C Programlama Dilinde Pointer Mantığı

Öncelikle pointerların makine seviyesinde ne anlama geldiğine bakalım. Bilgisayarda ana bellek **byte** boyutunda parçalara (**bellek hücrelerine**) bölünmüştür.

Her byte 8 bit saklayabilmektedir.

0	0	1	0	0	1	0	0

Her bit 0 veya 1 olabilir. Yani 1 byte 2^8 (256) farklı değer saklayabilir. 1 byte, sayı ise işaret durumuna göre [0, 256) ya da [-128, 128) arası değer alabilir.

bellek hücrelerini ayırt etmek için her bellek hücresine adres verilmiştir. Bellekte N tane byte varsa, adresler 0'dan N-1'e kadar numaralandırılmaktadır.

Bellek adresi	**Bellek içeriği**
0	01000100
1	01100101
2	01101110
3	01100101
4	01101101
	….
N-1	01100101

Değişkenler boyutlarına göre bellekte birden fazla byte yer kaplayabilir.

Adres	**İçerik**
	...
1000	
1001	
1002	
1003	
	...

Örnek; int -> 4 byte, double -> 8 byte

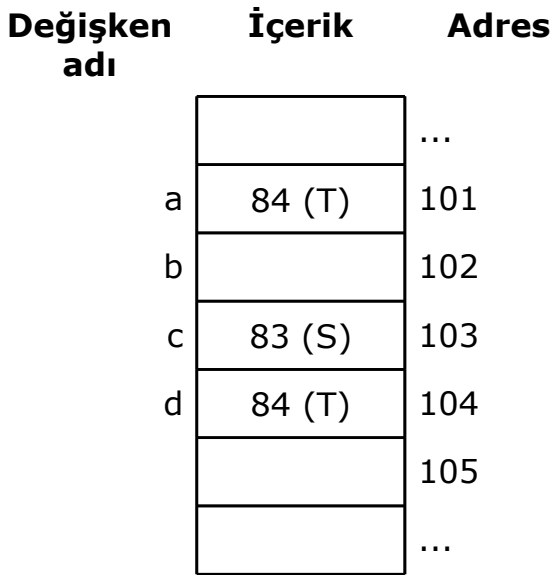
Değişkenlerin bellekte yerleşimi

Örneklerin basit olması için bu dokümanda tüm değişken tiplerinin boyutunun 1 byte olduğunu varsayıyoruz.

Aşağıdaki değişkenler programda tanımlanmış olsun:

```
char a = 'T'; // ascii kodu 84
char b;
char c = 'S';
char d = 'T';
```

Bu durumda bellekte aşağıdakine benzer bir yerleşim ortaya çıkacaktır.



Yukarıdaki bellek yerleşimine göre;

- a değişkeninin içeriği (değeri) 84'tür.
- a değişkeninin adresi 101'dir.

a değişkeninin değerini ekrana yazdırdığımızda:

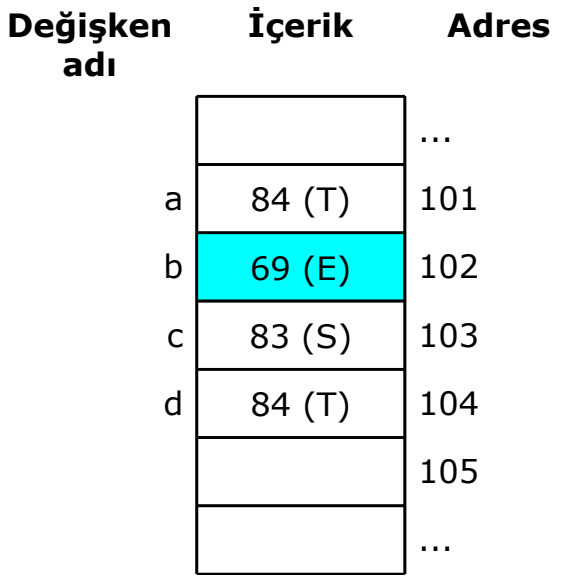
```
printf("%d\n", a);
```

a değişkeninin (101 numaralı hücrenin) içeriği okunur ve değeri ekrana yazılır.

