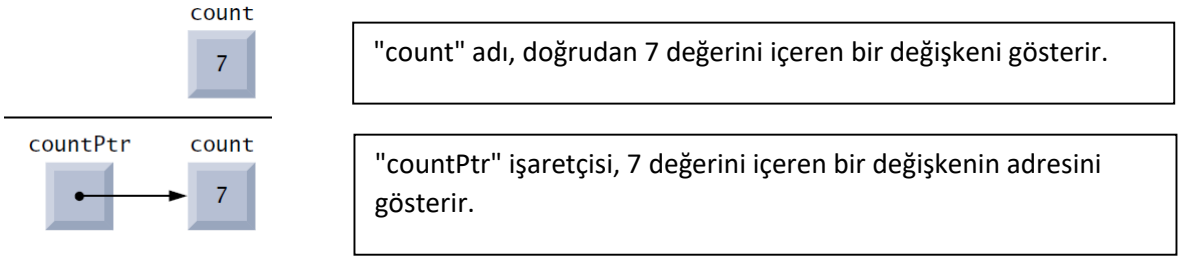
İşaretçiler (C Pointers)

Bu bölümde, C programlama dilinin en güçlü özelliklerinden biri olan işaretçiyi tartışacağız. İşaretçiler, C'nin ustalaşması en zor yetenekleri arasındadır. İşaretçiler, programların referansla geçiş gerçekleştirmesine, Fonksiyonlar arası ilişkide ve bağlantılı listeler, sıralar, yığınlar ve ağaçlar gibi yürütme sırasında büyüyüp küçülebilen dinamik veri yapıları oluşturmasına ve işlemesine olanak tanır.

İşaretçiler, değerleri bellek adresleri olan değişkenlerdir. Normalde, bir değişken doğrudan belirli bir değer içerir. Ancak bir işaretçi, belirli bir değer içeren bir değişkenin adresini içerir. Bu anlamda, bir değişken adı doğrudan bir değeri gösterir ama bir işaretçi dolaylı olarak bir değeri gösterir. (Şekil 7.1). Bir işaretçi aracılığıyla bir değere ulaşmaya dolaylı yol denir.



Şekil 7.1 Bir değişkene doğrudan ve dolaylı olarak gösterme.

İşaretçiler, tüm değişkenler gibi, kullanılmadan önce tanımlanmalıdır.

```
int *countPtr, count;
```

Bu tanım, "countPtr" değişkeninin int türünde olduğunu (yani, bir tamsayı işaretçisi) olduğunu belirtir. Bir tanımda * bu şekilde kullanıldığında, tanımlanan değişkenin bir işaretçi olduğunu gösterir. İşaretçiler, herhangi bir türdeki nesneleri işaret edecek şekilde tanımlanabilir.

İşaretçiler tanımlandıklarında başlatılmalıdır. Bir işaretçi NULL, 0 veya bir adres olarak başlatılabilir. NULL değerine sahip bir işaretçi hiçbir şeyi işaret etmez. NULL, <stddef.h> üstbilgisinde (ve <stdio.h> gibi diğer birçok üstbilgide) tanımlanan sembolik bir sabittir. Bir işaretçiyi 0 olarak başlatmak, bir işaretçiyi NULL olarak başlatmakla eşdeğerdir, ancak değişkenin bir işaretçi türünde olduğu gerçeğini vurguladığı için NULL tercih edilir. 0 atandığında, önce uygun türde bir işaretçiye dönüştürülür. 0 değeri, doğrudan bir işaretçi değişkenine atanabilecek tek tamsayı değeridir.

