6. Veri Tipleri

Bir veri tipi, verinin alabileceği değerleri ve bu değerler üzerinde tanımlanmış olan işlemleri belirtir. Bir programlama dili gerçek dünya problemlerindeki nesnelere karşılık gelecek çeşitlilikte veri tipleri ve yapılarına sahip olmalıdır.

İlk ortaya çıkan programlama dillerinde, örneğin Fortran90 öncesi sürümlerde, bağlı listeler ve ikili ağaçlar dizilerle oluşturulmaktaydı. COBOL ile kullanıcılar yüksek hassasiyette ondalık sayılar ve kayıt tipinde veri tanımlayabildiler. PL/I ile kayan noktalı sayılar ve tamsayılar için de kesinlik tanımı gelmiş oldu. Daha sonra bu Fortran ve Ada'ya da adapte edildi. PL/I'da birçok veri tipi bulunsa da asıl gelişme Algol 68 ile geldi: Birkaç basit tip ve birkaç esnek veri yapısı ile kullanıcının herbir ihtiyacı için ayrı bir veri yapısı tanımlayabilmesi mümkün oldu.

Kullanıcı tanımlı veri tiplerinden bir adım ötesi 80'lerin ortalarından beri tasarlanan dillerin çoğunda desteği olan soyut veri tipleridir. Yüksek seviyeli dillerin desteklediği tüm veri tipleri soyut tiplerdir. Veri üzerinde tanımlı işlemlerin detayları kullanıcıdan gizlenmiştir.

Buyurgan dillerdeki iki temel yapısal veri tipi dizi (array) ve kayıttır (record). Listeler, fonksiyonel programlama dillerinin ana unsurudur. Fonksiyonel programlamanın artan popülerliği Python ve C# gibi buyurgan dillere de listelerin eklenmesine sebep olmuştur.

Değişkenleri “tanımlayıcılar” açısından düşünmek daha uygun olur. Tanımlayıcı bir değişkenin özelliklerinin tamamıdır. Bir gerçekleştirimde tanımlayıcı, değişkenin özelliklerini tutan bellek alanıdır. Tüm özellikler statik ise tanımlayıcılara sadece derleme zamanında ihtiyaç duyulur. Dinamik özellikler için ise tanımlayıcıların bir kısmı ya da tamamı çalışma zamanında kullanılır.

Değişken kelimesi kullanılırken dikkatli olunmalıdır. Buyurgan dilleri kullanan biri bir ismi bir değişken olarak düşünebilir. Bazı dillerde isimler bir tipe sahip olmayabilirler. İsim kavramı değişkenlerin bir özelliği olarak düşünülmelidir.

6.1 Basit Veri Tipleri

Başka veri tipleri kullanılarak tanımlanamayan veri tiplerine ilkel veri tipleri denir. Neredeyse tüm programlama dilleri bir dizi ilkel veri tipi tanımına sahiptir.

6.1.1. Nümerik Tipler

6.1.1.1. Integer

Integer en bilinen ilkel nümerik veri tipidir. Bazı dillerde çeşitli boyutlarda Integer tanımları desteklenmektedir. Örneğin Java'da 4 işaretli Integer tipi bulunur: byte, short, int ve long. C++ ve C#'ta işaretsiz tamsayı da bulunmaktadır.

Integer tipler genelde donanım tarafından desteklenir. Bu durumun dışında kalan bir örnek Python'daki long integer tipidir. Bu tipteki değerler sonsuz uzunlukta olabilir. Bir tamsayı işlemi sonucunda oluşan, Integer tipine sığmayacak büyüklükteki veri long integer tipinde saklanır.

6.1.1.2. Kayan Noktalı Sayılar

Reel sayıları temsil ederler, çoğu zaman yaklaşık değerler kullanılır. Örneğin pi ve e sayıları kayan noktalı sayılarla doğru şekilde gösterilemezler. Kayan noktalı veri tipi ile ilgili problem, aritmetik işlemler sonucunda kesinlik kaybı oluşmasıdır.

Çoğu dilde iki kayan noktalı tip bulunur ve genelllikle float ve double olarak adlandırılırlar. Float 4 byte standart uzunlukta olup, double en az float'ın iki katı uzunluktadır.

6.1.1.3. Complex

Bazı dillerde (Fortran ve Python) karmaşık veri tipi desteklenmektedir. Bu diller karmaşık sayı aritmetiğini de desteklerler.

