

# 1906002132015

# Programlama Dilleri Temelleri

BAİBÜ Bilgisayar Müh.

## Ders 3

Dr. Araş. Gör. İsmail Hakkı Parlak

Kaynak: Watt, David A., Programming Language Design Concepts, Wiley

Şekiller kaynak kitaptan kopyalanmıştır.

# Bağlar ve Kapsam

- Programlama dilleri, programcılarının **tanımlayıcıları (identifiers)** değerler, değişkenler ve prosedürler gibi varlıklara bağlayan **bildirimler (declarations)** yazmasına olanak tanır.
- İyi seçilmiş tanımlayıcılar (identifiers), bir programın anlaşılmasını kolaylaştırmaya yardımcı olur.
- Bir tanımlayıcıyı tek bir yerde bir varlığa bağlamak ve bu tanımlayıcıyı diğer birçok yerde varlığı belirtmek için kullanmak, programın değiştirilmesini kolaylaştırmaya yardımcı olur.

## 3.1 Bağlar ve Ortamlar

- `i++` ifadesi (expression) `i` tanımlayıcısını, `f(i)` ifadesi `f` ve `i` tanımlayıcılarını kullanır. Bu ifadeler kendi başlarına anlaşılamaz. Anlamları bu tanımlayıcıların bildirimlerine dayanır.
- Bir tanımlayıcı (identifier) 1 kere bildirilir (declared),  $n$  kere kullanılır.
- Benzer şekilde `m = i + 1;` veya `print(3 * i);` gibi komutların anlamı, komutlarda kullanılan tanımlayıcıların bildirimlerine dayanır.

## 3.1 Bağlar ve Ortamlar

- **Bağlama (Binding)**, bir tanımlayıcı ile bir değer, değişken veya prosedür gibi bir varlık arasındaki sabit bir ilişkidir.
- **Bir ortam (environment)** (veya ad alanı - name space), bir dizi bağlamadır. Her ifade veya komut belirli bir ortamda yorumlanır ve ifade veya komutta kullanılan tüm tanımlayıcıların o ortamda bağlamaları olmalıdır.
- **Bağlanabilir varlık (bindable entity)**, bir tanımlayıcıya bağlı olabilen varlıktır.
  - C'nin bağlanabilir varlıkları: tipler, değişkenler, fonksiyon prosedürleridir.
  - JAVA'nın bağlanabilir varlıkları değerler, yerel değişkenler, örnek (instance) ve sınıftır değişkenleri, yöntemler (methods), sınıflar ve paketler.

## 3.2 Kapsam (Scope)

- Bir bildirimin kapsamı, bildirimin program metni üzerinde etkili olduğu kısımdır.
- Bir bağlamanın kapsamı, bağlamanın uygulanabildiği program metni kısımdır.

## 3.2 Kapsam (Scope)

```
// C program1
```

```
int k = 3, m = 5;
```

```
void f(int x) {  
    printf("%d", x);  
    int k = 7; // OK  
    printf("%d", k);  
    printf("%d", m);  
}
```

```
x++; // Hata!
```

```
double m = 11; // Hata!
```

```
int main() {  
    int i = 9;  
    f(i);  
    int m = 8; // OK  
    return 0;  
}
```

