190600213201 5
Programlama Dilleri Temelleri

BAİBÜ Bilgisayar Müh.

Ders 2

Dr. Öğr. Üyesi İsmail Hakkı Parlak

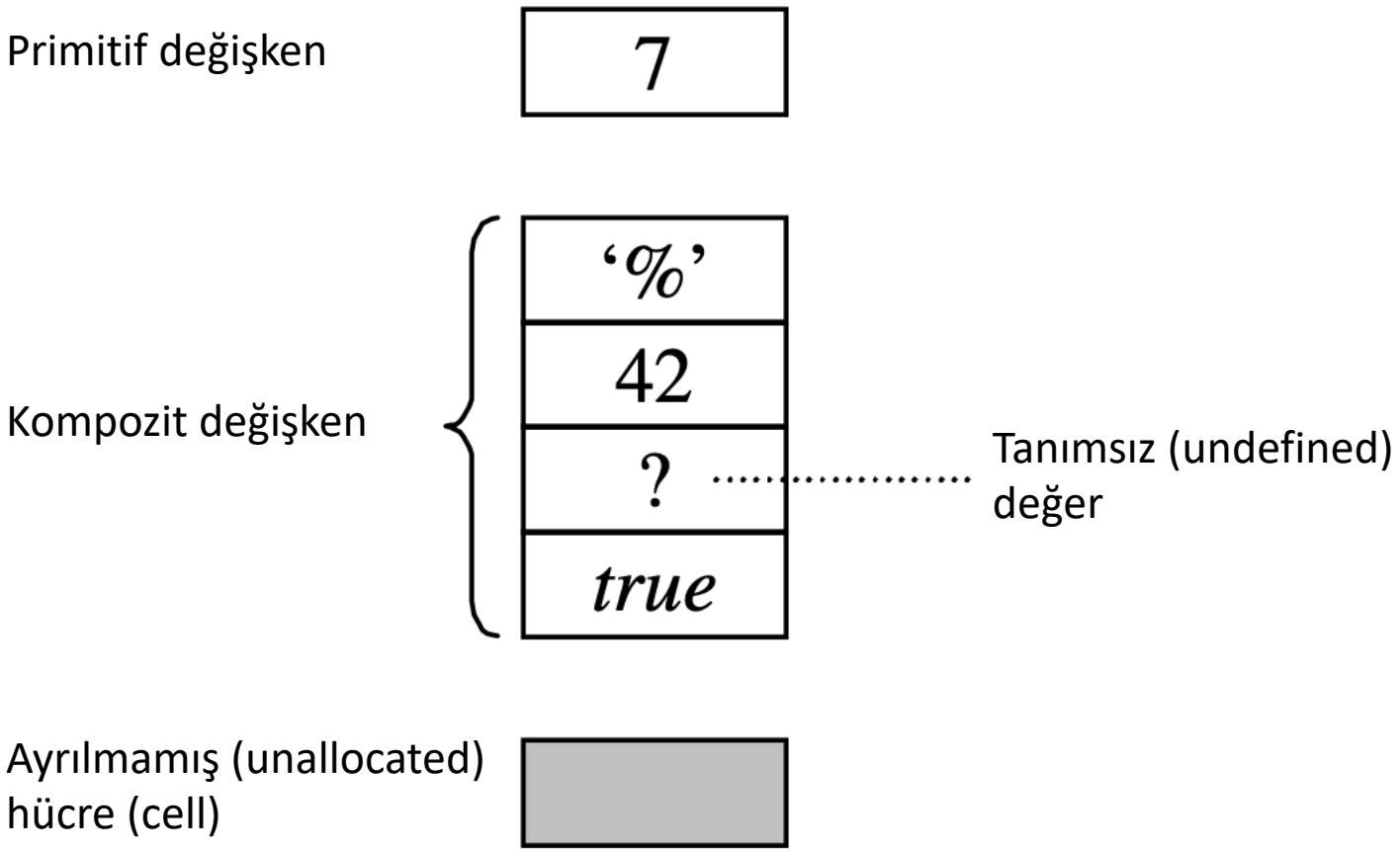
Kaynak: Watt, David A., Programming Language Design Concepts, Wiley

Şekiller kaynak kitaptan kopyalanmıştır.

