ITU ACM Student Chapter Course Program

Introduction to C

Week 4

Instructor

Gökalp Akartepe

Prepared by

Mehmet Yiğit Balık & Mihriban Nur Koçak & Emir Oğuz

Pointers(Göstericiler)

Bir değişken bir değer içerir ancak bir pointer bir değişkenin bellekteki konumunu belirtir. Pointerlarla işlem yaparken kullanılan bazı operatörler vardır.

Bir pointer tanımlanırken kullanılan üç farklı yazım düzeni vardır:

```
int *pt      int* pt      int  *  pt
```

Tüm bu yazım düzenlerinin işlevi aynıdır. Kullanıcı kendi tercihine göre herhangi birini kullanabilir.

Burada int olarak belirtilmiş kısım tanımladığımız pointer hangi değişken tipinin adresini temsil edecekse, o değişken tipini belirtmek için kullanılır. Bir pointer var olan tüm veri tiplerinin adresini temsil edebilir.

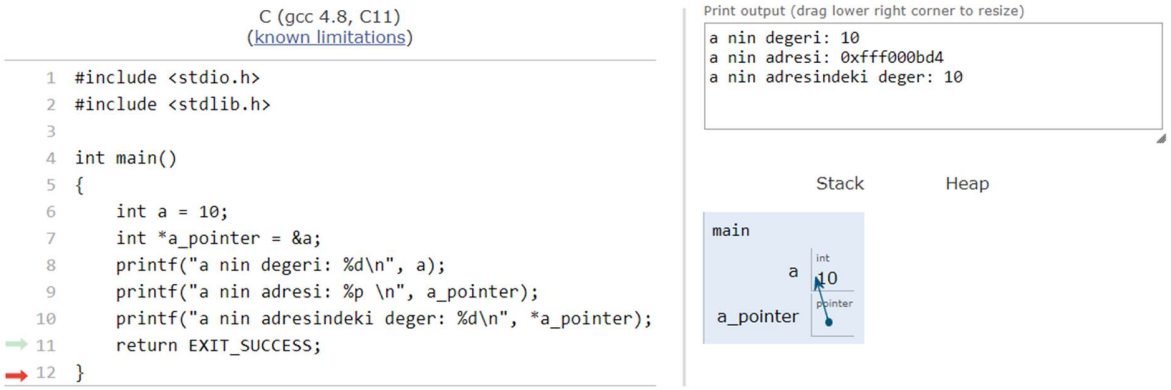
```
double* pt      char* pt      float* pt
```

- **&** operatörü :
  - Bir değişkenin önüne konularak kullanıldığında (**&var**) bu değişkenin adresini temsil eder.
- * operatörü
  - Bir pointer tanımlamak istendiğinde pointer'ın adresini temsil edeceği değişkenin veri tipini belirttikten sonra, pointer'ın isminden önce (**int* pt**) (**int *pt**) (**int * pt**) kullanılır.
  - Bir pointer'ın temsil ettiği değere ulaşılmak istendiğinde pointer isminin önüne konularak (***pt**) bu değere ulaşılır (**dereference**)

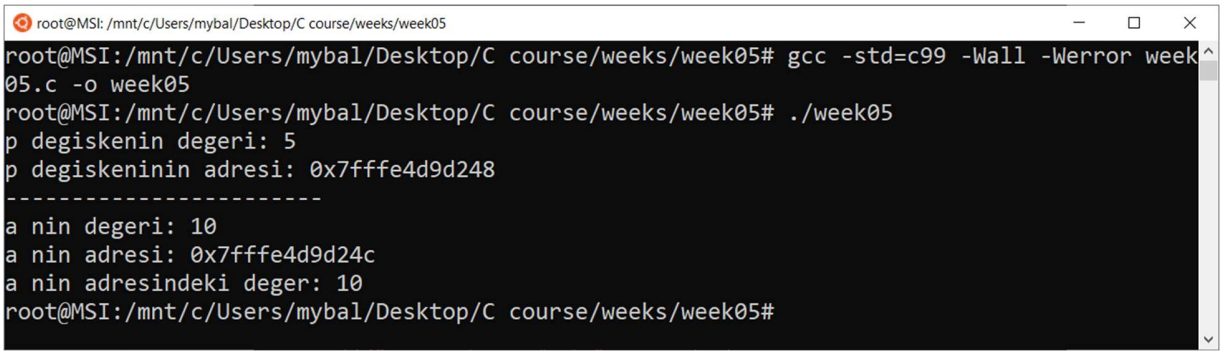
Not: *Bir adres değeri yazdırılmak istendiğinde format olarak **%p** formatı kullanılır.*

