Ders 11: *Gösterici (Pointer) Kavramı*

```
###################- (%99)
```

En son güncelleme: Wed, 30 Nov 2011 13:22:02 +0200

- Giriş
- 11.1 Değişken ve Bellek Adresi
- 11.2 Gösterici Nedir?
- 11.3 Gösterici Aritmetiği
- 11.4 Gösterici ve Diziler arasındaki İlişki
- 11.5 Fonksiyon Parametresi Olan Göstericiler
- 11.6 Geri Dönüş Değeri Gösterici Olan Fonksiyonlar
- 11.7 Fonksiyon Göstericileri
- 11.8 `NULL` Gösterici
- 11.9 `void` Tipindeki Gösterici

Giriş

Hemen hemen bütün programlama dillerinin temelinde *gösterici* (pointer) veri tipi bulunmaktadır. Bir çok dil gösterici kullanım sunmamıştır veya çok sınırlı olarak sunmuştur. Fakat C Progrmalama Dili'nde göstericiler yoğun olarak kullanılır. Hatta gösterici kav bel kemiğidir. Kavranması biraz güç olan göstericiler için -latife yapılıp- C kullanıcılarını "gösterici kullanabilenler ve kullanmayanla iki gruba ayıranlar da olmuştur. Özetle, bir C programcısı gösterici kavramını anlamadan C diline hakim olamaz.

Türkçe yazılan C kitaplarda *pointer* kelimesi yerine aşağıdaki ifadelerden biri karşılaşılabilir:

pointer = işaretçi = gösterici = gösterge

Anlatımda, *gösterici* terimini kullanacağız.

11.1 Değişken ve Bellek Adresi

Bilgisayarın ana belleği (RAM) sıralı kaydetme gözlerinden oluşmuştur. Her göze bir adres atanmıştır. Bu adreslerin değerleri 0 ila olduğu üst değere bağlı olarak değişebilir. Örneğin 1GB MB bir bellek, 1024*1024*1024 = 1073741824 adet gözden oluşur. Değ bellekte işgal ettiği alanın bayt cinsinden uzunluğu `sizeof()` operatörüyle öğrenildiğini hatırlayın. (bkz: Program 2.1).

Bir programlama dillinde, belli bir tipte değişken tanımlanıp ve bir değer atandığında, o değişkene dört temel özellik eşlik eder:

1. değişkenin adı
2. değişkenin tipi
3. değişkenin sahip olduğu değer (içerik)
4. değişkenin bellekteki adresi

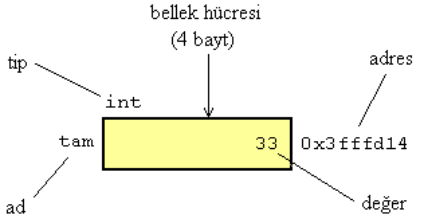
Örneğin `tam` adlı bir tamsayı değişkenini aşağıdaki gibi tanımladığımızı varsayalım:

```
int tam = 33;
```

Bu değişken için, `int` tipinde bellekte (genellikle herbiri 1 bayt olan 4 bayt büyüklüğünde) bir hücre ayrılır ve o hücreye 33 sayısı ikili sitemindeki karşılığı olan 4 baytlık (32 bitlik):

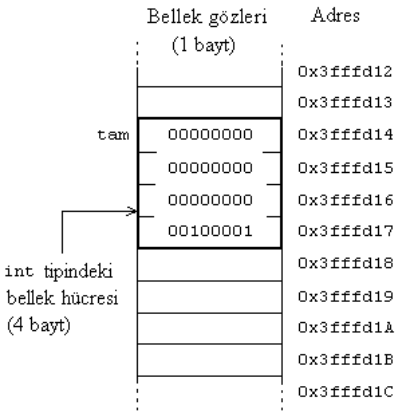
00000000 00000000 00000000 00100001

sayısı elektronik olarak yazılır. `tam` değişkenine ait dört temel özellik Şekil 11.1'deki gibi gösterilebilir:



Şekil 11.1: *Bir değişkene eşlik eden dört temel özellik*

Bellek adresleri genellikle onaltılık (hexadecimal) sayı sisteminde ifade edilir. `0x3fffd14` sayısı onluk (decimal) sayı sisteminde 67 karşık gelir. Bunun anlamı, `tam` değişkeni, program çalıştığı sürece, bellekte 67108116. - 67108120. numaralı gözler arasındaki 4 baytlı edecek olmasıdır. Şekil 11.1'deki gösterim, basit ama anlaşılır bir tasvirdir. Gerçekte, `int` tipindeki `tam` değişkeninin bellekteki yerle (değeri) Şekil 11.2'de gösterildiği gibi olacaktır.