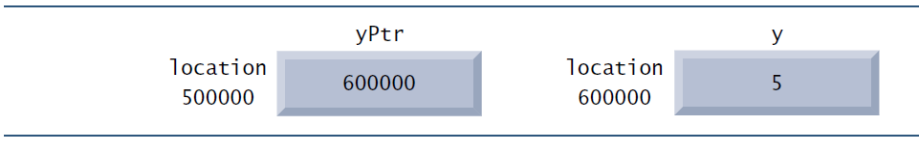
Adres (&) Operatörü, işleneninin adresini döndüren tekli bir operatördür. Örneğin, şu tanımları varsayarsak

```
int y = 5;
int *yPtr;
```

Aşağıdaki ifade, y değişkeninin adresini yPtr işaretçi değişkenine atar.

```
yPtr = &y;
```

Şekil 7.3, y tamsayı değişkeninin 600000 konumunda ve yPtr işaretçi değişkeninin 500000 konumunda saklandığını varsayarak, işaretçinin bellekteki temsilini gösterir. Adres operatörünün işleneni bir değişken olmalıdır; adres operatörü sabitlere veya ifadelere uygulanamaz.



Şekil 7.3 Bellekte y ve yPtr gösterimi.

Genellikle dolaylı operatör veya referans kaldırma operatörü olarak adlandırılan tekli * operatörü, işleneninin (yani bir işaretçinin) işaret ettiği nesnenin değerini döndürür. Örneğin, aşağıdaki ifade y değişkeninin değerini yazdırır. *'ın bu şekilde kullanılmasına bir işaretçinin referansının kaldırılması denir.

```
printf("%d", *yPtr);
```

Şekil 7.4, işaretçi (pointer) operatörü (& ve *) kullanımını göstermektedir. printf dönüştürme belirleyicisi %p, çoğu platformda bellek konumunu onaltılık (hexadesimal) bir tamsayı olarak verir. Programın çıktısında, a'nın adresinin ve aPtr'nin değerinin çıktıda aynı olduğuna dikkat edin, böylece a'nın adresinin gerçekten aPtr işaretçi değişkenine atandığı görülür (satır 8). & (ampersan) ve * işleçleri (operatörleri) birbirinin tümleyenidir; her ikisi de aPtr'ye herhangi bir sırayla (satır 18) arka arkaya uygulandığında, aynı sonuç yazdırılır. Burada * operatörünün a değişkeninin bulunduğu adresteki veriyi yani a değerini yazdırdığına dikkat edelim. Çıktıda gösterilen adresler sistemler arasında değişiklik gösterecektir. Şekil 7.5, bu noktaya tanıtılan operatörlerin önceliğini ve ilişkilendirilebilirliğini listelemektedir.

```
1  // Fig. 7.4: fig07_04.c
2  // Using the & and * pointer operators.
3  #include <stdio.h>
4
5  int main(void)
6  {
7     int a = 7;
8     int *aPtr = &a; // set aPtr to the address of a
9
10    printf("The address of a is %p"
11           "\nThe value of aPtr is %p", &a, aPtr);
12
13    printf("\n\nThe value of a is %d"
14           "\nThe value of *aPtr is %d", a, *aPtr);
15
16    printf("\n\nShowing that * and & are complements of "
17           "each other\n&*aPtr = %p"
18           "\n*&aPtr = %p\n", &*aPtr, *&aPtr);
19 }
```

```
The address of a is 0028FEC0
The value of aPtr is 0028FEC0

The value of a is 7
The value of *aPtr is 7

Showing that * and & are complements of each other
&*aPtr = 0028FEC0
*&aPtr = 0028FEC0
```

Şekil 7.4 & ve * işaretçi işleçlerini kullanma.

Operators		
() [] ++ *(postfix)* -- *(postfix)*	soldan sağa	Son ek
+ - ++ -- ! * & *(type)*	sağdan sola	tekli
* / %	soldan sağa	çarpımsal
+ -	soldan sağa	additive
< <= > s>=	soldan sağa	ilişkisel
== !=	soldan sağa	eşitlik
&&	soldan sağa	mantıksal VE
||	soldan sağa	mantıksal VEYA
?:	sağdan sola	koşullu
= += -= *= /= %=	sağdan sola	atama
,	soldan sağa	virgül

Şekil 7.5 Şimdiye kadar tartışılan operatörlerin önceliği ve ilişkilendirilebilirliği.