## 3.2.1 Blok Yapısı

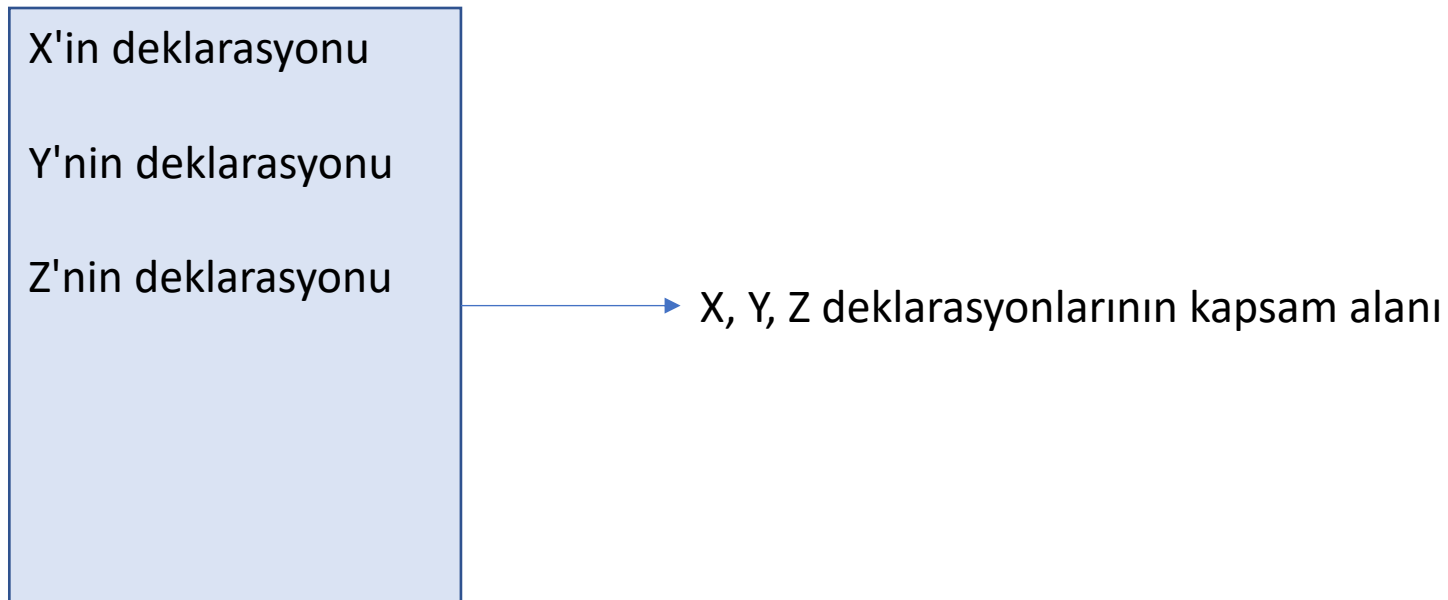
- Blok, içindeki tüm bildirimlerin kapsamını sınırlayan bir program yapısıdır. Her programlama dilinin kendi blok biçimleri vardır:
- Bir C programının blokları, blok komutları (`{ . . . }`), fonksiyon gövdeleri, kaynak dosyalar ve bir bütün olarak programdır.
- Bir JAVA programının blokları, blok komutları (`{ ... }`), yöntem gövdeleri, sınıf bildirimleri, paketler ve bir bütün olarak programdır.

## 3.2.1.1 Monolitik Blok Yapısı

- Tek blok tüm programdır, bu nedenle her bildirimin kapsamı tüm programdır.
- Tüm bildirimler globaldir.
- COBOL'un eski versiyonları.
- Uzun bir program metninde tüm tanımlayıcılar (identifiers) birbirinden farklı olmalıdır.



## 3.2.1.1 Monolitik Blok Yapısı



## 3.2.1.2 Düz (Flat) Blok Yapısı

- Düz blok yapısına sahip bir dilde, program birbirleriyle örtüşmeyen birkaç bloğa bölünmüştür. Örn: FORTRAN
- Her prosedür ve her global değişken ayrı bir tanımlayıcıya sahip olmalıdır.

```
int i, k;  
int foo () { int x; int i; ... }  
int bar () {int x; int k; ...}
```

## 3.2.1.2 Düz (Flat) Blok Yapısı

```
int i, k;  
int foo () { int x; int i; ... }  
int bar () {int x; int k; ...}
```

i, k, foo, bar'ın deklarasyonu

foo  
x, i'nin deklarasyonu

bar  
x, k'nın deklarasyonu

### 3.2.1.3 Yuvalanmış (Nested) Blok Yapısı

- Yuvalanmış (nested) blok yapısına sahip bir dilde, bloklar diğer blokların içine yerleştirilebilir.
- C'de, fonksiyon gövdeleri üst üste gelemez, ancak blok komutları fonksiyon gövdeleri içinde serbestçe iç içe yerleştirilebilir.
- JAVA, yöntem (method) gövdelerinin ve iç sınıfların bir sınıf içinde yuvalanabileceği daha az kısıtlayıcı bir iç içe blok yapısına sahiptir.

## 3.2.2 Kapsam ve Görünürlük

- Bir programda iki farklı tanımlayıcı (identifier) kullanımı görülebilir:
- Bağlama: tanımlayıcı / bir  $X$  varlığına bağlanır.
  - `int n = 7;`
- Uygulama:  $/$ 'ya bağlı  $X$  varlığı kullanılır.
  - `printf ("%d", n * 3);`
- Programda 1'den çok blok varsa aynı tanımlayıcı / birden çok blokta yeniden deklare edilebilir.
- Yuvalanmış (nested) blokların her birinde tanımlayıcı / deklare edilebilir.

## 3.2.2 Kapsam ve Görünürlük

- Eğer iç blokta / deklare edilmemişse, l'nın iç ve dış bloktaki uygulamaları aynı / deklarasyonuna karşılık gelir.
- Eğer iç blokta l deklare edildiyse, iç bloktaki l'nın uygulamaları dış bloktaki / deklarasyonunu **gizler**.

