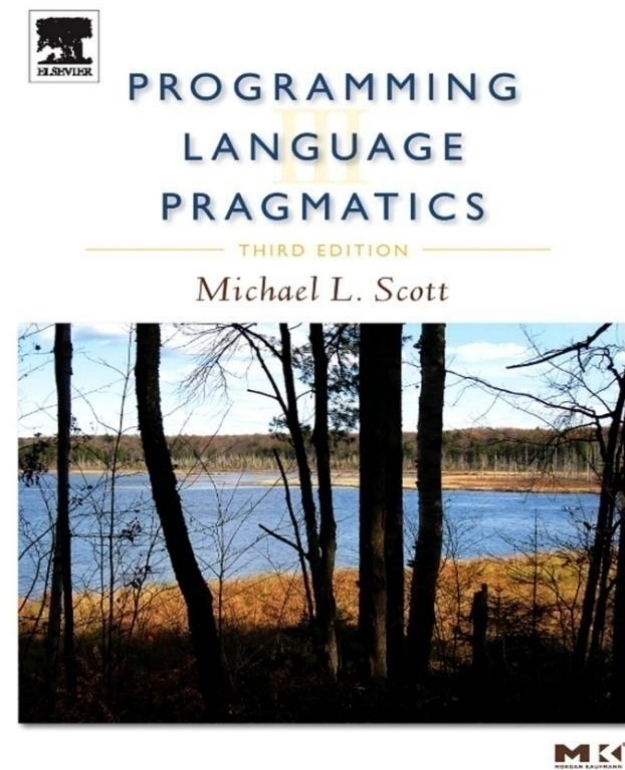
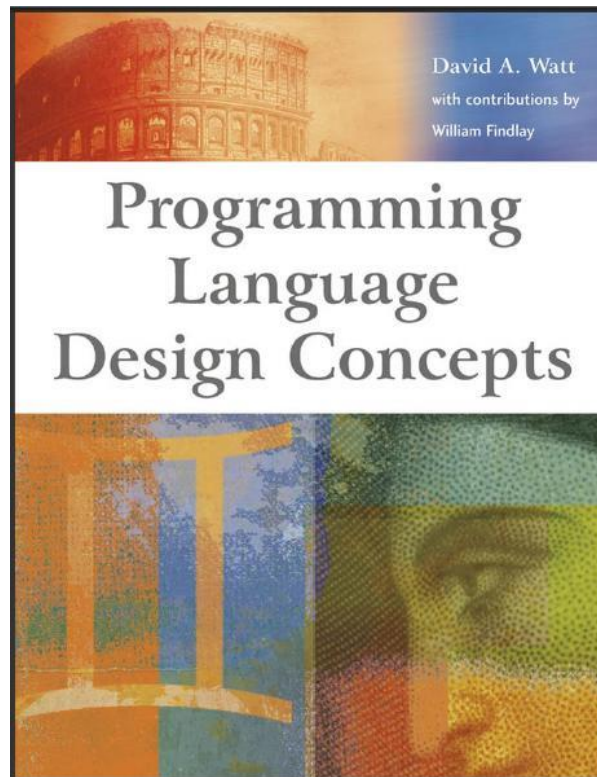
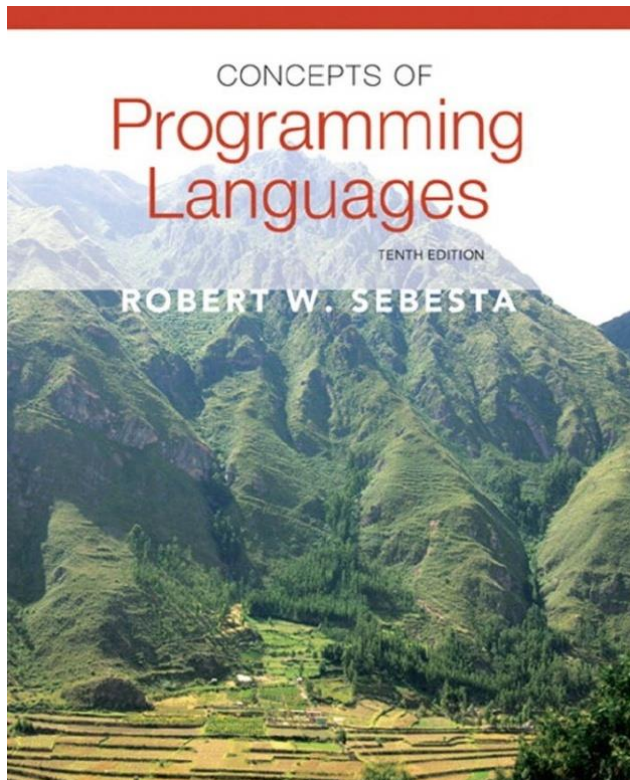


Bölüm 5: Adlar, Bağlama, Tip Kontrolü ve Kapsamlar



BÖLÜM 5 - Konular

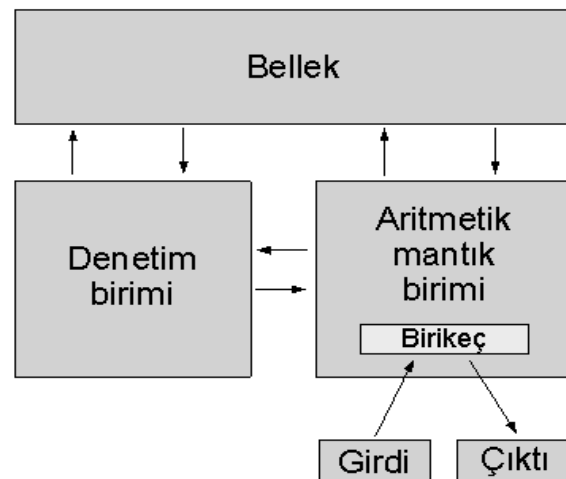
2

1. Giriş
2. Değişkenler
3. Sabitler
4. İşlemciler
5. İfadeler
6. Deyimler
7. Bağlama (Binding) Kavramı
8. Tip Kontrolü
9. İsim Kapsamları
10. Referans Çevreleri

5.1. Giriş

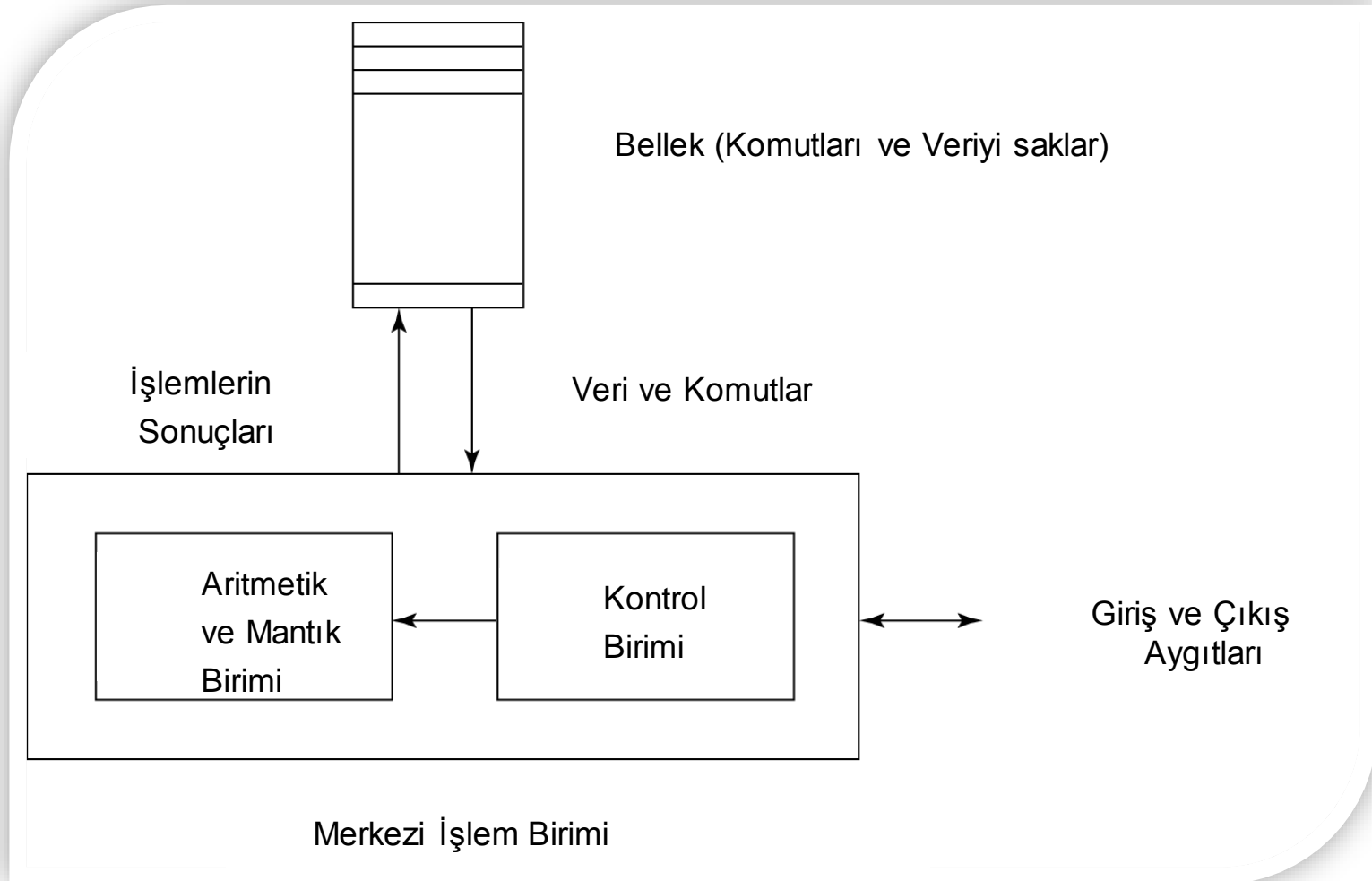
3

- ❑ Geleneksel bilgisayar mimarisi ***von Neumann*** mimarisi olarak adlandırılır. von Neumann mimarisi, veri ve komutları tek bir depolama biriminde bulunduran bilgisayar tasarı örneğidir.
- ❑ Bu mimari, her bellek hücresinin özgün bir adres ile tanımlandığı ana bellek kavramına dayanmaktadır.



Von Neumann Mimarisi

4



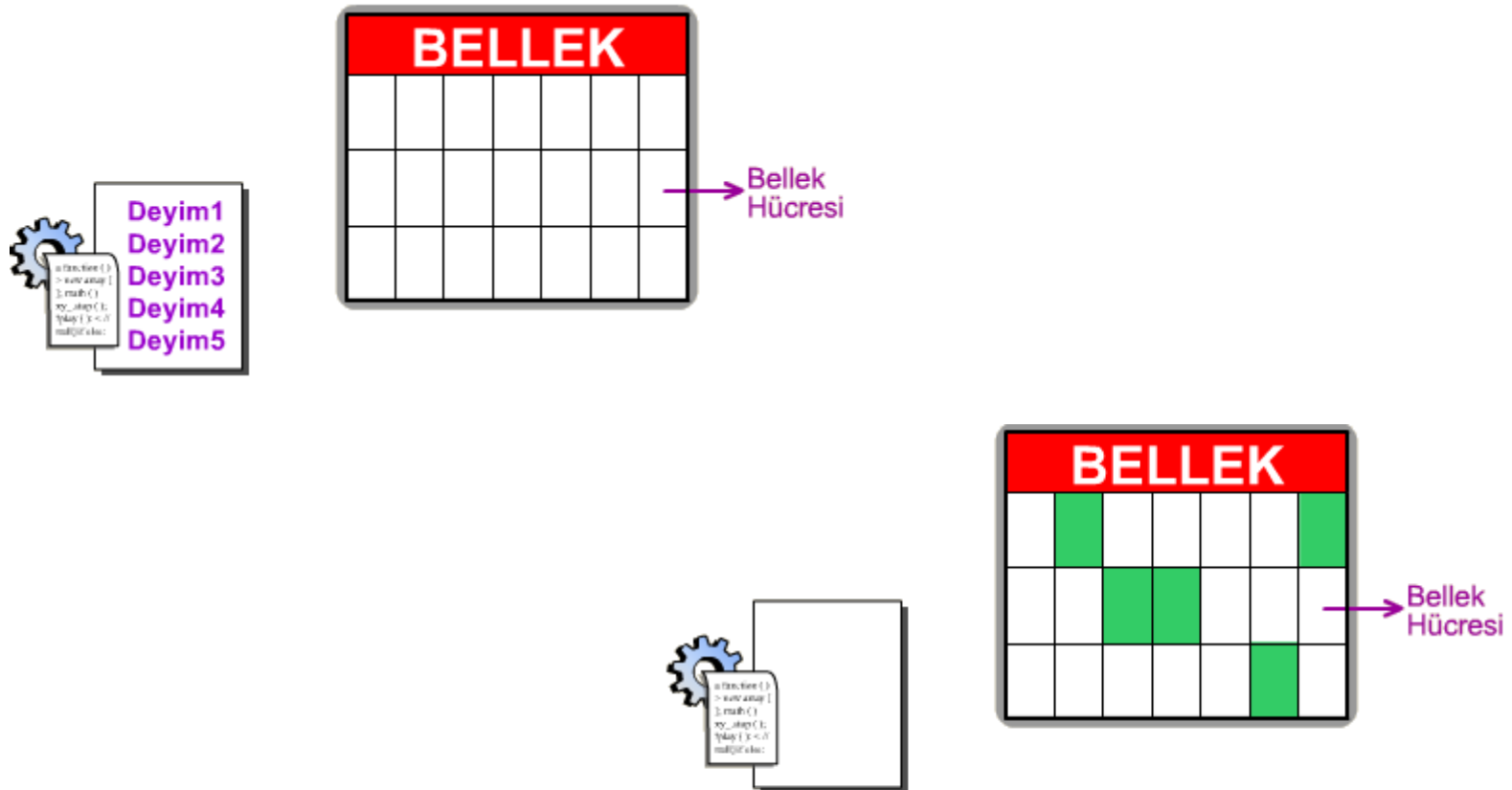
5.1. Giriş

5

- Bir bellek hücresinin içeriği, bir değerin belirli bir yöntemle göre kodlanmış gösterimidir.
- Bu içerik, programların çalışması sırasında okunabilir ve değiştirilebilir.
- *Imperative (zorunlu)* programlama, *von Neumann* mimarisindeki bilgisayarlara uygun olarak programların işlem deyimleri ile bellekteki değerleri değiştirmesine dayanır.

5.1. Giriş

6



Imperative programlama, programların işlem deyimleri ile bellekteki değerleri değiştirmesine dayanır.

5.1. Giriş

7

- **Atama İşlemi**
- Imperative programlamada en temel işlem atama işlemidir. Atama sembolü programlama dillerinde farklı şekilde gösterilebilir, ancak tüm programlama dillerinde atama sembolünün anlamı, sağ taraftaki değerin sol taraftaki değişkene aktarılmasıdır.
- `a = 10;` Java, C, C++, C#, PL/I, BASIC, Fortran vs.
- `a := 10;` Pascal, Algol, Ada vs.

Daha karmaşık atamalar:

- 1. Çok hedefe(PL/I)
 $A, B = 10$
- 2. Koşullu hedefler(C, C++, ve Java)
 $(first == 1) ? total : subtotal = 0$
- 3. Birleşik atama komutları(C, C++, ve Java)
 $sum += next;$
- 4. Tekli atama komutu(C, C++, ve Java)
 $a++;$
 - $a++$: iki tekli işleç bir değişkene uygulandığında değerlendirme, sağdan soladır. Bu nedenle:
 - $a++ \rightarrow -(a++)$ şeklinde değerlendirilir.
- 5. C, C++, ve Java "=" işlecini aritmetik işlemci olarak görürler.
 Örneğin:
 $a = b * (c = d * 2 + 1) + 1$
 Bu yapı ALGOL 68'den alınmıştır.

Bir ifade olarak atama komutu

- C, C++, ve Java'da, atama komutu sonuç döner.
- Bu nedenle ifadelerde işlenen olarak da yer alabilir.

- ▣ Örneğin:

```
while ((ch= getchar()) !=EOF) {...}
```

- Dezavantaj

- ▣ Başka tip yan etkiler:

```
a = b + (c = d/b++)-1;
```

- ▣ Yazım hatası olasılığı:

```
if(x = y) ...
```

```
if(x == y) ...
```

Çok farklı anlamları var. Java ve C# "if" içinde atamaya izin vermez.

Karışık biçimli atamalar

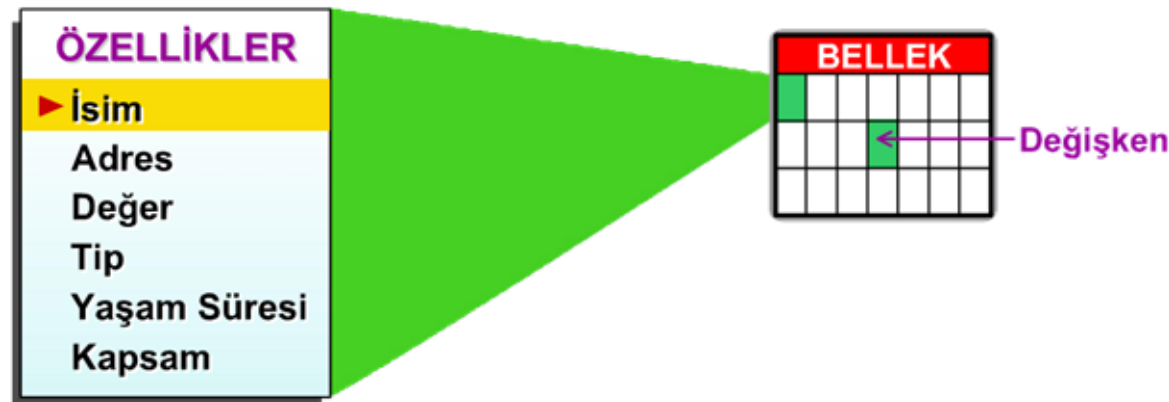
10

- ❑ FORTRAN, C, ve C++, her türlü sayısal değerler, sayısal değişkene atanabilir, gerekli çevrimler yapılır.
- ❑ Pascal'da interger'lar real'lara (gerçel) atanabilir, fakat tersi doğru değil. Bunun için gerçel sayının üste mi tamlanacağı, aşağıya mı indirileceğinin belirlenmesi gerekir.
- ❑ C# ve Java'da sadece genişletici örtülü çevrimler atamalarda çalışır.
- ❑ Ada'da atama için örtülü çevrim yoktur.

5.2. Değişkenler

11

- Bir değişken, bir veya daha çok bellek hücrelerinin soyutlamasıdır.