6.1.1.4. Decimal

Ondalıklı sayıların kesin bir şekilde ifade edildiği tiplerdir. COBOL, C#, F# dillerinde tanımlıdır. Kayan noktalı sayılarla kesin bir şekilde ifade edilemeyen sayılar bu veri tipi ile ifade edilebilir.

6.1.1.5. Boolean

Tüm tiplerin en basiti Boolean veri tipidir çünkü sadece iki değer alır. İlk defa Algol 60'ta ortaya çıkmıştır. O zamandan beri çoğugenel amaçlı dilde bu veri tipi bulunmaktadır. C89'da sıfır false, sıfırdan farklı değerler de true olarak işlem görür. C99 ve C++'ta Boolean veri tipi bulunmasına rağmen, nümerik ifadeler de Boolean'mış gibi kullanılabilir. Java ve C#'ta bu seçenek yoktur.

6.1.1.6. Karakter

Karakterler bilgisayarda nümerik olarak 8-bitlik ASCII (American Standard Code of Information Interchange) kodlarıyla tutulurlar. 0-127 arası kodlarla 128 farklı karakter belirtilebilir. ISO 8859-1 başka bir 8-bitlik kod standardıdır ancak 256 farklı karakterin gösterimine izin vermektedir. Ada95+ sürümleri, bu kodu kullanmaktadır.

Gelişen teknoloji ve değişen gereksinimler karşısında ASCII karakterler yetersiz kalmıştır. 1991'de Unicode Consortium 16-bitlik UCS-2 standardını yayımlamıştır. Unicode çoğu doğal dilin karakterlerini kapsar. Java, JavaScript, Perl, Python, C# ve F# Unicode karakterlerini kullanır.

91'den sonra Unicode Consortium ve ISO birlikte UCS-4 veya UTF-32 adlı 4-byte'lık karakter setlerini geliştirmişlerdir.

6.1.2. String

Bir dizi karakter, string olarak adlandırılır. En bilinen string işlemleri atama, birleştirme (catenation), alt string referansı, karşılaştırma ve desen eşlemedir (pattern matching).

Stringlerde atama ve karşılaştırma işlemleri farklı uzunluktaki stringler üzerinde yapılırsa olay karmaşıklaşır. Örneğin uzun bir string daha kısa bir stringe atandığında ya da tam tersi olduğunda oluşabilecek sorunlar programcının üreteceği basit ve mantıklı çözümlerle giderilebilir.

Desen eşleme bazı dillerde doğrudan desteklenmiştir. Bazılarında bir fonksiyon veya sınıf kütüphanesi ile kullanılabilir.

Eğer String basit bir tip olarak tanımlanmadıysa bir karakter dizisi olarak tanımlanabilir. Bu yaklaşım C ve C++ dillerinde mevcuttur. Örn: char str[] = “apples”

C ve C++'ta bulunan string işlem fonksiyonları güvenli değildirler ve birçok programlama hatasına yol açabilirler.

Örn:
strcpy(dest,src);

Bu örnekte strcpy komutu ile, src stringi, dest stringine kopyalanacaktır. dest stringinin uzunluğunun 20, src'nin uzunluğunun da 50 karakter olduğunu varsayarsak, 50 karakterin 20'si dest'e geri kalan 30 karakter de dest'ten sonraki bellek bölgesine yazılabilir. Strcpy fonksiyonu dest'in uzunluğunu bilmez. Dolayısıyla kopyalama işleminin dest'ten sonraki bellek bölgesine yapılmayacağının bir garantisini veremez. C++ programcıları standart kütüphanedeki string sınıfını kullanmalı, C string kütüphanesini ve char arrayleri kullanmaktan kaçınmalıdır.

Java'da stringler String (immutable) sınıfı ile desteklenir. Bu sınıfta sabit string değerleri vardır. Bir de StringBuffer sınıfı vardır ve bu sınıfta değişebilir stringler desteklenir.

Python'da string basit bir tiptir. Arama, değiştirme, metinde bir karakterin yerini bulma, birleştirme gibi bir çok string işlemi desteklenir. String tipi tıpkı Java'daki gibi değişmez (immutable)' dir. Immutable kavramı için bu adrese bakabilirsiniz: http://www.ilkayilknur.com/immutable-nesne-kavrami/

6.1.2.1. String Uzunluk Seçenekleri

1) Bir string yaratılırken belirlenen uzunluk sabitse bunlar statik uzunluklu stringler olarak adlandırılır. Java'nın String sınıfı, Python, C++ standat kütüphanesi, C# ve F# için .NET sınıf kütüphanesi, sabit uzunluklu stringleri desteklerler.

