BLM-112 PROGRAMLAMA DİLLERİ II

Ders-2 İşaretçiler (Pointer) (Kısım-1)

Yrd. Doç. Dr. Ümit ATİLA
umitatila@karabuk.edu.tr
http://web.karabuk.edu.tr/umitatilla/

Hafıza Yapısı

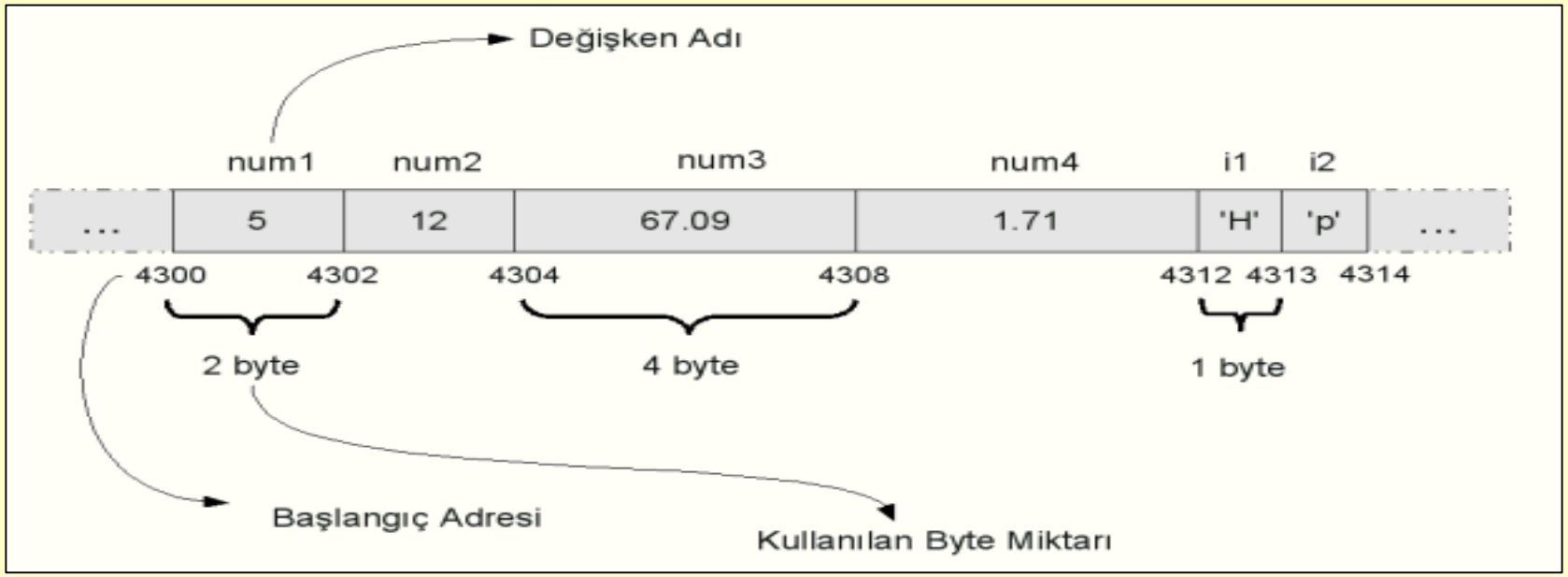
- Bir değişken tanımlandığında arka planda bilgisayarın hafızasında bir konuma yerleştirilir.
- Hafıza küçük hücrelerden oluşmuş bir blok olarak düşünülebilir.
- Bir değişken tanımlandığında bellek bloğundan gerekli miktarda hücre ilgili değişkene aktarılır.
- Hafızada değişken için ne kadar hücre ayrılacağı değişkenin tipine göre değişir.

Hafıza Yapısı

```c
#include <stdio.h>

int main(void)
{
    // Degiskenler tanımlanıyor:
    int num1, num2;
    float num3, num4;
    char i1, i2;
    // Degiskenlere atama yapiliyor:
    num1 = 5;
    num2 = 12;
    num3 = 67.09;
    num4 = 1.71;
    i1 = 'H';
    i2 = 'p';
    return 0;
}
```

Hafıza Yapısı

- Hücrelerden oluşan bellek yapısını verilen kod parçası için uygularsak.
  - int veri tipi 2 bayt, float tipinin 4 bayt, char tipinin de 1 bayt yer kapladığını varsayalım.
  - Her bir hücre 1 bayt alanı temsil etsin.
  - Değişkenler için ayrılan hafıza alanı 4300 adresinden başlasın.



