

1906002132015

Programlama Dilleri Temelleri

BAİBÜ Bilgisayar Müh.

Dr. Öğr. Üyesi. İsmail Hakkı Parlak

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

Ders 2: Değerler (Values) ve Veri Tipleri (Types)

- Değer (Value): Bir program tarafından işlenebilen herhangi bir varlıktır.
 - Saklanabilirler (store), değerlendirilebilirler (evaluate), argüman olarak kullanılabilirler (arguments), fonksiyonlardan döndürülebilirler (return), ...
- 2.1 Veri tipleri (Types): Aynı türe ait değerler.
 - C: int, float, array, pointer, char, struct, union, ...
 - Java: boolean, string, object, ...
 - Haskell: int, float, list, tuple, ...
- $v \in T$
- v : değer, T : veri tipi

Değerler (Values) ve Veri Tipleri (Types)

- Her değer seti bir Tip olmak zorunda değildir. Tip üzerinde tanımlı işlem(ler), o tipe ait tüm değerler üzerinde benzer şekilde çalışmalıdır.
- $\{false, true\} \rightarrow$ işlemler: *and, or, not*
- $\{..., -2, -1, 0, +1, +2, ...\} \rightarrow$ işlemler: $+$, $-$, $*$, $/$, ...
- $\{13, true, Ekim\} \rightarrow$ işlemler: ?

2.2 Primitif (Primitive) Tipler

- Daha basit türlere parçalanamayan tiplerdir.
- Her programlama dilinde dilde yerleşik temel primitif tipler bulunur.
- C: int, float, char, pointer
- Python: int, boolean, string
- Nicelik (Cardinality): T 'de olabilecek tüm farklı elemanların toplam sayısı. $\#T$ olarak gösterilir.
- $\#Boolean = 2$
- $\#Character = 256$
- $\#Integer = 2^{32}$ (4 Byte = 32 Bit)

Kullanıcı Tanımlı (User Defined) Primitif Tipler

- Numaralandırılmış (Enumerated) tipler:

```
enum Sonuc {  
    OK, FAIL, PENDING  
}  
  
public class Test {  
  
    public static void main(String[] args) {  
        Sonuc s = Sonuc.PENDING;  
        // ...  
        if (s == Sonuc.OK) {  
            // ...  
        }  
    }  
}
```

2.3 Kompozit (Bileşik) Tipler

- Kompozit tipler, kompozit değerlerden oluşan tiplerdir.
- Kompozit değerler daha basit değerlerin bir araya getirilmesiyle oluşur.
- struct, tuple, array, object, list, ...
- Kompozit değerler aşağıdaki yapısal kavramlardan oluşur:
 - Kartezyen çarpımlar (tuple)
 - Eşleştirmeler (array)
 - Ayrık bileşimler (cebirsel tipler, objeler)
 - Özyinelemeli tipler (list, tree)

2.3.1 Kartezyen Çarpımlar

- Struct, tuple, ...
- $S \times T = \{ (x, y) \mid x \in S; y \in T \}$
- $\#(S \times T) = \#S \times \#T$
- C:

```
struct Ogrenci {  
    int no;  
    char isim[20];  
};
```

Ogrenci = int × char[20]

- Python:

```
ogr = (123456, "Adem")
```

ogr = int × string

2.3.2 Eşleştirmeler (Mappings)

- $m: S \rightarrow T$

m , S kümesindeki x değerini T kümesindeki y değeri ile eşleştirir. $y = m(x)$

- $\#(S \rightarrow T) = \#T \wedge \#S$

$S = \{x1, x2\}, T = \{y1, y2, y3\} \Rightarrow S \rightarrow T = [\{x1 \rightarrow y1, x2 \rightarrow y1\}, \{x1 \rightarrow y1, x2 \rightarrow y2\}, \{x1 \rightarrow y1, x2 \rightarrow y3\}], \{x1 \rightarrow y2, x2 \rightarrow y1\}, \{x1 \rightarrow y2, x2 \rightarrow y2\}, \dots, \{x1 \rightarrow y3, x2 \rightarrow y3\}]$

Diziler (Arrays)

- Diziler endekslenmiş sıralı bileşenlerden oluşan yapıdır.
- $S \rightarrow T$ (S: index, T: değer)
- $\#S$: *Dizinin uzunluğu*
- `int a[3] = { 3, 5, 2};`
- $a \in (\{0, 1, 2\} \rightarrow \text{int})$

Fonksiyonlar (Functions)

```
def is_even(num):  
    if num % 2 == 0:  
        return True  
    else:  
        return False
```

- Integer → Boolean

```
def is_bigger(num1, num2):  
    if num1 > num2:  
        return True  
    else:  
        return False
```

- Integer x Integer → Boolean

2.3.3 Ayırık Bileşimler (Disjoint Unions)

- $S+T = \{ \textit{left } x \mid x \in S \} \cup \{ \textit{right } y \mid y \in T \}$
- $S = \{ 'a', 'b' \}, T = \{ 'a', 'b', 'c' \}$
 $S + T = \{ \textit{left } 'a', \textit{left } 'b', \textit{right } 'a', \textit{right } 'b', \textit{right } 'c' \}$
- $\#(S + T) = \#S + \#T$

```
enum Accuracy {exact, inexact};
```

```
struct Number {  
    Accuracy acc;  
    union {  
        int ival;  
        float rval;  
    } content;  
};
```

2.4 Özyinelemeli (Recursive) Tipler

- Tanımında kendisi geçer.
- Listeler (Lists): Değerler silsilesidir. Değerler aynı tipteyse homojen, değilse heterojendirler.
- İçinde eleman olmayan listeye boş liste denir.
- Listenin içindeki eleman sayısı listenin uzunluğudur.