3.1 Değişkenler (Variables) ve Depolama (Storage)

- Zorunlu (imperative), nesne yönelimli (oop) ve eşzamanlı (concurrent) programlama dillerinde, değişken bir değer için bir kapsayıcıdır ve istendiği kadar sık incelenebilir ve güncellenebilir.
- Imperative programlamadaki değişkenler matematikteki değişkenler gibi davranmazlar.
- Değişkenlerin nasıl davrandığını anlamak için nasıl saklandıklarını bilmeliyiz.

3.1 Değişkenler (Variables) ve Depolama (Storage)



3.1 Değişkenler (Variables) ve Depolama (Storage)

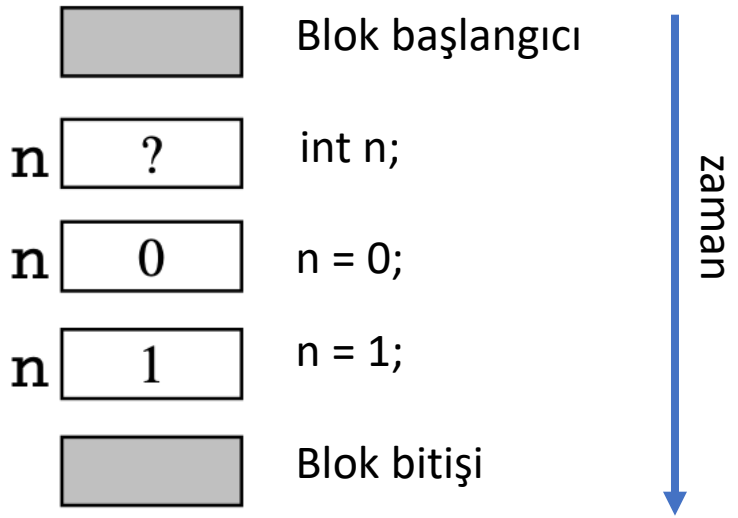
- Bir depolama alanı (store), her biri benzersiz bir adrese sahip olan bir depolama hücreleri (storage cells) topluluğudur.
- Her depolama hücresinin, tahsis edilmiş (allocated) veya tahsis edilmemiş (unallocated) olan bir durum bilgisi (status) bulunur.
- Tahsis edilen her depolama hücresinin, depolanabilir bir değer veya tanımsız olan bir içeriği vardır.
- Depolanabilir bir değer (storable value), tek bir depolama hücresinde depolanabilen değerdir.
- C:
  - int, char, pointer -> depolanabilir değer
  - struct, array, union, function -> depolanabilir **olmayan** değer

3.2 Basit (Simple) Değişkenler

- Basit bir değişken, saklanabilir bir değer içerebilen bir değişkendir. (Tek bir depolama hücresini kaplar.)

Bir C bloğu:

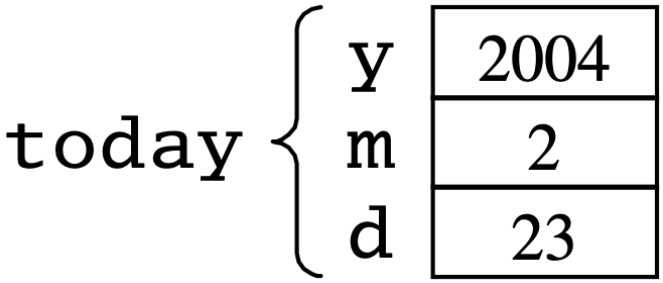
```
{
    int n;
    n = 0;
    n = n+1;
}
```



3.3 Kompozit Değişkenler

- Kompozit değişken, kompozit türden bir değişkendir. Her kompozit değişken, bir grup bitişik depolama hücresinde depolanır.

```
struct Date {
  int y, m, d;
};

Date today;

today.m = 2;
today.d = 23;
today.y = 2004;
```



3.3.1 Bütünsel ve Selektif Güncelleme

- Bütünsel güncellemede kompozit bir değişkenin değeri tek seferde güncellenir.
- Selektif (seçici) güncellemede kompozit değişkenin tek bir bileşeni güncellenir.

```
struct Date {
  int y, m, d;
};

Date d1, d2;

d1.m = 2;
d1.d = 23;
d1.y = 2004;

d2 = d1;
```