## 3.2.2 Kapsam ve Görünürlük

```
int main() {  
    int n = 1, k = 3;  
    printf("%d", n);  
  
    {  
        int n = 5;  
        printf("%d", n);  
        printf("%d", k);  
    }  
  
    return 0;  
}
```

## 3.2.3 Statik ve Dinamik Kapsam

```
int s = 2;

int f(int x) { return s * x; }

void p(int y) { printf("%d", f(y)); }

void q(int z) {
    int s = 3;
    printf("%d", f(z));
}

int main() {
    p(5);
    q(5);
    return 0;
}
```

- Statik kapsamlı bir dilde bir prosedürün gövdesi, prosedürün tanımlandığı ortamda çalışır.
- Örn: C, C++, JAVA, Haskell
- Dinamik kapsamlı bir dilde prosedürün gövdesi prosedürün çağırıldığı ortamda çalışır.
- Örn: Lisp



## 3.3 Bildirimler (Declarations)

- Bir bildirim, bağlamalar oluşturmak için detaylandırılacak bir yapıdır.
- Deklarasyon (Bildirim, Declaration):
  - `extern int n;`
  - `int topla(int, int);`
- Tanım (Definition): Tek görevi bağlama oluşturmaktır.
  - `int n;`
  - `int topla(int a, int b) { return a + b; }`

## 3.3.1 Tip Deklarasyonları

- Bir tür bildirimi, bir tanımlayıcıyı bir türe bağlar. İki tür tip bildirimini ayırt edebiliriz:
- Bir tür tanımı (type definition), bir tanımlayıcıyı mevcut bir türe bağlar.
  - **typedef char\*** Text;
- Yeni tür bildirimi (new-type declaration), bir tanımlayıcıyı mevcut herhangi bir türe eşdeğer olmayan yeni bir türe bağlar.
  - **struct** Kitap {Text baslik, **int** baskiNo};

## 3.3.2 Sabit Deklarasyonları

- Sabit (constant) bir bildirim, bir tanımlayıcıyı sabit bir değere bağlar.
- Sabit bir bildirim tipik olarak “const I = E;” biçimindedir ve I tanımlayıcısını E ifadesinin değerine bağlar.
- `const int taban = 10;`
- `#define PI 3.14`

## 3.3.3 Değişken Deklarasyonları

- Bir değişken bildirimi, en basit haliyle, tek bir değişken oluşturur ve bu değişkene bir tanımlayıcı bağlar.
- Çoğu programlama dili, bir değişken bildiriminin birkaç değişken oluşturmaya ve bunları farklı tanımlayıcılara bağlamasına da izin verir.
- `int n = 3;`
- `int j = 5, k = 7;`

## 3.3.4 Prosedür Tanımları

- Bir prosedür tanımı, bir tanımlayıcıyı bir prosedüre bağlar.

```
bool even (int n) {  
    return (n % 2 == 0);  
}
```

### 3.3.5 Özyinelemeli (Recursive) Deklarasyonlar

- Özyinelemeli bir bildirim, kendi ürettiği bağlamaları kullanan bir bildirimdir.
- Deklarasyonun gövdesi, deklarasyonun tanımlayıcısına erişebilir.
  - `I = Body`
- Çoğu modern programlama dili özyinelemeli deklarasyonları destekler.
- Bir dizi prosedür tanımları karşılıklı olarak özyinelemeli yapılabilir (mutually recursive).

### 3.3.5 Özyinelemeli (Recursive) Deklarasyonlar

**cift** :: Int -> Bool

**cift** 0 = True

**cift** n = **tek** (n - 1)

**tek** :: Int -> Bool

**tek** 0 = False

**tek** n = **cift** (n - 1)

## 3.4.1 Blok Komutları

- Bir blok komutu (block command), yerel bir bildirim (veya bildirimler grubu - local declaration)  $D$  ve bir alt komut  $C$  (subcommand) içeren bir komut biçimidir.  $D$  tarafından üretilen bağlamalar yalnızca  $C$ 'yi yürütmek için kullanılır.
- C, Java: {  $D$   $C$  }

```
// JAVA
```

```
if (x > y) {  
    int z = x;  
    x = y;  
    y = z;  
}
```

$z$  değişkeninin ömrü *if* bloğunun ömrü kadardır.



## 3.4.2 Blok İfadeler (Expressions)

- Blok ifadesi, yerel bir bildirim (veya bildirimler grubu) D ve bir alt ifade C içeren bir ifade biçimidir. D tarafından üretilen bağlamalar yalnızca E'yi değerlendirmek için kullanılır.
- C, C++ ve ADA'da bir fonksiyon prosedürünün gövdesi bir blok ifadesidir.

```
float area (float x, y, z) {  
    float s = (x + y + z)/2.0;  
    return sqrt(s*(s-x)*(s-y)*(s-z));  
}
```