
            if list(i) > list(j) then
                temp := list(i);
                list(i) := list(j);
                list(j) := temp;
            end if;
        end loop; -- for j
    end loop; --for i
End Generic_Sort

procedure Integer_Sort is new Generic_Sort
    (element => INTEGER; vector => INTEGER_ARRAY);
```

# Cinsine özgü altprogramlar (örnekler)

101

- Yukarıdaki somutlaştırmaya kadar jenerik tanım (bir önceki sayfa) için bir kod üretilmez. Ne zaman ki yukarıdaki somutlaştırma gerçekleştirilir, derleyici `Generic_Sort`'un tamsayıları sıralayan versiyonu olan `Integer_Sort` alt fonksiyonunu üretir.
- Ada'da fonksiyon isimlerinin parametre olamadığı düşünülürse, bu yöntem çok amaçlı fonksiyon hazırlamak açısından da faydalıdır. Bir sonraki örneğe bakalım.

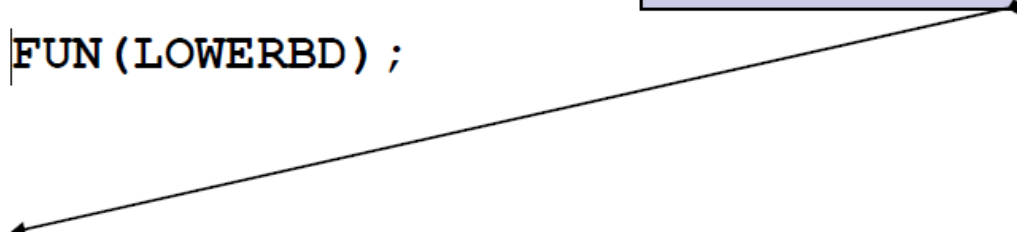
# Cinsine özgü altprogramlar (örnekler)

102

```
generic
with function FUN(X : FLOAT) return FLOAT;
procedure INTEGRATE(LOWERBD : in FLOAT;
                    UPPERBD : in FLOAT;
                    RESULT : out FLOAT) ;
procedure INTEGRATE(LOWERBD : in FLOAT;
                    UPPERBD : in FLOAT;
                    RESULT : out FLOAT) is

    FUNVAL : FLOAT;
begin
    ...
    FUNVAL := FUN(LOWERBD) ;
    ...
end;
INTEGRATE_FUN1 is new INTEGRATE(FUN => FUN1) ;
```

FUN1 fonksiyonunun entegralini alan özel program.

A diagram consisting of a long arrow pointing from the text 'FUN1' in the line 'FUNVAL := FUN(LOWERBD) ;' to a box containing the text 'FUN1 fonksiyonunun entegralini alan özel program.'

# Cinsine özgü altprogramlar (örnekler)

103

## 2. C++

□ Şablon fonksiyonlar (Template functions)

□ **template** <şablon parametreleri>

şablon parametreleri:

**class** tanımlayıcı (identifier) – tip isimleri için kullanılır.

tip\_adı tanımlayıcı – şablon fonksiyona değer geçmek için kullanılır.

□ Örneğin

```
template <class Type>
```

```
    Type max(Type first, Type second) {  
        return first > second ? first : second;  
    }
```

□ "Type" fonksiyonun işleyeceği veri tipini gösterir. Bu fonksiyon ">" işlecinin işleyebileceği bütün veri tipleri ile çalışır.

# Cinsine özgü altprogramlar (örnekler)

104

## □ Örnek:

```
template <class Type>
    Type max(Type first, Type second) {
        return first > second ? first : second;
    }
```

## □ Yukarıdaki işlem makrolar ile de yapılabilirse de, yan etkilerine dikkat etmek gerekir:

- **#define max(a,b) ((a) > (b)) ? (a) : (b)**
- ancak bu makro **max(x++,y)** şeklinde kullanılırsa:
- **((x++) > (y)) ? (x++) : (y)** şeklinde hatalı kod üretir. (**x** iki defa büyütülüyor!)
- Aynı sorun şablon alt programda yok.
- C++ şablon programları örtülü olarak, ismi ile çağırıldığında veya & işleci ile adresi alındığında somutlaştırılır (instantiated).

□

```
int a,b,c;
float d,e,f;
...
c = max(a,b);
f = max(d,e);
```



# Cinsine özgü altprogramlar (örnekler)

105

```
template <class Type>
void generic_sort(Type list[], int len) {
    int top, bottom;
    Type temp;

    for (top = 0; top < len - 2; top++)
        for (bottom = top + 1; bottom < len - 1;
            bottom++) {
            if (list[top] > list[bottom]) {
                temp = list [top];
                list[top] = list[bottom];
                list[bottom] = temp;
            } /** for (bottom = ... sonu
        } /** generic_sort sonu

/** şablon fonksiyonun somutlaştırılması
float flt_list[100];
. . .
generic_sort(flt_list, 100);
```

# Cinsine özgü altprogramlar (örnekler)

106

## □ Java 5.0

- Java 5.0 ve C++'arasındaki soysal farklar:

1. Java 5.0'de soysal parametreler sınıflandırılmış olmalıdır
2. Java 5.0 'de soysal yöntemler sadece bir kere doğru olarak soysal yöntemler olarak örneklendirilmelidir
3. Sınırlamalar sınıf aralıklarına göre belirtilmelidir ki soysal yöntemler soysal parametreler olarak geçebilsin
4. Soysal parametrelerin joker türleri

# Cinsine özgü altprogramlar (örnekler)

107

## □ Java 5.0 (devam)

```
public static <T> T doIt(T[] list) { ... }
```

- parametreler sosyal elemanların dizileridir (T türün ismidir)

- Bir çağırım:

```
doIt<String>(myList);
```

Sosyal parametrelerin sınırları olabiliir:

```
public static <T extends Comparable> T  
doIt(T[] list) { ... }
```

Sosyal tür `Comparable` arabirimini uygulayan bir sınıf olmalıdır

# Cinsine özgü altprogramlar (örnekler)

108

## □ Java 5.0 (devam)

### ▣ Joker türleri

`Collection<?>` Derleme sınıfları için bir joker türüdür