Hafıza Yapısı

- Bir değişken tanımlandığında hafızada onun için gereken alan rezerve edilir.
- Örn. ***int num1*** tanımlaması, bellekte uygun bir yerde ***2 bayt*** alanın ***num1*** değişkeni için ayrılmasını sağlar.
- Daha sonra num1 değişkenine 5 değeri atandığında ayrılan hafıza alanına 5 değeri kaydediliyor.
- Aslında, *num1* ile ilgili yapacağınız bütün işlemler, 4300 adresiyle 4302 adresi arasındaki bellek hücrelerinin değişmesiyle alakalıdır.
- Değişken dediğimiz; uygun bir bellek alanının, bir isme revize edilip, kullanılmasından ibarettir.

Pointer Tanımlama

- Bir veri bloğunun bellekte bulunduğu adresi içeren (gösteren) veri tipidir.

  veri tipi *p;

- p değişkeni <veri tipi> ile belirtilen tipte bir verinin bellekte saklandığı **adresi** içerir.

  int *iptr;

  float *fptr;

- Dikkat edilmesi gereken tek nokta; pointer'ı işaret edeceği değişken tipine uygun tanımlamaktır.
- Yani float bir değişkeni, int bir pointer ile işaretlemeye çalışmak yanlıştır!

Pointer Tanımlama

- Bir pointer'ın var olan bir değişkenin bulunduğu adresi göstermesi için değişkenin adresinin pointer'a atanması gerekir.
- Bunun için değişkenin hafızada tutulduğu adresin bilinmesi gerekir.
- Bu da adres operatörü (&) ile mümkündür.
  - &y → y değişkeninin adresini verir.

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

Pointer Tanımlama

- Pointer bir değişkenin adresinin gösterir şekilde atandıktan sonra o pointer, ilgili değişkeni işaret eder.
- Eğer bahsettiğimiz değişkenin sahip olduğu değeri pointer ile göstermek veya değişken değerini değiştirmek isterseniz, pointer başına '*' getirerek işlemlerinizi yapabilirsiniz.
- Pointer başına '*' getirerek yapacağınız her atama işlemi, değişkeni de etkileyecektir.

Pointer Tanımlama

```c
#include <stdio.h>

int main(void)
{
   int i;
   int *iptr;
   i = 5;
   iptr = &i;

   printf("i adresi    %p\n", &i);
   printf("iptr degeri %p\n", iptr);

   printf("i degeri    %d\n", i);
   printf("*iptr degeri %d\n", *iptr);

   getchar();
   return 0;
}
```

Pointer Tanımlama (Değişkene Pointer ile Erişme)

- Pointer kullanarak, değişkenlerin sakladığı değerleri de değiştirebiliriz.
- Bir işaretçinin gösterdiği adresteki veriye erişmek için işaretçi değişkeninin önüne * karakteri eklenir.

```c
#include<stdio.h>
int main()
{
    int i;
    int *iptr;
    iptr = &i;
    *iptr = 8;
    printf("i değişkeninin değeri %d\n", i);
    printf("iptr adresinin içeriği %d\n", *iptr);

    getchar();
    return 0;
}
```

Pointer Tanımlama
(Değişkenleri Pointer ile İlişkilendirme)

```c
#include<stdio.h>
int main( void )
{
    // int tipinde değişken tanımlıyoruz:
    int xyz = 10, k;
    // int tipinde pointer tanımlıyoruz:
    int *p;

    // xyz değişkeninin adresini pointer'a atıyoruz.
    // Bir değişken adresini '&' işaretiyle alırız.
    p = &xyz;

    // k değişkenine xyz'nin değeri atanır. Pointer'lar değer tutmaz.
    // değer tutan değişkenleri işaret eder.
    //Başına '*' koyulduğunda, işaret ettiği değişkenin değerini gösterir.
    k = *p;

    return 0;
}
```

Pointer Tanımlama

- Bir pointer'in işaret ettiği değişkeni program boyunca sürekli değiştirebilirsiniz.

```c
#include<stdio.h>
int main( void )
{
    int x, y, z;
    int *int_addr;
    x = 41;
    y = 12;
    // int_addr x degiskenini isaret ediyor.
    int_addr = &x;
    // int_addr'in isaret ettigi degiskenin sakladigi deger aliniyor. (yani x'in degeri)
    z = *int_addr;
    printf( "z: %d\n", z );
    // int_addr, artik y degiskenini isaret ediyor.
    int_addr = &y;
    // int_addr'in isaret ettigi degiskenin sakladigi deger aliniyor. (yani y'nin degeri)
    z = *int_addr;
    printf( "z: %d\n" ,z );

    return 0;
}
```

