190600213201 5
Programlama Dilleri Temelleri

BAİBÜ Bilgisayar Müh.

Ders 3

Dr. Öğr. Üyesi İsmail Hakkı Parlak

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

Şekiller kaynak kitaptan kopyalanmıştır.

4 Bağlar ve Kapsam

- Programlama dilleri, programcıların **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.

4.1 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.

4.1 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, fonksiyonlar.
  - JAVA'nın bağlanabilir varlıkları: değerler, yerel değişkenler, örnek (instance) ve sınıftır değişkenleri, yöntemler (methods), sınıflar ve paketler.

4.2 Kapsam (Scope)

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

4.2 Kapsam (Scope)

```
// C programı

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;
}
```

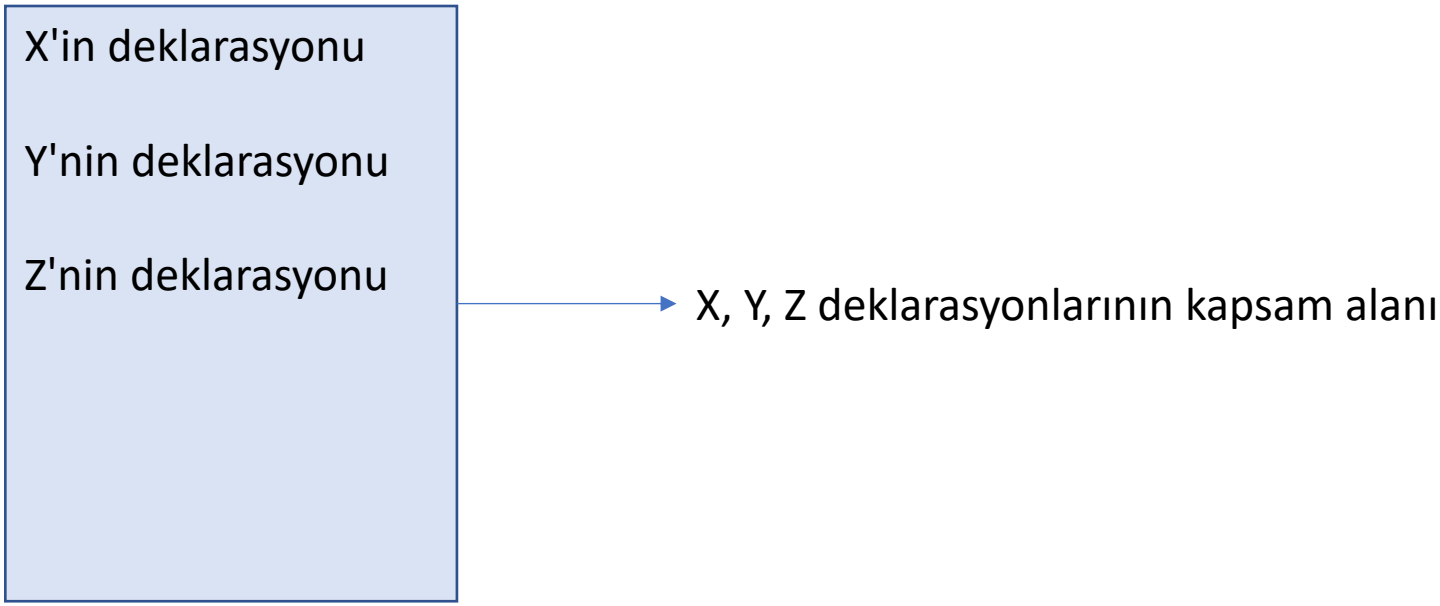
4.2.1 Blok Yapısı

- 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.

4.2.1.1 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.

4.2.1.1 Monolitik Blok Yapısı



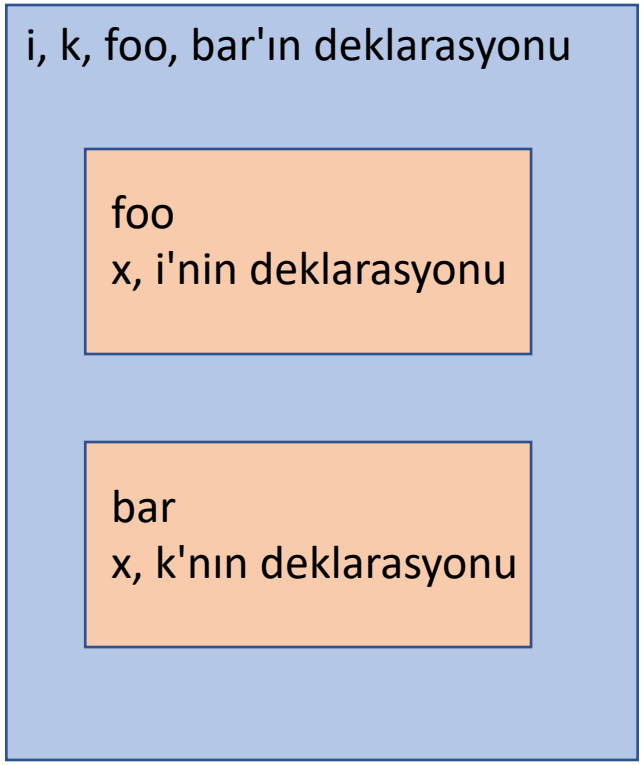
4.2.1.2 Düz (Flat) Blok Yapısı

- Düz blok yapısına sahip bir dilde, program birbiriyle 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; …}
```



4.2.1.3 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.

4.2.2 Kapsam ve Görünürlük

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

4.2.2 Kapsam ve Görünürlük

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

4.2.2 Kapsam ve Görünürlük

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

  {
    int n = 5;
    printf("%d", n);
    printf("%d", k);
  }

  return 0;
}
```

4.3 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; }`

4.3.1 Tip Deklarasyonları

- Bir tip bildirimi, bir tanımlayıcıyı bir tipe bağlar.
- Bir tip tanımı (type definition), bir tanımlayıcıyı mevcut bir tipe bağlar.
  - `typedef char* Text;`
- Yeni tip bildirimi (new-type declaration), bir tanımlayıcıyı mevcut herhangi bir tipe eşdeğer olmayan yeni bir tipe bağlar.
  - `struct Kitap {Text baslik, int baskiNo};`

4.3.2 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`

4.3.3 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 oluşturmasına ve bunları farklı tanımlayıcılara bağlamasına da izin verir.
- `int n = 3;`
- `int j = 5, k = 7;`

4.3.4 Prosedür Tanımları

- Bir prosedür tanımı, bir tanımlayıcıyı bir prosedüre bağlar.

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

4.3.5 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).

4.3.5 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)
```

4.4 Bloklar

4.4.1 Blok komutlar

- 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.

4.4.2 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. Haskell'deki *let - in* bloğu.
- 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));
}
```