```
void printCollection(Collection<?> c) {  
    for (Object e: c) {  
        System.out.println(e);  
    }  
}
```

-Herhangi bir derleme sınıfı için çalışabilir

# Cinsine özgü altprogramlar (örnekler)

109

## □ C# 2005

- Java 5.0 ile benzer soysal yöntemleri destekler
- Bir fark: etkin tür parametreler derleyici tanımlanmamış türüne ulaşabilirse, bir çağrı atlanabilir

■ Diğer – C# 2005 jokerleri (wildcard) desteklemez

# Cinsine özgü altprogramlar (örnekler)

110

## □ F#

- Soysal tür dışında bir parametre türü veya bir işlemin dönüş türü saptanamıyorsa-*otomatik genelleştirme*
- Böyle türler bir kesme işareti veya bir har ile belirtilir,örn., 'a
- Fonksiyonlar genel parametreleri tanımlayabilir

```
let printPair (x: 'a) (y: 'a) =  
    printfn "%A %A" x y
```

- %A her tür için format koddur

-Bu parametreler kısıtlı tür değildir

# Cinsine özgü altprogramlar (örnekler)

111

- F# (devam)
  - Eğer parametrelerin fonksiyonları aritmetik operatörler ile kullanılıyorsa hatta parametreler soysal olarak belirtilse bile
  - Tür sonuç çıkarmaları(inferencing)ve tür zorlamalarının(coercions) eksikliği yüzünden, F# soysal fonksiyonları C++, Java 5.0+,ve C# 2005+'dan çok daha az kullanışlıdır

# Cinsine Özgü Altprogramlarda Tasarımla ilgili Etmenler

112

## 1. Yan etkilere izin verilecek mi?

- ▣ a. İki yönlü parametreler (Ada izin vermez)
- ▣ b. Lokal olmayan referans (hepsi izin verir)

Yukarıdakilere izin verildiği zaman yan etkiler (parametrelerin değişme olasılığı) kaçınılmazdır.

## 2. Ne tip dönülen değere izin verilecek?



# Altprogramlarda Tasarımla ilgili etmenler

113

- Olası dönülen tipler için Dil örnekleri:
  1. FORTRAN, Pascal – sadece basit tipler dönerler.
  2. C – fonksiyonlar ve dizilimler hariç her tip (fonksiyonlar ve dizilimler göstericiyle dönülür).
  3. Ada – her tip (altprogramlar tip değildir bu nedenle dönülmez ama göstericisi dönülebilir).
  4. C++ – C gibi, ayrıca kullanıcı tanımlı tipler ve "class" dönülebilir.
  5. Java ve C# – Bu iki dilde fonksiyonlar yoktur ancak onlar gibi işlem gören nesnelerin metotları vardır. Bu dillerde bütün tipler ve sınıflar ("class") dönülebilir. Metotlar tip olmadığından dönülemez.
  6. JavaScript – Bazı dillerde fonksiyonlar veri nesneleri gibidir ve bu nedenle parametrede geçilebileceği gibi değer olarak da dönülebilirler. Birçok fonksiyonel dilde de bu böyledir.

## 8.5. Kullanıcı tanımlı fazla yüklü işleçler (overloaded operators)

114

- Fazla yüklü işleçler bütün programlama dillerinde bulunur.
- Kullanıcılar C++ ve Ada'da bunları daha fazla da yükleyebilirler (C++'ın devamı niteliğindeki Java'ya bu özellik taşınmamıştır).

# Kullanıcı tanımlı fazla yüklü işleçler

115

- Örnek vektör iç çarpımı (vektörün aynı yöndeki elemanlarının çarpımlarının toplamı) (Ada) varsayalım `VECTOR` tipi `INTEGER` tipi elemanları olan bir dizilim (array) olsun:

```
function "*" (A, B : in VECTOR) return INTEGER is
    SUM : INTEGER := 0;
begin
    for INDEX in A'range loop
        SUM := SUM + A(INDEX) * B(INDEX);
    end loop;
    return SUM;
end "*";
```

- Aynı örneğin C++ benzerinin prototipi ise aşağıdaki gibi olurdu:  
`int operator *(const vector A, const vector B, int boy);`
- Kullanıcı tanımlı fazla yüklü işleçler iyi mi yoksa kötü müdür? Kişisel olarak beğeniye kalmış.
- Ancak programın okunabilirliğini azaltır. "a\*b" ifadesinin sayısal bir çarpım mı yoksa vektör çarpımı mı hatırlamak zor olabilir.
- Farklı ekiplerin hazırladığı program parçacıkları işleçlere çelişen görevler yükleyebilirler, ayırt edilmesi zor olur.

## 8.6. Kapatmalar (closure)

116

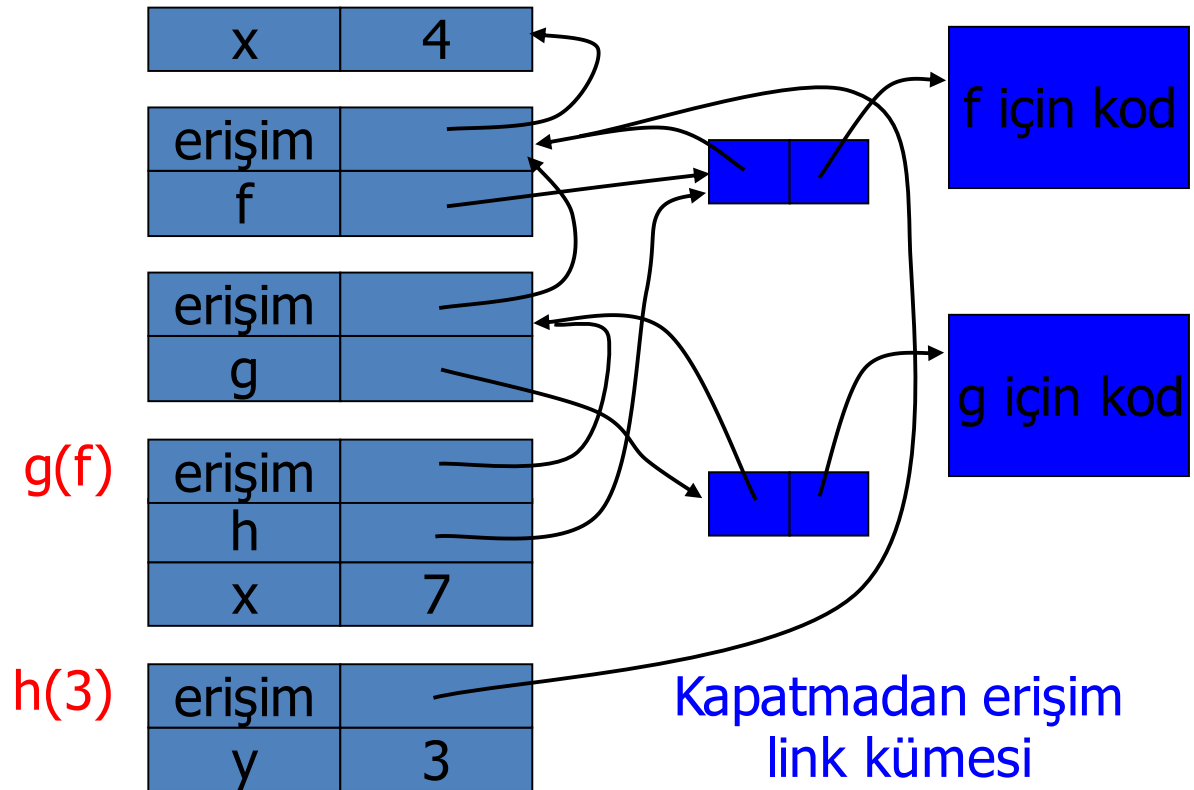
- Kapatma (closure): Bir altprogramın ve referansal platformun nerde tanımlandığıdır
  - Altprogram rastgele bir yerden çağrılabilir değilse referansal platform gereklidir
  - Statik-kapsamlı dil yuvalanmış altprogramlara izin vermez kapatmalara ihtiyacı yoktur
  - Kapatmalara sadece eğer altprogram yuvalanmış kapsamının içindeki değişkene erişebildiği zaman ihtiyaç duyulur ve herhangi bir yerden çağrılabilir
  - Kapatmaları desteklemek için, bir implemantasyonda (uygulamada) bazı değişkenlere sınırsız ölçüde değer verilebilir (çünkü bir altprogram normalde hayatta olmayan bir değişkene erişebilir)

# Fonksiyon argümanı ve kapatmalar

117

Erişim linkli yürütme zamanı stack