Pointer Boyutu

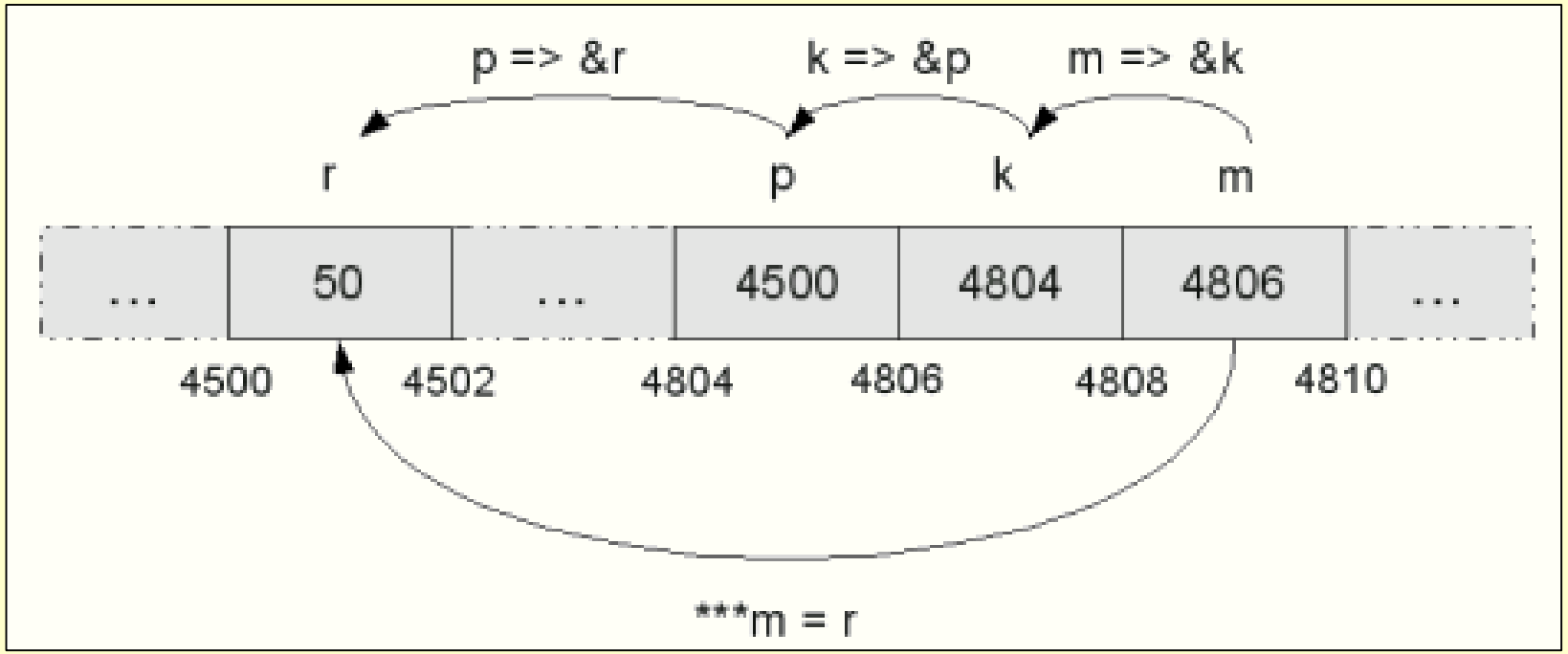
- Pointer'lar genelde sabit boyutta yer kaplar. Örneğin 32 bit bir sistemde genellikle pointerlar 32 bit olur.

```c
#include<stdio.h>
int main()
{
    double i;
    double *iptr;

    iptr = &i;
    printf("i boyutu: %d\n", sizeof(i));
    printf("iptr boyutu: %d", sizeof(iptr));

    getchar();
    return 0;
}
```

Pointer Tutan Pointer'lar

- Pointer'lar, gördüğümüz gibi değişkenleri işaret ederler.
- Pointer'da bir değişkendir ve onu da işaret edecek bir pointer yapısı kullanılabilir.
- Pointer değişkenini işaret edecek bir değişken tanımlıyorsanız; başına '**' getirmeniz gerekir.
- Buradaki yıldız sayısı değişebilir. Eğer, pointer işaret eden bir pointer'i işaret edecek bir pointer tanımlamak istiyorsanız, üç defa yıldız (***) yazmanız gerekir.