- l-value: Değişkenin adresidir
- r-value: Değişkenin değeridir.
- Değişkenlerin özellikleri aşağıda kısaca açıklanmıştır:

5.2. Değişkenler

12

- **İsim:** Bir değişkenin ismi, programda bir tanımlama deyimi ile tanıtıldıktan sonra, programdaki diğer deyimler tarafından o değişkene başvuru için kullanılır.
- **Adres:** Bir değişkenin bellekte bulunduğu yerin adresi değişkenin belirleyici bir özelliğidir. Bir çok programlama dilinde, programın farklı bölümlerinde, aynı ismin farklı adresler ile ilişkilendirilmesi olasıdır. Örneğin iki alt programın yerel değişken olarak *toplam* değişkenini tanımlaması durumunda, her iki alt programdaki *toplam* değişkeni, bir birinden bağımsız iki adres ile ilişkili olup, iki farklı değişken olarak nitelendirilecektir.
 - Bir değişken çalışma süresi boyunca farklı zamanlarda farklı adreslere sahip olabilir
 - Bir değişken bir program içerisinde farklı yerlerde farklı adreslere sahip olabilir
 - Eğer iki değişken adı aynı bellek konumuna erişmek için kullanılabiliyorsa, bunlara takma isim (*alias*) adı verilir.
 - Takma adlar (Aliases) okunabilirlik açısından zararlıdır (program okuyucuları hepsini hatırlamak zorundadır)

5.2. Değişkenler

13

- Takma adlar nasıl oluşturulabilir:
 - İşaretçiler (Pointers), referans değişkenleri (reference variables), C ve C++ bileşimleri (unions), (ve geçiş parametrelerle- Bölüm 9 da bahsedilecek)
 - Takma adlar için orijinal gerekçelerin bazıları artık geçerli değildir; örn. FORTRAN'da belleğin yeniden kullanımı
 - Bunlar dinamik ayırma ile değiştirilir

5.2. Değişkenler

14

- **Değer:** Bir değişkenin değeri, bellekteki değişken ile ilgili adreste, belirli bir yöntemle göre kodlanmış olarak saklanır. Değişkenlerin değerleri, bir programın çalıştırılması sırasında değiştirilebilirler.
- **Tip:** Bir değişkenin tipi, o değişkenin alabileceği değerleri ve o tip için belirlenmiş işlemleri belirler. Kayan-nokta (floating point) olduğu durumda, tip aynı zamanda duyarlılığı da belirler.
- **Yaşam Süresi:** Bir değişkenin yaşam süresi, bir bellek adresinin bir değişken ile ilişkili kaldığı süredir.
- **Kapsam:** Bir değişkenin kapsamı, o değişkenin isminin tanımlı olduğu program deyimlerini göstermektedir. Değişkenlerin kapsamı, durağan veya dinamik olarak belirlenebilir.

5.2. Değişkenler

15

DEĞİŞKEN ÖZELLİKLERİ

İSİM	Okunabilirlik
ADRES	Aliasing
DEĞER	Bellekte belirli yöntemle göre kodlanmış
TİP	Tip uyumsuzluğu, tip dönüşümü
YAŞAM SÜRESİ	Bellekle ilişkili kaldığı süre
KAPSAM	Geçerli olduğu deyimler

5.2.1. İsimler

16

- ❑ İsimler programlama dillerinde, değişkenlerin yanı sıra; etiketler, altprogramlar, parametreler gibi program elemanlarını tanımlamak için kullanılırlar.
- ❑ İsimleri tasarlamak için programlama dillerinde farklı yaklaşımlar uygulanmaktadır.



5.2.1.1. En Fazla Uzunluk

17

- Programlama dillerinde bir ismin en fazla kaç karakter uzunluğunda olabileceği konusunda farklı yaklaşımlar uygulanmıştır.
- Önceleri programlama dillerinde bir isim için izin verilen karakter sayısı daha sınırlı iken, günümüzdeki yaklaşım, en fazla uzunluğu kullanışlı bir sayıyla sınırlamak ve çoklu isimler oluşturmak için altçizgi "_" karakterini kullanmaktır.



değişken_xy_z...
uzunluk

5.2.1.1. En Fazla Uzunluk

18

□ Uzunluk

Eğer çok kısa
ise, anlaşılmaz

Programlama Dili	İzin Verilen Maksimum İsim Uzunluğu
FORTRAN I	maksimum 6
COBOL	30
FORTRAN 90, ANSI C	31
Ada	limit yoktur ve hepsi anlamlıdır (significant)
Java	limit yoktur ve hepsi anlamlıdır (significant)
ANSI C	31
C++	limit yoktur fakat konabilir

5.2.1.2. Küçük-Büyük Harf Duyarlılığı (Case Sensitivity)

19

Küçük-Büyük Harf Duyarlı

TOPLAM ≠ toplam
toplam ≠ ToPlaM

Küçük-Büyük Harf Duyarsız

TOPLAM = toplam
tOpLaM = toplam

- Birçok programlama dilinde, isimler için kullanılan küçük ve büyük harfler arasında ayırım yapılmazken, bazı programlama dilleri (Örneğin; C, C++, Java) isimlerde küçük-büyük harf duyarlılığını uygulamaktadır.
- Bu durumda, aynı harflerden oluşmuş isimler derleyici tarafından farklı olarak algılanmaktadır.
- Yandaki örnekte görüldüğü gibi; *TOPLAM*, *toplam*, ve *ToPlaM*, üç ayrı değişkeni göstermektedir.

5.2.1.2. Küçük-Büyük Harf Duyarlılığı (Case Sensitivity)

20

- Büyük küçük harf duyarlılığı
 - Dezavantaj: okunabilirlik (benzer görünen isimler farklıdır)
 - C++ ve Java'da daha kötüdür, çünkü önceden tanımlanmış isimler karışık büyük-küçüklüktedir (örn. **IndexOutOfBoundsException**)
 - C, C++, ve Java isimleri büyük küçük harfe duyarlıdır
 - Diğer dillerdeki isimler değildir

5.2.1.3. Özel Kelimeler

21

- ❑ Özel kelimeler, bir programlama dilindeki temel yapılar tarafından kullanılan kelimeleri göstermektedir.
- ❑ Okunabilirliğe yardımcı olmak için; ifade cümlelerini sınırlamak ve ayırmak için kullanılır



Anahtar Kelime
Keyword

REAL ELMA

REAL= 87.6

- ❑ **Anahtar Kelime**
- ❑ Bir anahtar kelime (*keyword*), bir programlama dilinin sadece belirli içeriklerde özel anlam taşıyan kelimelerini göstermektedir.
 - ▣ Örneğin FORTRAN'da *REAL* kelimesi, bir deyimin başında yer alıp, bir isim tarafından izlenirse, o deyimin tanımlama deyimi olduğunu gösterir. *REAL ELMA* gibi. Eğer *REAL* kelimesi, atama işlemcisi "=" tarafından izlenirse, bir değişken ismi olarak görülür. *REAL = 87.6* gibi. Bu durum dilin okunabilirliğini azaltır.

5.2.1.3. Özel Kelimeler

22



- **Ayrılmış Kelime:**
- Ayrılmış kelime (*reserved word*), bir programlama dilinde bir isim olarak kullanılamayacak özel kelimeleri göstermektedir. Örneğin Pascal'da, *for*, *begin*, *end* gibi kelimeler, isim olarak kullanılamaz ve ayrılmış kelime olarak nitelendirilir.

5.2.2. Veri Tipi Kavramı

23

- Bir **veri tipi**, aynı işlemlerin tanımlı olduğu değerler kümesini göstermektedir. Bir değişkenin tipi, değişkenin tutabileceği değerleri ve o değerlere uygulanabilecek işlemleri gösterir.
- Örneğin; tamsayı (*integer*) tipi, dile bağımlı olarak belirlenen en küçük ve en büyük değerler arasında tamsayılar içerebilir ve sayısal işlemlerde yer alabilir.
- Veri tipleri, programlama dillerinde önemli gelişmelerin gerçekleştiği bir alan olmuş ve bunun sonucu olarak, programlama dillerinde çeşitli veri tipleri tanıtılmıştır.
- Tipler **ilkel** (*temel-primitive*) ve **yapısal** (*composite*) olarak gruplandırılabilir.

5.2.2. Veri Tipi Kavramı

24

- ❑ **İlkel tipler**, çoğu programlama dilinde yer alan ve diğer tiplerden oluşmamış veri tiplerini göstermektedir.



C

- ❑ bool
- ❑ char
- ❑ int
- ❑ float

Ada

- Boolean = {false, true}
- Character = { . . . , 'a', . . . , 'z', . . . , '0', . . . , '9', . . . , '?', . . . }
- Integer = { . . . , -2, -1, 0, +1, +2, . . . } -> {-2 147 483 648, . . . , +2 147 483 647}
- Float = { . . . , -1.0, . . . , 0.0, . . . , +1.0, . . . }

5.2.2. Veri Tipi Kavramı

25

Temel C++ değişken tipleri

Keyword	Nümerik aralık		Ondalık kısım	Bellek alanı byte
	Alt sınır	Üst sınır		
char	-128	127	yok	1
short	-32,768	32,767	yok	2
int	-2,147,483,648	2,147,483,647	yok	4
long	-2,147,483,648	2,147,483,647	yok	4
float	3.4×10^{-38}	3.4×10^{38}	7	4
double	1.7×10^{-308}	1.7×10^{308}	15	8
long double	3.4×10^{-4932}	1.1×10^{4932}	19	10

5.2.2. Veri Tipi Kavramı

26

- **Yapısal tipler** ise çeşitli veri tiplerinde olabilen bileşenlerden oluşmuştur. Bir yapısal tipin elemanları, tipin bileşenlerini oluşturmaktadır. Bir yapısal tipteki her bileşenin, tip ve değer özellikleri bulunmaktadır.



5.2.2. Veri Tipi Kavramı-Yapısal Tipler

27

□ Örnek

Ada örneği

```
type Month is (jan, feb, mar, apr, may,  
    jun,jul, aug, sep, oct, nov, dec);  
type Day_Number is range 1 .. 31;  
type Date is  
    record  
        m: Month;  
        d: Day_Number;  
    end record;
```

Java Örneği

```
enum Month {jan, feb, mar, apr, may, jun,  
    jul, aug, sep, oct, nov, dec};  
struct Date {  
    Month m;  
    byte d;  
};
```

5.3. Sabitler

28

- Bir **sabit**, belirli bir tipteki bir değerin kodlanmış gösterimini içeren ancak programın çalıştırılması sırasında değiştirilemeyen bellek hücresine veya hücrelerine verilen isimdir. Bir sabit genellikle ilkel tipte bir değerdir. Örneğin 568, bir tamsayı sabittir.
- Bir değişken, bir bellek yerine bağlandığında bir değere de bağlanıyorsa ve daha sonra bu değer değiştirilemiyorsa o değişkene **isimlendirilmiş sabit** denir. İsimlendirilmiş sabitlerin tanımlanması için, bir çok dilde (örneğin Pascal, C# vs) **const** tanımlayıcısı kullanılır. C'de ise isimlendirilmiş sabit tanımlamak için **#define** kullanılır.



Değişken



Sabit



İsimlendirilmiş
Sabit

5.3. Sabitler

29

- ❑ Sabit bir değerin programda birçok kez yinelenmesi durumunda, isimlendirilmiş sabitlerin kullanılması okunabilirlik ve değiştirilebilirlik açısından yararlıdır. Örneğin 3.14159 değeri yerine *pi* isminin kullanılması, programın okunabilirliğini artırır.
- ❑ Bir başka örnek olarak, 50 elemanlı bir diziyi işleyen bir programı düşünelim. Bu programda birçok kez (örneğin, dizi tanımlamada, döngülerde vb.) dizi sınırına başvuru yer alır. Bu değerin programın başında isimlendirilmiş sabit olarak tanımlanması, programın okunabilirliğini ve güvenilirliğini artırır.
- ❑ Programları parametrelerle ifade etmek için kullanılır.
- ❑ İsimlendirilmiş sabitlere değer bağlama (binding) statik (**manifest constants**) veya dinamik olabilir (5.7.'de anlatılacak)

5.4. İşlemciler (Operatörler)

30

- İşlemciler, genel özelliklerine ve işlenenlerin niteliğine göre iki şekilde sınıflandırılabilirler.

- **İşlemcilerin Genel Özelliklerine Göre**

- İşlenen sayısı,
- İşlemcinin yeri,
- Öncelik
- Birleşmelilik (associativity)

- **İşlenenlerin Niteliğine Göre;**

- Sayısal işlemciler,
- İlişkisel işlemciler
- Mantıksal işlemciler

5.4.1. İşlemcilerin Genel Özellikleri

31

- **İşlenen Sayısı (arity)** : Bir işlemci, alabileceği işlenen sayısına göre **tekli** (unary), **ikili** (binary), **üçlü** (ternary) veya **çoklu** (n-ary) olabilir.
 - ▣ `p = &a;` (C – unary, prefix)
 - ▣ `i = -5;` (C – unary, prefix)
 - ▣ `i++;` (C – unary, postfix)
 - ▣ `i = i + j;` (C – binary, infix)
 - ▣ `i = i + j + 5;` (C – binary, infix)
 - ▣ `(plus i j 5)` (LISP – nary, prefix)
 - ▣ `sonuc = (sayi_1 % 2 == 1) ? "tek" : "cift" ;` (java,C, ternary)

5.4.1. İşlemcilerin Genel Özellikleri

32

prefix
+ab
infix
a+b
postfix
ab+

-5
&sum
d=++b

A+B

ptr^
a++

İkili işlemciler
işlenenler
arasındadır

- **İşlemcinin Yeri** : Çoğu işlemci işlenenleri arasına yazılmakla birlikte, bazı işlemciler, işlenenlerinden önce veya sonra da yazılabilirler. İşlemciler bir ifadede, işlenenlerden önce (*prefix*), işlenenler arasında (*infix*) ve işlenenlerden sonra (*postfix*) olmak üzere üç şekilde yer alabilirler.

➤ Genellikle tekli işlemciler, işlenenlerden önce (örneğin; -5, &toplam gibi) veya sonra yazılır (örneğin; ptr^ gibi). "^" işlemcisinin görevi, bir gösterge ile gösterilen değeri belirtmektir. Bu işlemci, hem tekli hem de işlenenden sonra yazılan (postfix) işlemcidir.

➤ İkili işlemciler ise işlenenlerin arasında yazılır.

5.4.1. İşlemcilerin Genel Özellikleri

33

- ❑ **Öncelik:** İşlemcilerin öncelikleri, birden çok işlemcinin yer aldığı bir ifadede, parantez kullanılmadığında, bir ifadenin bileşenlerinin değerlendirilme sırasını belirler.
- ❑ İkili işlemciler, *infix* gösterimde, " $a+b$ " de olduğu gibi işlenenleri arasına yazılır. Ancak *infix* gösterimdeki sorun, birden çok işlemcinin birlikte yer aldığı bir ifadede görülür.
- ❑ Örneğin; " $a+b*c$ " gibi bir ifadenin değerlendirilmesi nasıl olacaktır? Sonuç, " a " ve " $b*c$ " nin toplamı mı; yoksa " $a+b$ " ve " c " nin çarpımı mıdır? Bu soruların yanıtları işlemcilerin **öncelik** ve **birleşmelilik (associativity)** kavramları ile açıklanabilir.

5.4.1. İşlemcilerin Genel Özellikleri

34

- **Öncelik:**
- Sayısal ifadeler, ilişkisel ifadelerin işlenenleri olabileceği ve ilişkisel ifadeler de Boolean ifadelerin işlenenleri olabileceği için, üç işlemci grubunun kendi aralarında öncelikleri vardır.
- İlişkisel işlemcilerin önceliği, her zaman sayısal işlemcilerden düşüktür.

$$X+20 \leq k*2$$

- İlişkisel ifadeler ise mantıksal ifadeler için bir operand olabileceğinden ilişkisel ifadeler mantıksal ifadelerden önce yapılmalıdır

5.4.1. İşlemcilerin Genel Özellikleri

35

	FORTTRAN	PASCAL	C	ADA
En yüksek öncelik	** (exponentiation)	*, / , div , mod	++, -- (postfix)	** , abs
	*, /	+ , -	++, -- (prefix)	*, / , mod
	+, -		Tekli (unary) +, -	Tekli (unary) +, -
			*, / , %	İkili (Binary) +, -
			İkili (Binary) +, -	
En düşük öncelik				

5.4.1. İşlemcilerin Genel Özellikleri

36

ADA Programlama Dili	
En yüksek	**, abs, not
	*, /, mod, rem
	+, - (tekli)
	+, -, & (ikili)
	=, /=, >, <, <=, >=, in, not in
En düşük	AND, OR, XOR, AND THEN, OR ELSE

$(A < B) \text{ and } (A > C) \text{ or } X = 0$ Doğru

$A < B \text{ and } A > C \text{ or } X = 0$ Yanlış

5.4.1. İşlemcilerin Genel Özellikleri

37



- Her işlemcinin, programlama dili tasarlanırken önceden belirlenmiş bir önceliği vardır. Daha yüksek bir öncelik düzeyinde yer alan bir işlemci, işlenenlerini daha düşük bir düzeydeki bir işlemciden önce alır.

- Klasik öncelik seviyeleri
 - ✓ Parantezler
 - ✓ Tekli operatörler
 - ✓ ** (Eğer dil destekliyorsa, üs alma)
 - ✓ *, /(çarpma,bölme)
 - ✓ +, -(toplama,çıkarma)

Geleneksel bir kural olarak, sayısal işlemcilerden "*" ve "/", toplama "+" ve çıkarmadan "-" daha yüksek önceliğe sahiptir. Bu nedenle yukarıdaki ifadede, "*" işlemcisi, işlenenlerini "+" dan önce alır ve " $a+b*c$ ", " $a+(b*c)$ " ye eşittir.

5.4.1. İşlemcilerin Genel Özellikleri

38

- **Birleşmelilik (Associativity):** Bir ifadede aynı öncelik düzeyinde iki işlemci bulunuyorsa, hangi işlemcinin önce değerlendirileceği dilin birleşmelilik kuralları ile belirlenir. Bir işlemci, sağ veya sol birleşmeli olabilir.

Örnek

$$4-2-1$$

1. işlem

2. işlem

- **Sol birleşmeli:** Bir işlemcinin birden çok kez yer aldığı bir ifadedeki alt ifadeler, soldan sağa olarak gruplanırsa, işlemci sol birleşmeli olarak adlandırılır. Aritmetik işlemcilerden "+", "-", "*" ve "/" sol birleşmelidir.

Örnek

$$2^{3^4} = 2^{81}$$

- **Sağ birleşmeli:** Öte yandan, eğer bir işlemcinin birden çok kez yer aldığı bir ifadedeki alt ifadeler, sağdan sola gruplanırsa işlemci, sağ birleşmeli olarak adlandırılır. Üs alma işlemcisi sağ birleşmelidir.

5.4.1. İşlemcilerin Genel Özellikleri

39

- ❑ **Birleşmelilik (Associativity)**
- ❑ Tipik birleşmelilik kuralları
 - Soldan sağa, ******(hariç, burada sağdan sola),
 - Zaman zaman tekli operatörlerin birleşmeliliği sağdan sola olabilir (ör., FORTRAN)
- ❑ APL dili farklıdır; bu dilde tüm operatörler eşit önceliklere sahiptir ve operatörlerin birleşmeliliği sağdan soladır.
- ❑ Öncelik ve birleşmelilik kuralları parantezlerle geçersiz kılınabilir

5.4.1. İşlemcilerin Genel Özellikleri

40

Birleşmelilik (Associativity)

Programlama Dili	Birleşme Kuralı	İşlemciler
FORTRAN	Sol Birleşmeli	*, /, +, -
	Sağ Birleşmeli	**
PASCAL	Sol Birleşmeli	Bütün işlemciler
	Sağ Birleşmeli	
C	Sol Birleşmeli	Postfix++, postfix--, *, /, %, ikili +, ikili-
	Sağ Birleşmeli	Prefix++, pretfix--,tekli +, tekli-
C++	Sol Birleşmeli	*, /, %, ikili +, ikili-
	Sağ Birleşmeli	++, --,tekli +, tekli-
ADA	Sol Birleşmeli	** dışındakiler
	Sağ Birleşmeli	** birleşme özelliği yok

5.4.2. Niteliğine Göre İşlemciler

41

- İşlemciler, işlenenlerin niteliğine göre *sayısal*, *ilişkisel* veya *mantıksal işlemciler* olabilirler.
- **Sayısal işlemciler**, sayısal işlenenlere uygulanan işlemcilerdir. Kullanılan semboller programlama dillerinde farklılık gösterebilmekle birlikte, genel olarak üs alma, toplama, çıkarma, mod, çarpma, bölme gibi işlemcilerdir.