```
int x = 4;  
fun f(y) = x*y;  
fun g(h) =  
  let  
    int x=7  
  in  
    h(3) + x;  
  g(f);
```



# Özet: Function Arguments

118

- Use closure to maintain a pointer to the static environment of a function body
- When called, set access link from closure
- All access links point “up” in stack
  - ▣ May jump past activ records to find global vars
  - ▣ Still deallocate activ records using stack (lifo) order

# Return Function as Result

119

- Language feature
  - ▣ Functions that return “new” functions
  - ▣ Need to maintain environment of function
- Example

```
function compose(f,g)
    {return function(x) { return g(f (x)) }};
```
- Function “created” dynamically
  - ▣ expression with free variables
    - values are determined at run time
  - ▣ function value is closure =  $\langle \text{env}, \text{code} \rangle$
  - ▣ code *not* compiled dynamically (in most languages)

# Example: Return fctn with private state

120

ML

```
fun mk_counter (init : int) =  
  let val count = ref init  
      fun counter(inc:int) =  
        (count := !count + inc; !count)  
      in  
        counter  
      end;  
val c = mk_counter(1);  
c(2) + c(2);
```

- Function to “make counter” returns a closure
- How is correct value of **count** determined in **c(2)** ?



# Example: Return fctn with private state

121

JS

```
function mk_counter (init) {  
    var count = init;  
    function counter(inc) {count=count+inc; return count};  
    return counter};  
var c = mk_counter(1);  
c(2) + c(2);
```

Function to “make counter” returns a closure

How is correct value of **count** determined in call **c(2)** ?

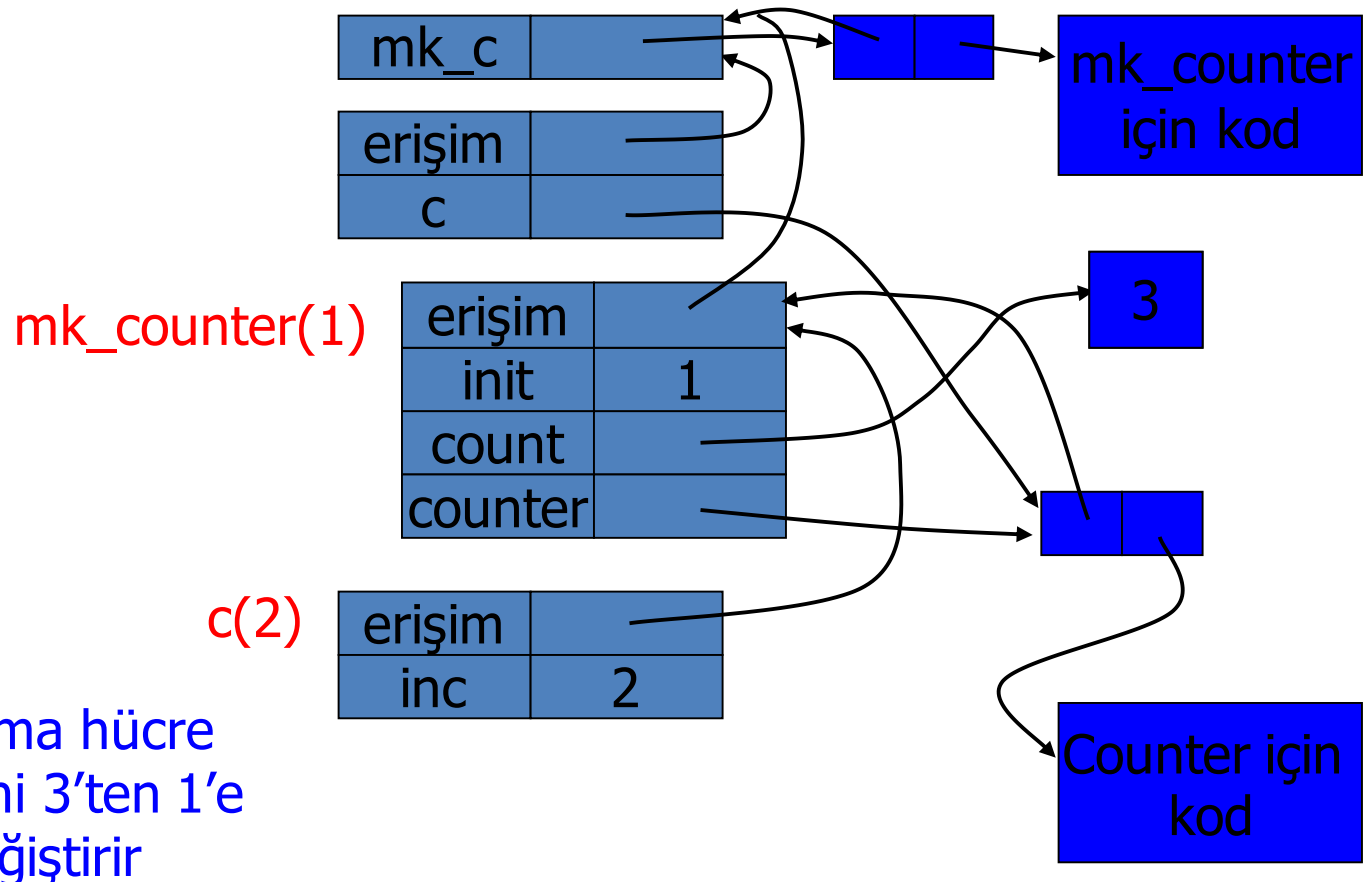
# Function Results and Closures

122