Bir fonksiyona bağımsız değişkenleri iletmenin iki yolu vardır: değere göre geçirme ve referansa göre geçirme. Ancak, C'deki tüm bağımsız değişkenler değere göre iletilir. Ama C'de, referansa göre geçişi gerçekleştirmek için işaretçiler ve dolaylı operatör kullanılır. Değiştirilmesi gereken bağımsız değişkenlere sahip bir fonksiyon çağrılırken, bağımsız değişkenlerin adresleri iletilir.

Bu normalde, değeri değiştirilecek değişkene adres operatörünün (&) uygulanmasıyla gerçekleştirilir. & operatörü kullanılarak diziler aktarılmaz çünkü C otomatik olarak dizinin belleğindeki başlangıç konumuna geçer (bir dizinin adı &diziAdı[0] ile eşdeğerdir). Bir değişkenin adresi bir fonksiyona iletildiğinde, çağıranın belleğindeki o konumdaki değeri değiştirmek için fonksiyonda dolaylı operatör (*) kullanılabilir.

Şekil. 7.6 ve 7.7, bir tamsayının küpünü alan bir fonksiyonun iki sürümünü sunar—cubeByValue ve cubeByReference. Şekil 7.6'daki 14. satır, değişken sayısını değere göre cubeByValue fonksiyonuna iletir. cubeByValue fonksiyonu bağımsız değişkeninin küplerini alır ve yeni değeri bir dönüş ifadesi kullanarak ana değere geri iletir. Yeni değer main'deki numaraya atanır (satır 14).

```c
// Fig. 7.6: fig07_06.c
// Cube a variable using pass-by-value.
#include <stdio.h>

int cubeByValue(int n); // prototype

int main(void)
{
   int number = 5; // initialize number

   printf("The original value of number is %d", number);

   // pass number by value to cubeByValue
   number = cubeByValue(number);

   printf("\nThe new value of number is %d\n", number);
}

// calculate and return cube of integer argument
int cubeByValue(int n)
{
   return n * n * n; // cube local variable n and return result
}
```

```
The original value of number is 5
The new value of number is 125
```

Şekil 7.6 Değere göre geçiş kullanarak bir değişkenin küpünü alan program.

Aşağıdaki program ise “number” değişkeninin bulunduğu adresin içerisindeki veriye göre işlem yapar. (yani bu adresteki veri de number değişkenine atanan veri olacağından bu değer de number değişkenine atanan değer olacaktır.) Buradaki amaç verinin bulunduğu adresi kullanarak nasıl işlem yapılacağını göstermektir.

```
1  // Fig. 7.7: fig07_07.c
2  // Cube a variable using pass-by-reference with a pointer argument.
3
4  #include <stdio.h>
5
6  void cubeByReference(int *nPtr); // function prototype
7
8  int main(void)
9  {
10     int number = 5; // initialize number
11
12     printf("The original value of number is %d", number);
13
14     // pass address of number to cubeByReference
15     cubeByReference(&number);
16
17     printf("\nThe new value of number is %d\n", number);
18 }
19
20 // calculate cube of *nPtr; actually modifies number in main
21 void cubeByReference(int *nPtr)
22 {
23    *nPtr = *nPtr * *nPtr * *nPtr; // cube *nPtr
24 }
```

```
The original value of number is 5
The new value of number is 125
```

Şekil 7.7 İşaretçi bağımsız değişkeniyle referansa göre geçiş kullanarak bir değişkenin küpünü alma.