Sembol	Anlamı	Örnek	Sonuç
\wedge	üs alma	$2 \wedge 3$	8
$*$	çarpma	$2 * 3$	6
$/$	bölme	$8 / 2$	4
\backslash	tam bölme	$5 \backslash 2$	2
MOD	kalan	$5 \text{ MOD } 2$	1
$+$	toplama	$5 + 2$	7
$-$	çıkarma	$5 - 2$	3

5.4.2. Niteliğine Göre İşlemciler

42

- **İlişkisel işlemciler**, Bir ilişkisel işlemci, iki işleneninin değerlerini karşılaştırır ve eğer mantıksal (*Boolean*) tipi dilde tanımlı bir veri tipi ise mantıksal tipte bir sonuç oluşturur.
 - Genellikle, ilişkisel işlemcilerle kullanılabilen işlenenler; sayısal, karakter veya sıralı (*ordinal*) tiplerde olurlar.

	Pascal	C
Eşit	=	==
Eşit değil	<>	!=
Küçük	<	<
Büyük	>	>
Küçük veya eşit	<=	<=
Büyük veya eşit	>=	>=

5.4.2. Niteliğine Göre İşlemciler

43

- **Mantıksal İşlemciler**, sadece mantıksal (Boolean) işlenenleri alırlar ve mantıksal değerler, DOĞRU ve YANLIŞ üretirler.
 - Mantıksal işlemciler, **AND-&** (ve), **OR-|** (veya), **NOT-~** (değil), **XOR-^** (dışlayan veya– aynı ise 0, farklı ise 1) gibi işlemleri de içerebilirler. Mantıksal işlemcilerde öncelik sıralaması genellikle NOT, AND ve OR şeklindedir.
 - İlişkisel işlemcilerin önceliği, her zaman sayısal işlemcilerden düşüktür.

Örnek

$$\underbrace{a+10}_{\text{önce}} > \underbrace{b*5}_{\text{önce}}$$

NOT: İlk önce aritmetik işlem, ardından mantıksal işlem yapılır.

5.4.2. Niteliğine Göre İşlemciler

44

- Diğer dillerden farklı olarak Pascal'da, mantıksal işlemcilerin önceliği, ilişkisel işlemcilerden yüksektir.

Örnek

$a > 0$ or $b < 100$

NOT: Öncelik "or"da olduğu için ifade geçersizdir. Bu yüzden doğru söz dizimi;

$(a > 0)$ or $(b < 100)$

5.4.2. Niteliğine Göre İşlemciler (Özet)

45

SAYISAL İŞLEMCİLER

Sembol	İşlev	Formül	Sonuç
*	Çarpma	4*2	8
/	Bölme ve tamsayı bölme	64/4	16
%	Modül veya kalan	13%6	1
+	Toplama	12+9	21
-	Çıkarma	80-15	65

İLİŞKİSEL İŞLEMCİLER

Anlamı	C++	PASCAL
Büyüktür	>	>
Küçüktür	<	<
Eşittir	==	=
Eşit değildir	!=	<>
Büyük veya eşittir	>=	>=
Küçüktür veya eşittir	<=	<=

MANTIKSAL İŞLEMCİLER

C Dili Mantıksal İşlemcileri		
&&	VE	AND
	VEYA	OR
!	DEĞİL	NOT



5.4.3. İşlemci Yükleme

46

- İşlemcilerin anlamlarının, işlenenlerin sayısına ve tipine bağlı olarak belirlenmesine **işlemci yüklemesi** (*operator overloading*) denir.
 - Sayısal işlemciler, programlama dillerinde sıklıkla birden çok anlamda kullanılırlar. Örneğin "+", hem tamsayı hem de kayan-noktalı toplama için kullanılır ve bazı dillerde, sayısal işlemlere ek olarak karakter dizgilerin birleştirilmesi için de kullanılır. İşlemci yüklemelerinde, "+" da olduğu gibi, benzer anlamlarda olmayabilir.
 - Bir diğer örnek olan '-' işlemcisi, hem bir sayısal değerin negatif olduğunu belirtmek için tekli işlemci olarak, hem de ikili bir işlemci olan sayısal çıkarma işlemini göstermek için kullanılır. Her ne kadar işlemcinin iki kullanımında anlamsal yakınlık varsa da, bir ifadede yanlışlıkla birinci işlenenin unutulması, derleyici tarafından hata olarak algılanmayacak ve ikinci işlenen negatif değer olarak kabul edilecektir. Bu ve benzeri işlemci yüklemeleri, fark edilmesi güç hatalara neden olabilmektedir.

5.4.3. İşlemci Yükleme

47

$2 + 3 = 5$	+, tam sayı değerler için toplama işlemi anlamına gelir.
<code>"a" + "b" = "ab"</code>	+, karakter deyimleri için birleştirme anlamına gelir.
$2.5 + 3.8 = 6.3$	+, kayan noktalı sayılar için toplama işlemi anlamına gelir.



İşlemcilerin yüklenmesi, programlama dillerinin ifade yeteneğini artırmakla birlikte, okunabilirliği ve güvenilirliği azaltabilmektedir.

5.4.3. İşlemci Yükleme

48

- Bazısı potansiyel olarak sorunludur (örn., C ve C++ da ‘*’ ikili olarak çarpı, tekli olarak adresleme (pointer))
 - ▣ Derleyicinin hata belirlemesindeki kayıplar (derleyici operant eksiklerini fark etmeli) ($a*b$ yerine $*b$ yazmak gibi)
 - ▣ Bazı okunabilirlik kayıpları (okunabilirliği zorlaştırır)
- C++, C#, ve F# kullanıcı tarafından tanımlanan operatörlere izin verir.
 - ▣ Böyle operatörler anlamlı kullanıldığında, okunabilirliğe bir yardımı olabilir (metot çağrılarından kaçınmak, ifadeler doğal görünür)
 - ▣ Potansiyel problemler:
 - Kullanıcılar anlamsız işlemler tanımlayabilir
 - Operatörler anlamlı bile olsa, okunabilirlik zarar görebilir

5.5. İfadeler

49

- ❑ İfadeler bir programlama dilinde hesaplamaları belirtmede temel araçtır.
- ❑ İfadelerin değerlendirmesini anlamak için, operatörlerin sırası ve işlenenlerin (operant) değerlendirmesine aşına olmamız gerekir.
- ❑ Emirsel dillerin temeli atama ifadeleridir.

5.5. İfadeler

50

- **İfadeler**, yeni değerler oluşturmak için değerleri ve işlemcileri birleştirmeye yarayan sözdizimsel yapılardır.
 - Bir ifade; bir sabit, bir değişken, parantezler, bir değer döndüren bir fonksiyon çağırımı veya bir işlemciden oluşabilir. Bir dilin tip sistemi, bir tipi dildeki ifadelerle ilişkilendirmek için bir dizi kural sunar. Bu kurallar, dildeki her işlemcinin doğru kullanımının belirlenmesini sağlar.
 - Sayısal ifadeler, programlarda atama deyimlerinde, altprogram parametrelerinde vb. birçok deyimde, mantıksal ifadeler ve ilişkisel ifadeler ise özellikle seçimli deyimlerde olmak üzere birçok deyimde yer alabilir.

5.5. İfadeler

51

Sayısal İfadeler

Bir ifade değerlendirilince bir değer oluşturur. Her ifadenin bir tipi olmalıdır. Örneğin "1+2" nin tipi integer, dolayısıyla sayısal olmaktadır.

Örnek: $(a + b) * 5$

İlişkisel İfadeler

Bir ilişkisel ifadenin iki işleneni ve bir ilişkisel işlemcisi vardır.

Örnek: `sayac >= 10`

Mantıksal İfadeler

Mantıksal ifadeler, mantıksal değişkenler, mantıksal sabitler, ilişkisel ifadeler ve mantıksal işlemcilerden oluşurlar.

Örnek: `sayac >= 25 OR toplam > 150`

5.5. İfadeler

52

- Aritmetik İfadeler için Tasarım Sorunları
 - ▣ İşlemcilerin (Operatörlerin) öncelik kuralları?
 - ▣ İşlemcilerin (Operatörlerin) birleşmelilik kuralları?
 - ▣ Operantların sırasının değerlendirilmesi?
 - ▣ Operant değerlendirmenin yan etkileri?
 - ▣ İşlemci (Operatör) yükleme?
 - ▣ İfadelerdeki tip karıştırılması?

5.5. İfadeler

53

□ Ruby'de ifadeler

- ▣ Tüm aritmetik, ilişkisel, atama operatörleri, ve hatta dizi indeksleme, kaydırma ve bit şeklindeki mantık operatörleri metotlar olarak sağlanır
 - Bunun sonuçlarından biri bu operatörlerin uygulama programları tarafından geçersiz kılınabilmesidir.

□ Scheme (ve Common LISP) de ifadeler

- Tüm aritmetik ve mantık işlemleri belirgin bir şekilde alt programlar tarafından çağrılır.
- $a + b * c$ ifadesi `(+ a (* b c))` olarak kodlanır.

5.5. İfadeler

54

- Şartlı İfadeler
 - ▣ C-tabanlı diller (ör., C, C++)
 - ▣ Bir örnek:

```
average = (count == 0)? 0 : sum / count
```

- ▣ Aşağıdakine eş değerdir:

```
if (count == 0)
    average = 0
else
    average = sum /count
```

5.5. İfadeler

55

- ***operant değerlendirme sırası***
 1. Değişkenler: Bellekten değerini al
 2. Sabitler: bazen bellekten alınır bazen de makine kodunun içine konulmuş olur ve kendiliğinden gelir.
 3. Parantezli ifadeler: İçindeki tüm operant ve operatörler ilk olarak işlenir
 4. Fonksiyonlara referanslar: En ilginç durum bir operantın bir fonksiyon çağrısı yapması durumudur (İşlenme sırası çok önemli) (yan etkilerinden dolayı önemli)

5.5. İfadeler

56

- **Fonksiyonel Yan Etkiler**
- Bir fonksiyon iki yönlü bir parametreyi veya lokal olmayan bir değişkeni değiştirdiğinde meydana gelir.
- Örnek:
 - Bir ifadede çağrılmış bir fonksiyon ifadenin başka bir operantını değiştirdiğinde ortaya çıkar; bir parametre değişim örneği:

```
int fun(int *u)
{
    *u = *u/2;
    return *u; }

a = 10;
b = a + fun(&a); /* fun, parametresini değiştiriyor */
```


- ❑ Global değişkenlerle de aynı sorun.
- ❑ Fonksiyonların birden çok değer dönmeleri gerektiğinden, fonksiyonları parametrelerini değiştiremez veya global değişkenleri değiştiremez yapmak pratik değil.
- ❑ İşlenenlerin işlenmesi sırasını belirlemek de derleyicilerin optimizasyonlarını bozacağından pratik değil.
- ❑ Bununla birlikte, Java'da işlenenler soldan sağa işlenirler ki, burada bahsettiğimiz sorunla karşılaşılmaz

5.5. İfadeler

58

- **Fonksiyonel Yan Etkiler Problemi için 2 muhtemel çözüm:**
 1. Fonksiyonel yan etkileri iptal etmek için dil tanımlaması yapılır
 - Fonksiyonlarda 2 yönlü parametre olmayacak
 - Fonksiyonlarda global değişken olmayacak
 - **Avantajı:** Çalışması
 - **Dezavantajı:** Tek yönlü parametrelerin kararlılığı ve global değişkenlerin olmayışı (fonksiyonların birden çok değer döndürmeleri ihtiyacından dolayı pratik değil)
 2. Operantların işlem sırasını belirlemek için dil tanımlaması yapılır
 - **Dezavantajı :** Bazı derleyicilerin optimizasyonunu sınırlar
 - Java operantların soldan sağa işlenmesine izin verdiğinden bu problem oluşmaz.

5.5. İfadeler

59

- **İfadelerdeki Hatalar**
- Sebepler
 - ▣ Aritmetiğin doğal sınırları örn: sıfıra bölme
 - ▣ Bilgisayar aritmetik sınırları örn: taşma (overflow)
- Çalışma zamanında sık sık ihmal edilir.

5.5. İfadeler

60

□ İlişkisel İfadeler

- ▣ İlişkisel operatörler ve çeşitli tipteki operantların kullanımı
- ▣ Bazı mantıksal işaretlerin ölçümü
- ▣ Operatör sembolleri dillere göre değişiklik gösterir. (`!=`, `/=`, `~=`, `.NE.`, `<>`, `#`)

□ JavaScript ve PHP 2 ek ilişkisel operatöre sahiptir, `===` and `!==`

- Operantlarını zorlamamaları dışında kuzenlerine benzer, `==` ve `!=`,
- Ruby eşitlik ilişki operatörü için `==` kullanır

5.5. İfadeler

61

□ Mantıksal İfadeler

- Hem operantlar hem de sonuçlar mantıksaldır
- C mantıksal tipe sahip değil ve bunun için int tipini kullanır. (0 → yanlış, değilse doğru)
- C ifadelerinin tuhaf bir özelliği:
- $a < b < c$ doğru bir ifade, ama sonuç umduğumuz şeyi vermeyebilir:
 - Soldaki operatörler işlendiğinde, 0 veya 1 üretir
 - Ölçülen sonuç o zaman 3. operant ile karşılaştırılır (ör:, c)

5.5. İfadeler

$(13 * a) * (b / 13 - 1)$ $(a \geq 0) \ \&\& \ (b < 10)$	$\text{if } (i < \text{len} \ \&\& \ a[i] \neq 0)$ $\quad a[i] *= 2;$
---	--

- Kısa Devre Tespiti
- Bir ifadede operant/operatörlerin tüm hesaplamalarını yapmaksızın sonucun bulunmasıdır.
- Örnek: $(13 * a) * (b / 13 - 1)$
Eğer $a == 0$ ise, diğer kısmı hesaplamaya gerek yok ($b / 13 - 1$)
- Kısa Devre Olmayan Problem

```
index = 0;  
while (index <= length) && (LIST[index] != value)  
    index++;
```

 - $\text{index} = \text{length}$ olduğunda, $\text{LIST}[\text{index}]$ indeksleme problemi ortaya çıkaracak (LIST dizisi $\text{length} - 1$ uzunluğunda varsayılmış)

5.5. İfadeler

63

- ❑ Kısa Devre Tespiti(devam)
- ❑ C, C++, ve Java: kısa devre tespitinin bütün mantıksal operatörler (&& ve | |) için yapar, ama bit düzeyinde mantıksal operatörler (& ve |) için yapmaz.
- ❑ Ruby, Perl, ML, F#, ve Python'da tüm mantık operatörleri için kısa devre tespiti yapılır.
- ❑ Ada: Programcının isteğine bağlıdır (kısa devre **'and then'** ve **'or else'** ile belirtilir)
- ❑ Kısa devre tespiti ifadelerdeki potansiyel yan etki problemini ortaya çıkarabilir
örnek. $(a > b) \quad || \quad (b++ \ / \ 3)$
- ❑ $a > b$ olduğu sürece b artmayacak

5.5. İfadeler

64

□ Atama İfadeleri

□ Genel Sentaks

`<target_var> <assign_operator> <expression>`

□ Atama operatörü

`=` Fortran, BASIC, C-tabanlı diller

`:=` Ada

- `=` eşitlik için ilişkisel operatörler aşırı yüklendiğinde kötü olabilir (o zaman C-tabanlı diller ilişkisel operatör olarak neden `'=='` kullanır?)

5.5. İfadeler

65

□ Atama İfadeleri: Şartlı Amaçlar (Perl)

```
($flag ? $total : $subtotal) = 0
```

Hangisi eşittir

```
if ($flag) {  
    $total = 0  
} else {  
    $subtotal = 0  
}
```

5.5. İfadeler

61

- **Atama İfadeleri: Birleşik Atama Operatörleri**
- Atama formu için yaygın olarak bir stenografi metodu belirtmek gerekir
- ALGOL'de tanımlı; C ve C-tabanlı diller de benimsemiş.
 - ▣ Örnek:

$a = a + b$

Aşağıdaki gibi yazılabilir.

$a += b$

5.5. İfadeler

67

- **Atama İfadeleri : Tekli Atama Operatörleri**
- C-tabanlı dillerdeki tekli atama operatörleri atama ile artış ve azalış işlemlerini birleştirir
- Örnekler

`sum = ++count` (count arttırıldı, daha sonra sum' a **aktarıldı**)

`sum = count++` (count sum' a **aktarıldı**, ondan sonra arttırıldı)

`count++` (count arttırıldı)

`-count++` (count arttırıldı ondan sonra negatifi alındı)

5.5. İfadeler

68

- **Bir İfade olarak Atama**
- C-tabanlı diller, Perl, ve JavaScript'te atama durumu bir sonuç üretir ve bir operant olarak kullanılabilir,

```
while ((ch = getchar()) != EOF) {...}
```

`ch = getchar()` başarılı; sonuç (`ch` a aktar) **while** döngüsü için şartsal bir değer olarak kullanılır

- Dezavantaj: Başka tip yan etkiler.

5.5. İfadeler

64

❑ Çoklu Atamalar

- ❑ Perl, Ruby, ve Lua çok hedefli ve çok kaynaklı atamalara izin verir

```
($first, $second, $third) = (20, 30, 40);
```

Hatta, aşağıdaki geçerlidir ve bir yer değiştirme uygulanır:

```
($first, $second) = ($second, $first);
```

5.5. İfadeler

70

- **Fonksiyonel Dillerde Atama**
- Fonksiyonel dillerde tanıttıcılar (identifier) sadece değer adlarıdır.
- ML
 - ▣ İsimler `val` ve değer ile sınırlıdır İsimler
 - `val fruit = apples + oranges;`
 - Eğer fruit için başka bir val izlenecekse, o yeni ve farklı bir isimde olmalıdır.
- F#
 - ▣ F#' yeni bir skop (scope) yaratmanın dışında ML 'in `val` ile aynıdır .

5.5. İfadeler

71

- **Karışık Biçim Ataması**
- Atama ifadeleri karışık biçimde olabilir
- Fortran, C, Perl, ve C++'ta, her tip sayısal değer her tip sayısal değişkene atanabilir
- Java ve C#'ta, sadece genişletici atama zorlaması yapılır
- Ada'da, atama zorlaması yoktur.

5.5. İfadeler

1-72

Atama deyimi

<hedef_değişken> <atama_işlemcisi> <ifade>

Sum=++count

Count=count+1
Sum=count

Sum, total=0

Sum=0 ve total=0

puan+=500;

Puan=Puan+500

Sum=count ++

Sum=count
Count=count+1

count ++

count+1

F ? count1:count2=0

F=1 ise count1=0
F=0 ise count2=0

5.6. Deyimler

73

- **Deyimler**, bir programdaki işlemleri göstermek ve akışı yönlendirmek için kullanılan yapılardır.
 - Deyimler **basit** veya **birleşik** olabilirler.
 - Basit deyimlere örnek olarak atama deyimi verilebilir.
 - Birleşik deyimler ise, bir dizi deyimin tek bir deyime soyutlanmasını sağlarlar. Birleşik bir deyimde yer alan deyimleri belirlemek için basit deyimlerden ayrı bir sözdizime gereksinim vardır. Örneğin Pascal'da, birleşik deyimler *begin* ve *end* anahtar kelimeleri arasında, C de “{ }” parantezleri arasında gruplanır.
 - Programlarda akışı yönlendirmek için seçimli deyimler (*if-then-else* ve *case* deyimleri gibi) ve yinelemeli deyimler (*while* ve *for* deyimleri gibi) kullanılabilir.