```

fun mk_counter(init: int) =
  let val count = ref init
      fun counter(inc: int) = (count := !count + inc; !count)
    in counter end
end;
val c = mk_counter(1);
c(2) + c(2);

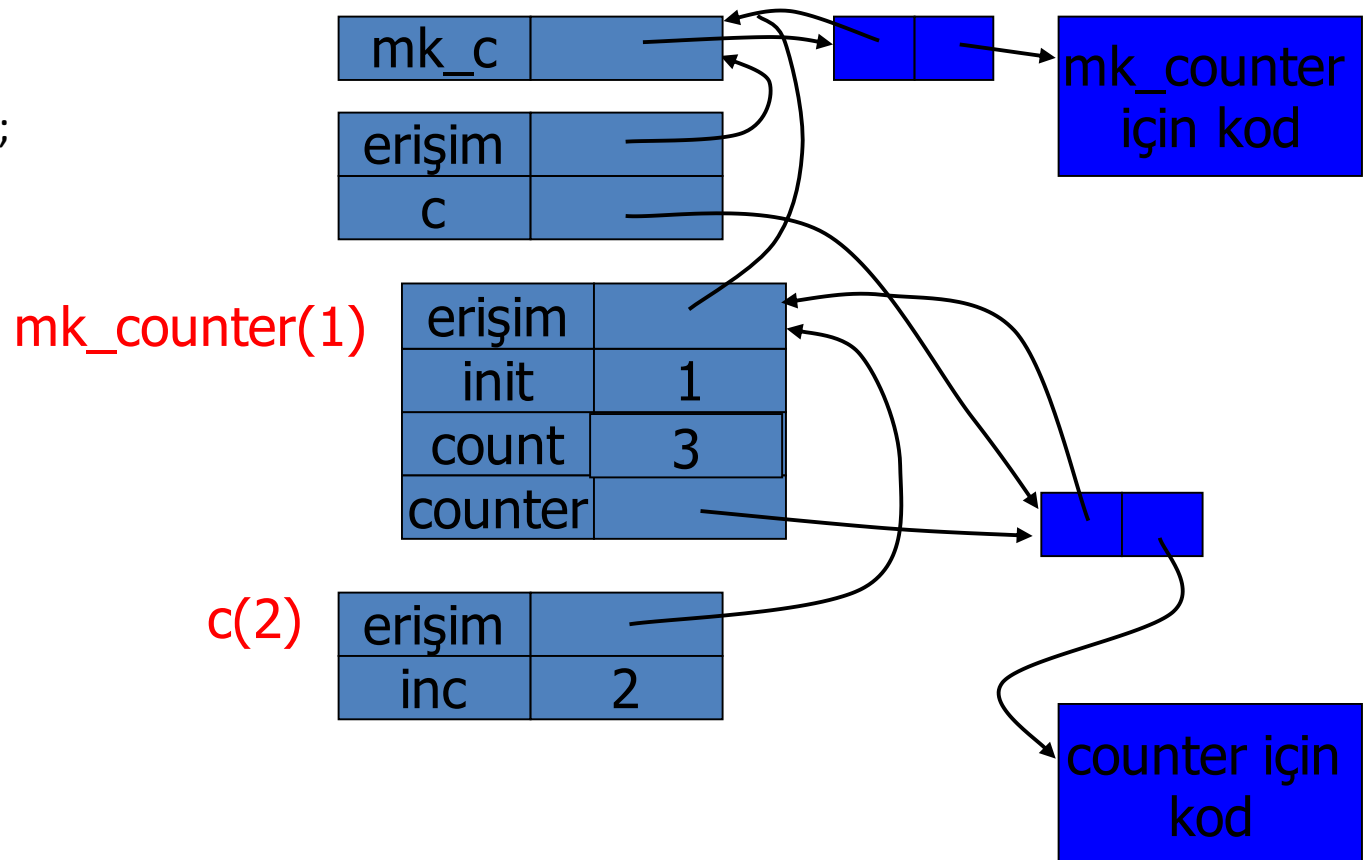
```



# Function Results and Closures

123

```
function mk_counter(init) {
  var count = init;
  function counter(inc) {count=count+inc; return
    count};
  return counter;
}
var c = mk_counter(1);
c(2) + c(2);
```



# Kapatmalar (devam)

124

❑ C,

```
void apply_to_A (int(*f)(int),
                 int A[],int A_size)
{
    int i;
    for (i=0; i < A_size; i++)
        A[i]=f(A[i]);
}
```

❑ Scheme:

```
(define apply-to-L (lambda (f l)
  (if (null? l) '()
      (cons (f (car l)) (apply-to-L f (cdr l))))))
```

❑ ML:

```
fun apply_to_L (f, l) =
  case l of
    nil => nil
  | h :: t => f (h) :: apply_to_L(f, t);
```

# Kapatmalar (devam)

125

## □ Bir JavaScript kapatması:

```
function makeAdder(x) {  
    return function(y) {return x + y;}  
}  
  
...  
var add10 = makeAdder(10);  
var add5 = makeAdder(5);  
document.write("add 10 to 20: " + add10(20) +  
               "<br />");  
document.write("add 5 to 20: " + add5(20) +  
               "<br />");
```

- Kapatılması `makeAdder` tarafından döndürülen adsız işlem

# Kapatmalar (devam)

126

## □ C#

- Yuvalanmış bir temsilci kullanarak C# 'de aynı kapatmayı yazabiliriz
- `Func<int, int>` (return türü) specifies a delegate that takes an `int` as a parameter and returns an `int`

```
static Func<int, int> makeAdder(int x) {  
    return delegate(int y) {return x + y;};  
}  
  
...  
Func<int, int> Add10 = makeAdder(10);  
Func<int, int> Add5 = makeAdder(5);  
Console.WriteLine("Add 10 to 20: {0}", Add10(20));  
Console.WriteLine("Add 5 to 20: {0}", Add5(20));
```

## 8.7. Etkinlik Kayıtları

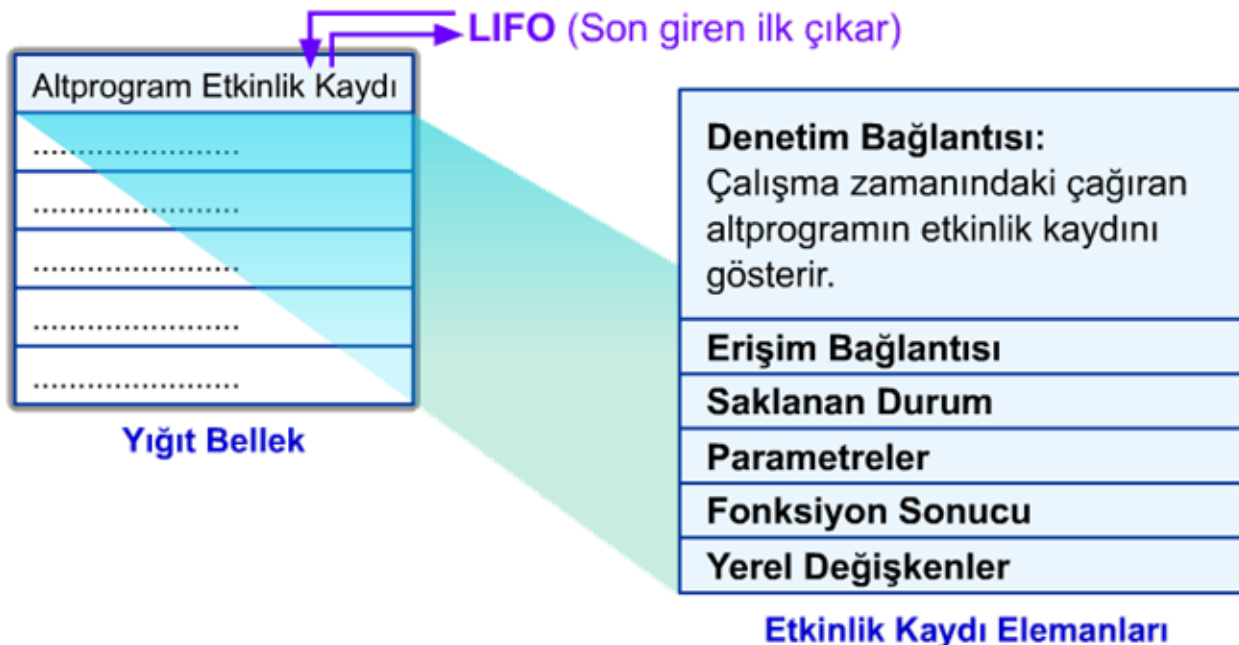
127

- Bir altprogramdaki değerler ve değişkenler için bellek ataması, altprogram çağrıldığı zaman, yani denetim altprograma aktarıldığında yapılır.
- ALGOL60 ve izleyen dillerde, altprogramlar arasında parametre iletişimi çalışma zamanındaki belleğin yığıt bölümü aracılığıyla gerçekleşir. Yığıt bellek, yerel değişkenler ve altprogramın çalışması süresince kullanılan parametreler gibi bir altprogram çağırımıyla ilişkili bilgiyi saklamak için kullanılır.
- Yığıt, en son giren en ilk çıkar (*last in first out, LIFO*) mantığında çalışan bir veri yapısı olduğu için ve aynı anda tek bir altprogram etkin olabildiği için, altprogramların çalışma tekniğine uygundur.

## 8.7. Etkinlik Kayıtları

128

- Etkin olan her altprogram için yığıt bellekte bir **etkinlik kaydı** (*activation record*) oluşturulur.
- Altprogramlarda kullanılan yerel değişkenler için her yordama ilişkin etkinlik kaydı kapsamında, yığıt bellekte bellek atanır.
- Aşağıdaki şekilde görülen etkinlik kaydı, yerel değişkenler, resmi parametreler ve diğer gerekli bilgi için bellek içerir.





## 8.7. Etkinlik Kayıtları

129

- Altprogramların çağrılmasında, en önce serbest bırakılacak bellek yeri, en son atanan bellek yeridir.
- Pascal ve C'de bir altprogram etkin olunca yerel değişkenler için yığıt bellekte yer ayrılır ve etkinlik bitince de geri verilir. Bir altprogram çağrımında gerçekleşen işlemler aşağıda gösterilmiştir:

```
call ortalama (a,b)
procedure ortalama (c: integer; d: integer);
begin
    ....
    ....
end;
```

Altprogram Çağrımında  
Gerçekleşen İşlemler

1. Gerçek parametreler, yığıta yerleştirilir.
2. Altprogram sona erdiğinde çalışacak program deyimini gösteren dönüş adresi, yığıta saklanır.
3. Yığıt indisi, yerel değişkenleri tutacak kadar yer ayrıldıktan sonra gerektiği kadar artırılır.
4. Altprogram kodu etkin duruma geçer.

5	Yerel Değişkenler
4	Dönüş Adresi
3	Gerçek Parametreler
2	.....
1	.....
0	.....

Yığıt İndisi: 4 → 5

## 8.7. Etkinlik Kayıtları

130

- Bir altprogram çağırımı sona erdiğinde gerçekleşen işlemler aşağıda gösterilmiştir:

```
call ortalama (a,b)
procedure ortalama (c: integer; d: integer);
begin
    .....
    .....
end;
```

Altprogram Çağırımında  
Gerçekleşen İşlemler

Altprogram Sona Erdiğinde  
Gerçekleşen İşlemler

1. Yığıt indisi, yerel değişkenler için kullanılan yer kadar azaltılır.
2. Dönüş adresi yığıttan alınır ve programın çalışmasının bu noktadan devam etmesi için, komut göstergesi (instruction pointer) ayarlanır.
3. Yığıt indisi, gerçek parametreler için kullanılan yer kadar azaltılır.

2	.....
1	.....
0	.....

Yığıt İndisi: 3 → 2

## 8.8. Özyinelemeli Bir Altprogramın Çoklu Çağırımı

131

- *Imperative* programlarda yinelemeli yapılara sıklıkla gerek duyulur ve bu amaçla döngüler kullanılır. Ancak matematiksel bir temele dayanan özyineleme de programlama açısından yararlı bir tekniktir.
- Bir altprogramı oluşturan deyimlerin her çalışması, altprogramın etkin olması olarak nitelendirilir.
- Eğer bir altprogram, kendi altprogram deyimleri içersinden etkin duruma getirilebiliyorsa, **özyinelemeli** (*recursive*) olarak adlandırılır.
- Özyinelemeli altprogramların aynı anda etkin olan birden çok kopyası olabilir. Bu nedenle özyinelemeli altprogramların bulunduğu programlama dillerinde, her altprogram etkinliğinin ayrı etkinlik kaydı bulunur.
- Özyinelemeyi faktöriyel fonksiyonu ile açıklayalım. Faktöriyel fonksiyonunun matematiksel tanımı yandaki şekilde verilmiştir.

## 8.8. Özyinelemeli Bir Altprogramın Çoklu Çağırımı

132

### □ C'de Faktöriyel Fonksiyonu:

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

### □ Pascal'da Faktöriyel Fonksiyonu:

```
function fakt(n:integer):integer
begin
    if n=0 then fakt:=1 else fakt:=n*fakt(n-1)
end;
```

## 8.8. Özyinelemeli Bir Altprogramın Çoklu Çağırımı

133

- **Scheme dilinde Faktöriyel Fonksiyonu:**
- Programlama mantığı olarak özyineli olan ve dolayısıyla döngülerin tamamının yerine özyineli fonksiyon yazıldığı Lisp dilinde ve bu dilin bir türevi olan Scheme dilinde ise aşağıdaki şekilde kodlanabilir:

```
(define fakt
  (lambda (n)
    (if (= n 1) 1
        (* n (fakt (- n 1))))))
```

- **Prolog dilinde Faktöriyel Fonksiyonu:**