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int p = 5;
    printf("p degiskenin degeri: %d\n", p);
    printf("p degiskeninin adresi: %p\n", &p);
    printf("------------------------\n");
    int a = 10;
    int *a_pointer = &a;
    printf("a nin degeri: %d\n", a);
```

```c
    printf("a nin adresi: %p \n", a_pointer);
    printf("a nin adresindeki deger: %d\n", *a_pointer);
    return EXIT_SUCCESS;
}
```



(C tutor adlı sitenin görselleştirmesi)



Örnekte de görüldüğü üzere pointer bir değişkenin adresini ifade eder ve bu adres kullanarak değişkenin değerine de ulaşılabilir. Dolayısıyla değişkenlerin değerleri kendi adreslerini temsil eden pointerlar kullanılarak değiştirilebilir.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a = 10;           // ilk deger
    int *a_pointer = &a; // a'nin adresi
    printf("a'nin ilk degeri: %d\n", a);
```

```c
    *a_pointer = 20; //a'nin degeri adresi uzerinden degistirildi
    printf("a'nin degistirilmis degeri: %d\n", a);
    return EXIT_SUCCESS;
}
```

```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week05
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week05# gcc -std=c99 -Wall -Werror week
05.c -o week05
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week05# ./week05
a'nin ilk degeri: 10
a'nin degistirilmis degeri: 20
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week05#
```

Kullanılan operatörlerin işlevlerini karıştırmamak adına, sık yapılan bazı atama hataları aşağıda gösterilmiştir. Tüm bu atamaları yaparken dikkat edilmesi gereken her verinin ancak ve ancak kendisiyle aynı veri tipindeki bir veriye atanabildiğidir.

```c
int c;
int *pc;
// pc bir adres fakat c degil
pc = c; // HATA

// &c bir adres fakat *pc degil
*pc = &c; // HATA

//  &c ve pc adres
pc = &c;

// c ve *pc deger
*pc = c;
```

Şu ana kadar sıkça kullanılan **scanf** fonksiyonu da aslında çalışması için bir adres değerine ihtiyaç duyar.

```
scanf("%d",&a)
```

Burada görüldüğü üzere scanf fonksiyonunda tırnak içinde yazdırmak istenilen değerin formatı belirtildikten sonra, hangi değişkenin değeri yazdırılmak isteniyorsa onun adı yazılır ve önüne bir **&** işareti konulur. Burada **&a**, a değişkeninin adresini temsil eder.

Aşağıdaki örnekte scanf fonksiyonu ile bir pointer'a adresini temsil edeceği bir değer atanmıştır. Burada pointer scanf fonksiyonu ile kullanılırken önüne **&** işareti konulmamıştır. Çünkü halihazırda bir pointer bir değişkenin adresini temsil eder.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a = 5;
    int *p = &a;
    scanf("%d", p);
    printf("%d\n", a);
    return EXIT_SUCCESS;
}
```

```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week05
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week05# ./week05
5
5
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week05#
```

Pointers and Arrays (Göstericiler ve Diziler)

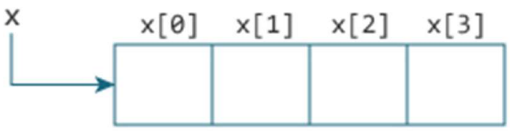
C dilinde pointerlar ve diziler arasında derin bir ilişki vardır. Her dizi bir pointer olarak kolaylıkla ifade edilebilir. Dizinin adı aslında dizinin ilk elemanının adresini temsil eder. Dizinin her bir elemanı ise hafızada farklı bir adreste depolanır. Dolayısıyla her bir dizi elemanının adresi bir diğerinden farklıdır.

```c
#include <stdio.h>
```

```c
#include <stdlib.h>
int main()
{
    int x[4];
    int i;

    for (i = 0; i < 4; ++i)
    {
        printf("&x[%d] = %p\n", i, &x[i]);
    }

    printf("x'in adresi: %p\n", x);