5.6. Deyimler

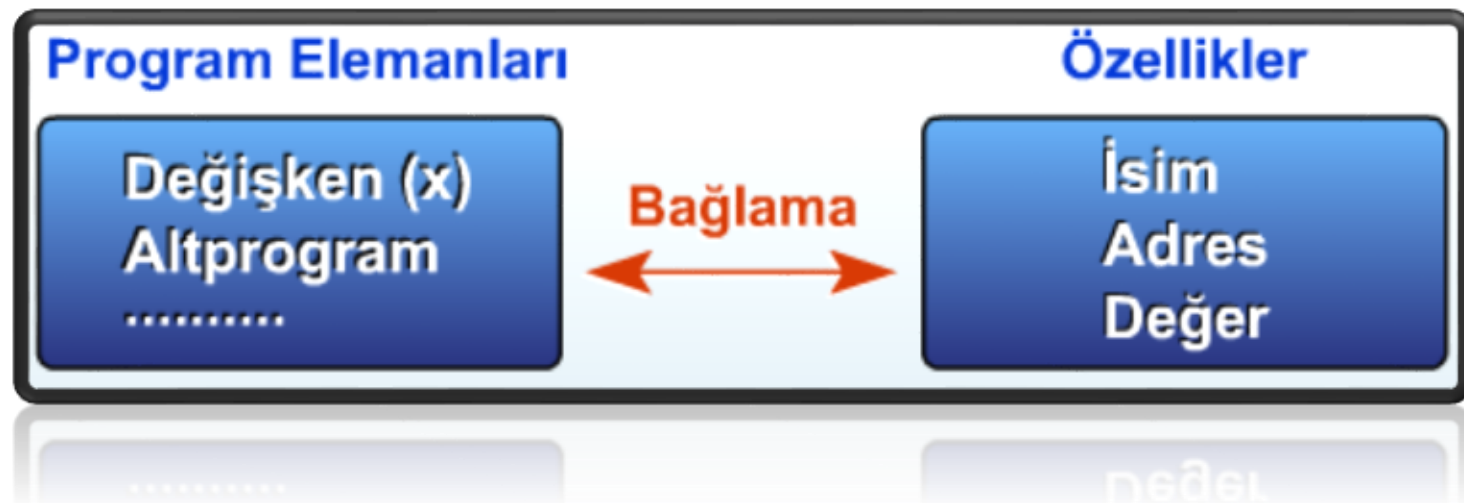
74

- **Altprogramlar**, bir dizi deyimin gruplanmasını ve bir isim ile gösterilmesini sağlarlar.
- Altprogramlar, bir başlık, yerel tanımlamalar bölümü ve işlemlerin yer aldığı bir gövde ile tanımlanır.
 - Altprogram başlığı, altprogram ismini ve varsa tipleriyle birlikte altprogramın parametrelerini belirtir.
 - Altprogram gövdesi, altprogram etkin olduğunda çalıştırılacak deyimlerden oluşur. Altprogramlar ve parametre aktarımları daha sonra incelenecektir.

5.7. Bağlama (Binding) kavramı

75

- Bir program elemanı ile bir özellik arasında ilişki kurulmasına **bağlama** (*binding*) denir.
- Çeşitli programlama dillerinde, özelliklerin program elemanlarına **bağlanma zamanı** ve bu özelliklerin **durağan** (*static*) veya **dinamik** (*dynamic*) olması açısından farklılıklar göstermektedir.

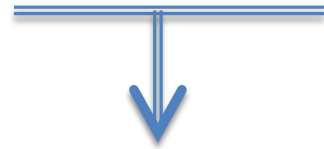


5.7. Bağlama (Binding) kavramı

76

- Bir değişkenin *l-değeri* onun adresidir
- Bir değişkenin *r-değeri* onun değeridir
- Tanım: Bağlama bir ilişkilendirmedir, bir özellik ve bir varlık arasında, veya bir işlem ve bir sembol arasındaki gibi.

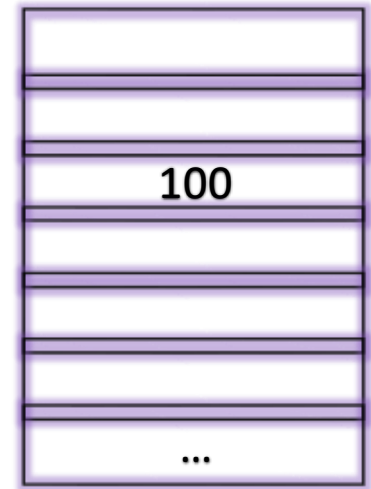
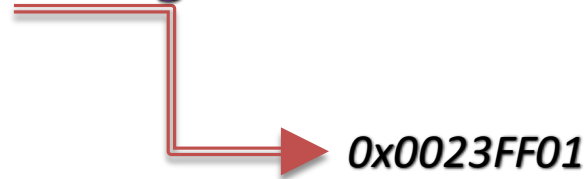
```
int counter = 100;
```



Değişken

- *İsim (Identifier): "counter"*
- *Tip: int*
- *Değer (R-değeri): 100*
- *Adres (L-değeri): 0x0023FF01*

Bağlama



Hafıza

5.7.1. Bağlama Zamanı

78

- Bir programlama dilinde çeşitli bağlamalar farklı zamanlarda gerçekleşebilir.
- **Durağan (Statik) Bağlama:**
 1. **Dil Tasarım Zamanında Bağlama:** Bazı özellikler dil tasarımı sırasında bağlanır. Örneğin, " * " sembolünün çarpım sembolü olması dil tasarım sırasında belirlenen bir özelliktir.
 2. **Dil Gerçekleştirim (implemantation) Zamanında Bağlama:** Dil gerçekleştirim zamanı, bir programlama dili için derleyicinin hazırlandığı zamandır. Örneğin, bir çok programlama dilinde " integer " tipinin alabileceği değerler dil gerçekleştirim sırasında belirlenir.

5.7.1. Bağlama Zamanı

79

□ Durağan (Statik) Bağlama:

3. **Derleme Zamanında Bağlama:** Derleme zamanı, bir programın makine koduna çevrildiği zamanı göstermektedir. Bir programın derlenmesi sırasında kaynak koda göre çeşitli bağlamalar gerçekleşebilir. Örneğin, C veya Pascal'da bir değişkenin, bir veri tipi ile bağlanması derleme zamanında yapılır.

□ Dinamik Bağlama:

4. **Çalışma Zamanında Bağlama:** Çalışma zamanı, bir programın çalışması sırasında geçen zamanı göstermektedir. Örneğin, değişkenlerin değerler ile bağlanması çalışma zamanında gerçekleşir ve bu bağlama, çalışma süresince bir çok kez değişebilir.

5.7.1. Bağlama Zamanı

80

- İlk üç grupta olduğu gibi bağlamanın çalışma zamanından önce gerçekleştiği durumda bağlama, çalışma zamanında değiştirilemez ve durağan (static) bağlama olarak adlandırılır.
- Çalışma zamanında gerçekleşen bağlamalar ise, çalışma süresince değiştirilebilirler ve dinamik (dynamic) bağlama olarak adlandırılırlar.
- Aşağıda, C'deki bir atama deyimi için gerçekleşen bağlamalar ve bağlama zamanları örneklenmiştir.

5.7.1. Bağlama Zamanı

81

Örnek

```
int puan;  
.....  
puan = puan + 3;
```

Dil Tasarım Zamanında Bağlama	Dil Gerçekleştirim Zamanında Bağlama	Derleme Zamanında Bağlama	Çalışma Zamanında Bağlama
<ul style="list-style-type: none">• puan değişkeninin alabileceği tipler• + sembolünün alabileceği anlamlar	<ul style="list-style-type: none">• puan değişkeninin alabileceği değerler	<ul style="list-style-type: none">• puan değişkeninin programdaki tipi• + sembolünün anlamı	<ul style="list-style-type: none">• puan değişkeninin o deyimdeki değeri

5.7.1. Bağlama Zamanı

82

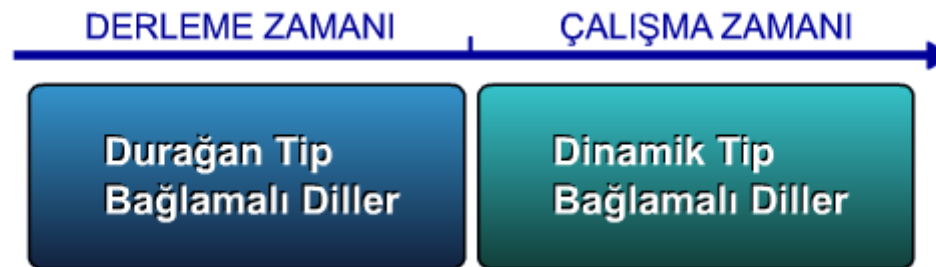


int hesap; ... hesap=hesap+10;	
Hesap için olası tipler	Dilin tasarım zamanında
Hesap değişkeninin tipi	Dilin derlenmesi zamanında
Hesap değişkeninin olası değerleri	Derleyici tasarım zamanı
Hesabın değeri	Bu deyimin yürütülmesi zamanında
+ işlemcisinin muhtemel anlamları	Dilin tanımlanması zamanında
+ işlemcisinin bu deyimdeki anlamı	Derlenme süreci
10 literalinin ara gösterimi	Derleyici tasarımı zamanında
Hesap değişkeninin alacağı son değer	Çalışma zamanında

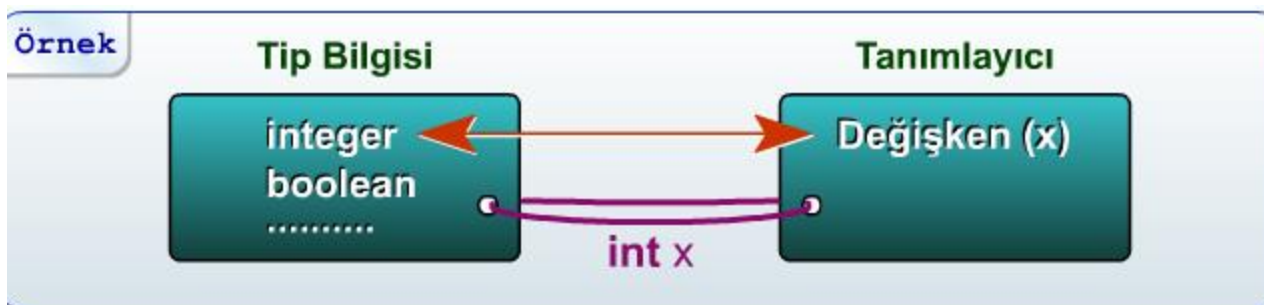
5.7.2. Tip Bağlama

83

- Tip bilgisi, bir tanımlayıcı ile ilişkilendirilince, tanımlayıcı, o tipe bağlanmış olur. Örneğin, birçok programlama dilinde bir programda bir değişkene başvuru yapılmadan önce, değişkenin bir veri tipi ile bağlanması gereklidir.



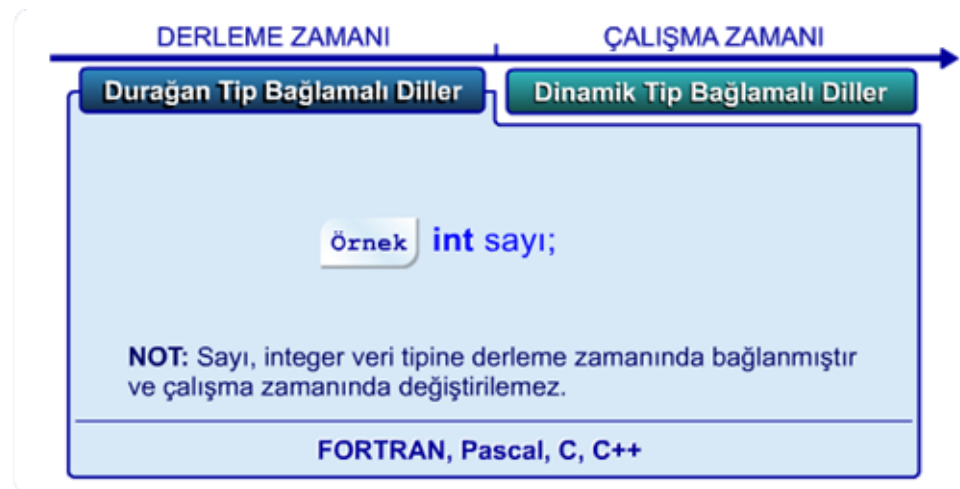
- Tip bağlamaları durağan veya dinamik olarak gerçekleşebilir.



5.7.2.1. Durağan Tip Bağlama

84

- Tiplerin isimlerle derleme zamanında bağlandığı diller, **durağan tip bağlamalı diller** olarak nitelendirilirler.
- Durağan tip bağlamalı dillerde bir değişken, *integer* tipi ile bağlanmışsa, değişkenin gösterdiği değer çalışma zamanında değişse de, gösterdiği değerın tipi her zaman *integer* olmalıdır. Örneğin FORTRAN, Pascal, C ve C++'da bir değişkenin tip bağlaması durağan olarak gerçekleşir ve çalışma süresince değiştirilemez.



5.7.2.1. Durağan Tip Bağlama

85

- Durağan tip bağlamalı dillerde derleyici, tip hatalarını, program çalıştırılmadan önce yakalar.
- Durağan tip bağlamaları, **örtülü** (*implicit*) ve **dışsal (açık)** (*explicit*) olmak üzere iki tür tanımlama ile gerçekleştirilebilir.

5.7.2.1. Durağan Tip Bağlama

86

- **Örtülü tanımlama**
- Örtülü tanımlamada, tanımlama deyimleri kullanılmaz ve değişkenlerin tipleri, varsayılan (*default*) kurallar ile belirlenir. Bu durumda bir değişken isminin programda ilk kullanıldığı deyim ile değişkenin örtülü tip bağlaması oluşturulur.

Örnek

PL/I'da dışsal olarak tanımlanmadığı durumda, isimleri "I" ve "N" arasındaki harflerle başlayan değişkenler tamsayı tipi ile bağlanır.

BASIC'te son karakteri "\$" olan tanımlayıcılar, karakter tipi ile bağlanırlar.

5.7.2.1. Durağan Tip Bağlama

87

- Örtülü tanımlamalar, program geliştirme sırasında yazılabilirlik alanında programcıya yardımcı olsalar da, yazım yanlışlığı gibi hataların derleme sırasında yakalanmasını engelledikleri için ve programcının tanımlamayı unuttuğu değişkenlere varsayılan olarak tip bağlanması ile programda fark edilmesi güç hatalara yol açabildikleri için, programlama dilinin güvenilirliğini azaltırlar.
- Ancak örtülü ve dışsal tanımlama ile tip bağlama, anlamsal açıdan aynıdır.
- PL/I, BASIC, FORTRAN ve PERL gibi dillerde örtülü tanımlamalar bulunmasına karşın günümüzde çoğu programlama dili, değişkenlerin dışsal olarak tanımlanmasını gerektirmektedir.

5.7.2.1. Durağan Tip Bağlama

88

- ❑ **Dışsal (Açık) tanımlama**
- ❑ Dışsal tanımlamada, bir değişken, programda yer alan bir tanımlama deyimi ile belirli bir tip ile bağlanır.

Örnek

C'de aşağıdaki tanımlama deyimlerinde toplam değişkeni integer tipi ile, ortalama değişkeni ise float tipi ile bağlanmaktadır.

```
int toplam;
```

```
float ortalama;
```


5.7.2.2. Dinamik Tip Bağlama

89

- Bir programlama dilinde bir değişkenin tipi çalışma zamanında, değişkenin bağlandığı değer ile belirleniyorsa, dil, **dinamik tip bağlamalı** olarak nitelendirilir.
- Dinamik tip bağlamalı dillerde bir değişken, atama sembolünün sağ tarafında bulunan değer, değişkenin veya ifadenin tipine bağlanır ve değişkenin tipi, çalışma zamanında değişkenin yeni değerler alması ile değiştirilir.

5.7.2.2. Dinamik Tip Bağlama

90

DERLEME ZAMANI

ÇALIŞMA ZAMANI

Durağan Tip Bağlamalı Diller

Dinamik Tip Bağlamalı Diller

Örnek

```
notlar <--- 99.5, 76.2, 89.4  
notlar <--- 100
```

- İlk atama deyiminde notlar, 3 elemanlı gerçel sayıların bulunduğu tek boyutlu bir dizinli değişkendir.
- İkinci deyimde notlar, tamsayı bir değerden oluşan basit bir değişken haline gelmiştir.

APL, LISP, Smalltalk, JavaScript ve PHP

5.7.2.2. Dinamik Tip Bağlama

91

- Değişkenlere Dinamik olarak tip bağlanması, programlama açısından esneklik sağlar.
- Örneğin, Dinamik tip bağlamalı bir dilde bir sıralama programındaki değişkenlerin tipleri çalışma zamanında belirlenebileceği için, tek bir program, farklı tipteki değerlerin sıralanması amacıyla kullanılabilir.
- Halbuki, C veya Pascal gibi durağan tip bağlamalı programlama dillerinde, bir sıralama programı sadece tek bir veri tipi için yazılabilir ve bu veri tipi program geliştirilirken bilinmelidir.
- Dinamik tip bağlamalı diller, genellikle yorumlayıcı ile gerçekleştirilirler

5.7.2.2. Dinamik Tip Bağlama

92



Programlamada esneklik sağlamakla birlikte dinamik tip bağlama, önemli dezavantajlar da taşımaktadır. Bunlar aşağıda özetlenmiştir:

1. Dinamik tip bağlamalı dillerde, dillerin derleyicilerinin hata yakalama yeteneği, durağan tip bağlamalı bir dilin derleyicisine göre zayıftır.
2. Dinamik tip bağlama, programlara durağan olarak tip denetimi yapılmasını önler.