Pointer Tutan Pointer'lar



Pointer Aritmetiği

- İşaretçi değişkenler üzerinde toplama ve çıkartma işlemleri (++, --) geçerlidir. Ancak eklenecek değer tamsayı olmalıdır.
- İşaretçi değişkenin değeri 1 arttırıldığı zaman değişken bir sonraki veri bloğunu işaret eder.
- Değişkenin alacağı yeni değer işaretçi değişkenin ne tip bir veri bloğunu işaret ettiğine bağlıdır.

Pointer Aritmetiği

- int i , *iPtr;
- iPtr = &i; // iPtr örneğin 1000 nolu adresi gösteriyorsa
- **(*iPtr) ++; // Bu işlem 1000 nolu adresin içeriğini 1 artırır.**
- **iPtr ++; // BU işlem iPtr nin 1004 nolu adresi göstermesini sağlar**
- (*iPtr) +=2; // Bu işlem 1000 nolu adresin içeriğini 2 artırır.
- (*iPtr) =7; // Bu işlem 1000 nolu adresin içeriğini 7 yapar.
- *(iPtr+2) = 5; //iPtr 1000 nolu adresi gösteriyorsa 1008 nolu adresin içeriğini 5 yapar.

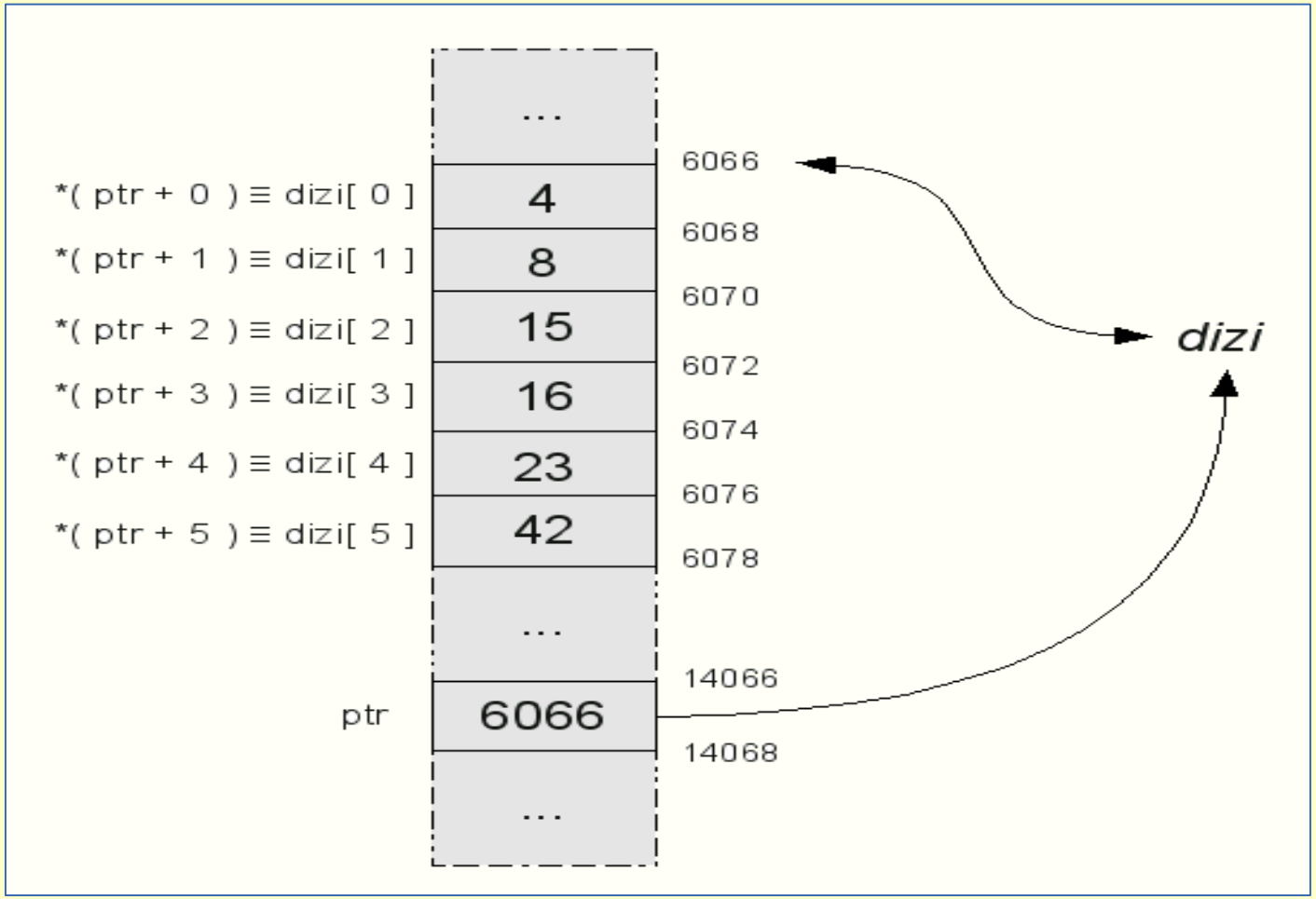
Pointer Aritmetiği

```c
#include<stdio.h>
int main( void )
{
    int i, *iPtr;
    double y, *yPtr;

    iPtr = &i;
    printf("iPtr gosterdigi adres: %d \n", iPtr);
    iPtr ++;  //int tipi için bir sonraki adres bloğu 4 bayt fazlası.
    printf("iPtr gosterdigi adres: %d \n\n", iPtr);

    yPtr = &y;
    printf("yPtr gosterdigi adres: %d \n", yPtr);
    yPtr ++;  //double tipi için bir sonraki adres bloğu 8 bayt fazlası.
    printf("yPtr gosterdigi adres: %d ", yPtr);

    getchar();
    return 0;
}
```