Şekil 11.2: `tam` *adlı değişkenin bellekteki gerçek konumu ve ikilik düzendeki içeriği*

Değişkenin saklı olduğu adres, `&` karakteri ile tanımlı *adres operatörü* ile öğrenilebilir. Bu operatör bir değişkenin önüne konursa, o değ ile değil adresi ile ilgileniliyor anlamına gelir. Aşağıdaki program parçasının:

```
int tam = 33;

printf("icerik: %d\n",tam);
printf("adres : %p\n",&tam);
```

çıktısı:

```
icerik: 33
adres : 3fffd14
```

şeklindedir. Burada birinci satır `tam` değişkeninin içeriği, ikinci ise adresidir. Adres yazdırılırken `%p` tip belirleyicisinin kullanıldığına di

11.2 Gösterici Nedir?

Gösterici, bellek alanındaki bir gözün adresinin saklandığı değişkendir. Göstericilere veriler (yani değişkenlerin içeriği) değil de, o ve saklı olduğu hücrenin başlangıç adresleri atanır. Kısaca gösterici adres tutan bir değişkendir.

Bir gösterici, diğer değişkenler gibi, sayısal bir değişkendir. Bu sebeple kullanılmadan önce program içinde bildirilmelidir. Göst değişkenler şöyle tanımlanır:

```
tip_adı *gösterici_adı;
```

Burada *`tip_adı`* herhangi bir C tip adı olabilir. Değişkenin önünedeki `*` karakteri yönlendirme (indirection) operatörü olarak adl değişkenin veri değil bir adres bilgisi tutacağını işaret eder. Örneğin:

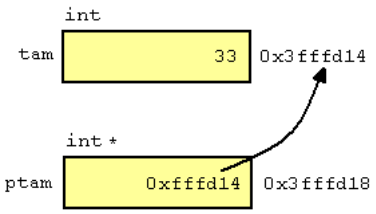
```
char  *kr;              /* tek bir karakter için */
int   *x;               /* bir tamsayı için */
float *deger, sonuc;    /* deger gösterici tipinde, sonuc sıradan bir gerçel değ
```

Yukarıda bildirilen göstericilerden; `kr` bir karakterin, `x` bir tamsayının ve `deger` bir gerçel sayının bellekte saklı olduğu yerlerin adresler

Bir göstericiye, bir değişkenin adresini atamak için adres operatörünü kullanabiliriz. Örneğin tamsayı tipindeki `tam` adlı bir değişk gösterici olsun. Derleyicide, aşağıdaki gibi bir atama yapıldığında:

```
int *ptam, tam = 33;
.
.
.
ptam = &tam;
```

`ptam` göstericisinin `tam` değişkeninin saklandığı adresi tutacaktır. Bu durum Şekil 11.3'deki gibi tasvir edilir.



Şekil 11.3: *Göstericinin bir değişkenin adresini göstermesi*

Şekil 11.3'deki gösterimde, `ptam` göstericisinin içeriği `tam` değişkeninin içeriği (33) değil adresidir (0x3fffd14). Ayrıca, `ptam` değişkeni, bir hücrede saklandığına ve bu hücrenin `int` değil `int *` tipinde bir bölge olduğuna dikkat ediniz. Buraya kadar anlatılanlar, Pr özetlenmiştir.

Program 11.1: *Bir değişkenin içeriğini ve adresini ekrana yazdırma*

```
01: /* 10prg01.c: ilk gösterici programı */
02:
03: #include <stdio.h>
04:
05: int main()
```

```
06: {
07:    int *ptam, tam = 33;
08:
09:    ptam = &tam;
10:
11:    printf("tam:  icerik = %d\n", tam);
12:    printf("tam:   adres = %p\n",&tam);
13:    printf("tam:   adres = %p\n",ptam);
14:
15:  return 0;
16: }
```

7. satırda değişkenler bildirilmiştir. 9. satırdaki atama ile `tam` değişkeninin *adresi*, `ptam` göstericisine atanmıştır. Bu satırdan itib değişkeninin gösterir. 11. satıda `tam`'ın içeriği (33 sayısı), 12. ve 13. satırda `tam`'ın adresi, `%p` tip karakteri ile, ekrana yazdırılmıştır incelendiğinde, `&tam` ve `ptam` içereriğinin aynı anlamda olduğu görülür.