Bağlama Zamanı

Durağan Tip Bağlama	Dinamik Tip Bağlama
Derleme Zamanında	Bir değişkenin tipi çalışma zamanında, değişkenin bağlandığı değer ile belirleniyorsa
Bir değişken, <i>integer</i> tipi ile bağlanmışsa	Bir değişken, atama sembolünün sağ tarafında bulunan değer, değişkenin veya ifadenin tipine bağlanır ve değişkenin tipi, çalışma zamanında değişkenin yeni değerler alması ile değiştirilir. $A=1.5$ $A=14$ Avantaj: Esneklik (örneğin sıralama)
FORTRAN, Pascal, C ve C++'da bir değişkenin tip bağlaması durağan olarak gerçekleşir ve çalışma süresince değiştirilemez.	APL, LISP, SMALLTALK, SNOBOL4
Derleyici, tip hatalarını, program çalıştırılmadan önce yakalar.	Derleyicinin hata yakalama yeteneği zayıftır. Statik tip kontrolü yapılamaz. Yorumlayıcı kullanırlar

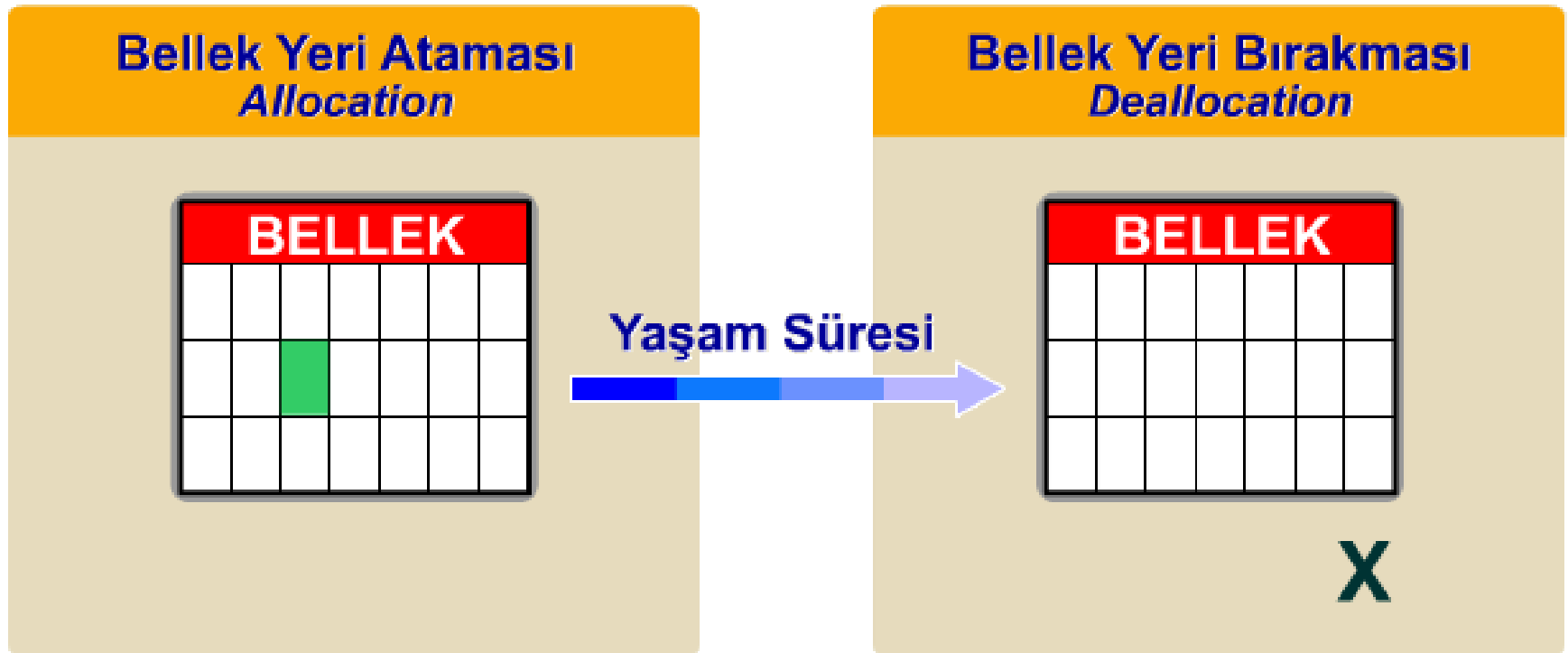
5.7.3. Bellek Bağlama

94

- Bir değişkene bağlanan bir bellek hücresi, kullanılabilir bellek hücreleri arasından seçilir ve bu işleme **bellek yeri ataması** (*allocation*) denir.
- Bir değişkenin kullandığı bellek hücrelerini geri vermesi ise **belleğin serbest bırakması** (*deallocation*) işlemi olarak nitelendirilir.
- Bir değişkenin **yaşam süresi**, o değişkenin belirli bir bellek yerine bağlı kaldığı süredir. Bu nedenle bir değişkenin yaşam süresi, bir bellek hücresi ile bağlandığı zaman başlar ve bellek hücreni bıraktığı zaman sona erer.

5.7.3. Bellek Bağlama

95

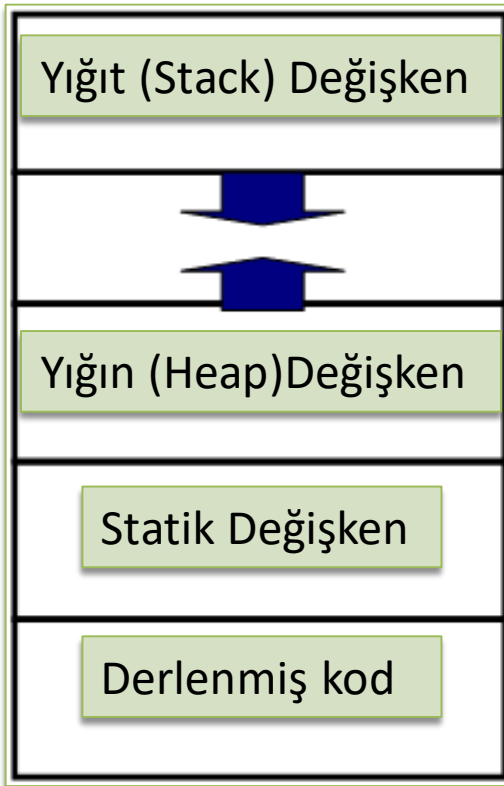


Pascal-*dispose*

Java-otomatik

C'deki malloc fonksiyonu

C++ 'daki new işlemcisi



Etkinlik (*activation*) kaydı

Aynı bellek bölümünün
yeniden kullanılabilmesi

Doğrudan adresleme

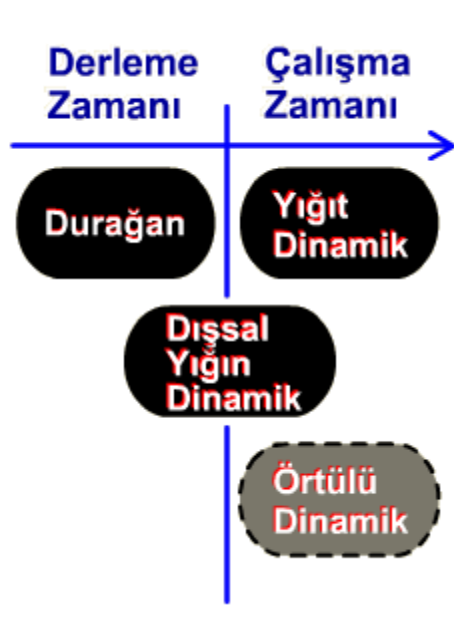
5.7.3.1. Bellek Düzeni

97

- Bellek yeri bağlamalarının anlaşılması için program çalışma zamanındaki bellek düzeninin incelenmesi yararlı olacaktır. Bir programın çalışması süresince bellek, çeşitli bölümlere ayrılmıştır.
 - ▣ **Derlenmiş Program:** Derlenmiş program kodlarının tutulduğu bölüm
 - ▣ **Genel Değişkenler:** Programdaki global değişkenleri içerir.
 - ▣ **Yığıt Bellek:** Program çalışırken etkin olan her alt program için etkinlik kaydı bu bölümde tutulur
 - ▣ **Yığın Bellek:** Dinamik bellek değerleri için kullanılır. Böylece aynı bellek bölümü, programda farklı zamanda yeniden kullanılabilir.

5.7.3. 2. Değişkenlerin Bellek Yeri Bağlamalarına Göre Sınıflanması

98



- Değişkenler, bellek yeri bağlamalarına ve yaşam sürelerine göre öncelikle **yığıt tabanlı değişkenler** ve **yığın tabanlı değişkenler** olarak iki gruba ayrılırlar.
- Buna ek olarak değişkenler, **durağan** (*static*) **değişkenler**, **yığıt dinamik** (*stack-dynamic*) **değişkenler**, **dışsal yığın** (*explicit heap*) **dinamik değişkenler** ve **örtülü** (*implicit*) **dinamik değişkenler** olarak dört grupta incelenebilir.

5.7.3. 2. Değişkenlerin Bellek Yeri Bağlamalarına Göre Sınıflanması

99

□ Durağan (static) Değişkenler :

- Durağan değişkenler, bellek hücrelerine programın çalışması başlamadan bağlanırlar ve programın çalışması bitinceye kadar o bellek hücrelerine bağlı kalırlar. Bu değişkenler için gerekli bellek çalışma zamanından önce ayrılır.
- Durağan değişkenler zaman açısından etkinlik sağlarlar. Ancak, bellek yerinin durağan olarak değişkenlere bağlanması, programlamada esnekliği azaltmaktadır. Sadece durağan değişkenlerin bulunduğu bir programlama dilinde, özyinelemeli (*recursive*) altprogramlar gibi programlama teknikleri kullanılamaz. C ve C++'da değişkenler, tanımlanma deyiminde static tanımlayıcısı kullanılarak, durağan değişkenler olarak tanımlanabilir.



5.7.3. 2. Değişkenlerin Bellek Yeri Bağlamalarına Göre Sınıflanması

100

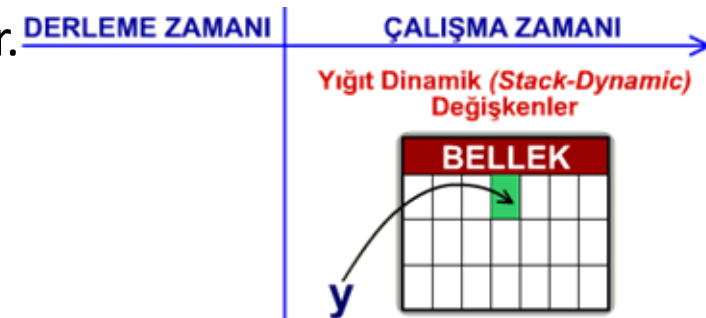
- **Durağan (*static*) Değişkenler (devam):**
 - ▣ Örnek: bütün FORTRAN 77 değişkenleri, C statik değişkenleri.
 - ▣ C++, C#, ve Java'da “class” içinde yapılan “static” tanımlamaları belleğin yaşam süresini değil, onun bir “class” değişkeni olduğunu, bir nesnenin anlık değişkeni olmadığını gösterir.
- **Avantajları:** verimli, hızlı (doğrudan adresleme), geçmişe duyarlı alt program desteği.
- **Dezavantaj:** esnek değil (özyineleme yok).

5.7.3. 2. Değişkenlerin Bellek Yeri Bağlamalarına Göre Sınıflanması

101

□ Yığıt Dinamik (*stack-dynamic*) Değişkenler:

- Yığıt dinamik değişkenler için bellek gereksinimi derleme zamanında hesaplanamaz ve bu tür değişkenler için bellek yeri, çalışma zamanında bellekteki yığıt bellekten ayrılır. Yığıt dinamik değişkenlerin bellek yeri bağlamaları kendilerine ilişkin tanımlama deyimleri çalıştığında gerçekleşir. Tanımlandığı blok aktif kaldığı sürece yaşar.
- Sayısal değişkenlerin bellek adresi hariç bütün özellikleri statik olarak belirlenmiştir.
- ALGOL 60 ve izleyen dillerde değişkenler, varsayılan olarak yığıt_dinamik değişkenlerdir. Örneğin; Pascal, C ve C++'da, tüm yerel değişkenler varsayılan olarak yığıt_dinamik değişkenlerdir.



NOT: Bellek yeri, çalışma zamanında bellekteki yığıt bellekten ayrılır.

5.7.3. 2. Değişkenlerin Bellek Yeri Bağlamalarına Göre Sınıflanması

102

- **Yığıt Dinamik (*stack-dynamic*) Değişkenler (devam):**
 - Örnek: C alt programlarının veya Java metotlarının lokal değişkenleri gibi.
 - Bazı dillerde her noktada yığıt dinamik tanımlama yapılabilir. Metotlar içinde tanımlanan Java, C++ ve C# değişkenlerinin hepsi yığıt dinamiktir. Aynı şekilde Ada'da alt-yordamlarda tanımlanan değişkenlerde (bellek yığını hariç) yığıt dinamiktir.
- **Avantaj:** Öz yinelemeye izin verir; depolamayı korur; az bellek harcanmasına neden olur.
- **Dezavantaj:**
 - Bellekten yer almak ve geri vermenin yarattığı işlem yükü.
 - Alt programlar geçmişe hassas değildir. Çıkılınca bütün bilgiler unutulur.
 - Verimsiz referanslar (Dolaylı adresleme)

5.7.3. 2. Değişkenlerin Bellek Yeri Bağlamalarına Göre Sınıflanması

103

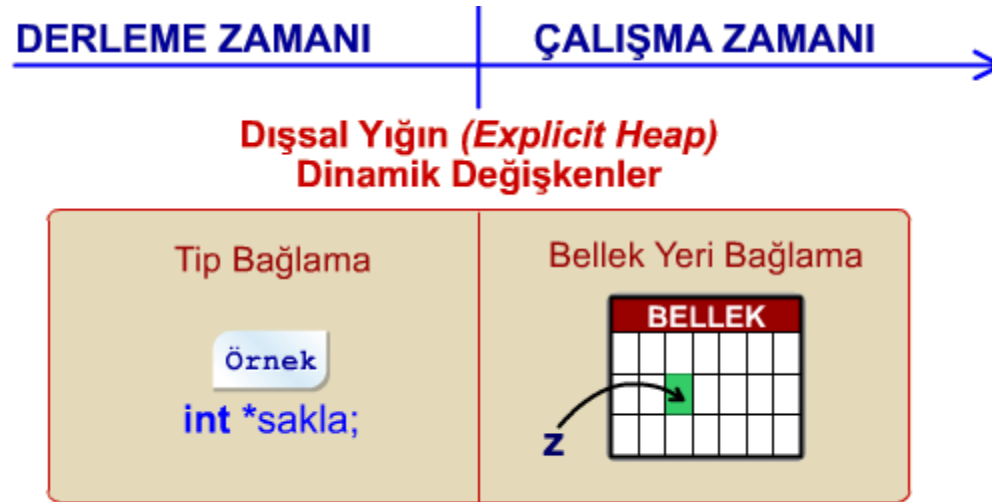
□ Dışsal (Açık) Yığın (*explicit heap*) Dinamik Değişkenler:

- Dışsal yığın dinamik değişkenler için bellek gereksinimi öngörülemez ve veriler çalışma zamanında gerek oldukça belleğe atanır.
- Dışsal yığın dinamik değişkenler için bellek yeri yığın bellekten alınır ve yığın belleğe geri verilir. Bu verilere sadece gösterge değişkenler aracılığıyla ulaşılabilir.
- Dışsal yığın dinamik değişkenlerin tip bağlaması derleme zamanında, bellek yeri bağlaması ise çalışma zamanında gerçekleşir.

5.7.3. 2. Değişkenlerin Bellek Yeri Bağlamalarına Göre Sınıflanması

104

□ Dışsal Yığın (*explicit heap*) Dinamik Değişkenler (devam):



NOT: Veriler çalışma zamanında gerek oldukça belleğe atanır.

- Bir dışsal yığın dinamik değişken, ya bir işlemci ile (örneğin; C++ 'daki new işlemcisi) ya da programlama dilinde bulunan bir altprogram ile (örneğin; C'deki malloc fonksiyonu) oluşturulur

5.7.3. 2. Değişkenlerin Bellek Yeri Bağlamalarına Göre Sınıflanması

105

□ Dışsal Yığın (*explicit heap*) Dinamik Değişkenler (devam):

int *sakla; —————→ sakla isimli gösterge değişken tanımlanıyor.

.....

sakla = new int; —————→ Sakla isimli dışsal yığın dinamik değişken yaratılıyor.
(**new** işlemcisi, tip ismini içermektedir.)

.....

delete sakla; —————→ değişken için ayrılan bellek geri veriliyor.

- Dışsal yığın dinamik değişkenler, bağlılı listeler ve ağaçlar gibi çalışma sırasında büyüyeabilen veya küçülebilen yapılar için uygun değişkenlerdir.

5.7.3. 2. Değişkenlerin Bellek Yeri Bağlamalarına Göre Sınıflanması

106

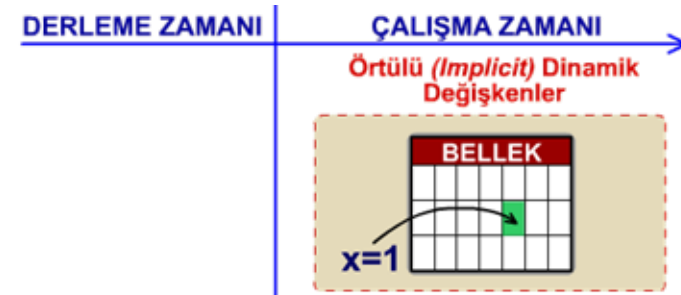
- **Dışsal Yığın (*explicit heap*) Dinamik Değişkenler (devam):**
- Örnek: C++ dinamik nesneleri (new ve delete ile):

```
int *intp; // gösterici tanımla  
...  
intp = new int; //yeni bellek ayır  
...  
delete intp; // yeni belleği sil (gerekli)
```
- Örnek: Bütün Java nesneleri: “delete” yok ancak kullanılmayan bellekleri toplayan örtülü bir çöp toplama yapısı var (implicit garbage collection).
- Örnek: C#’da da “delete” yok ancak örtülü olarak kullanılmayacaklar sisteme geri iade ediliyor.
- **Avantaj:** dinamik bellek yönetimi sağlar.
- **Dezavantaj:** yönetimi zor bu nedenle güvenilir değil.

5.7.3. 2. Değişkenlerin Bellek Yeri Bağlamalarına Göre Sınıflanması

107

- ❑ **Örtülü (*implicit*) Dinamik Değişkenler**
- ❑ Örtülü değişkenler, sadece bir değer aldıkları zaman yığın belleğe bağlanırlar. Bu tür değişkenlerin, tip ve bellek özellikleri her değer alışlarında yeniden belirlenebilir. Bu nedenle, esnek kod yazılmasını sağlarlar.
- ❑ Dezavantajları ise, tüm dinamik özelliklerin çalışma zamanında izlenmesi nedeniyle oluşan performans kaybı ve derleyicinin hata yakalama yeteneğinin azalmasıdır.
- ❑ Örtülü dinamik değişkenlerin yer aldığı bir dil APL'dir.



NOT: Örtülü değişkenler, sadece bir değer aldıkları zaman yığın belleğe bağlanırlar.

5.7.3. 2. Değişkenlerin Bellek Yeri Bağlamalarına Göre Sınıflanması

108

- ❑ **Örtülü (*implicit*) Dinamik Değişkenler**
- ❑ Örnek: Perl ve JavaScript'de bütün dizgi (string) ve dizilim (array) atamaları.
- ❑ Avantaj: Esneklik.
- ❑ Dezavantaj:
 - ▣ Yetersiz çünkü bütün özellikler dinamik.
 - ▣ Hata fark etme yetersizliği.

5.7.3. 2. Değişkenlerin Bellek Yeri Bağlamalarına Göre Sınıflanması

109

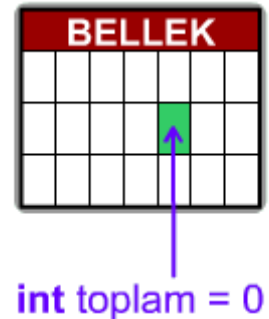


	<i>Statik</i>	<i>Stack</i>	<i>Heap</i>
Ada		Lokal değişkenler, altprogram parametreleri	<i>Implicit</i> : lokal değişkenler; <i>Explicit</i> : new (garbage collection)
C	Global değişkenler; statik lokal değişkenler	Lokal değişkenler, altprogram parametreleri	<i>Explicit</i> : malloc ve free
C++	C ile aynı, static sınıf üyeleri	C ile aynı	<i>Explicit</i> : new ve delete
Java		Sadece ilkel tipli lokal değişkenler	<i>Implicit</i> : her sınıf (garbage collection)
Fortran 77	Global değişkenler (bloklar), lokal değişkenler ve altprogram parametreleri (implementation dependent); SAVE , static bellek atamasını düzenler	Lokal değişkenler, altprogram parametreleri (implementation dependent)	
Pascal	Global değişkenler (derleyici bağımlı)	Global değişkenler (derleyici bağımlı), lokal değişkenler altprogram parametreleri	<i>Explicit</i> : new ve dispose

5.7.3.3. Değişken İkleme

111

- Bir değişkene bellek yeri bağlandığı zaman, bir değer verilirse bu işleme **değişken ilkleme** denir.
- Birçok programlama dilinde (Örneğin; PL/I, C) değişken ilkleme olasıdır.
- Pascal'da ise değişken ilkleme yapılamaz.