2) Bir stringin alabileceği maksimum uzunluk sabit ise fakat string değişken uzunluk alabiliyorsa bunlara sınırlı dinamik uzunluklu stringler diyoruz. C'de bu tip stringler desteklenir.

3) Maksimum uzunluk sınırı olmaksızın tanımlanabilen stringler dinamik uzunluklu stringler olarak adlandırılırlar. JavaScript, Perl, standart C++ kütüphanesi bu tip stringleri destekler. Bu tip string kullanımında dinamik bellek tahsis etme ve geri bırakmanın getirdiği ek yük vardır ama yüksek derecede esneklik sağlar.

6.2 Kullanıcı Tanımlı Sıralı (Ordinal) Tipler

Programlama dilleri tarafından desteklenen iki tane kullanıcı tanımlı sıralı tip vadır: Enumeration ve Subrange

6.2.1. Enumeration

C# örneği: enum days {Mon, Tue, Wed, Thu, Fri, Sat, Sun};

{} içindeki sabitlere enumeration sabitleri denir ve o tipteki değişkenin alabileceği değerlerin kümesidir. Enumeration sabitleri dolaylı yoldan indekslenir ve C# için bu indeks değerleri sıfırdan başlamaktadır.

days today= Tue;
today++; // today Wed oldu
Eğer dil izin veriyorsa today=5 dediğimizde today'in Sat olmasını bekleriz. C++'ta buna izin verilmemektedir.

6.2.2. Subrange

Subrange (alt aralık) kullanımı, yeni bir tip tanımlamadan ziyade varolan tiplerin alt kümeleri için yeni adlar tanımlamaktır. Pascal ve Ada dillerinde subrange tanımı desteklenmektedir. Örn: 12..14 integer için bir subrange'dir.

```
Type Days is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
subtype Weekdays is range Mon..Fri;
subtype Index is Integer range 1..100;
```

Ana kümede tanımlı bütün işlemler alt kümede de tanımlıdır. Sadece aralığın dışında bir atama yapılmasına izin verilmez.

```
Day1 : Days;
Day2 : Weekdays;
…
Day2 = Day1;  ==> Day1, Sat veya Sun değilse bu işlem kabul edilir.
```

6.3. Diziler (Arrays)

Bir dizi homojen verilerin bir araya geldiği, herbir elemanın dizinin ilk elemanına göre dizi içindeki yeri ile tanımlandığı veri yapısıdır.

Çoğu dilde (C, C++, Java, Ada, C#) dizinin tüm elemanları aynı tipte olmak zorundadır. Bu dillerde işaretçiler hep aynı tipte veriyi işaret etmek zorundadır.

Diğer bazı dillerde (JavaScript, Ruby, Python gibi) değişkenler bir nesneye referans verirken tip sınırı yoktur. Yine de bu diziler tek tip veri tutarlar ancak dizi elemanları farklı nesnelere referans verebilirler. Dizi yine homojendir.

Bazı dillerde dizinin bir elemanına ulaşmak içindizi değişkeninden sonra () içine elemanın indisi yazılır. Örneğin Ada dilinde: sum:= sum + B(I);

Parantezler aynı zamanda fonksiyon çağrısı için de kullanıldıklarından kafa karıştırıcır ve okunabilirliği azaltır. Fortran ve Ada dışındaki çoğu dilde dizi indisleri [] arasına yazılır.

6.4. Kayıt (Record)

Kayıt tipi, farklı tiplerdeki değişkenlerin tek tip altında tanımlanması ile oluşur. Bir kayıt tanımlarken kullanılan değişken tipleri basit veya karmaşık olabilir. Örneğin bir öğrenciye ait olarak tutulacak bilgiler, ad soyad (string), no (int), genel not ortalaması (float) vb. olabilir. C, C++ ve C# dillerinde struct denilen veri tipi ile kayıt tanımlaması yapılır.