Bir adresi bağımsız değişken olarak alan bir fonksiyon, adresi almak için bir işaretçi parametresi tanımlamalıdır. Örneğin, Şekil 7.7'de cubeByReference (satır 21) fonksiyonunun başlığı şöyledir:

```
void cubeByReference(int *nPtr)
```

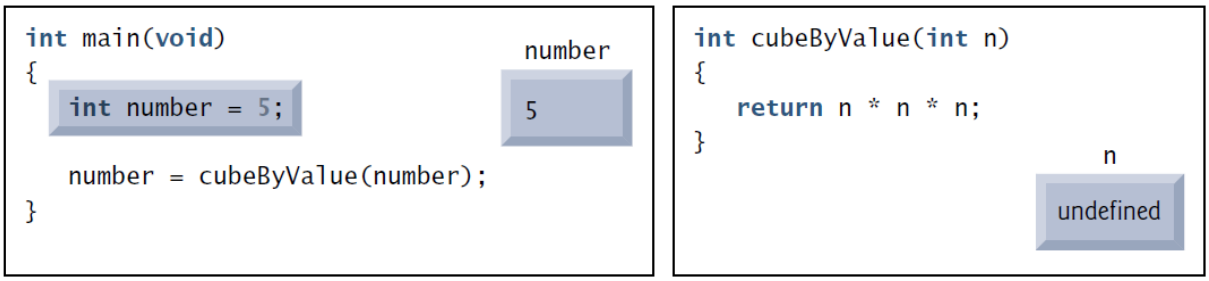
Başlık, cubeByReference'ın bir tamsayı değişkeninin adresini argüman olarak aldığını, adresi yerel olarak nPtr'de sakladığını ve bir değer döndürmediğini belirtir.

“cubeByReference” için fonksiyon prototipi (Şekil 7.7, satır 6) bir int * parametresi belirtir. Diğer değişken türlerinde olduğu gibi, fonksiyon prototiplerine işaretçi adlarını dahil etmek gerekli değildir. Dokümantasyon amacıyla dahil edilen adlar, C derleyicisi tarafından dikkate alınmaz.

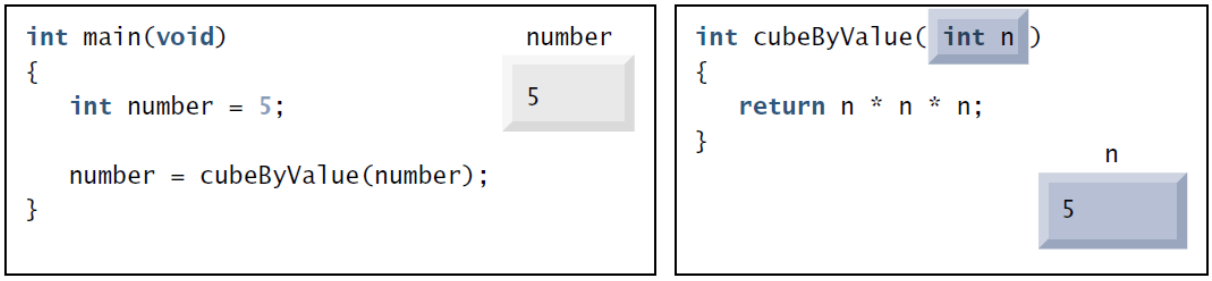
Argüman olarak tek boyutlu bir dizi bekleyen bir fonksiyon için, fonksiyonun prototipi ve başlığı, “cubeByReference” fonksiyonunun parametre listesinde (satır 21) gösterilen işaretçi gösterimini kullanabilir. Derleyici, bir işaretçi alan bir fonksiyon ile tek boyutlu bir dizi alan bir fonksiyon arasında ayrım yapmaz. Bu, elbette, fonksiyonun bir diziyi veya yalnızca tek bir değişkeni alırken kendisi için referans yoluyla geçiş gerçekleştireceğini "bilmesi" gerektiği anlamına gelir. Derleyici “int b[]” biçimindeki tek boyutlu bir dizi için bir fonksiyon parametresiyle karşılaştığında, derleyici parametreyi int b* işaretçi notasyonuna dönüştürür. İki form birbirinin yerine kullanılabilir.