Diziler ile Pointer Arası İlişki

- Bir dizi adı sabit bir pointer gibi düşünülebilir.
- Diziler ile pointer'lar yakından ilişkilidir.
- Pointer'lar değişkenleri gösterdikleri gibi, dizileri de gösterebilirler.
  - int dizi [6];
  - int *ptr;
  - Dizi ve pointer'ı eşitlemek için dizinin adı kullanılabilir. Çünkü dizi adı aslında o dizinin ilk elemanının adresidir.
    - ptr = dizi; //Artık ptr[0] ve dizi[0] eşittir.
    - Aynı işlemi ptr= &dizi [0] şeklinde de yapabiliriz.

Diziler ile Pointer Arası İlişki



Diziler ile Pointer Arası İlişki

- Dizi gösteren pointer'lar ile dizinin elemanlarına ulaşmak için:
  - *(ptr + n) → n sayısı dizinin n. indisini gösterir.
  - *(ptr + 4) → dizinin 4. indisindeki eleman yani dizi[4]
- Diğer alternatif gösterimler
  - ptr[4]
  - *(dizi + 4)

Diziler ile Pointer Arası İlişki

```c
#include<stdio.h>
int main( void )
{
    int elm;
    int month[ 12 ];
    int *ptr;
    ptr = month; // month[0] 'ın adresini ptr'ye ata
    elm = ptr[ 3 ]; // elm = month[ 3 ]
    ptr = month + 3; // ptr, month[ 3 ] adresini göstersin
    ptr = &month[ 3 ]; // ptr, month[ 3 ] adresini göstersin
    elm = ( ptr+2 )[ 2 ]; // elm = ptr[ 4 ] ( = month[ 7 ] ).
    elm = *( month + 3 );
    ptr = month; // month[0] 'ın adresini ptr'ye ata
    elm = *( ptr + 2 ); // elm = month[ 2 ]
    elm = *( month + 1 ); // elm = month[ 1 ]

    return 0;
}
```

Diziler ile Pointer Arası İlişki

```c
#include <stdio.h>
int main()
{
    int i[10], j;
    int *iptr;
    for (j=0; j<10; j++)
        i[j]=j;

    iptr = i;
    for (j=0; j<10; j++) {
        printf("%d ", *iptr);
        iptr++;
    }
    /* iptr artık dizinin başını göstermez */
    printf("\n%d \n",*(iptr-1));
    iptr = i;
    for (j=0; j<10; j++)
        printf("%d ", *(iptr+j));
    /* iptr hala dizinin başını gösterir */
    printf("\n%d",*iptr);
    getchar();
    return 0;
}
```

Diziler ile Pointer Arası İlişki

```c
#include <stdio.h>
int main()
{
    char *a="1234567890";
    char x[10];
    char *p1, *p2;
    printf("%s\n", a);
    p1 = a;
    p2 = x;
    while (*p1 != '\0') {
        *p2 = *p1;
        p1++;
        p2++;
    }
    printf("%s\n", x);
    getchar();
    return 0;
}
```

Diziler ile Pointer Arası İlişki

- Diziler pointer içerebilirler.
- Pointer tutan diziler sayesinde birden fazla diziye erişim yapılabilir.
- Pointer tutan diziye dizilerin başlangıç adreslerini atamak yeterlidir.
- Pointer tutan dizi üzerinde yapılan değişiklik orijinal diziyi etkiler.