ÇIKTI

```
tam:  icerik = 33
tam:   adres = 0x3fffd14
tam:   adres = 0x3fffd14
```

`tam` adlı değişkenin içeriğine `ptam` gösterici üzerinde de erişilebilir. Bunun için program içinde `ptam` değişkeninin önüne yönelendirm koymak yeterlidir. Yani `*ptam`, `tam` değişkeninin adresini değil içeriğini tutar. Buna göre:

```
*ptam = 44;
```

komutuyla, `ptam`'ın adresini tuttuğu hücreye 44 değeri atanır. Bu durum, Program 11.2'de gösterilmiştir.

Program 11.2: *Bir değişkenin içeriğini ve adresini ekrana yazdırma*

```
01: /* 10prg02.c: ikinci gösterici programı */
02:
03: #include <stdio.h>
04:
05: int main()
06: {
07:    int *ptam, tam = 33;
08:
09:    ptam = &tam;     /* ptam -> tam */
10:
11:    printf("&tam  = %p\n",&tam);
12:    printf("ptam  = %p\n",ptam);
13:    printf("\n");
14:
15:    printf("tam   = %d\n",tam);
16:    printf("*ptam = %d\n",*ptam);
17:    printf("\n");
18:
19:    *ptam = 44;      /* tam = 44 anlamında */
20:
21:    printf("tam   = %d\n",tam);
22:    printf("*ptam = %d\n",*ptam);
23:
24:  return 0;
25: }
```

ÇIKTI

```
&tam  = 0x3fffd14
ptam  = 0x3fffd14

tam   = 33
*ptam = 33

tam   = 44
*ptam = 44
```

Özetle `ptam = &tam` atamasıyla:

- `*ptam` ve `tam`, `tam` adlı değişkenin içeriği ile ilgilidir.
- `ptam` ve `&tam`, `tam` adlı değişkenin adresi ile ilgilidir.
- `*` yönlendirme ve `&` adres operatörüdür.

11.3 Gösterici Aritmetiği

Göstericiler kullanılırken, bazen göstericinin gösterdiği adres taban alınıp, o adresten önceki veya sonraki adreslere erişilmesi isteneb göstericiler üzerinde, aritmetik işlemcilerin kullanılmasını gerektirir. Göstericiler üzerinde yalnızca toplama (+), çıkarma (-), bir arttır eksiltme (--) operatörleri işlemleri yapılabilir.

Aşağıdaki gibi üç tane gösterici bildirilmiş olsun:

```
char   *kar;
int    *tam;
double *ger;
```

Bu göstericiler sırasıyla, bir karakter, bir tamsayı ve bir gerçel sayının bellekte saklanacağı adreslerini tutar. Herhangi bir anda, tuttuk sırasıyla 10000 (`0x2710`), 20000 (`0x4e20`) ve 30000 (`0x7530`) olsun. Buna göre aşağıdaki atama işelemlerinin sonucu:

```
kar++;
tam++;
ger++;
```

sırasyla 10001 (`0x2711`), 20004 (`0x4e24`) ve 30008 (`0x7538`) olur. Bir göstericiye ekleme yapıldığında, o anda tuttuğu adres ile eklenen

toplanmaz. Böyle olsaydı, bu atamaların sonuçları sırasıyla 10001, 20001 ve 30001 olurdu. Gerçekte, göstericiye bir eklemek, gösteric yerdeki veriden hemen sonraki verinin adresini hesaplamaktır.

Genel olarak, bir göstericiye *n* sayısını eklemek (veya çıkarmak), bekllekte gösterdiği veriden sonra (veya önce) gelen *n*. eleı hesaplamaktır. Buna göre aşağıdaki atamalar şöyle yorumlanır.

```
kar++;              /* kar = kar +   sizeof(char)    */
tam = tam + 5;      /* tam = tam + 5*sizeof(int)     */
ger = ger - 3;      /* ger = ger - 3*sizeof(double)  */
```

Program 11.3, bu bölümde anlatlanları özetlemektedir. İnceleyiniz.

Program 11.3: *Gösterici aritmetiği*

```
01: /* 10prg03.c: gösterici aritmetiği */
02:
03: #include <stdio.h>
04:
05:
06: int main()
07: {
08:    char   *pk, k = 'a';
09:    int    *pt, t = 22;
10:    double *pg, g = 5.5;
11:
12:    pk = &k;
13:    pt = &t;
14:    pg = &g;
15:
16:    printf("Onceki  adresler: pk= %p  pt= %p   pg= %p \n", pk, pt, pg);
17:
18:    pk++;
19:    pt--;
20:    pg = pg + 10;
21:
22:    printf("Sonraki adresler: pk= %p  pt= %p   pg= %p \n", pk, pt, pg);
23:
24:  return 0;
25: }
```