```
fakt(1,1).
```

```
fakt(N,F) :- N>0, N1 is N-1, fakt(N1,F1), F = N * F1.
```

- Bir programlama dilinin özyinelemeli altprogramları desteklemesi için yığıt dinamik değişkenlere gereksinimi vardır. Çünkü bir altprogram tamamlanmadan aynı altprogram yeniden çağrılmakta ve altprogramın değişkenleri için yeniden bellek atanması gerekmektedir.

## 8.9. Tip Kontrol Parametreleri (Type checking parameters)

134

- Güvenilirlik için çok önemli olduğu düşünülmektedir
- FORTRAN 77 ve orjinal C: yok
- Pascal, FORTRAN 90+, Java, ve Ada: daima girektirir
- ANSI C and C++: kullanıcıya bırakılmıştır
  - ▣ Prototipler
- Nispeten yeni diller Perl, JavaScript, ve PHP tip kontrolünü gerektirmez
- In Python ve Ruby, değişkenlerin tipleri yoktur (nesnelerin vardır), yani parametre tip kontrolü mümkün değildir

## 8.10. Eşyordamlar (Coroutines)

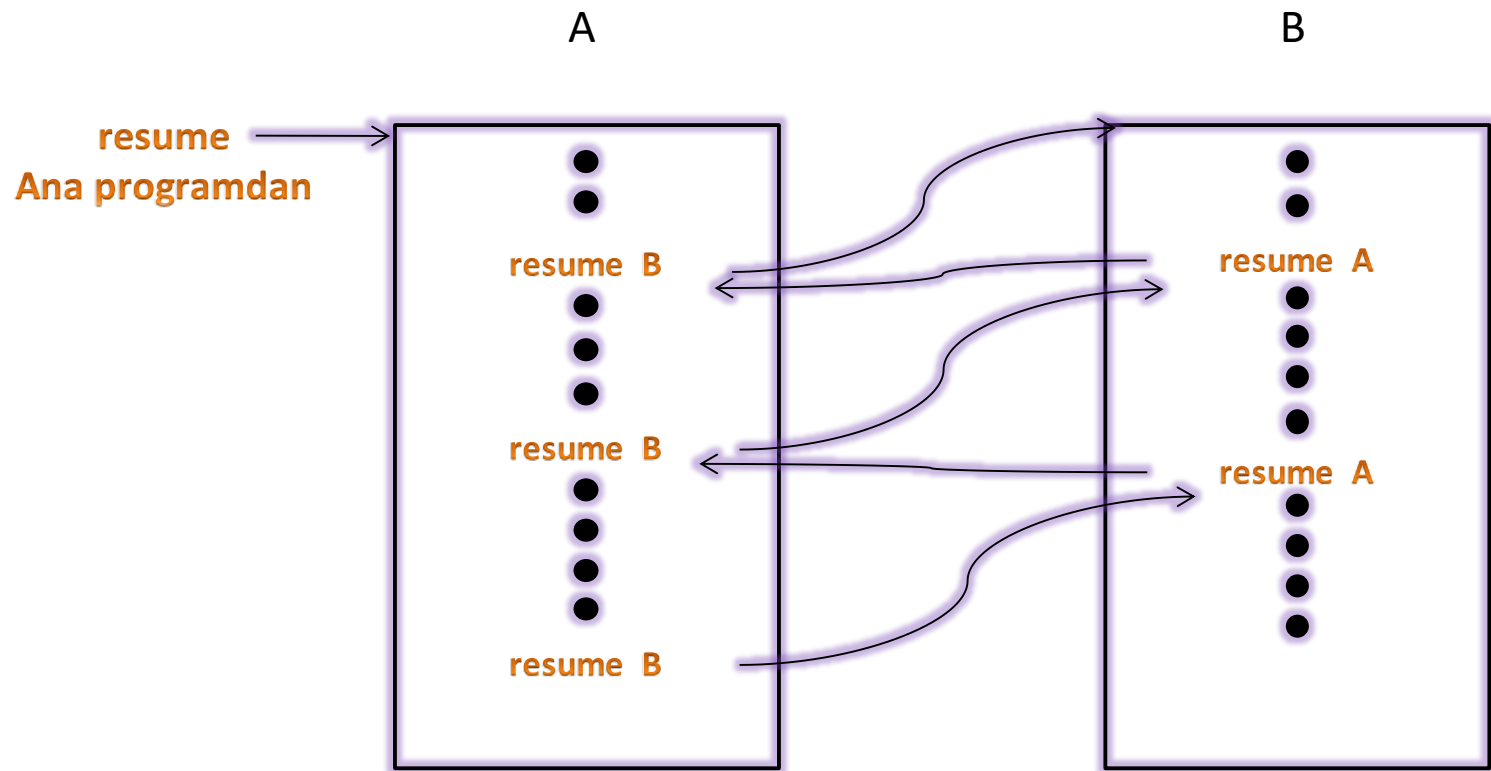
135

- ❑ İlk eşyordam kullanan programlama dili Simula 67 dir. Simula sistem simülasyonları (benzetim, gerçek sistemin bilgisayarda benzerini yaparak incelemek) için geliştirilmiştir. Daha sonra kullanan diller BLISS (Wulf et al., 1971), INTERLISP (Teitelman, 1975) ve Modula-2 (Wirth, 1985).
- ❑ Bir eşyordam (coroutine) birden çok giriş noktası olan ve bunları kendisi kontrol eden, daha önceki kullanılışlarını hatırlayan, çağırana eşit yetkilere sahip bir program birimidir.
- ❑ Bu nedenle simetrik kontrol da (symmetric control) da denir.
- ❑ Bu nedenle eşyordam çağrılmasında sürdür (resume) kullanılır.
- ❑ Eşyordama ilk giriş başından olur fakat daha sonraki girişler hep bırakılan yerden devam şeklinde olur.
- ❑ Tipik olarak eşyordamlar birbirlerin tekrar tekrar yürütürler.
- ❑ Eşyordamlar paralel işlemcili makinelerde yarı koştur zamanlı (quasi-concurrent execution) olarak çalışabilirler.