3.3.2 Statik, Dinamik ve Esnek Array'ler

- Statik bir dizi, dizin aralığı (index range) derleme zamanında (compile-time) sabitlenen bir dizi değişkenidir.

```
int nums[] = {10, 20, 30}; // dizin aralığı = 0-2

char msg[50]; // dizin aralığı = 0-49
```

3.3.2 Statik, Dinamik ve Esnek Array'ler

- Dinamik bir dizi, dizin aralığı (index range) array değişkenin yaratılma zamanında sabitlenen bir dizi değişkenidir.

```
// C++

// Statik
int a[5] = {1, 2, 3, 4, 5};

// Dinamik
int size;
cin >> size;
int *a = new int[size];
```

3.3.2 Statik, Dinamik ve Esnek Array'ler

- Esnek bir dizi, dizin aralığı sabit olmayan bir dizi değişkenidir. Esnek bir dizinin dizin aralığı, kendisine yeni bir dizi değeri atandığında değiştirilebilir.

Java

```
float[] v1 = {2.0, 3.0, 5.0, 7.0};
float[] v2 = {0.0, 0.0, 0.0};

v2 = v1;
```

3.4 Kopya Semantiği, Referans Semantiği

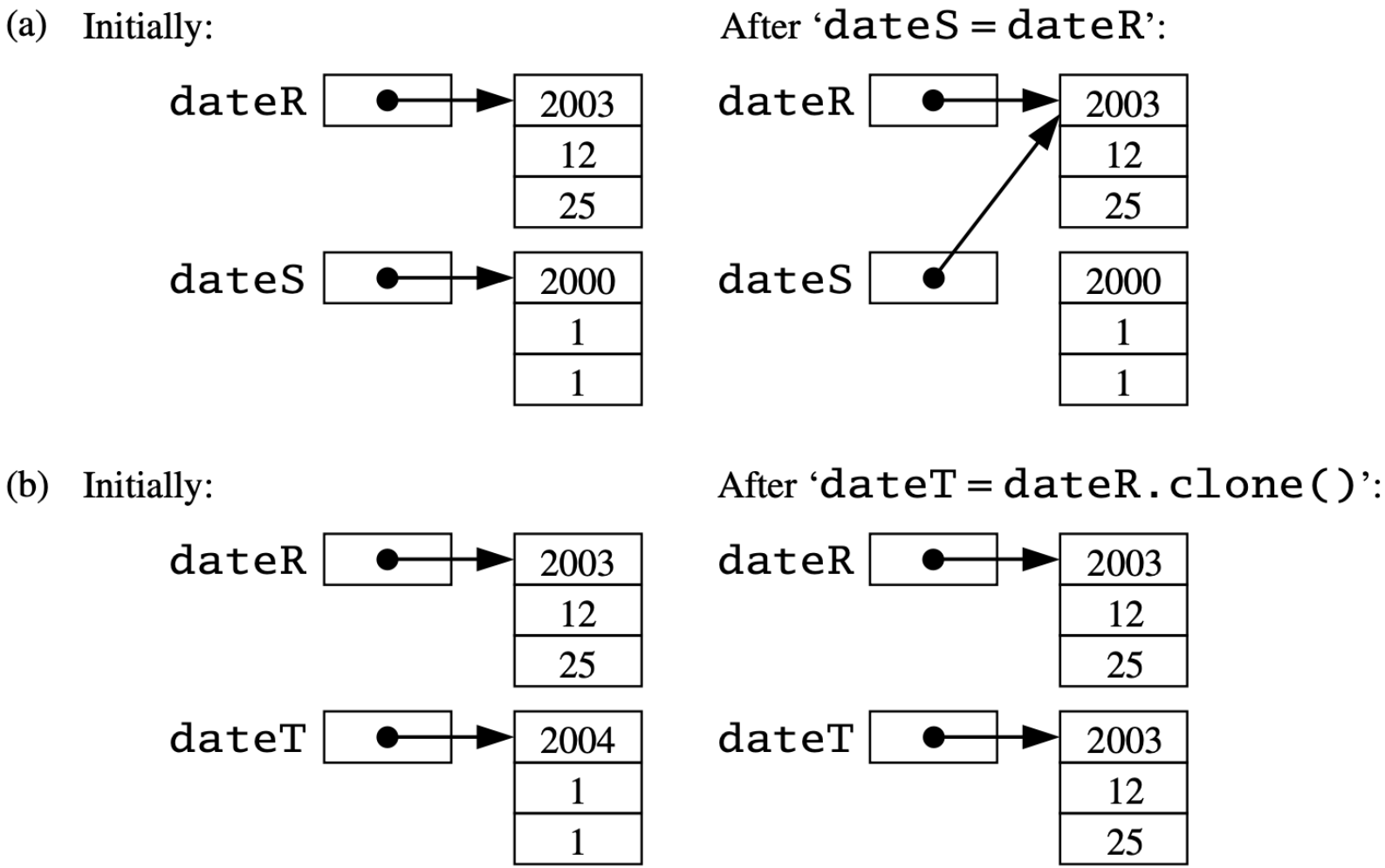
Bir program aynı türden bir değişkene kompozit bir değer atadığında ne olacağı kullanılan dile bağlıdır. 2 farklı olasılık bulunur:

- Kopya semantiği: Atama, kaynak bileşik değerin tüm bileşenlerini hedef bileşik değişkenin karşılık gelen bileşenlerine kopyalar.
- Referans semantiği: Atama, hedef bileşik (kompozit) değişkenin, atanan bileşik değeri gösteren bir pointer içermesini sağlar.

```
// C Struct : Kopya

struct Date {
  int m, d, y;
};

Date d1, d2;

d1 = {2, 23, 2004};
d2 = d1;
d1.m = 5; // d2.m değişmez
```

```
// Java Class : Referans

Date dateR = new Date(2003, 12, 25);
Date dateS = new Date(2000, 1, 1);

dateS = dateR;
dateR.d = 7; // dateS.d değişir

// Değerleri kopyalamak için .clone()
```

3.4 Kopya Semantiği, Referans Semantiği

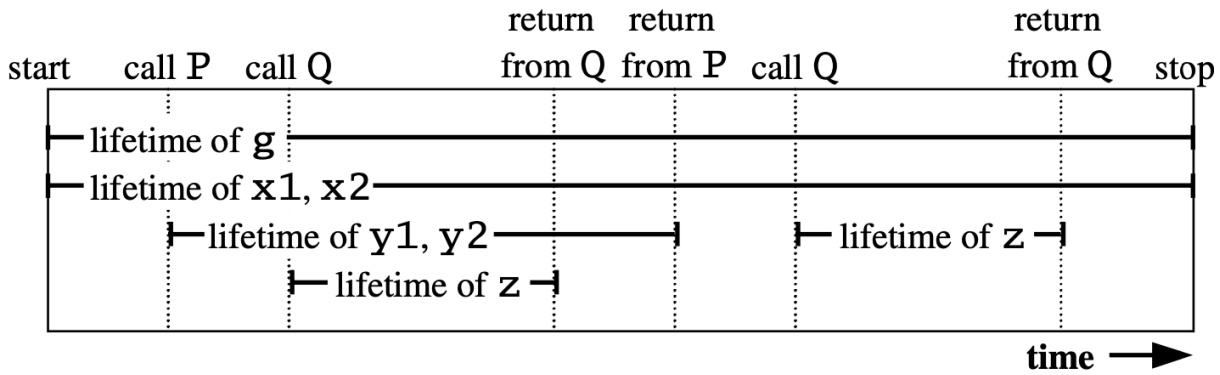


3.5 Değişken Ömrü

- Her değişken belirli bir zamanda yaratılır (veya tahsis edilir) ve daha sonra artık ihtiyaç duyulmadığında yok edilir (veya serbest bırakılır). Bir değişkenin yaratılması ve yok edilmesi arasındaki aralığa, değişkenin ömrü denir.
- Global bir değişkenin ömrü programın ömrü kadardır.
- Lokal bir değişkenin ömrü, kendi bloğunun ömrü kadardır.
- Yığın (Heap) değişkeninin ömrü isteğe bağlıdır. Ancak program ömründen uzun olamaz.
- Kalıcı (Persistent) bir değişkenin ömrü isteğe bağlıdır. Program sonlansa da yaşamaya devam edebilir.