PL/I

```
Declare toplam Fixed Dec Init(0);
```

Bu tanımlama deyimi ile tanımlama ve ilkleme gerçekleştirilmektedir.

C

```
int toplam = 0;
```

Bu deyim ile toplam değişkeni tamsayı olarak tanımlanmakta ve ilk değer olarak sıfır değerini almaktadır.

5.8.İsim Kapsamları

112



- Belirli isim tanımlarının etkin olduğu bir program alanı bir **isim kapsamı** (*name scope*) oluşturur. İsim kapsamları ile komutların isimlere ulaşmaları kısıtlanır. Bir isim için kapsam, ismin tanımlandığı noktadan başlar ve o programlama dilinin isim kapsamı kurallarına bağlı olarak sonraki bir noktaya kadar devam eder.
- Bir isim kapsamında tanımlanmış bir isim, o isim kapsamı için **yerel (lokal)**, çevreleyen kapsamlardan alınmış isimler ise **yerel olmayan** olarak kabul edilir. Programdaki en dış bloğa ilişkin bir isim ise genel (global) özelliktedir.
- Programlama dillerinde isimlerin kapsam bağlaması **durağan** veya **dinamik** olarak gerçekleşebilir.

5.8. 1. Durağan Kapsam Bağlama

113

- ❑ ALGOL 60 isimleri, yerel olmayan değişkenlere bağlamak için **durağan kapsam bağlama** olarak adlandırılan bir yöntem tanıtmıştır.
- ❑ Bu yöntemde değişkenlerin kapsamları, programın metinsel düzenine göre belirlenir. Yani bir değişkene olan başvuru, programın çalıştırılması gerekmeden, program metninin incelenmesi ile belirli bir değişken tanımına bağlanabilir.
- ❑ Bir isim referansını değişkene bağlayabilmek için isim tanımlanmış olmalıdır.
- ❑ Arama işlemi: lokalden başlayarak ve her seferinde kapsamı genişleterek, verilen ismin tanımını arama. Bu durumda kapsam en içteki alt programdan onu çevreleyen üst alt programlara doğrudur.
- ❑ ALGOL 60'ı izleyen çok sayıda dil (Ada, JavaScript, PHP), durağan kapsam bağlama kurallarını uygulamıştır.
- ❑ Durağan kapsam bağlamayı uygulayan bazı diller iç içe alt programları desteklerken (Ada, JavaScript, PHP), bazı diller desteklemez (C tabanlı diller gibi).
- ❑ İç içe alt programları desteklemeyen dillerde bile blok içleri ayrı kapsama alanlarıdır.
- ❑ Statik kapsamı çevreleyen kapsam onun atasıdır. En yakın ataya, ebeveyn (parent) denir.

5.8. 1. Durağan Kapsam Bağlama

114

- **Durağan Kapsam Bağlamayı Uygulayan Dillerdeki Yapı:**
- Bu tür dillerdeki yapıyı aşağıdaki şekilde görülen örnek bir Pascal programı ile inceleyelim.

```
Procedure nothesap;  
  Var sayac: integer;  
  Procedure yazdır;  
    ① Begin  
      ...sayac....  
    End;  
  Procedure toplar;  
    ② Var sayac: integer;  
    Begin  
      .....  
    End;  
  ③ Begin  
    .....  
  end;
```

- ① Sayac için tanımlama ilk olarak yazdır altprogramında aranır.
- ② Eğer sayac değişkeninin tanımı bulunamazsa, arama, yazdır altprogramını metinsel olarak içeren altprogram içinde devam eder.
- ③ Eğer sayac için orada da bir tanımlama bulunamazsa, arama, hiyerarşiye göre bir üstteki altprogram içinde (örnek, nothesap) devam eder.
- ④ Eğer, hiyerarşinin en üstüne kadar bir tanımlama bulunamazsa tanımsız değişken hatası oluşur.

5.8. 1. Durağan Kapsam Bağlama

115

- Durağan kapsam bağlama kurallarına göre, *yazdır* altprogramı içinde *sayac* değişkene yapılan başvurular, *yazdır* altprogramında *sayac*'a ilişkin bir tanımlama bulunmadığı için, *nothesap* altprogramında tanımlanmış *sayac*'a yapılmaktadır.

5.8. 1. Durağan Kapsam Bağlama

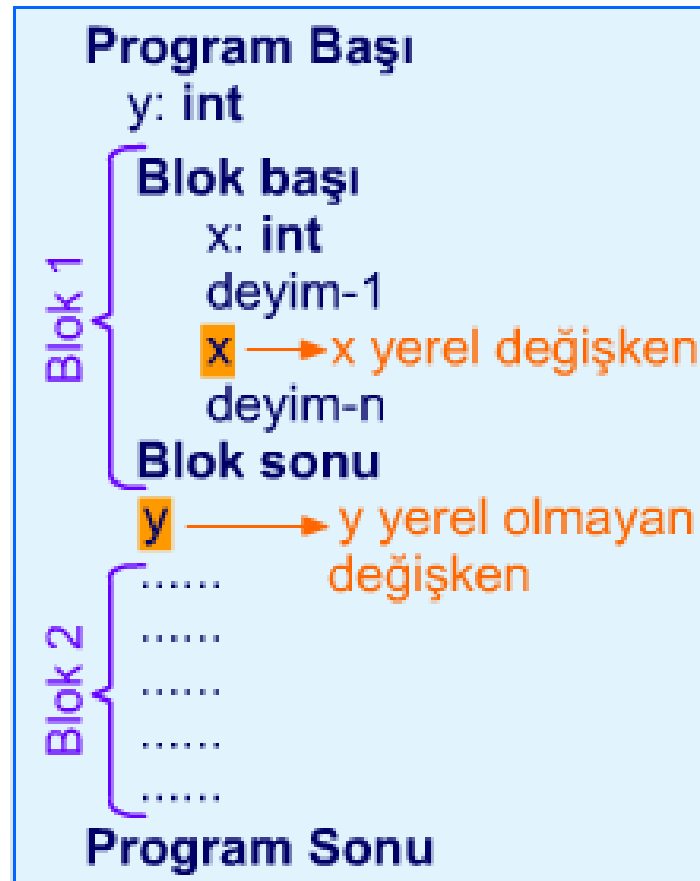
116

- ❑ **Blok Yapısı**
- ❑ Bir program birimi içinde tanımlanmış değişkenler, o birim için **yerel değişkenler** 'dir. O program birimi içinde görünür olan, ancak o birimde tanımlanmamış değişkenler ise **yerel olmayan değişkenler** olarak adlandırılmaktadır.
- ❑ Deyimlerin bir araya getirilmesi ile oluşturulan ve bu deyimlere özgü yerel değişkenlerin tanımlanabildiği kod bölümüne **blok** adı verilir.
- ❑ Geçici tanımlamalar yapmak için bir bloktan yararlanmak, programları yapılandırmak için uygun bir yaklaşımdır. Blok yapısında bir program, her biri yeni kapsamlar oluşturan yuvalanmış bloklara bölünmüştür.

5.8. 1. Durağan Kapsam Bağlama

117

□ Blok Yapısı



İç içe yuvalanmış statik kapsam örneği

118

```
class MyClass {
```

```
    int myVar1 = 100, myVar2 = 200;
```

```
    void myMethod1() {
```

```
        System.out.println(myVar1);
```

```
        {  
            int myVar1 = 200;  
            System.out.println(myVar1);  
            System.out.println(this.myVar1);  
        }
```

```
        System.out.println(myVar1);
```

```
    }
```

```
    class InnerClass {
```

```
        int myVar1 = 300;
```

```
        void myMethod2() {
```

```
            System.out.println(myVar1);
```

```
            System.out.println(myVar2);
```

```
            System.out.println(MyClass.this.myVar1);
```

```
        }
```

```
    }
```

```
}
```

myVar1 ve myVar2
kapsamı

Lokal değişken myVar1'i
tanımlamak için bir *blok*

Gizli değişken myVar1'e
ulaşma

Lokal değişken myVar1'in
kapsamı

myVar2 bu blokta lokal-
olmayan bir değişkendir

- InnerClass, myVar2 'nin *statik parent'idir*
- MyClass, myVar2'nin *statik dedesidir (ancestor)*

5.8. 1. Durağan Kapsam Bağlama

119

- ❑ **Blok Yapılı Diller**
- ❑ İsim kapsamları oluşturan altprogramların yuvalanabildiği programlama dilleri, **blok yapılı** olarak nitelendirilir.
- ❑ Pascal, altprogramların yuvalanmasına izin verdiği için blok yapılı bir dil olarak nitelenmesine karşın, altprogram olmayan bloklar, Pascal programlarında yer alamaz.

5.8. 1. Durağan Kapsam Bağlama

120

```
If (puan[i] < puan [j])  
{  
    int sakla;  
    sakla = puan[i];  
    puan[i] = puan [j];  
    puan[j] = sakla;  
}
```

- ❑ **C'de Blok Yapısı:**
- ❑ C'de isimsiz bloklar bulunabilir ama altprogramlar yuvalanamaz.
- ❑ C ve C++'da birleşik (*compound*) deyimler blok olarak nitelendirilir ve bu bölümlerde tanımlamalar yapılarak yeni kapsamlar tanımlanabilir. Birleşik deyimler " { }" arasındaki deyimlerdir.
- ❑ Yandaki şekilde C deyimleri, bir bloğu göstermektedir.

5.8.1. Durağan Kapsam Bağlama

121

- **C'de Blok Tanımı ile İsim Kapsamı Oluşturulması:**
- Bir blokta oluşturulan kapsamlar, altprogramlar tarafından oluşturulan kapsamlarla aynı özellikleri taşırlar.
- Aşağıdaki şekilde C kodunda, *say1* ve *say2* değişkenleri, tüm main programı boyunca görünür olacak şekilde tanımlanmışlardır. Ancak programda yer alan blokta *say1* ve *say2* için yeni tanımlamalar olursa, blokta tanımlanan *say1* ve *say2* değişkenleri blok içinde, main de tanımlanmış *say1* ve *say2*'yi gizler. Gizlenmiş değişkenler, blok içinde görünür olmasalar da, var olmaya devam ederler.

5.8.1. Durağan Kapsam Bağlama

122

```
include <stdio.h>
main()
{
    int say1,say2;
    scanf ("%d %d",&say1,&say2);
    if say1 < say2 {
        int gec;
        gec =say1;
        say1=say2;
        say2 =gec;
    }
    printf("%d %d", say1,say2);
}
```

Blok2

Blok1

Blok1, ana programda tanımlı değişkenleri gizleyen bir tanımlama yapmamakta, sadece blok içinde görünür olan "gec" değişkenini tanımlamaktadır.

- Bloklarda tanımlanmış değişkenler, yığıt dinamik değişkenler oldukları için bellek yerini sadece o blok çalışırken tutarlar.

5.8.1. Durağan Kapsam Bağlama

123

- Lokalde tanımlanmış aynı isimli değişken, dışarıda ata tarafından tanımlı değişkene erişimi keser:

```
void sub() {  
    int count;  
    ...  
    while ( ... ) {  
        int count=1;  
        count ++;  
        ...  
    }  
}
```

- C++ ve Ada bu tip erişilmez verilere kapsamı belirterek erişim imkanı sağlar.
- Ada'da: **unit.name**
- C++'da: **class_name::name**

5.8.1. Durağan Kapsam Bağlama

124

Statik kapsam, örnekler

- C++ değişken tanımlarının fonksiyon içinde herhangi bir yerde yapılmasına izin verir. Fonksiyonun içinde ama bir blok içinde olmayan tanımlar, fonksiyon içinde tanımlandığı noktadan fonksiyonun sonuna kadar tanımlanmış sayılırlar.
- C’de benzer tanımların fonksiyon başında yapılması zorunludur.
- C++, Java ve C# “class”ları içinde tanımlanan değişkenler farklılıklar gösterir:
 - Eğer herhangi bir metodun içinde tanımlanmadıysa, bütün class içinde tanımlıdır. “public”se dışarıdan da erişilebilir.
 - Bir metod içinde tanımlandıysa, tanımlandığı bloktaki değerini kullanır.
 - C#, C++ tipi göstericileri destekler. Ancak bunlar güvenliği bozduklarından bunları kullanan ‘metot’ların ‘unsafe’ olarak tanımlanması zorunludur

1. A,B :ornek; X: Sub1;
2. A,B :ornek; Y: Sub 2; X: Sub3;
3. A,B, :ornek; X,Y: Sub2;
4. A,B, :ornek;

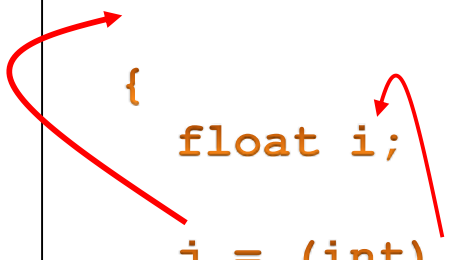
```

procedure ornek is
  A, B : Integer;
  ...
  procedure Sub1 is
    X: Integer;
    begin -- Sub1
    ... <===== 1
    end;
  procedure Sub2 is
    X,Y : Integer;
    ...
    procedure Sub3 is
      X : Integer;
      begin -- Sub3
      ... <===== 2
      end;
    begin -- Sub2
    ... <===== 3
    end;

  begin -- Example
  ... <===== 4
endl;

```

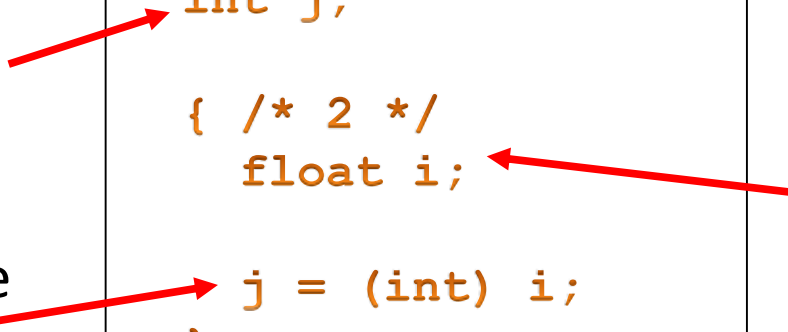
```
int i;  
  
{  
    int j;  
  
    {  
        float i;  
        j = (int) i;  
    }  
}
```



j scope 1'de
sınırlı

j scope 2'de
serbest

```
{ /* 1 */  
    int j;  
  
    { /* 2 */  
        float i;  
        j = (int) i;  
    }  
}
```



i scope 2'de
sınırlı

5.8.1. Durağan Kapsam Bağlama

127

- **Durağan Kapsam Bağlamadaki Sorunlar**
 - Blok yapısı, bir altprogramın ayrıştırılması için kolay ve etkin bir yoldur. Ancak durağan kapsam bağlamada, altprogramların yuvalanması sonucu gereğinden fazla genel değişken kullanımı olabilir.
 - Bir programda genel olarak tanımlanan değişkenler, gerekli olup olmamasına bakılmadan, tüm altprogramlara görünür olacakları için güvenilirliği azaltırlar.

5.8.1. Durağan Kapsam Bağlama

128

- **Durağan Kapsam Bağlama Değerlendirmesi**
- Örneğe bakalım:

Varsayalım ki

MAIN, A ve B yi çağırır

A, C ve D yi çağırır

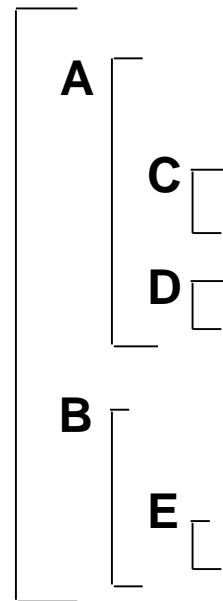
B, A ve E yi çağırır

5.8.1. Durağan Kapsam Bağlama

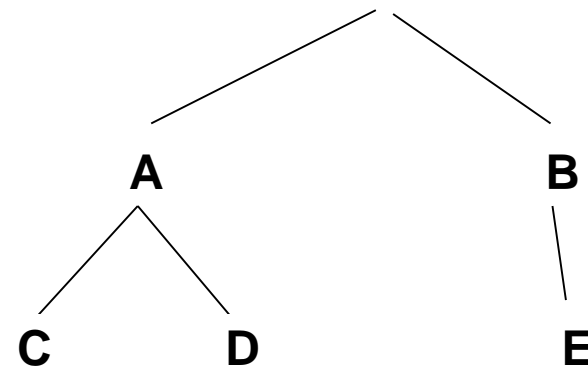
129

Durağan Kapsam Örneği

MAIN



MAIN

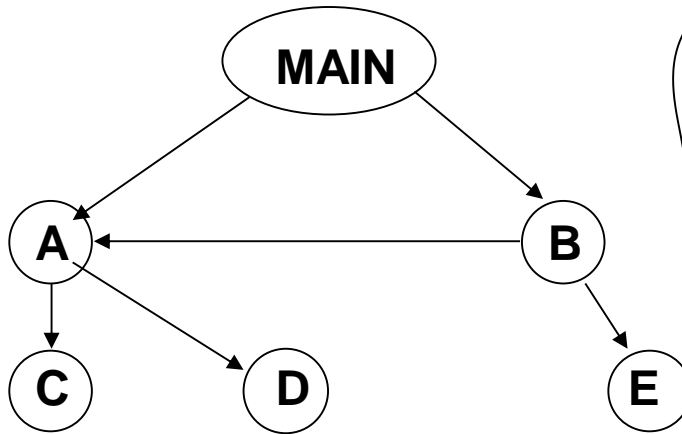


5.8.1. Durağan Kapsam Bağlama

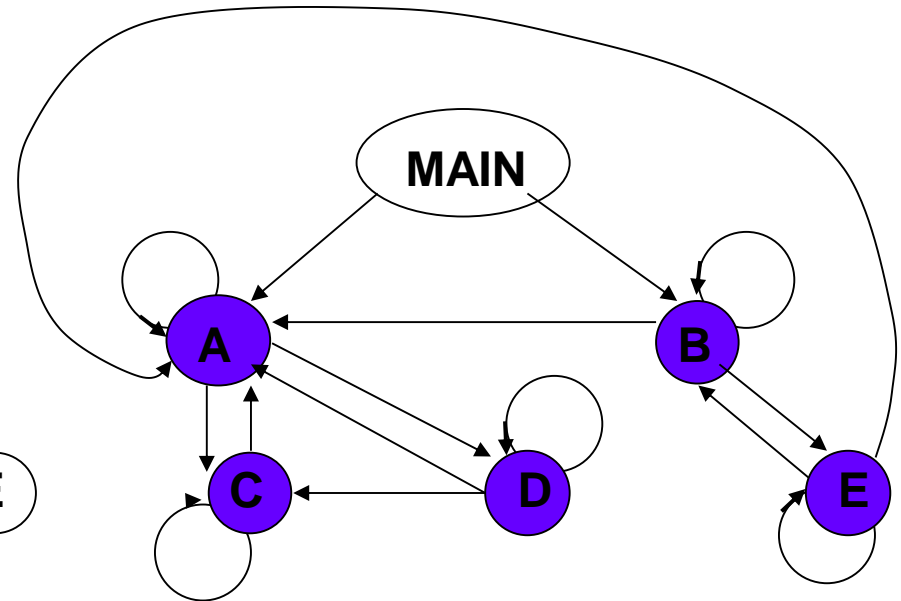
130

Durağan Kapsam Örneği

Programın istenen altprogram
çağırma yapısı



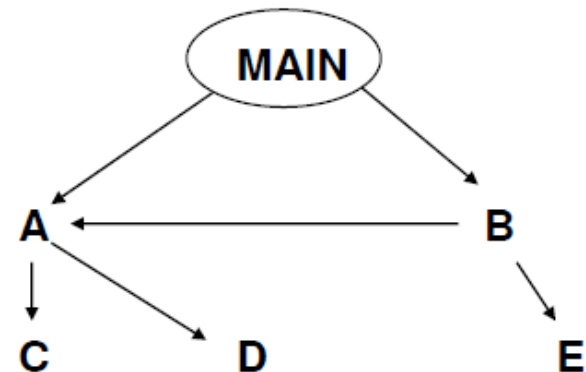
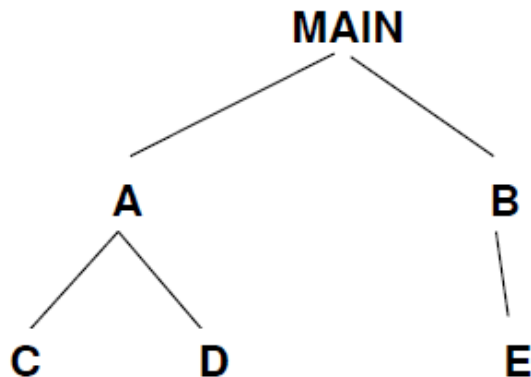
Programın potansiyel altprogram
çağırma yapısı