Örn. COBOL dilinde:

```
01 EMPLOYEE-RECORD.
        02 EMPLOYEE-NAME.
                05 FIRST PICTURE IS X(20).
                05 MIDDLE PICTURE IS X(10).
                05 LAST PICTURE IS X(20).
        02 HOURLY-RATE PICTURE IS 99V99.
```

COBOL dilinde satır başındaki numaralar seviye göstermektedir. Bir satırın kendinden sonraki satırda daha büyük bir seviye numarası varsa bir kayıt olduğu anlamına gelir. Yukarıdaki örnekte iç içe iki kayıt yapısı görmekteyiz. EMPLOYEE-NAME ve HOURLY-RATE sahalarına sahip bir EMPLOYEE-RECORD kaydı ve FIRST, MIDDLE ve LAST sahalarına sahip bir EMPLOYEE-NAME kaydı bulunmaktadır. PICTURE kelimesi bellek bölgesinin hangi formatta olduğunu belirtmeden önce kullanılır. X(20) ile belirtilen 20 karakterlik bir alfanumerik veridir. 99V99 ise iki tamsayı iki de ondalık basamağı olan nümerik veriyi ifade eder. Bu COBOL kaydında MIDDLE sahasına erişmek için MIDDLE OF EMPLOYEE-NAME OF EMPLOYEE-RECORD yazmak gerekir.

Yukarıda COBOL'da tanımlanmış olan kayıt Ada dilinde şu şekilde tanımlanır:

```
type Employee_Name_Type is record
        First : String (1..20);
        Middle : String (1..10);
        Last : String (1..20);
end record;
type Employee_Record_Type is record
        Employee_Name: Employee_Name_Type;
        Hourly_Rate: Float;
end record;
Employee_Record: Employee_Record_Type;
```

Middle sahasına erişmek için Ada'da Employee_Record.Employee_Name.Middle ifadesini kullanmak gerekir.
C ve C++'ta da struct içindeki veriye . işareti kullanılarak erişilebilir.

Java ve C#'ta kayıtlar sınıf olarak tanımlanabilir.

6.5. Demet (Tuple)

Kayıt tipine benzeyen ancak elemanların isimlerinin olmadığı bir veri yapısıdır. Python'da değiştirilemez (immutable) tuple tipi bulunur.

myTuple = (3, 5.8, 'elma') örneğinde olduğu gibi demete ailt elemanlar aynı tipte olmak zorunda değildir. myTuple[1] ile demetin ilk elemanına ulaşırız. Burada indislerin 1'den başladığı unutulmamalıdır.

6.6. Liste (List)

Listeler ilk defa ilk fonksiyonel programlama dili olan LISP'te desteklenmiştir. Fonksiyonel dillerin daima bir parçası olmuşlardır ancak son yıllarda buyurgan dillerde de kullanılmaktadır.

Scheme ve LISP'teki listeler şöyle tanımlanır: (A B C D)
İç içe liste tanımlamak için: (A (B C) D)

Python'da myList = [3, 5.8, 'elma'] ifadesi dizi tanımlama için bir örnektir. Indisler sıfırdan başlamaktadır. x= myList[1] ifadesi ile x'in değeri 5.8 olur. Listeden bir eleman silmek için del komutu kullanılır. Örneğin, del myList[1] ifadesi ile dizinin ikinci elemanı silinir.

6.7. Union

Union kullanarak, aynı bellek bölgesine birden fazla değişken adı ile ulaşabilmek mümkün olmaktadır. Union'ın boyutu içerdiği değişkenlerin en büyüğü olarak belirlenir. C dilinde bir aliasing örneği olarak Union kullanımından 5. ünitede bahsetmiştik.

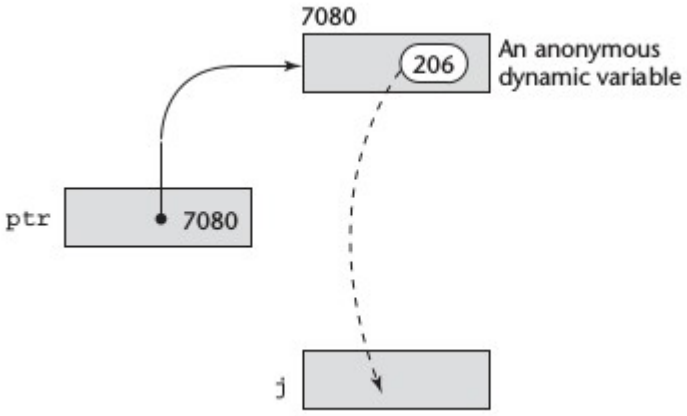
6.8. İşaretçi ve Referans Tipleri

İşaretçi tipi değişkenler bellek adresleri ve nil aralığında değerler alabilmektedirler. Nil, geçerli bir adres değeri değildir ve bir işaretçinin o anda bir bellek hücresine işaret edemeyeceği anlamına gelir. İşaretçiler iki farklı amaç için tasarlanmışlardır. Bunlardan biri dolaylı adresleme diğeri de dinamik bellek yönetimidir.