Programda b değişkenine 'E' değerini atadığımızda:

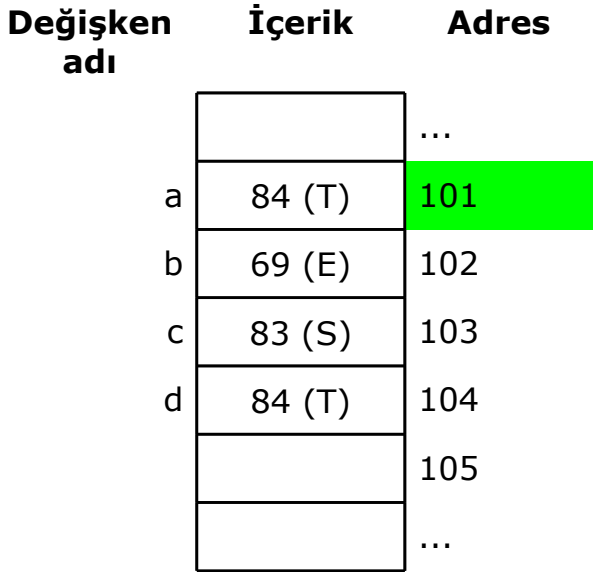
```
b = 'E';
```

b değişkeninin (102 numaralı adresin) içeriğine '**E' (69)** yazılır.



Adres Operatörü

Bir değişkenin bulunduğu adresi okumak için **&** operatörü kullanılmaktadır.



a değişkeninin adresini ekrana yazdırırsak

```
printf("%d\n", &a);
```

Örnekteki bellek yerleşimine göre, 101 çıktısını üretir. Gerçek bilgisayarda bellek çok daha büyük olduğu için 6293892 gibi bir sayı görebilirsiniz.

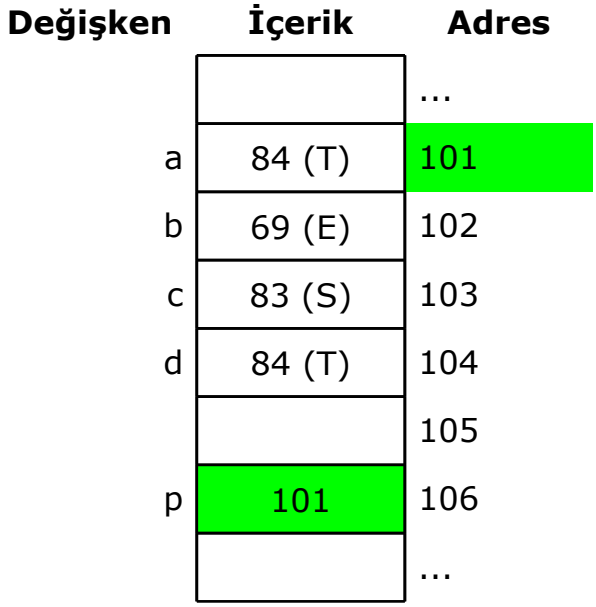
Adres değeri %d ile yazdırıldığında derleyici tarafından tip uyuşmazlığı uyarısı verilmektedir. Burada örneklerin basit olması açısından %d kullanılmıştır.

Programlarda adres değerinin hatasız bir biçimde yazdırılması için %p kullanılmalıdır. %p ile yazdırıldığında hexadecimal (16'lık tabanda) çıktı üretmektedir. Örneğin 0022FF4C gibi bir sayı görebilirsiniz.

İşaretçiler

İşaretçiler önceki başlıktaki bellek adresi mantığından ortaya çıkmaktadır. Bellek adresi sayısal bir değer olduğundan dolayı değişkenlerin içerisinde saklanabilir.

örneğin a değişkeninin adresi p isimli başka bir değişkende saklanıldığında program belleği aşağıdaki g,ibi görünecektir.



Bir adres değeri değişkende saklamak istenilirse, değişken pointer tipinde tanımlanmalıdır.

```
char * p; // char tipindeki değişkenin adresini saklayan degisken
```

p değişkeninin içerisine a'nın adresi aşağıdaki gibi atanabilir:

```
p = &a;
```

Artık p değişkeni (p pointer'ı), a değişkenini göstermektedir. p pointerını kullanarak 101 adresi (a değişkeni) üzerinde işlem yapabiliriz.