5.8.1. Durağan Kapsam Bağlama

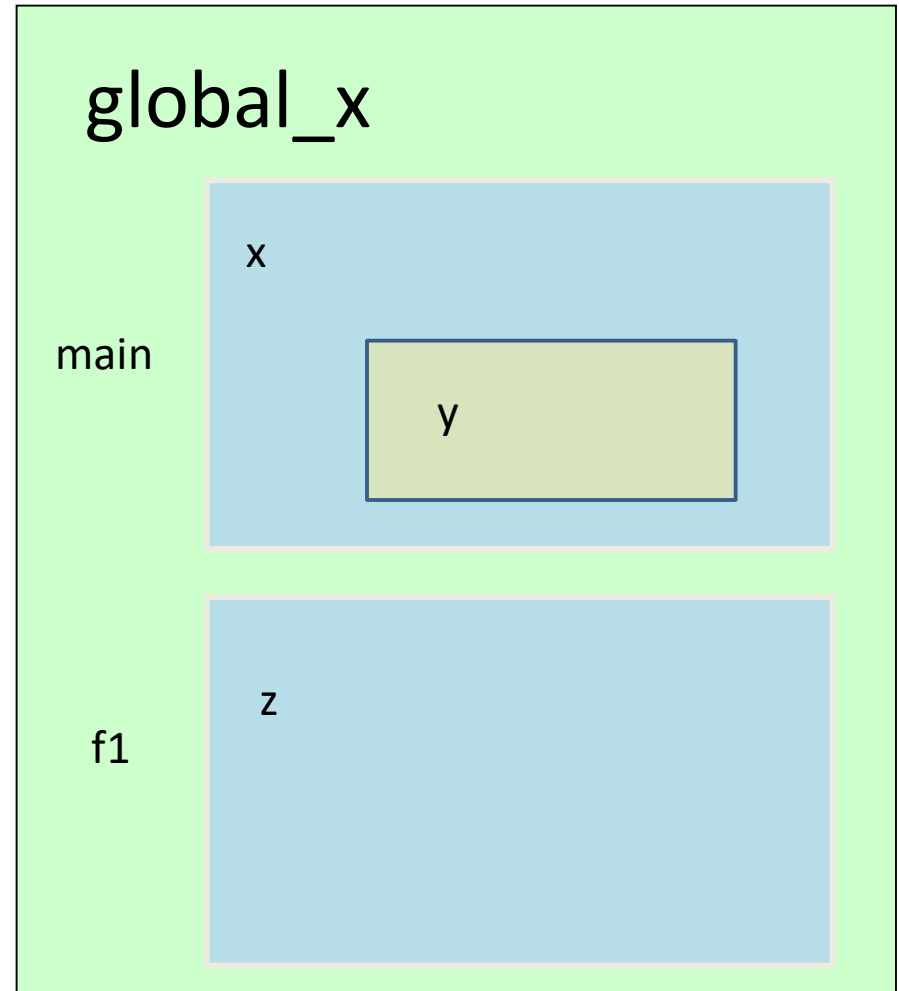
131

- Şartın değiştiğini varsayalım öyle ki D, B'deki bazı veriye erişmek zorunda olsun
- Çözümler:
 - D'yi B'nin içine koy (fakat o zaman C artık onu çağıramaz ve D, A'nın değişkenlerine erişemez)
 - D'nin ihtiyacı olan veriyi B'den MAIN'e taşı (fakat o zaman bütün prosedürler onlara erişebilir)
- Prosedür erişim için aynı problem
- Sonuçta: Durağan kapsam çoğunlukla birçok globale teşvik eder



Durağan Kapsam: C++

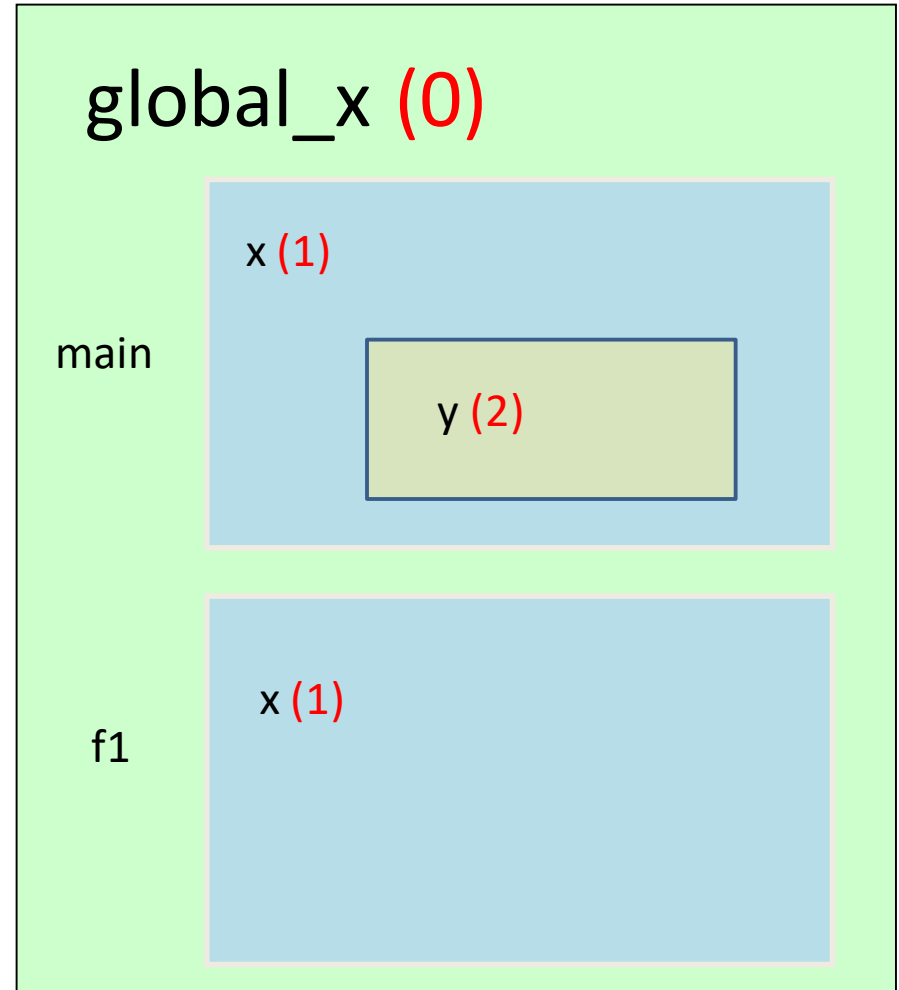
```
int global_x;  
  
int main()  
{   int x;  
    ...  
    {   int y;  
        ...  
    }  
}  
  
void f1()  
{   int z;  
    ...  
}
```



Durağan Kapsam: C++

Kapsamlar basitçe bir sayıyla işaretlenir.
Her { yeni bir kapsam getirir – daha yüksek
sayı

```
int global_X;  
  
int main()  
{   int x;  
    ....  
    {   int y;  
        ...  
    }  
  
}  
void f1()  
{   int z;  
...  
}
```



5.8.2. Dinamik Kapsam Bağlama

134

- Bir ismin kapsamının, altprogramların fiziksel yakınlıklarına göre değil, altprogramların çağrılma sırasına göre çalışma zamanında belirlenmesi **dinamik kapsam bağlama** olarak adlandırılır.
- Değişkenlere çağrılma sırasında ters yönde arama yapılarak başvurulur.
- Bu durumda bir isim tanımı, çalışma sırasında aynı isimde yeni bir tanımlama bulunana kadar, kendisinden sonra çalıştırılan tüm komutlarda geçerlidir.
- APL, SNOBOL4, Perl ve LISP'in ilk sürümleri, dinamik kapsam bağlamayı uygulayan dillerdir. Perl ve Common Lisp her iki kapsamı da kullanabilirler.

Dinamik Kapsam

135

Ör: **MAIN** - declaration of x
 SUB1
 - declaration of x -
 ...
 call SUB2
 ...
 SUB2
 ...
 - reference to x -
 ...
 ...
 call SUB1
 ...

MAIN, SUB1'i çağırır
SUB1, SUB2'yi çağırır
SUB2, x'i kullanır

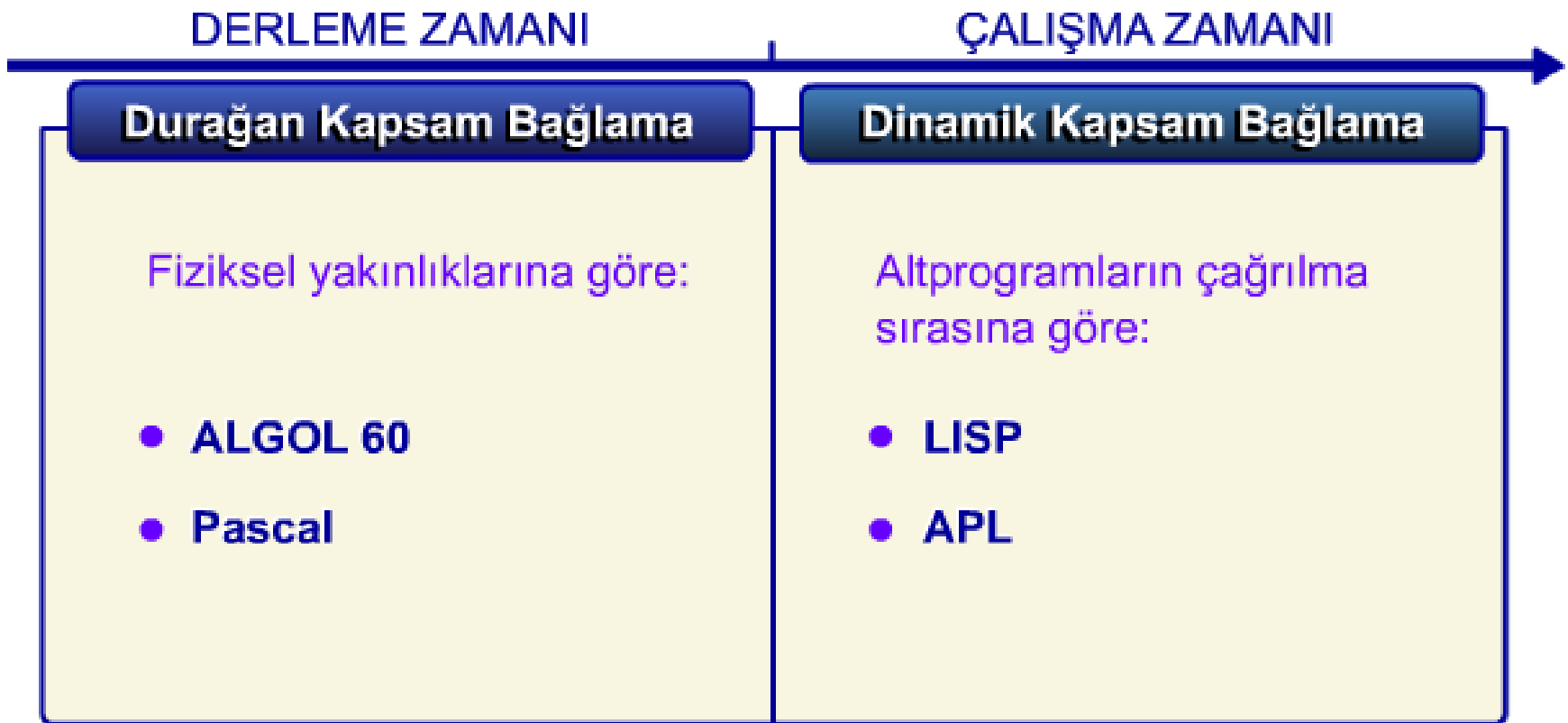


X'e referans SUB1'in x'inedir

- **Avantaj:** elverişlilik
- **Dezavantaj:** zayıf okunabilirlik

5.8.2. Dinamik Kapsam Bağlama

136



Dinamik Kapsam

```
int x = 10;
```

```
void f1();
```

```
void main()
```

```
{
```

```
    int x = 20;
```

```
    f1();
```

```
    cout << x;
```

```
}
```

```
void f1()
```

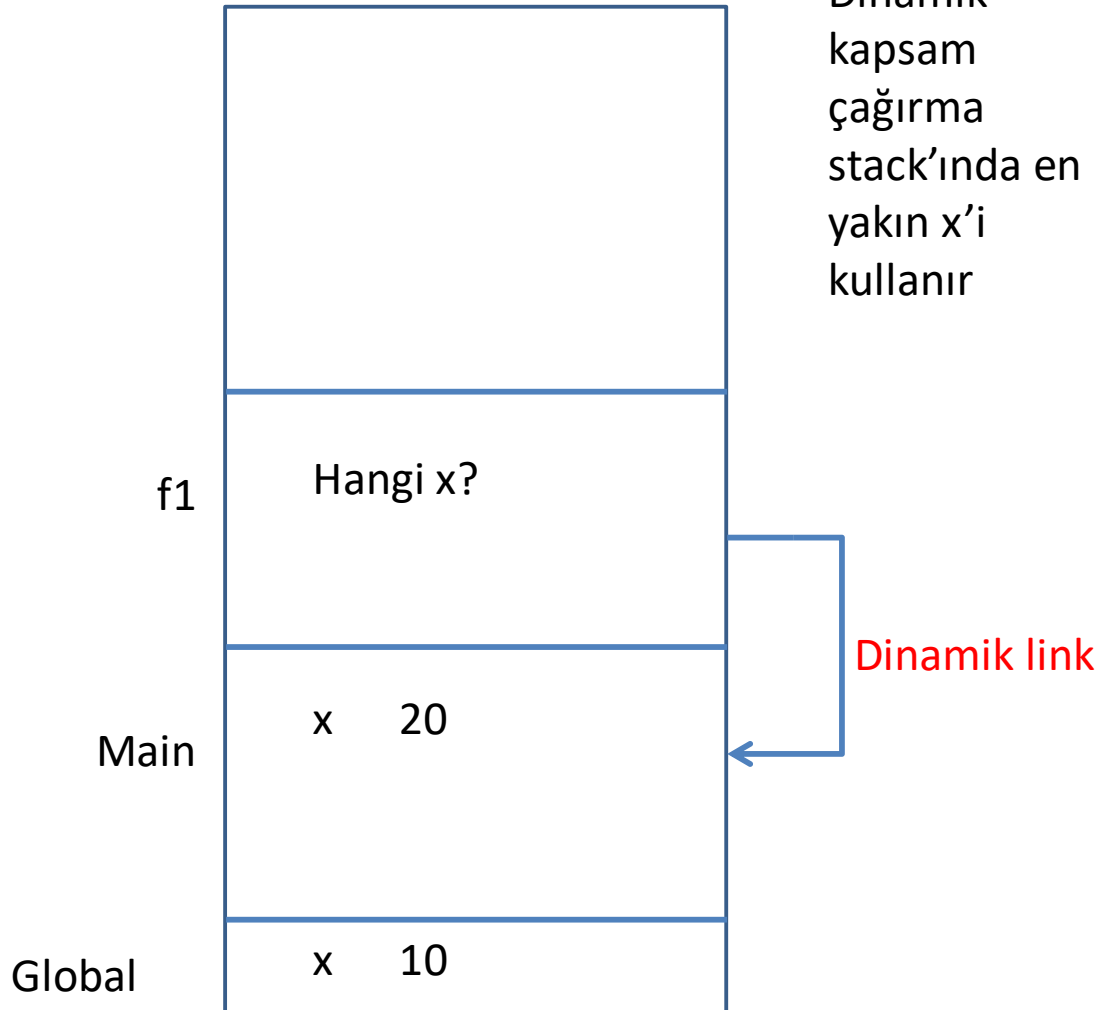
```
{
```

```
    cout << x;
```

```
}
```

Çıktı nedir?

Dinamik
kapsam
çağırma
stack'ında en
yakın x'i
kullanır



5.8.2. 1. Pascal'da Dinamik Kapsam Bağlama

138

```
Procedure nothesap;  
  Var sayac: integer;  
  Procedure yazdır;  
    Begin  
      ...sayac....  
    End;  
  Procedure topla;  
    Var sayac: integer;  
    Begin  
      .....  
    End;  
Begin  
.....  
end;
```

- Durağan Kapsam Bağlama bölümünde incelenen yandaki *yazdır* altprogramını, dinamik kapsam bağlama kurallarına göre yeniden inceleyelim:
- *yazdır* altprogramda *sayac* değişkenine olan başvurunun hangi *sayac* değişkenine olduğu, çağrım sırasına bağlı olduğu için derleme zamanında belirlenemez.
- Çalışma zamanında *yazdır* altprogramında bir deyimin *sayac* değişkenine başvuru yapması durumunda hangi *sayac* değişkenine başvuru yapıldığını belirlemek için;

5.8.2. 1. Pascal'da Dinamik Kapsam Bağlama

139

```
Procedure nothesap;  
  Var sayac: integer;  
  Procedure yazdır;  
    Begin  
      ...sayac....  
    End;  
  Procedure topla;  
    Var sayac: integer;  
    Begin  
      .....  
    End;  
Begin  
  .....  
end;
```

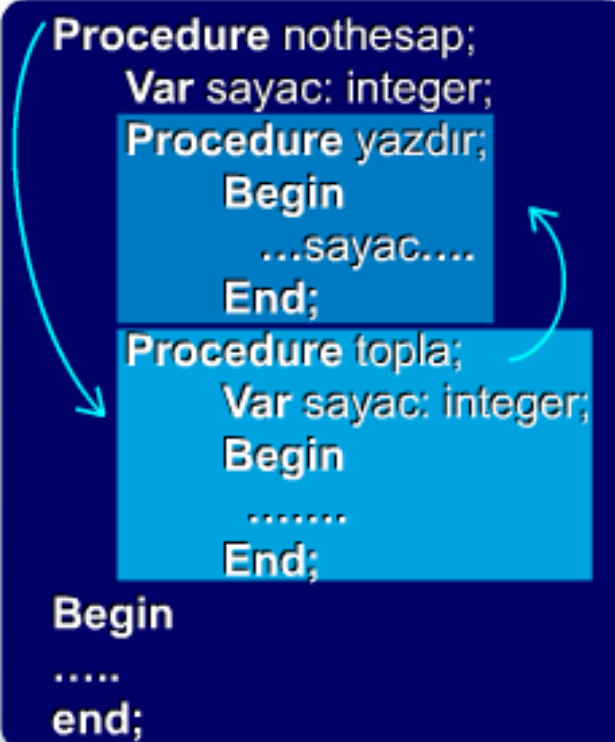
- 1) Öncelikle yerel tanımlamalar aranmaya başlanır.
- 2) Eğer yerel değişkenlerde ilgili tanımlama bulunamazsa, çağıran altprogramın tanımlamaları incelenir.
- 3) Bu şekilde ilgili tanımlama bulunana kadar aramaya devam edilir.
- 4) Dinamik kapsam bağlama kurallarına göre hiçbir altprogramda tanımlama bulunamazsa, çalışma zamanı hatası verilir.

