ALGORİTMA VE PROGRAMLAMA II

HAFTA#5

YZM 1106

Celal Bayar Üniversitesi Hasan Ferdi Turgutlu
Teknoloji Fakültesi

Genel Bakış…

- İşaretçiler ve Diziler
- Fonksiyon Parametresi Olan İşaretçiler
- Fonksiyon Geri Dönüş Değeri Olan İşaretçiler
- Struct ve İşaretçiler
- Fonksiyon İşaretçileri
- Void Tipindeki İşaretçiler

4. BÖLÜM

İşaretçiler - Bölüm 2

İşaretçiler ve Diziler

- Diziler ve işaretçiler, C'de özel bir biçimde ilişkilidirler.
- **Birbirleri yerine** hemen hemen her yerde kullanılabilirler.
- Bir dizi ismi, **sabit bir işaretçi** olarak düşünülebilir.
- Bu yüzden, *bir dizinin herhangi bir elemanına* işaretçi ile de erişilebilir.

İşaretçiler ve Diziler (devam…)


```
int kutle[5], *p, *q;
```

```
p = kutle;
```


1. Elemanın adresi p işaretçisine atanıyor

```
p = &kutle[0];
```


1. Elemanın adresi p işaretçisine atanıyor

```
q = &kutle[4]
```


5. Elemanın adresi q işaretçisine atanıyor

Not: Dizi adı bir işaretçi olduğu için doğrudan aynı tipteki bir işaretçiye atanabilir.

İşaretçiler ve Diziler (devam…)

- Ayrıca, **i** bir tamsayı olmak üzere aşağıdaki iki ifade aynı anlamdadır.
  - `kutle[i];`
  - `*(p + i);`
- Bunun sebebi, p işaretçisi kutle dizisinin başlangıç adresini tutmuş olmasıdır.
- p+i işlemi ile i+1. elemanın adresi, ve *(p+i) ile de bu adresteki değer hesaplanır.
  - ***(p+i);** → ***p nin gösterdiği adresten i blok ötedeki sayıyı hesapla***

İşaretçiler ve Diziler (devam…)

Örnek – Birlikte Yapalım…

```
int main()
{
    int kutle[5] = {5, 10, 15, 20, 25};
    int *p;

    p = kutle;

    printf("*p degeri = %d\n", *p);
    printf("*(p + 1) degeri = %d\n", *(p + 1));
    printf("*(p + 2) degeri = %d\n", *(p + 2));
    printf("*(p + 3) degeri = %d\n", *(p + 3));
    printf("*(p + 4) degeri = %d\n", *(p + 4));

    return 0;
}
```

İşaretçiler ve Diziler (devam…)

Örnek: `*(p+3);`

- Bu deyimde 3, işaretçinin offsetidir.
- İşaretçi, dizinin başlangıç adresini gösterirken, ve offset değeri dizi belirteciyle eştir. offset dizinin hangi elemanının kullanılacağını belirtir
- Bu gösterime **işaretçi/offset gösterimi** denir.
- Parantezler gereklidir çünkü * operatörünün önceliği + operatörünün önceliğinden yüksektir.
- Parantezler olmadan yukarıdaki ifade, *p'ye 3 eklerdi. (*yani, p'nin dizinin başlangıcını gösterdiği düşünülürse*, **kutle[0]'a 3 eklenirdi.)**

Örnek1:İşaretçi/Dizi Offset Yazdırma

- Aşağıdaki ekran çıktısının C programlama dilinde kodunu yazalım.

- int b[] = {10, 20, 30, 40};
- **pb** işaretçisi tanımla.
- **3 yöntem için 3 tane for** döngüsü kur.

```
YONTEM 1: DIZI INDISLERI
b[0]:   10
b[1]:   20
b[2]:   30
b[3]:   40

YONTEM 2: ISARETCI OFFSET
*(pb + 0 ):     10
*(pb + 1 ):     20
*(pb + 2 ):     30
*(pb + 3 ):     40

YONTEM 3: DIZI/ISARETCI OFFSET
*(b + 0 ):      10
*(b + 1 ):      20
*(b + 2 ):      30
*(b + 3 ):      40
```

Örnek1:İşaretçi/Dizi Offset Yazdırma


```c
int main()
{
    int i;
    int b[] = {10, 20, 30, 40};

    int *pb = b;

    printf("YONTEM 1: DIZI INDISLERI\n");
    for (i=0;i<=3;i++)
    {
      printf("b[%d]:\t%d\n",i,b[i]);
    }

    printf("\nYONTEM 2: ISARETCI OFFSET \n");
    for (i=0;i<=3;i++)
    {
      printf("*(pb + %d ):\t%d\n",i,*(pb + i));
    }

    printf("\nYONTEM 3: DIZI/ISARETCI OFFSET \n");
    for (i=0;i<=3;i++)
    {
      printf("*(b + %d ):\t%d\n",i,*(b + i));
    }

    return 0;
}
```