Şimdi değer ile geçiş işlemini (pass-by-value) şekilsel olarak açıklayalım.

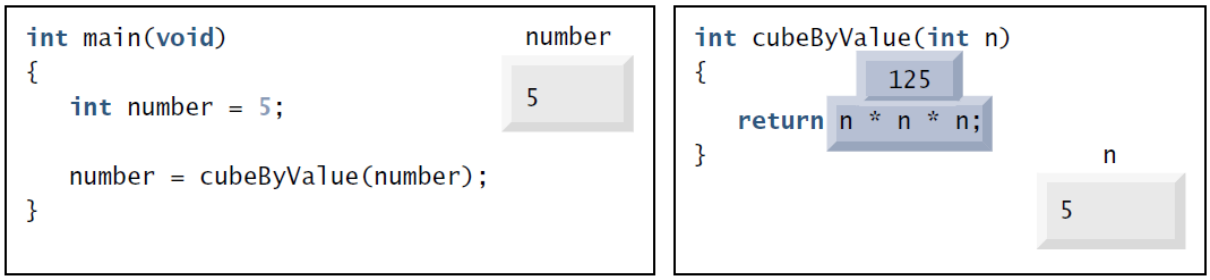
- Adım 1: Ana fonksiyon (main function) cubeByValue'u çağırmadan önce:



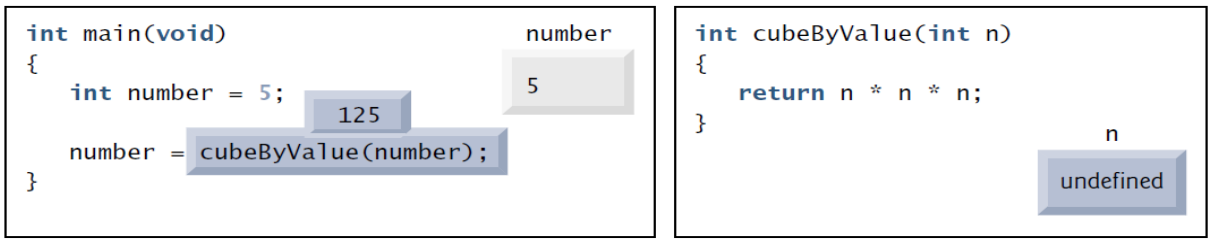
- Adım 2: cubeByValue çağrıyı aldıktan sonra:



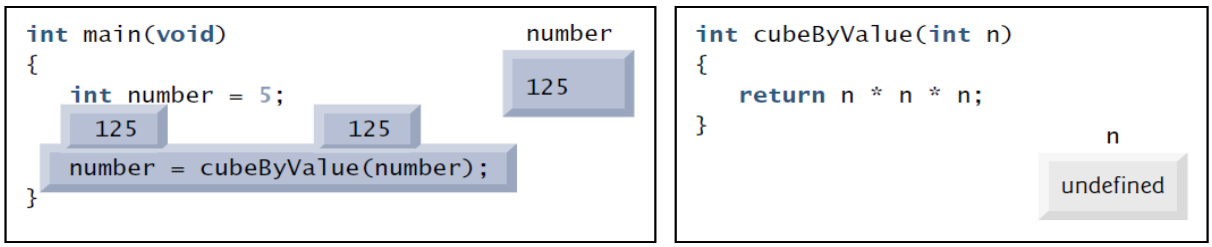
- Adım 3: cubeByValue küp parametresi n'den sonra ve cubeByValue ana parametreye dönmeden önce:



- Step 4: cubeByValue main'e döndükten sonra ve sonucu sayıya atamadan önce:

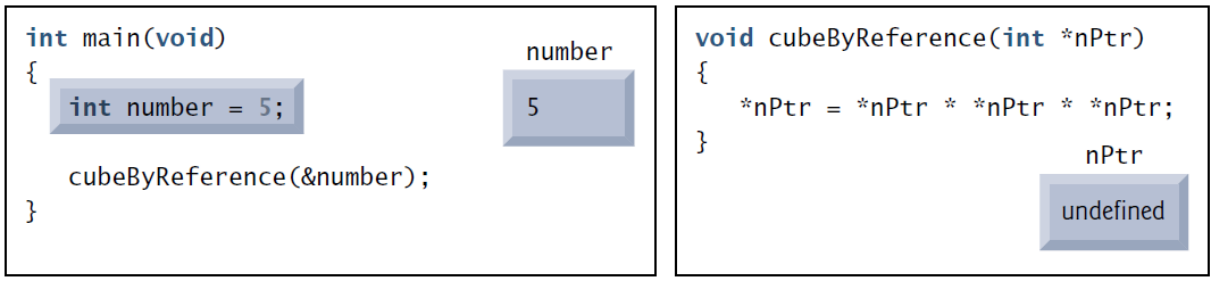


- Adım 5: Ana fonksiyon “number” değişkenine atamayı tamamladıktan sonra:

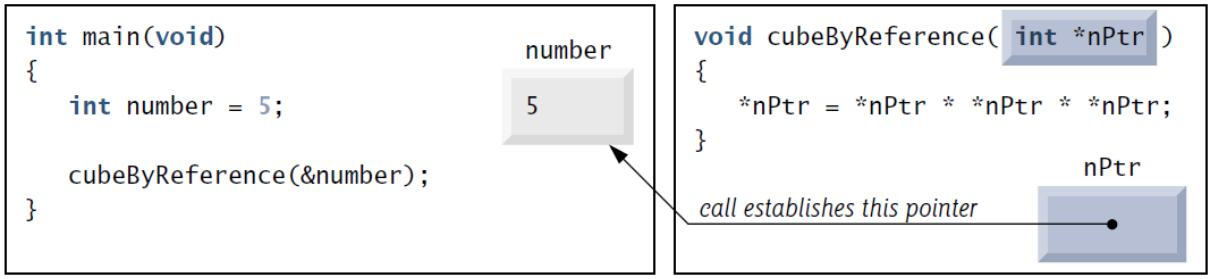


Şekil 7.8 Tipik bir değer ile geçiş (pass-by-value) analizi.

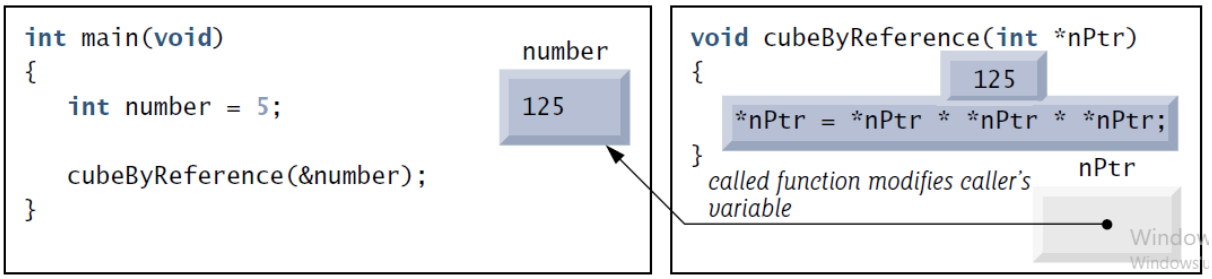
- Adım 1: Ana fonksiyon cubeByReference'ı çağırmadan önce:



- Adım 2: cubeByReference çağrıyı aldıktan sonra ve *nPtr küpünü almadan önce:



- Adım 3: *nPtr 'nin küp'ü alındıktan sonra ve program kontrolü ana konuma dönmeden önce:



Şekil 7.9 Bir işaretçi bağımsız değişkeniyle tipik bir referansla göndermenin (pass-by-reference) analizi.