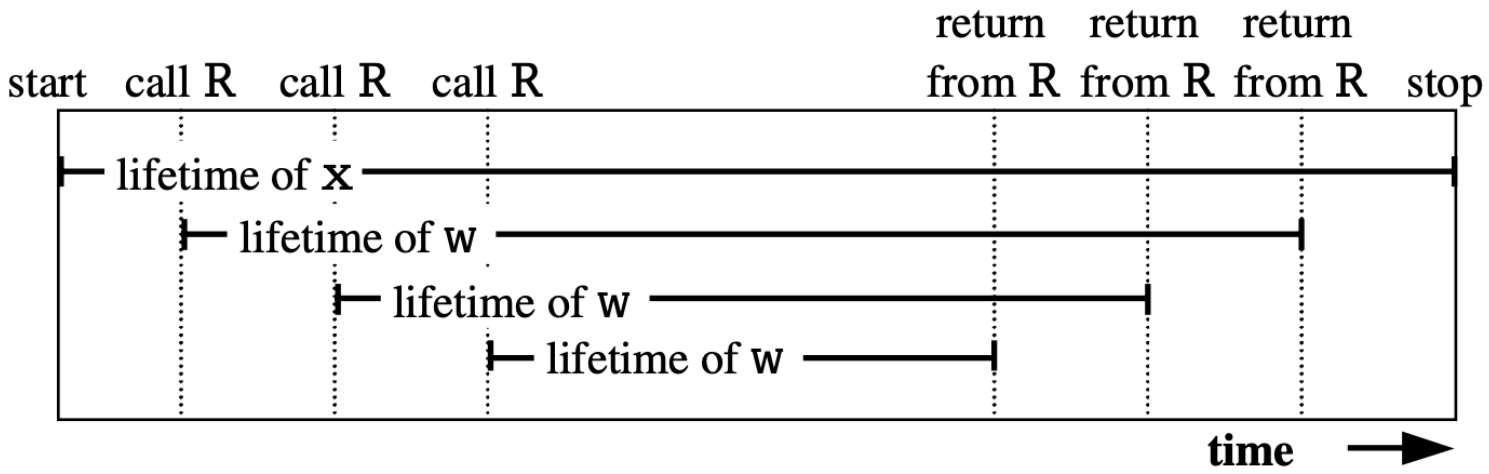
3.5.1 Global ve Lokal Değişkenler

```
int g;

void main () {
     int x1;
     float x2;
     ...
     P();
     ...
     Q();
     ...
}

void P () {
     float y1;
     int y2;
     ...
     Q();
     ...
}

void Q () {
     int z;
     ...
}
```



3.5.1 Global ve Lokal Değişkenler

```
void main () {
    int x;
    ...
    R();
    ...
}

void R () {
    int w;
    ...
    R();
    ...
}
```



3.5.2 Yığın (Heap) Değişkenleri

- İsteğe bağlı yaratılan ve yok edilen değişkenlerdir. C pointer'ları gibi (malloc, free). C++ : new, delete

```
void main () {
    int *a;
    a = malloc(sizof(int));
    a = 7;
    free(a);
}
```

- Yığın değişkenleri için gereken tahsis etme (allocation) ve yok etme (deallocation) işlemleri otomatik gerçekleşmez. Programcı bu işlemleri belirtmelidir.

3.5.3 Kalıcı Değişkenler

- Dosyalar, veri tabanları, …
- OOP: Serialization

```
# Python

import pickle

my_dict = {"Ali": [1,2,3]}

with open("s_file.pickle", "wb") as f:
    pickle.dump(my_dict, f)

with open("s_file.pickle", "rb") as f:
    my_dict_recovered = pickle.load(f)
```

3.6 Pointer'lar

- Pointer (işaretçi), bir değişkene referanstır. Pointer'lara bazen referanslar denir.
- Null Pointer, referansı olmayan özel bir pointer değeridir.

```
struct IntNode {
    int elem;
    IntNode* next;
}

IntNode* p;
```

3.6.1 Pointer'lar ve Recursive Tipler

- Pointer'lar ve yığın değişkenleri Listeler ve Ağaçlar gibi recursive (öz yinelemeli) yapıları ifade etmek için kullanılabilir.
- Genelde imperative dillerde recursive yapılar gerçekten recursive olmaktansa pointer'lar aracılığıyla gerçekleştirilir.