Örnek2:Ortalama Bul

- 5 elemanlı 10, 20, 30, 40, 50 değerlerine sahip b dizisi tanımlayalım.
- Aşağıdaki fonksiyonu kullanarak bu elemanların **ortalamasını bulup,** main fonksiyonu içerisinde yazdıralım.

```
float OrtalamaBul(float dizi[], int n)
```

- Fonksiyonun içerisinde diziye erişirken **işaretçi** kullanalım.

Örnek2:Ortalama Bul


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

float OrtalamaBul(float dizi[], int n);

int main()
{
    float ortalama = 0;
    float b[] = {10, 20, 30, 40, 50};

    ortalama = OrtalamaBul(b, 5);
    printf("Ortalama: %f", ortalama);
    return 0;
}

float OrtalamaBul(float dizi[], int n)
{
    int i;
    float *pdizi, toplam = 0;
    pdizi = dizi;

    for(i=0; i<=n; i++)
    {
        toplam += *(pdizi + i);
    }

    return (toplam / n);
}
```

Fonksiyon Parametresi Olan İşaretçiler

- C programlama dilinde fonksiyon parametreleri
  - **Değer** geçerek (**pass by value**)
  - **Adres** geçerek (**pass by reference**)

  olarak geçilebilirler.
- Şu ana kadar gördüğümüz fonksiyon kullanımında geçirilen parametreler, *fonksiyon içerisinde değiştirilse bile*, fonksiyon çağırıldıktan sonra **bu değişim çağrılan yerdeki değerini değiştirmez**.

Fonksiyon Parametresi Olan İşaretçiler
(devam…)

- Fakat, bir parametre adres geçerek aktarılırsa, fonksiyon içindeki **değişiklikler geçilen parametreyi etkiler**.
- Adres geçerek aktarım, işaretçi kullanmayı zorunlu kılar.

Örnek3: İşaretçi Olan Fonksiyon Parametresi

- main fonksiyonu içerisinde int türündeki x değişkenine 55 değerine atayan.
- Aşağıdaki fonksiyonları yazın.

```
void f1(int  n);
void f2(int *n);
```

- f1 fonksiyonu kendisine gelen n değişkenine 66 değerini atasın, f2 fonksiyonu kendisine gelen n işaretçisine 77 değerini atasın.
- main içerisinde f1 ve f2 fonksiyonlarını çağırarak x'i parametre olarak geçirin.
  - f2 fonksiyonuna **f2(&x);** olarak parametre geçirilecektir.
- Her fonksiyon çağırımı sonrasında main fonksiyonu içerisinde x değişkenini yazdırın.

Örnek3: İşaretçi Olan Fonksiyon Parametresi (devam…)


```c
#include <stdlib.h>

void f1(int n);
void f2(int *n);

int main()
{
    int x = 55;
    printf("x in degeri\n");
    printf("Fonksiyonlar cagrilmadan once: %d\n",x);

    /* f1 fonksiyonu çağrılıyor...*/
    f1(x);
    printf("f1 cagirildiktan sonra        : %d\n",x);

    /* f2 fonksiyonu çağrılıyor...*/
    f2(&x);
    printf("f2 cagirildiktan sonra        : %d\n",x);

    return 0;
}
```

Örnek3: İşaretçi Olan Fonksiyon Parametresi (devam…)


```
/* Değer geçerek aktarım */
void f1(int n)
{
    n = 66;
    printf("f1 fonksiyonu icinde          : %d\n",n);
}

/* Adres geçerek aktarım */
void f2(int *n)
{
    *n = 77;
    printf("f2 fonksiyonu icinde          : %d\n",*n);
}
```

Örnek3: İşaretçi Olan Fonksiyon Parametresi (devam…)

- x değişkeni f1(x) ve f2(&x) fonksiyonlarına, sırasıyla değer ve adres geçerek aktarılmıştır.
- f1 içinde x (`n = 66;` işlemi ile) değişime uğramış, fakat çağrılma işleminin sonucunda, *x'in değeri değişmemiştir*.
- Ancak f2 içinde x (`*n = 77` işlemi ile) değişimi sonucunda, *x'in değeri çağrıldıktan sonrada korunmuştur*.
- Yani, adres geçerek yapılan aktarımda, f2'ye aktarılan değer değil adres olduğu için, yollanan x parametresi f2 içinde değişikliğe uğrayacak ve bu değişim çağrıldığı satırdan itibaren devam edecektir.

İşaretçiler Kullanılarak Takas (Swap)

- İşaretçi kullanılarak "pass by reference" ile parametre aktarmayı en iyi anlatan örneklerden birisi de **swap işlemidir**.
- Swap iki değişkenin değerlerinin yer değiştirmesi anlamına gelmektedir.