    return EXIT_SUCCESS;
}
```

```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week05
x'in adresi: 0x7fffe85761c0root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week05# ./we
gcc -std=c99 -Wall -Werror week05.c -o week05
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week05# ./week05
&x[0] = 0x7ffffedc5120
&x[1] = 0x7ffffedc5124
&x[2] = 0x7ffffedc5128
&x[3] = 0x7ffffedc512c
x'in adresi: 0x7ffffedc5120
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week05#
```

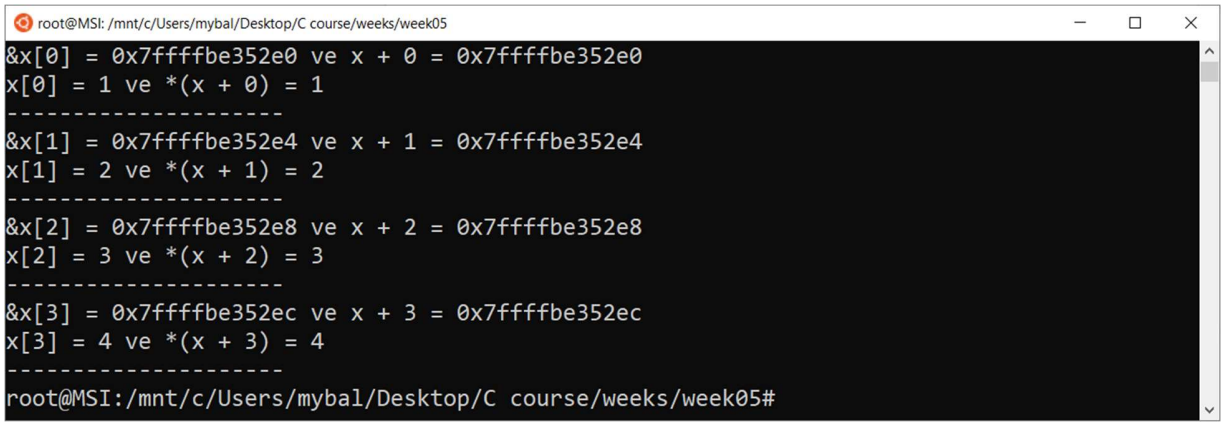
Dizinin adı dizinin ilk elemanının adresini temsil eder. Dizinin ikinci elemanının adresine ulaşmak için ise dizinin adına aritmetik olarak **bir** eklenir. Bu işlemlere bu şekilde dizinin adını arttırarak ve azaltarak devam edilebilir. Buna **pointer aritmetiği** adı verilir.



- `&x[0]` eşittir `x` ve `x[0]` eşittir `*x`.
- `&x[1]` eşittir `x+1` and `x[1]` eşittir `*(x+1)`.
- `&x[2]` eşittir `x+2` and `x[2]` eşittir `*(x+2)`.
- ...

- Kısaca, `&x[i]` eşittir `x+i` ve `x[i]` eşittir `*(x+i)`.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int x[4] = {1, 2, 3, 4};

    for (int i = 0; i < 4; ++i)
    {
        printf("&x[%d] = %p ve x + %d = %p\n", i, &x[i], i, x + i);
        printf("x[%d] = %d ve *(x + %d) = %d\n", i, x[i], i, *(x + i));
        printf("----------------------\n");
    }

    return EXIT_SUCCESS;
}
```




```c
#include <stdio.h>
#include <stdlib.h>

int main()
```

```c
{
    int x[5] = {1, 2, 3, 4, 5};
    int *ptr;

    // ptr'a x dizisinin 2. elemaninin adresi atandi
    ptr = &x[1]; // 2
    printf("*ptr = %d\n", *ptr);
    ptr++;                                  // 3
    printf("*ptr = %d \n", *ptr);           // 3
    printf("*(ptr+1) = %d \n", *(ptr + 1)); // 4
    printf("*(ptr-1) = %d\n", *(ptr - 1));  // 2

    return EXIT_SUCCESS;
}
```

```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week05
05.c -o week05
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week05# ./week05
*ptr = 2
*ptr = 3
*(ptr+1) = 4
*(ptr-1) = 2
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week05#
```

Yukarıdaki örnekte ilk olarak pointer'in x dizisinin ikinci elemanına ataması yapıldı. Sonrasında bu pointer'ın tuttuğu değer **dereference** edilerek bastırıldı. Sonrasında pointer aritmetik olarak bir arttırıldı. Bu işlem sonucunda artık pointer dizinin ikinci elemanını değil üçüncü elemanını tutmaya başladı. Pointer'ın güncel olarak tuttuğu değer(**x[2]**) **dereference** edilerek bastırıldı. Son olarak pointer'ın güncel olarak tuttuğu değerin(**x[2]**) bir fazlası(**x[3]**) **dereference** edilerek bastırıldı. Fakat dikkat edilmesi gereken kısım, bu bastırma işleminden sonra pointer'ın tuttuğu değerde(**x[2]**) bir değişiklik olmadı.

```c
ptr = &x[1]; // 2
printf("*ptr = %d\n", *ptr);
ptr++;                                  // 3
printf("*ptr = %d \n", *ptr);           // 3
printf("*(ptr+1) = %d \n", *(ptr + 1)); // 4
```