2.4.1 Listeler (Lists)

- Bazı liste işlemleri:
 - Baş elemanı seçme
 - Son elemanı seçme
 - Uzunluğunu öğrenme
 - Boş olma durumunu öğrenme
 - Başka bir liste ile birleştirme
- $\text{Integer-List} = \text{nil Unit} + \text{cons}(\text{Integer} \times \text{Integer-List})$
- $\text{Integer-List} = \{\text{nil}()\} \cup \{\text{cons}(i, l) \mid i \in \text{Integer}; l \in \text{Integer-List}\}$
- *nil*: boş liste; *cons*: boş olmayan liste

2.4.2 Stringler

- String = karakter dizisi.
- "Hello World", "X", ""
- Stringler primitif tip, liste veya karakter dizisi olarak sınıflandırılabilir.
- Çoğu modern programlama dilinde String'lere dair uzunluk bulma, karşılaştırma, karakter veya alt String seçme, birleştirme gibi işlemler mevcuttur.

2.5 Tip Sistemleri (Type Systems)

- Bir boolean ile bir stringi çarparsak ne olur? Tip hatası (type error)!
- Bir programlama dilinin tip sistemi, değerleri tipler halinde gruplar.
- Tip sistemleri programcının veriyi efektif şekilde ifade etmesini sağlar.
- Operasyonlar (aritmetik, mantıksal, array erişimi, vb.) gerçekleştirilmeden önce tip kontrolü (type checking) yapılır.

2.5.1 Statik vs Dinamik Tipler

- Statik tipli bir dilde, her değişken ve her ifadenin sabit bir türü vardır. Bu bilgiler kullanılarak, tüm işlenenler **derleme zamanında (compile time)** tip denetimine tabi tutulur. Hızlıdır, güvenlidir, katıdır. C, C++, Java, ...
- Dinamik tipli bir dilde, değerlerin sabit türleri vardır, ancak değişkenlerin ve ifadelerin sabit türleri yoktur. **Çalışma zamanında (run time)** denetlenir. Yavaştır, güvenilir değildir, esnektir. PHP, Python, JavaScript, ...

2.6 İfadeler (Expressions)

- İfade, değerlendirildiğinde bir değer elde edilen yapıdır.
- İfadeler çeşitli şekillerde oluşturulabilir.
 - değişmezler (literals)
 - yapılar (constructions)
 - fonksiyon çağrıları (function calls)
 - koşullu ifadeler (conditional expressions)
 - yinelemeli ifadeler (iterative expressions)
 - sabit ve değişken erişimleri (constant and variable accesses)

2.6 İfadeler

- 2.6.1 Literaller (Değişmezler): Bir tipteki sabit değerler.
 - `true`, `"naber?"`, `3.14`, `-12`, `'Y'`
- 2.6.2 Yapılar (Constructions): Komponentlerin bir araya gelmesiyle oluşmuş kompozit ifadelerdir.
 - `{30, Aylar.Şubat}`
 - `int numaralar[] = {5, 3, 2, 7}`
 - `new Person(35, "Dante")`

2.6 İfadeler

- 2.6.3 Fonksiyon Çağrılarını: Bir argüman listesine bir metot uygulayarak bir sonuç elde eder.
 - `int yas = yasiniHesapla(1, 1, 2000);`
- Operatörler de bir fonksiyon olarak kabul edilebilir.
 - $a * b + c \rightarrow +(c, (*(a, b)))$
 - `!, %, &&, ...`
- C++, Haskell gibi diller kullanıcısına, operatörlere yeni anlamlar yüklemesine olanak sağlar.

2.6 İfadeler

- 2.6.4 Koşullu (Conditional) İfadeler: Bir koşula bağlı olan bir değer hesaplar.
- `String hava = sicaklik < 15 ? "Soğuk" : "Sıcak";`
- `if (sicaklik < 15) { ... }`
- `switch (sonuc) {
 case 0:
 return "Yazi";
 case 1:
 return "Tura";
 default:
 return "Dik";
}`

2.6 İfadeler

- 2.6.5 Yinelemeli (Iterative) İfadeler: Bir dizi değer üzerinde hesaplama yaparlar ve bir sonuç dönerler.
- `[x**2 for x in range(10)]`
- 2.6.6 Sabit ve Değişken Erişimleri (Constant and Variable Accesses): Kullanıcının tanımladığı sabitler ve değişkenler, değerlendirildiklerinde kendi içeriklerini verirler.
- `int sayi = 3;`
- `#define PI 3.14`