5.8.2. 1. Pascal'da Dinamik Kapsam Bağlama

140

Örnek

```
Procedure nothesap;  
  Var sayac: integer;  
  Procedure yazdır;  
    Begin  
      ...sayac....  
    End;  
  Procedure topla;  
    Var sayac: integer;  
    Begin  
      .....  
    End;  
Begin  
  ....  
end;
```



- Nothesap önce topla altprogramını çağırırsa ve
- Topla altprogramı da yazdır altprogramını çağırırsa;
sayac değişkenine olan başvuru iki adımda aranır.

5.8.2. 1. Pascal'da Dinamik Kapsam Bağlama

141

Örnek

```
Procedure nothesap;  
  Var sayac: integer;  
  Procedure yazdır;  
    Begin  
      ...sayac....  
    End;  
  Procedure topla;  
    Var sayac: integer;  
    Begin  
      .....  
    End;  
Begin  
  .....  
end;
```

- Nothesap önce topla altprogramını çağırırsa ve
- Topla altprogramı da yazdır altprogramını çağırırsa;
sayac değişkenine olan başvuru iki adımda aranır.
 1. Yazdır altprogramındaki sayac değişkeni için yerel tanımlamalar aranır.

5.8.2. 1. Pascal'da Dinamik Kapsam Bağlama

142

Örnek

```
Procedure nothesap;  
  Var sayac: integer;  
  Procedure yazdır;  
    Begin  
      ...sayac....  
    End;  
  Procedure topla;  
    Var sayac: integer;  
    Begin  
      .....  
    End;  
  Begin  
    ....  
end;
```

- Nothesap önce topla altprogramını çağırırsa ve
- Topla altprogramı da yazdır altprogramını çağırırsa;
sayaç değişkenine olan başvuru iki adımda aranır.
 1. Yazdır altprogramındaki sayac değişkeni için yerel tanımlamalar aranır.
 2. Daha sonra, yazdır'ı çağırın topla altprogramı aranır.

5.8.2. 1. Pascal'da Dinamik Kapsam Bağlama

143

Örnek

```
Procedure nothesap;  
  Var sayac: integer;  
  Procedure yazdır;  
    Begin  
      ...sayac...  
    End;  
  Procedure topla;  
    Var sayac: integer;  
    Begin  
      .....  
    End;  
Begin  
  ....  
end;
```


- Nothesap önce topla altprogramını çağırırsa ve
- Topla altprogramı da yazdır altprogramını çağırırsa;
sayac değişkenine olan başvuru iki adımda aranır.
 1. Yazdır altprogramındaki sayac değişkeni için yerel tanımlamalar aranır.
 2. Daha sonra, yazdır'ı çağıran topla altprogramı aranır.
 3. Topla altprogramında sayac değişkeni için bir tanımlama bulunduğu için, sayac değişkenine olan başvuru, topla altprogramındaki tanımlamaya ilişkindir.

5.8.2. 1. Pascal'da Dinamik Kapsam Bağlama

144

Örnek

```
Procedure nothesap;  
  Var sayac: integer;  
  Procedure yazdır;  
  Begin  
    ...sayac....  
  End;  
  Procedure topla;  
  Var sayac: integer;  
  Begin  
    .....  
  End;  
Begin  
  ....  
end;
```



- Nothesap önce topla altprogramını çağırırsa ve
- Topla altprogramı da yazdır altprogramını çağırırsa;
sayac değişkenine olan başvuru iki adımda aranır.
 1. Yazdır altprogramındaki sayac değişkeni için yerel tanımlamalar aranır.
 2. Daha sonra, yazdır'ı çağıran topla altprogramı aranır.
 3. Topla altprogramında sayac değişkeni için bir tanımlama bulunduğu için, sayac değişkenine olan başvuru, topla altprogramındaki tanımlamaya ilişkindir.
- Daha sonra, nothesap içinden yazdır altprogramı yeniden çağırılırsa, bu durumda, sayac değişkeni nothesap'ta tanımlanmış sayac değişkenine başvuru yapar.

Statik isim kapsam

Değişkenlerin kapsamları, programın metinsel düzenine göre, fiziksel yakınlığa göre, belirlenir.

ALGOL 60'ı izleyen çok sayıda dilde tanımlıdır. Altprogramlar iç içe yuvalanabilir. (C++ ve FORTRAN hariç)

1. Altprogramların yuvalanması sonucu gereğinden fazla genel değişken kullanımı olabilir.
2. Bir programda genel olarak tanımlanan değişkenler tüm altprogramlara görünebilir olacakları için güvenilirlik azalmaktadır.

Dinamik Kapsam

Bir ismin kapsamının, altprogramların fiziksel yakınlıklarına göre değil, altprogramların çağrılma sırasına göre çalışma zamanında belirlenmesi **dinamik kapsam bağlama** olarak adlandırılır.

LISP, APL dillerinin ilk sürümleri

1. Bir altprogramda bir değişkene yapılan başvuru, deyim her çalışmasında farklı değişkenleri gösterebilir.
2. Programların anlaşılabilirliğini azaltmaktadır

Dinamik Kapsam Bağlama Örneği

MAIN

- declaration of x

SUB1

- declaration of x -

...

call SUB2

...

SUB2

...

- reference to x -

...

...

call SUB1

...

MAIN SUB1'i
çağırır
SUB1 SUB2'yi
çağırır
SUB2 x'i
kullanır

- Durağan Kapsam
 - ▣ x'e referans MAIN'in x'inedir
- Dinamik Kapsam
 - ▣ x'e referans SUB1'in x'inedir
- Dinamik kapsamanın değerlendirilmesi:
 - ▣ **Avantaj:** elverişlilik
 - ▣ **Dezavantaj:** zayıf okunabilirlik
- Perl ve Common Lisp'te dinamik kapsam vardır

ÖRNEK

```

program L;
  var n: char;           {n, L' de bildirilmiş }

  procedure W;
    begin
      writeln(n) {W de n 'ye başvuru var.}

    end;

  procedure D;
    var n: char;         {D de n tekrar bildirilmiş}

    begin
      n:= "D";

      W;                  { D' deki W' i çağırdı}

    end;

  begin {L}
    n:= "L";

    W;                    {Ana program L' den W' u çağırdı}

    D;

  end.

```

Statik kapsam bağlama
kuralına göre

L
L

Dinamik kapsam bağlama
kuralına göre

L
D

Örnek: Aşağıdaki program parçasının çıkışını

- a) statik kapsam bağlama kurallarına göre
- b) Dinamik kapsam bağlama kurallarına göre bulunuz.

```
int x;  
  
int main() {  
    x = 2;  
    f();  
    g();  
}  
  
void f() {  
    int x = 3;  
    h();  
}  
  
void g() {  
    int x = 4;  
    h();  
}  
  
void h() {  
    printf("%d\n",x);  
}
```

Dinamik kapsam bağlamaya göre çıkış: 3 4

Statik kapsam bağlamaya göre çıkış: 2 2

Statik-dinamik kapsam örnek

149

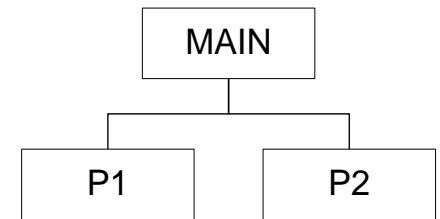
```
program MAIN;  
  var a : integer;  
  
  procedure P1;  
  begin  
    print a;  
  end; {of P1}  
  
  procedure P2;  
  var a : integer;  
  begin  
    a := 0;  
    P1;  
  end; {of P2}  
  
begin  
  a := 7;  
  P2;  
end. {of MAIN}
```

statik (lexical)

Lokal olmayan değişkenler program yapısına bağlı olarak sınırlıdır

Lokal değilse, “dışarı” bir seviyeye git

→ örnek 7 yazdırır

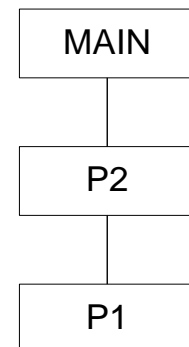


dinamik

Lokal olmayan değişkenler çağırma sırasına bağlı olarak sınırlıdır

Lokal değilse, çağırma noktasına git

→ örnek 0 yazdırır



İç içe kapsam örneği

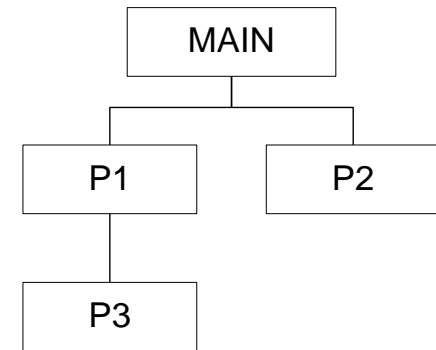
150

```
program MAIN;  
  var a : integer;  
  
  procedure P1(x : integer);  
    procedure P3;  
      begin  
        print x, a;  
      end; {of P3}  
    begin  
      P3;  
    end; {of P1}  
  
  procedure P2;  
    var a : integer;  
    begin  
      a := 0;  
      P1(a+1);  
    end; {of P2}  
  
  begin  
    a := 7;  
    P2;  
  end. {of MAIN}
```

Çoğu dil iç içe yuvalanmış prosedürlere izin verir

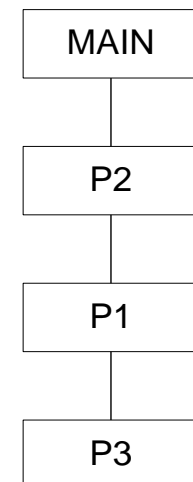
statik kapsam

→ örnek 1, 7 yazdırır



dinamik kapsam

→ örnek 1, 0 yazdırır



5.8.2. Dinamik Kapsam Bağlama

151

□ Dinamik Kapsam Bağlamanın Sorunları

- Bir altprogramdaki içeride tanımlanmamış değişken programın sürecine göre farklı altprogramlardaki farklı tanımlara gönderme yapıyor olabilir.
- Altprogramlardaki değişkenleri başka altprogramların beklenmedik değiştirmelerinden korumak çok zor. Güvenilirlik çok düşüyor.
- Yerel olmayan değişkenlerin kullanım sırasında tip kontrolünü yapmak zor.
- Dinamik kapsamlı bir programı okumak pratikte çok zor. Her türlü dinamik kapsam öngörülemez.
- Yerel olmayan değişkenlere erişim çok fazla zaman aldığından, program yavaşlıyor.

5.9. Tip Kontrolü

152

- İşlenen (operands) ve işleç (operators) tanımlarını genişletirsek: altprogramlar (subprograms) işletmen, parametreleri işlenenler; atamalar (assignments) işletmen, değişkenler ve ifadeler işlenenler şeklinde tanımlanabilir.
- Tanım: İşlenenlerin işletmenlere uygunluğuna bakmak tip kontrolü (**Type checking**) olarak adlandırılır.
- Tanım : Bir uygun tipli işlenen, ya işletmenin tanımına uygundur veya dilin yapısı içinde örtülü olarak uygun tipe çevrilebilir.
- Tanım : Tip hatası: işlenen işletmene uygun değilse tip hatası oluşur.
- Örnek:

```
int i;  
float f;  
...  
f = 3.14 * i;
```

```
int *ip;  
float f;  
...  
f = 3.14 * ip;
```


5.9. Tip Kontrolü

153

- Eğer tip bağlanmaları statikse, tip kontrolü de statiktir.
- Eğer tip bağlanmaları dinamikse, tip kontrolü de dinamik olmak zorundadır.
- Tanım: Bir programlama dili eğer tip hatalarının hepsini fark ediyorsa bu dile **kesin tiplendirilmiş (strongly typed)** dil denir.
- Bir dildeki bütün isimlerin önceden tanımlanmış olması kesin tiplendirilmiş olması için yeterli değildir. Çünkü programın yürütülmesi sırasında farklı veriler konulabilir.

5.9.1. Kesin tiplendirme

154

- Kesin tiplendirmenin avantajı: değişkenlerin yanlış kullanımının fark edilmesini ve hatalı sonuçları engeller.
- Bazı dillerin kesin tiplendirme durumu:
 - ▣ FORTRAN 77 kesin tiplendirilmiş değildir: parametreler, **EQUIVALENCE**
 - ▣ Pascal değildir.
 - ▣ C ve C++ değildir: parametre tip kontrolü engellenebilir, “union” tipler kontrol edilmez.
 - ▣ Ada hemen hemen (kontrol edilmeyen çevrimlerin istenebiliyor olması zayıflık) (Java, C# benzer şekilde)
- Çevirme kuralları kesin tip kontrolünü zayıflatır. (C++ karşı Ada)
- Java’nın çevirme kuralları C++’ın yarısı kadar olsa da, kesin tip kontrolü Ada’nın yanında zayıftır. Örneğin Java’da bir tam sayı değişkeni ile gerçel sayı değişkeni toplanırken tam sayı otomatik gerçeğe çevrilir ve bu yapılırken kesin tiplendirme bozulmuş olur.

5.9.2. Tip Uyumluluğu

155

- İki tip 'tip' uyumluluğu bulunmaktadır:
 - ▣ İsim tipi uyumluluğu;
 - ▣ Yapısal tip uyumluluğu.
- İsim tipi uyumluluğu: eğer iki değişken aynı tanımlamada tanımlanmış veya tanımlamalarında aynı tip tanımlama kullanılmışsa.
- Uygulanması kolay ancak hayli sınırlandırıcı, Ada örneğini inceleyelim:
 - ▣ Sınırlı tam sayılar ile tam sayılar uyumlu değil:

```
type indextype is 1..100;  
count: integer;  
index: indextype;
```
 - ▣ Fonksiyona geçirilen yapısal parametrelerin isim tipi uyumluluğu olması gerekirse, bu tanımlama her fonksiyonda yapılamayacağından, bir kez global tanımlanması gerekir (Pascal).

5.9.2. Tip Uyumluluğu

156

- Yapısal tip uyumluluğu (Structure type compatibility): Eğer yapıları (structure) aynıysa, iki değişken uyumlu tiplerdir.
- Daha esnek fakat uygulaması zor.

5.9.2. Tip Uyumluluğu

157

- Yapısal tiplerle ilgili aşağıdaki problemleri tartışalım:
 - ▣ Yapısal olarak aynı ama farklı alan adları kullanmış iki kayıt uyumlu mudur?
 - ▣ Diğer bütün özellikleri aynı ama indeksleri farklı iki dizilim aynı mıdır? (Örneğin [1..10] and [0..9])
 - ▣ Elemanları farklı yazılmış iki enumeration tip uyumlu mudur?
 - ▣ Tip uyumluluğu ile aynı yapıdaki farklı tipleri ayırt edemezsiniz (Örneğin farklı birimlerde hız (mph – km/h), ikisi de float).
 - Ada bu problemin üstesinden aşağıdaki gibi gelir:

```
type mph is new float;  
type kmph is new float;
```
 - ▣ Uyumlu Ada örnekleri:

Aşağıdaki örnek integer tipi ile uyumludur:

```
subtype small_int is integer range 1..99;
```

Aşağıdaki iki vector de uyumludur.

```
type vector is array (integer range <>) of integer;  
vector 1: vector (1..10);  
vector 2: vector (11..20);
```

5.9.2. Tip Uyumluluğu

158

□ Dil örnekleri:

- ▣ Pascal: genellikle yapısal tip uyumluluğu, fakat bazı durumlarda isim uyumluluğu da kullanılır (alt program parametrelerinde).
- ▣ C: yapısal, “structure”, “union” hariç.
- ▣ Ada: sınırlandırılmış isim uyumluluğu. Tanımlanmış yapılar uyumlu.
 - Türetilmiş tiplerde aynı yapıların farklı olması mümkün.
 - Genel tiplerin hepsi tek, hatta:

`A, B : array (1..10) of INTEGER:`
uyumlu değil.

- Uyumlu olması için:

```
type list10 is array (1 ..10) of Integer;  
A, B : list10;
```

5.9.2. Tip Uyumluluğu

159

- C, C++, Java, vs, için toplama operatörünün iki işlecinin farklı nümerik tiplerde olması önemli değildir. Uygun şekilde örtülü olarak çevrilirler.
- Ada'da aritmetik operatörler için örtülü çevirme yoktur.
- C 'struct' ve 'union' hariç bütün tipler için yapısal uyumluluk kullanır. Her 'struct' ve 'union' ayrı bir tiptir ve dolayısıyla uyumlu değildir. 'typedef' yeni bir tip yaratmaz, sadece isimlendirir.
- C++ isim tipi uyumluluğu kullanır.
- Yeni tip tanımlamaya izin vermeyen dillerde (Fortran, Cobol) isim tipi uyumluluğu kullanılamaz.

5.10. Kapsam ve Ömür

160

- Kapsam (Scope) ve Ömür (Lifetime) bazen yakından ilişkilidir, fakat farklı kavramlardır
- Bir C veya C++ fonksiyonundaki bir **static** değişkeni düşünelim

5.11. Referans Çevreleri

161

- ❑ Tanım: Bir ifadenin referans çevresi ifadede görünen bütün isimlerin koleksiyonudur
- ❑ Bir statik kapsamlı dil, lokal değişkenler artı bütün çevreleyen kapsamlardaki görünür değişkenlerin tümüdür
- ❑ Bir altprogramın çalıştırılması başlamış, ama henüz bitmemişse o altprogram aktiftir
- ❑ Bir dinamik kapsamlı dilde, referans platformu lokal değişkenler artı tüm aktif altprogramlardaki bütün görünür değişkenlerdir.

Özet

162

- Bu bölümde temel programlama elemanları incelenmiştir. Bu kapsamda, değişkenler, sabitler, işlemciler, ifadeler ve deyimler ele alınmıştır.
- İşlemcilerin alt bölümleri olan sayısal işlemciler, ilişkisel işlemciler, mantıksal işlemciler ve işlemci yükleme konuları açıklanmıştır.
- Ayrıca bu bölümde, programlama elemanlarıyla çeşitli özelliklerinin ilişkilendirilmesini sağlayan bağlama kavramı incelenmiştir. Bu kapsamda; durağan bağlama, dinamik bağlama, tip bağlama, bellek bağlama kavramları açıklanmıştır.
- İsim kapsamı konusu açıklanmış ve isim kapsamı içinde durağan kapsam bağlama ile dinamik kapsam bağlama alt konuları ele alınmıştır.

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, Programlama Dilleri Uzaktan Eğitim Notları
- Tuğrul Yılmaz, Programlama Dilleri Ders Notları
- Erkan Tanyıldızı, Programlama Dilleri Ders Notları
- David Evans, Programming Languages Lecture Notes
- Oscar Nierstrasz, Programming Languages Lecture Notes