```
int a = 10, b = 20, temp;
temp = a;
a = b;
b = temp;
```

İşaretçiler Kullanılarak Takas (Swap)

- Basit bir takas örneği yazalım:

```
int main()
{
    int a = 10, b = 20, temp;

    printf("a = %d, b = %d\n", a, b);
    temp = a;
    a = b;
    b = temp;

    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

Örnek4: İşaretçiler Kullanılarak Takas (Swap)

- Sizden main fonksiyonunda değerleri girilmiş iki değişken üzerine swap işlemini yapmanız istenmektedir. Bu işlem için kullanılacak fonksiyon prototipi aşağıdaki gibidir:

```
void takas(int *x, int *y)
```

- Takas öncesinde ve sonrasında main fonksiyonunda iki değişkenin de **adres** ve **değerlerini** yazdıralım.

Örnek4: İşaretçiler Kullanılarak Takas (Swap)


```
int main()
{
    int a, b;

    a = 22;
    b = 33;

    printf("takas oncesi : a=%d    b=%d\n",a,b);
    printf("takas oncesi : &a=%p    &b=%p\n",&a,&b);

    takas(&a, &b);

    printf("\ntakas sonrasi: a=%d    b=%d\n",a,b);
    printf("takas sonrasi : &a=%p    &b=%p\n",&a,&b);

    return 0;
}


void takas(int *x, int *y)
{
    int temp;

    temp = *x;
    *x = *y;
    *y =  temp;
}
```

```
takas oncesi : a=22    b=33
takas oncesi : &a=0028FF0C    &b=0028FF08

takas sonrasi: a=33    b=22
takas sonrasi : &a=0028FF0C    &b=0028FF08
```

Fonksiyon Geri Dönüş Değeri Olan İşaretçiler

- Fonkiyonların geri dönüş değeri bir işaretçi olabilir.
- Bu durumda, *fonksiyon bir değer değil* adres döndürecek demektir.
- Prototip aşağıdaki gibidir:

```
int *Fonksiyon(int a)
```

Örnek: Fonksiyon Geri Dönüş Değeri Olan İşaretçiler - BüyükBul

- Adım 1: Klavyeden iki tamsayı giriniz.
- Adım 2: Bu sayılardan büyük olanını bulmak ve için aşağıdaki prototipte bir fonksiyon yazınız. main() fonksiyonu içerisinde büyük olanının değerini ekrana yazdırınız.

```
int *BuyukBul(int a, int b)
```

Örnek: Fonksiyon Geri Dönüş Değeri Olan İşaretçiler - BüyükBul


```
int *BuyukBul(int a, int b);
int main()
{
    int s1, s2;
    int *p;

    printf("s1:");
    scanf("%d", &s1);

    printf("s2:");
    scanf("%d", &s2);

    p = BuyukBul(s1, s2);

    printf("Buyuk sayi:%d",*p);
    return 0;
}

int *BuyukBul(int a, int b)
{
    return (a > b) ? &a : &b;
}
```

Örnek: Fonksiyon Geri Dönüş Değeri Olan İşaretçiler - MaximumAdres


```
double x[6] = {1.1, 3.3, 7.1, 5.4, 0.2, -1.5};
```

tanımlayınız.

- Adım 1: Önce bu dizinin **indisleri**, dizi **değerleri** ve dizi elemanlarının **adresleri** main() fonksiyonunda ekrana yazdırılacaktır.
- Adım 2: Daha sonra, aşağıdaki prototipteki fonksiyon ile dizinin en büyük elemanı bulunacak ve bu elemanın adresi geriye döndürülerek ekrana yazdırılacaktır.

```
double* maxAdr(double a[], int boyut)
```

Örnek: Fonksiyon Geri Dönüş Değeri Olan İşaretçiler - MaximumAdres


```
int main()
{
    double x[6] = {1.1, 3.3, 7.1, 5.4, 0.2, -1.5};
    double *p;
    int k;
    // indis, dizi ve adresini ekrana bas
    for(k=0; k<6; k++)
    {
        printf("%d %lf %p\n", k, x[k], &x[k]);
    }

    p = maxAdr(x,6);

    printf("\nEn buyuk deger: %f\n", *p);
    printf("En buyuk adres: %p \n",  p);

    return 0;
}
```