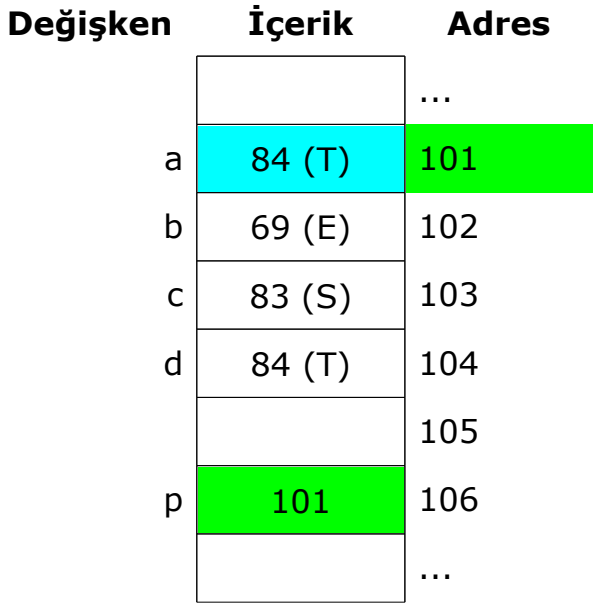
Pointer'ın gösterdiği adresin içeriğine erişim

p pointer'ının gösterdiği adresin içeriğine erişmek için***** operatörü kullanılır.

Örneğin p pointerının gösterdiği adresin içeriğini yazıdrırsak:

```
printf("%d\n", *p);
```

Örnekteki belleğe göre, ekrana 84 sayısını yazdıracaktır.



***p** yazıldığında yapılan işlem şudur:

p'nin değerini oku (101), o değere sahip bellek hücresini bul, içeriğine eriş

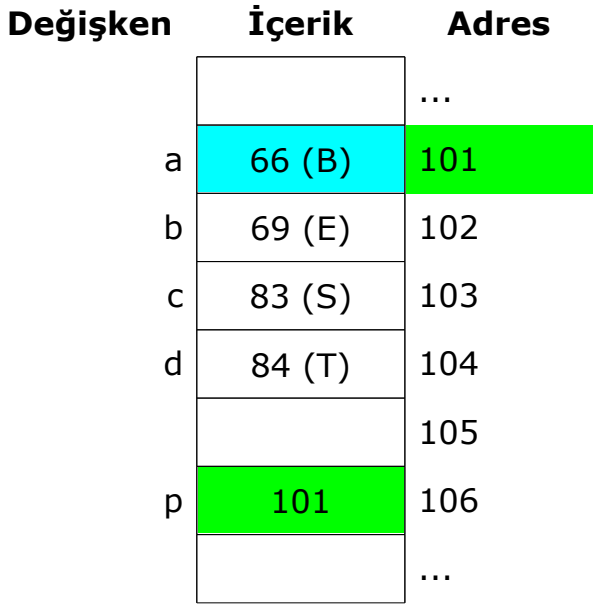
başka bir deyişle:

p'nin gösterdiği bellek hücresinin içeriğine eriş.

Aynı şekilde pointer kullanılarak değer atama işlemi de yapılabilmektedir. Aşağıdaki işlemi yaparsak:

```
*p = 66; // 'B'
```

p'nin gösterdiği bellek hücresinin içeriğine 66 yaz



Yukarıdaki işlem sonucunda 101 numaralı adresin içeriği değiştirilmiştir. Yani a değişkeninin değeri değişmiştir.

a'nın değerini yazdırırsak

```
printf("%d\n", a);
```

66 çıktısı üretilecektir.

Pointer tipindeki değişkenlerin kullanım alanları

Pointerlarların kullanım yerlerinden bazıları:

- fonksiyona parametre olarak gelen değişkenin değerini değiştirmek
- karakter dizileri
- dinamik (değişken boyutlu) diziler
- bağlı liste, ağaç vb. veriyapıları
- belleğin istenilen yerine erişim (sistem programlama)

Bu dokümandaki kodun tamami

```c
#include <stdio.h>

char a = 'T';
char b = 'E';
char c = 'S';
char d = 'T';

int main() {

    printf("a nin adresi: %d \n", &a);
    printf("b nin adresi: %d \n", &b);
    printf("c nin adresi: %d \n", &c);
    printf("d nin adresi: %d \n\n", &d);

    printf("a nin degeri: %d\n\n", a);

    char *p;
    p = &a;

    printf("p nin degeri : %d\n", p);
    printf("*p nin degeri: %d\n\n", *p);

    // p'nin gösterdiği bellek hücresinin içeriğine 66 yaz
    *p = 'B'; // ascii kodu 66

    printf("a nin degeri: %d\n", a);
    printf("*p nin degeri: %d\n", *p);
```

```
    return 0;
}
```