Diziler ile Pointer Arası İlişki

```c
#include <stdio.h>
int main()
{
    int i,j;
    char * ilkBaharAylar[3] ={"Mart","Nisan","Mayis"};
    char * yazAylar[3] ={"Haziran","Temmuz","Agustos"};
    char * sonBaharAylar[3] ={"Eylul","Ekim","Kasim"};
    char * kisAylar[3] ={"Aralik","Ocak","Subat"};

    char ** table[4];//char pointer(string) tutan dizileri tutan dizi
    table[0] = ilkBaharAylar;
    table[1] = yazAylar;
    table[2] = sonBaharAylar;
    table[3] = kisAylar;

    for(i=0;i<4;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%s\n",table[i][j]);
        }
    }

    getchar();
    return 0;
}
```

Fonksiyonu Değer ile Çağırma (Call by Value)

- Bir fonksiyona gönderilen parametrenin normalde değeri değişmez. Fonksiyon içinde yapılan işlemlerin hiçbiri argüman değişkeni etkilemez.
- Sadece değişken değerinin aktarıldığı ve argümanın etkilenmediği bu duruma, "***call by value***" veya "***pass by value***" adı verilir.

Fonksiyonu Değer ile Çağırma (Call by Value)

```c
#include <stdio.h>
void arttir(int);
int main14(void)
{
   int i;
   i = 5;
   printf("oncesi %d\n", i);
   arttir(i);
   printf("sonrasi %d\n", i);
   getchar();

   return 0;
}

void arttir(int k)
{
   k++;
}
```

Fonksiyonu Referans ile Çağırma (Call by Reference)

- Bir fonksiyondan birden fazla değer döndürülmesi veya fonksiyonun içerisinde yapacağımız değişikliklerin parametre değişkene yansıması gereken durumlar olabilir.
- İşte bu gibi zamanlarda, "***call by reference***" veya "***pass by reference***" olarak isimlendirilen yöntem kullanılır.
- Argüman değer olarak aktarılmaz; argüman olan değişkenin adres bilgisi fonksiyona aktarılır. Bu sayede fonksiyon içerisinde yapacağınız her türlü değişiklik argüman değişkene de yansır.

Fonksiyonu Referans ile Çağırma (Call by Reference)

```c
#include <stdio.h>
void increment(int *);
int main(void)
{
   int i;
   i = 5;
   printf("oncesi %d\n", i);
   increment(&i);
   printf("sonrasi %d\n", i);
   getchar();

   return 0;
}

void increment(int *k)
{
   (*k)++;
}
```

Fonksiyonu Referans ile Çağırma (Call by Reference)

- Eğer yazdığınız fonksiyon birden çok değer döndürmek zorundaysa, referans yoluyla aktarım zorunlu hâle geliyor.
- Çünkü return ifadesiyle sadece tek bir değer döndürebiliriz.
- Örneğin bir bölme işlemi yapıp, bölüm sonucunu ve kalanı söyleyen bir fonksiyon yazacağımızı düşünelim.
- Bu durumda, bölünen ve bölen fonksiyona gidecek argümanlar olurken; kalan ve bölüm geriye dönmelidir.
- return ifadesi geriye tek bir değer vereceğinden, ikinci değeri alabilmek için referans yöntemi kullanmamız gerekir.

Fonksiyonu Referans ile Çağırma (Call by Reference)

```c
#include<stdio.h>
int bolme_islemi( int, int, int * );
int main( void )
{
    int bolunen, bolen;
    int bolum, kalan;
    bolunen = 13;
    bolen = 4;
    bolum = bolme_islemi( bolunen, bolen, &kalan );
    printf( "Bolum: %d Kalan: %d\n", bolum, kalan );
    getchar();
    return 0;
}
int bolme_islemi( int bolunen, int bolen, int *kalan )
{
    *kalan = bolunen % bolen;
    return bolunen / bolen;
}
```

Kaynaklar

- Paul J. Deitel, "C How to Program", Harvey Deitel.
- Kaan Aslan, "A'dan Z'ye C Klavuzu 8. Basım", Pusula Yayıncılık, 2002.