İşaretçiler dizi ve kayıt gibi yapısal tipler değillerdir. Veri tutmaktan ziyade başka değişkenlere referans verirler. Burada iki tip değişken ortaya çıkıyor: referans tipi ve değer tipi.

Bir dilde işaretçilerin varlığı yazılabilirlik açısından olumludur. İşaretçi tipini destekleyen dillerde iki temel işlemi bulunur: atama ve referanstan ayrılma. Atama işlemi ile işaretçiye kullanılan bir bellek adresi atanmış olur.

j = *ptr ifadesinin şekilsel gösterimi yandadır. ptr içindeki adres değerinin 7080 olduğunu varsayalım. *ptr kullanımı ile ptr' nin işaret ettiği bellek bölgesinin içindeki değere ulaşmış oluyoruz.



Bir işaretçi kayıt tipinde bir değişkeni işaret ediyorsa kaydın sahalarına ulaşmak için C ve C++' ta iki yol bulunmaktadır: Kayıtta age diye bir saha olduğunu varsayıyoruz.
p->age veya (*ptr).age kullanımları ile istenen sahaya ulaşmış oluruz.
Ada dilinde aynı şeyi yapmak için p.age kullanımı yeterli olmaktadır.

6.8.1. İşareçiler ile İlgili Sorunlar

İşaretçilerin kullanıldığı ilk yüksek seviyeli dil PL/I' dır. İşaretçiler hem hem heap-dinamik değişkenlere hem de diğer program değişkenlerine işaret edebilirlerdi. PL/I'daki işaretçilerin esnekliği çok yüksekti ancak kullanımları birçok programlama hatasına sebep olabilmekteydi. Bu sorunlardan bazıları, sonradan ortaya çıkan dillerdeki işaretçi kullanım sorunları gibiydi. Bu tip problemleri en aza indirebilmek amacıyla Java'da işaretçilerin yerini referans tipleri almıştır.

6.8.1.1. Sallanan İşaretçiler (Dangling Pointers)

Bir işaretçi, artık geçerli olmayan bir bellek bölgesini işaret ediyorsabuna sallanan işaretçi denir. Sallanan işaretçiler birçok sebepten dolayı tehlikelidirler. Örneğin artık kullanılmayan bellek bölgesi yeniden bir değişkene tahsis edilmiş olabilir. Eğer yeni değişkenin tipi eski değişkenin tipinden farklı ise sallanan işaretçi üzerinden tip kontrolü geçersiz hale gelmektedir. Yeni değişkenin

tipi eskisi ile aynı bile olsa yeni değer işaretçinin daha önce gösterdiği değerden farklı olacaktır. Eğer işaretçi o değişkenin değerini de değiştirmek için kullanılıyorsa yeni değişkenin değeri bozulacaktır. Son olarak o bellek bölgesi geçici olarak sistem tarafından kullanılıyor ise o bölgede yapılacak değişiklik sistemin hatalı çalışmasına sebep olabilir.

Çoğu dilde aşağıdaki şekilde sallanan işaretçiler oluşmaktadır:
1) Yeni bir heap-dinamik değişken yaratılır ve p1 işaretçisi onu işaret etmek üzere ayarlanır.
2) p1' in değeri p2 işaretçisine atanır.
3) p1' in gösterdiği değişken (muhtemelen p1'e nil atanarak) belleğe geri bırakılır. Fakat p2 değiştirilmez. P2 bu durumda sallanan işaretçi olur. Eğer belleğe geri bırakma işi p1 ve p2 yi değiştirmeden yapıldıysa işaretçilerin ikisi birden sallanan işaretçi olurlar. Bu aynı zamanda aliasing problemidir çünkü p1 ve p2 birbirinin aliasıdır.