3.6.2 Dangling Pointers (Sarkan İşaretçiler)

- Sarkan bir işaretçi, yok edilmiş bir değişkenin işaretçisidir. Sarkan işaretçiler aşağıdaki durumlarda ortaya çıkar:
  - Yığın değişkeni yok edildikten sonra bir yığın değişkenine yönelik bir işaretçi hala mevcuttur.
  - Yerel bir değişkene yönelik bir işaretçi yerel değişkenin bildirildiği bloktan çıkışta hala yaşar (örneğin, global bir pointer değişkeni) .

3.6.2 Dangling Pointers (Sarkan İşaretçiler)

```
// C++

struct Date {
   int y, m, d;
};

Date* dateP = new Date;
dateP->y = 2023;
dateP->m = 10;
dateP->d = 16;

Date* dateQ = dateP;

delete dateQ;

cout << dateP->y;
```

```
// C++

int* f () {
   int i = 12;
   return &i;
}

int* p = f();
*p = 0;
```

C, C++ yok edilmiş Heap değişkenlerine erişim istendiğinde bunu engellemezler.

Java'da çöp toplayıcısı (garbage collector) mekanizması bu tarz hataları olabildiğince engellemek üzere geliştirilmiştir. Bir heap değişkeni, erişildiği süre boyunca yaşar.

3.7 Komutlar

- Komutlar değişkenleri değiştirmek üzere çalıştırılan (execute) yapılardır.
- Primitif komutlar:
  - Geçmeler (skips)
  - Atanmalar (assignments)
  - Prosedür çağrıları (procedure calls)
- Bileşke komutlar:
  - Sıralı komutlar (sequential)
  - Koleteral komutlar (collateral)
  - Şartlı komutlar (conditional)
  - Yinelemeli komutlar (iterative)

3.7.1 Geçmeler (Skips)

- C, C++: **;**

```
int i;

for (int i=0; i<10; i++)
    ;
```

3.7.2 Atamalar

- V = E
- V: değişken
- E: bir değer dönen ifade (expression)

```
a += 1;

b = c = 7;
```

3.7.3 Prosedür Çağrıları

- $P(E_1, \ldots, E_n)$;
- P: uygulanacak prosedür
- E_i: prosedürüm argümanları
- Uygun bir prosedür çağrısının net etkisi, değişkenleri güncellemektir. Prosedür bu etkiyi, argüman olarak iletilen değişkenleri güncelleyerek ve/veya global değişkenleri güncelleyerek başarabilir.

3.7.4 Sıralı Komutlar

- Sırayla uygulanırlar.
- 2 veya daha fazla komut.
- C1; C2; C3;
- Önce C1, sonra C2, sonra C3, …

3.7.5 Kolateral Komutlar

- Herhangi bir sırayla uygulanırlar.
- 2 veya daha fazla komut.
- C1, C2
- Komut uygulama sırasına compiler veya optimizer karar verebilir.
- m = 7, n = n + 1; // komut sırası önemsiz
- n = 7, n = n + 1; // komut sırası önemli

3.7.6 Şartlı Komutlar

- Koşullu bir komutun iki veya daha fazla alt komutu vardır ve bunlardan tam olarak biri yürütülmek üzere seçilir.
- if else, switch case, …

3.7.8 Yinelemeli Komutlar

- Baştan bilinebilen veya bilinemeyen bir sayı kadar tekrarlayan komutlar.
- Döngüler, for, while, …
- Loop (döngü) gövdesindeki (body) komutların her çalıştırılışına iterasyon (iteration) denir.

3.8 Yan Etkili (Side Effects) İfadeler

- Bir ifadeyi (expression) değerlendirmenin (evaluation) birincil amacı, bir değer elde etmektir. Ancak bazı zorunlu (imperative) ve nesne yönelimli (OO) dillerde, bir ifadeyi değerlendirmenin, değişkenleri güncelleme yan etkisi olması mümkündür.

```
enum Gender {female, male};
Gender g;

if (getchar(f) == 'F') g = female;
else if (getchar(f) == 'M') g = male;
else ...
```

3.8 Yan Etkili (Side Effects) İfadeler

- Kısaca yan etki bit state'i (durum, hal) değiştirmektir.
  - Bir değişkenin değerini değiştirmek
  - Diske veri yazmak
  - Bir butonu tıklanabilir / tıklanamaz olarak güncellemek.

```haskell
-- Haskell

f x = x ^ 2 -- S.E. yok
```

```c
// C

int i = 3;

int sonraki() {
    return ++i; // S.E. Var
}
```