# Eşyordamlar (Coroutines)

136

□ (a)

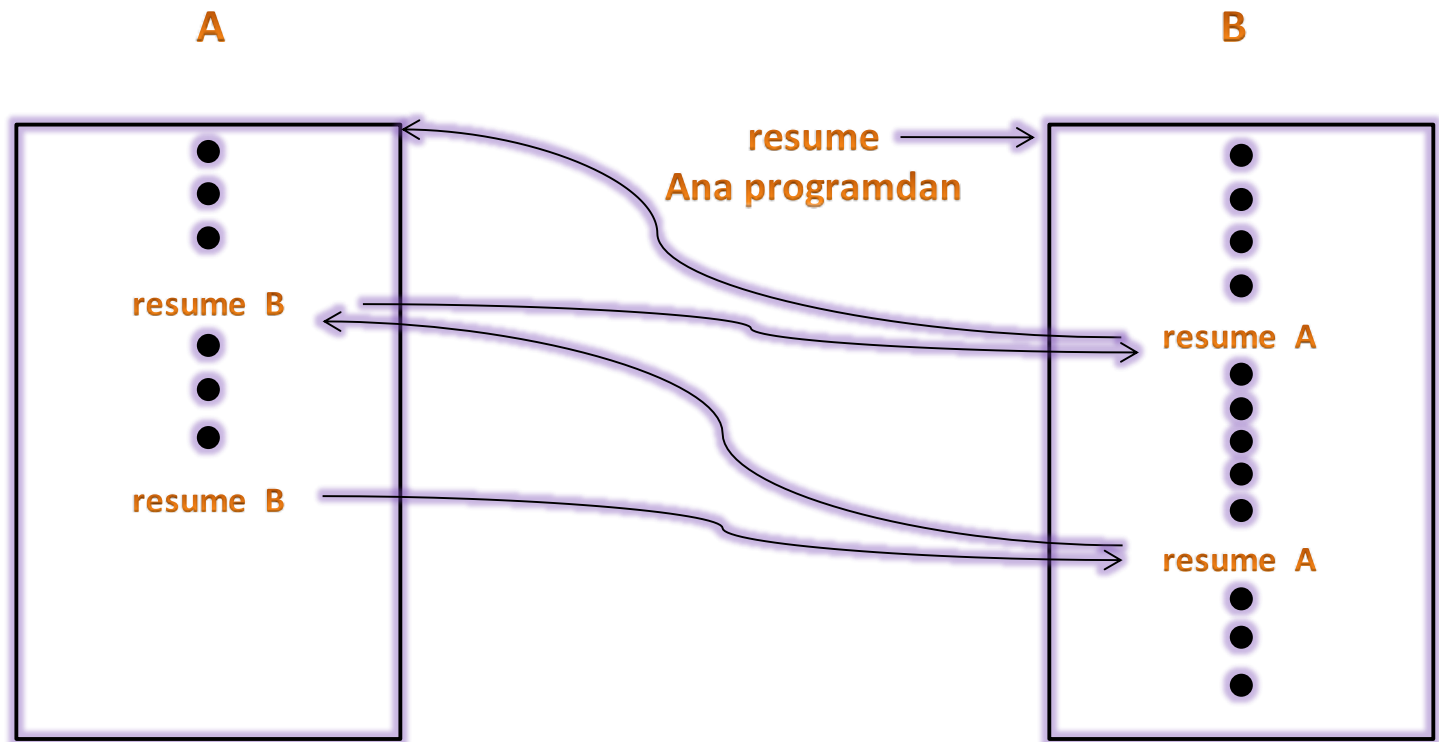




# Eşyordamlar (Coroutines)

137

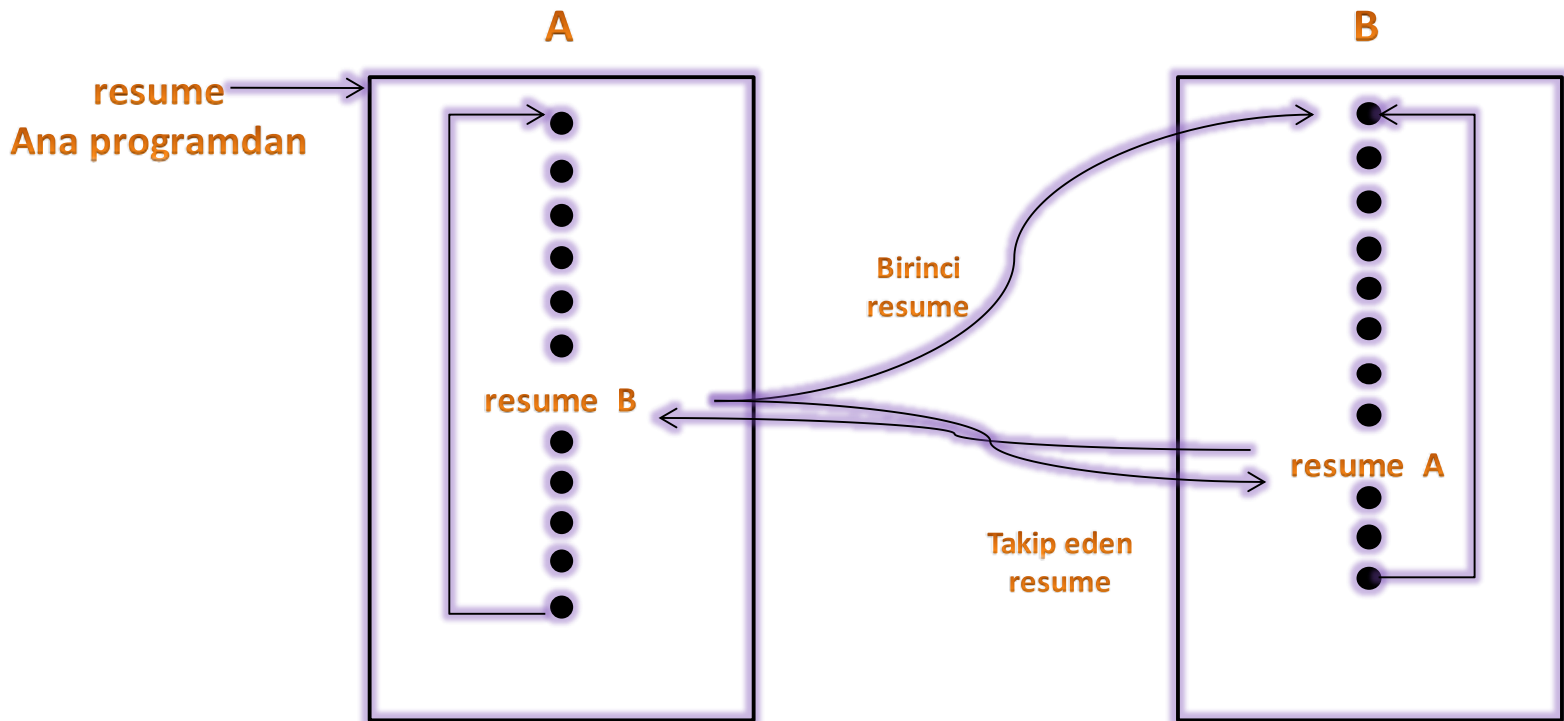
□ (b)



# Eşyordamlar (Coroutines)

138

□ (c)



# Ödev

139

- Call-by Need'i araştırınız
- C ve Java dilleri ile oluşturacağınız programlar için aşağıdaki işlemleri gerçekleştiriniz
  - ▣ a) Altprogramlar oluşturarak parametre aktarım yöntemlerini uygulayınız. Hangi yöntem neden uygulanamadı araştırıp yazınız.
  - ▣ b) Özyinelemeli altprogramlara örnek olarak
    - N sayısının toplamını
    - N sayısına kadar fibonacci sayılarının oluşturulmasını yapınız.
    - Bu hazırladığınız alprogramlarda değişken değerlerinin yığıt mantığına göre nasıl hafızada yer işgal ettiğini şematik olarak gösteriniz.

# Özet

140

- Bu bölümde, yazılım mühendisliği açısından altprogramlar ve altprogramların genel özellikleri incelenmiştir.
- Bu kapsamda; Parametreler, Parametre Aktarım Yöntemleri, Programlama Dillerinde Parametre Aktarımı, Parametre Aktarımında Tip Denetimi, Etkinlik Kayıtları, Özyinelenemeli Bir Altprogramın Çoklu Çağırımı ve Yüklenmiş Altprogramlar açıklanmıştır.

# Kaynaklar

- Roberto Sebesta, Concepts Of Programming Languages, International 10th Edition 2013
- David Watt, Programming Language Design Concepts, 2004
- Michael Scott, Programming Languages Pragmatics, Third Edition, 2009
- Zeynep Orhan, Programlama Dilleri Ders Notları
- Mustafa Şahin, Programlama Dilleri Ders Notları
- Ahmet Yesevi Üniversitesi, Uzaktan Eğitim Notları
- Erkan Tanyıldızı, Programlama Dilleri Ders Notları
- David Evans, Programming Languages Lecture Notes
- O. Nierstrasz, Programming Languages Lecture Notes
- Giuseppe Attardi, Programming Languages Lecture Notes
- John Mitchell, Programming Languages Lecture Notes