Örnek C++ kodu:

```
int * arrayPtr1;
int * arrayPtr2 = new int[100];
arrayPtr1 = arrayPtr2;
delete [] arrayPtr2; // burada arrayPtr1 sallanan işaretçi olmaktadır.
```

6.8.1.2. Kayıp Heap-Dinamik Değişkenler

Bellekten yer ayrılmış bir değişkeneerişilemem problemidir. Bu durumda kalan değişkenler çöp (garbage) olarak adlandırılırlar. Çünkü asıl amaçları için kullanılmamaktadırlar ve yeni bir kullanım için de onlara yeniden bir bellek bölgesi tahsis edilemez. Kayıp değişkenler genellikle aşağıdaki işlemler sonucunda oluşurlar.
1) İşaretçi p1, yeni yaratılan bir heap-dinamik değişkeni işaret edecek şekilde ayarlanır.
2) Daha sonra p1 başka bir heap-dinamik değişkeni işaret edecek şekilde ayarlanır.
Bu durumda birinci değişkeni kaybetmiş oluruz. Bu duruma bazen bellek sızıntısı da denir.

6.8.2. C ve C++'ta İşaretçiler

Örn 1:

```
int *ptr;
int count, init;
…
ptr = &init;
count = *ptr;
```

İşaretçinin bir değişkeni işaret edebilmesi için işaretçiye aynı tipte bir değişkenin adresi atanmalıdır. & işareti bir değişkenin adının önüne yazılarak o değişkenin adres değeri elde edilir. İşaretçi adının önünde kullanılan * işareti ise işaretçinin gösterdiği bellek adresinde bulunan değere ulaşabilmeyi sağlar. * işaretine dereferencing işareti de denir.

Örn 2:

```
int list[10];
int *ptr;
ptr=list; // list[0]'ın adresi ptry'yr atandı çünkü bir dizinin adı aynı zamanda ilk elemanının adresidir.
```

*(ptr+1) ile list[1] aynı değere sahiptir
*(ptr+index) ile list[index] aynı değere sahiptir
*ptr[index] ile list[index] aynı değere sahiptir

ptr+index dediğimizde index değeri ptr' ye doğrudan eklenmez. Önce bellek hücresinin boyutu ile ölçeklendirilir. Örneğin eğer ptr 4 bellek hücresi büyüklüğünde bir bellek hücresini işaret ediyorsa index 4 ile çarpılıp ondan sonra ptr' ye eklenir.

C ve C++'ta void * tipinde işaretçi tanımlanabilir. Bu tipteki işaretçilere dereferencing uygulanamaz. Bir bellek bölgesindeki veriyi başka bir bellek bölgesine taşımak istediğimizi düşünelim. Bu işlemi bir fonksiyon içinde yapacaksak fonksiyona void * tipinde parametreler göndermek doğru olur. Bu durumda bellek bölgesindeki verirnin tipinden bağımsız olarak işlem yapılabilir.

6.8.3. Referans Tipi

İşaretçilere benzer bir tiptir. En temel farkları işaretçi bir bellek bölgesini işaret ederken referans bir nesneyi veya bellekteki bir değeri işaret eder. Dolayısıyla referans üzerinde aritmetik işlem yapmak mantıklı değildir.

C++'ta
```
int result = 0;
int &ref_result = result; //ref_result sabit işaretçidir.
. . .
ref_result = 100;
```

Tüm Java sınıf instance'ları referans değişkenleri ile işaret edilir. Bu, Java'daki tek referans kullanımıdır. Java sınıf instance'ları dolaylı olarak belleğe geri bırakıldığından Java'da sallanan referans olmaz.

C#'da hem Java referansları hem de C++ işaretçileri kullanılabilir ancak işaretçi kullanımından kaçınılması gerektiği belirtilir. İşaretçi içeren alt programlar unsafe tanımlayıcısı içermek zorundadır.

Small-talk, Python, Ruby ve Lua gibi nesne tabanlı programlama dillerinde tüm değişkenler referanstır, değerlerine doğrudan erişilmez.

6.9. Tip Kontrolü

Bir işlemde, işleme girenlerin tiplerinin birbiri ile uyumlu olup olmadığının kontrolüdür. Uyumlulukta işleme girenler ya o işlem için uygundur ya da programlama dili tarafından otomatik olarak uygun tipe dönüştürülür. Bu otomatik tip dönüştürme işine zorlama (coercion) diyoruz.

Örneğin Java'da bir integer ve bir float toplanırken integer değişken floata dönüşmeye zorlanır ve bir kayan nokta ekleme işlemi yapılır.