Örnek: Fonksiyon Geri Dönüş Değeri Olan İşaretçiler - MaximumAdres


```
double* maxAdr(double a[], int boyut)
{
    double  ebd =  a[0];
    double *eba = &a[0];
    int i;
    for(i=1; i<boyut; i++)
    {
        if(a[i]>ebd)
        {
            ebd =  a[i]; // en büyük deger
            eba = &a[i]; // en büyük adres
        }
    }
    return eba;
}
```

Struct ve İşaretçiler

- Struct içerisindeki üye elemanları **bir işaretçi olabileceği** gibi **struct değişkenlerinin kendisi** de bir işaretçi olabilir.
- Kisi bir struct olmak üzere aşağıdaki gibi bir işaretçi tanımlanabilir ve üyelerine -> karakterleri ile ulaşılabilir.

```
Kisi *pk;
…
pk->Yas = 50;
```

Örnek: Struct ve İşaretçiler

- **Kisi** isimli bir struct tanımlayınız. **Isim** ve **Yas** üyelerine sahip olsun.
- Bu structtan
  - bir struct değişkeni ve
  - struct işaretçisi oluşturunuz.
- İşaretçiye struct atamalarını yapınız.
- Struct değişkenine değerler atayınız.
- İşaretçiye değerler atayınız.
- Ekranda yazdırınız.

Örnek: Struct ve İşaretçiler


```
typedef struct
{
    char Isim[50];
    int Yas;
} Kisi;

int main()
{
    Kisi k;
    Kisi *pk;

    pk = &k;
    strcpy(k.Isim, "ahmet");
    k.Yas = 19;

    pk->Yas = 20;
    printf("yas:%d\n", pk->Yas);


    (*pk).Yas = 21;
    printf("yas:%d\n", pk->Yas);
    return 0;
}
```

Fonksiyon İşaretçileri

- Fonksiyon işaretçileri, işaretçi (pointer) kavramının gücünü gösteren diğer bir uygulama alanıdır.
- Dizilerde olduğu gibi, **fonksiyon adları** da sabit işaretçidir.
- Fonksiyon kodlarının bellekte bir adreste tutulduğu şeklinde **düşünebiliriz**.
- Fonksiyon işaretçisi basit olarak fonksiyon adının saklandığı bellek adresini tutan bir **işaretçidir**.
- Fonksiyon işaretçileri sayesinde, fonksiyonlar başka fonksiyonlara *parametre olarak aktarılabilmektedirler*.

Fonksiyon İşaretçileri (devam…)

- Fonksiyonların bellekteki adresleri aşağıdaki kodlama kullanımıyla öğrenilebilir:

```
int f(int);       /* fonksiyon bildirimi */
int (*pf)(int); /*fonksiyon işaretçi bild.*/
pf = &f;          /*f'nin adresini pf'ye ata! */
```

Örnek: Fonksiyon İşaretçileri


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

int f(int n){
  int f=1, i;
  for(i=1; i<n; i++)
    f*=i;
  return f;
}

int main()
{
  int (*pf)(int);
  pf = &f;

  printf("Fonksiyonun adresi = %p\n", &f);
  printf("Fonksiyonun adresi = %p\n", pf);

  int sonuc = pf(4);
  printf("sonuc:%d\n",sonuc);

  return 0;
}
```

```
Fonksiyonun adresi = 00401334
Fonksiyonun adresi = 00401334
sonuc:6
```

void Tipindeki İşaretçiler

- void işaretçiler herhangi bir veri tipine ait olmayan işaretçilerdir.
- Bu özelliğinden dolayı, void işaretçi genel işaretçi (**generic pointer**) olarak da adlandırılır.
- void göstericiler, void anahtar sözcüğü ile bildirilir.

- **Örnek:**

```
void *adr;
```

KAYNAKLAR

- N. Ercil Çağıltay ve ark., C DERSİ PROGRAMLAMAYA GİRİŞ, Ada Matbaacılık, ANKARA; 2009.
- Milli Eğitim Bakanlığı "Programlamaya Giriş ve Algoritmalar Ders Notları", 2007
- C Programlama Dili, Şerafettin ARIKAN
- Problem Solving and Program Design in C, Hanly, Koffman
- http://www.AlgoritmaveProgramlama.com

Algoritma ve Programlama

İYİ ÇALIŞMALAR…