ÇIKTI

```
Onceki  adresler: pk= 0xbfbbe88f  pt= 0xbfbbe888   pg= 0xbfbbe880
Sonraki adresler: pk= 0xbfbbe890  pt= 0xbfbbe884   pg= 0xbfbbe8d0
```

11.4 Gösterici ve Diziler Arasındaki İlişki

C dilinde göstericiler ve diziler arasında yakın bir ilişki vardır. Bir dizinin adı, dizinin ilk elemanının adresini saklayan bir göstericidir. dizinin herhangi bir elemanına gösterici ile de erişilebilir. Örneğin:

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

şeklinde bir bildirim yapılsın. Buna göre aşağıda yapılan atamalar geçerlidir:

```
p = &kutle[0];    /* birinci elemanın  adresi p göstericisne atandı */
p = kutle;        /* birinci elemanın  adresi p göstericisne atandı */
q = &kutle[4];    /* son      elemanın adresi q göstericisne atandı */
```

İlk iki satırdaki atamalar aynı anlamdadır. Dizi adı bir gösterici olduğu için, doğrudan aynı tipteki bir göstericiye atanabilir. Ayrıca olmak üzere,

```
kutle[i];
```

ile

```
*(p+i);
```

aynı anlamdadır. Bunun sebebi, `p` göstericisi `kutle` dizisinin başlangıç adresini tutmuş olmasıdır. `p+i` işlemi ile `i+1`. elemanın adresi, v bu adresteki değer hesaplanır.

NOT

Bir dizinin, `i`. elemanına erişmek için `*(p+i)` işlemi yapılması zorunludur. Yani

```
*p+i;   /* p nin gösterdiği değere (dizinin ilk elemanına) i sayısını ekle */
*(p+i); /* p nin gösterdiği adresten i blok ötedeki sayıyı hesapla */
```

anlamındadır. Çünkü, `*` operatörü `+` operatörüne göre işlem önceliğine sahiptir.

Program 11.4'de tanımlanan fonksiyon kendine parameter olarak gelen *n* elemanlı bir dizinin aritmetik ortlamasını hesaplar.

Program 11.4: *Bir dizi ile gösterici arasındaki ilişki*

```
01: /* 10prg04.c: gösterici dizi ilişkisi */
02:
03: #include <stdio.h>
04:
05: double ortalama(double dizi[], int n);
06:
07: int main()
08: {
09:
```

```
10:    double a[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
11:    double o;
12:
13:    o = ortalama(a,5);
14:
15:    printf("Dizinin ortalaması = %lf\n",o);
16:
17:   return 0;
18: }
19:
20: double ortalama(double dizi[], int n)
21: {
22:    double *p, t=0.0;
23:    int i;
24:
25:    p = dizi;    /* veya p = &dizi[0] */
26:
27:    for(i=0; i<n; i++)
28:       t += *(p+i);
29:
30:    return (t/n);
31: }
```

ÇIKTI

```
Dizinin ortalaması = 3.300000
```

20. - 31. satırda tanımlanan fonksiyon aşağıdaki gibi de yazılabilirdi:

```
double ortalama(double dizi[], int n)
{
   double *p, t=0.0;

   for(p=dizi; p < &dizi[n]; p++)
      t += *p;

   return (t/n);
}
```

Bu fonksiyonda, döngü sayacı için (`i` değişkeni) kullanılmayıp, döngü içinde dizinin başlangıç adresi `p` göstericisine atanmış ve koşul k karşılaştırılması yapılmıştır. Bu durumda döngü, `p`'nin tuttuğu adresten başlar, ve `p`'nin adresi `dizi`nin son elemanının adresinden (`&diz` veya eşit olduğu sürece çevrim yinelenir.

11.5 Fonksiyon Parametresi Olan Göstericiler

C (ve C++) programlama dilinde fonksiyon parametreleri değer geçerek (pass by value) yada adres geçerek (pass by reference) ola Bölüm 8'deki uygulamalarda fonksiyonlara parametreler değer geçerek taşınmıştı. Bu şekilde geçirilen parametreler, fonksiyon içersi bile, fonksiyon çağrıldıktan sonra bu değişim çağrılan yerdeki değerini değiştirmez. Fakat, bir parametre adres geçerek aktarılısa, fonk değişikler geçilen parametreyi etkiler. Adres geçerek aktarım, gösterici kullanmayı zorunlu kılar.

Örneğin, Program 11.5'de fonksiyonlara değer ve adres geçerek aktarımın nasıl yapılacağı gösterilmiştir.

Program 11.5: *Bir değişkenin içeriğini ve adresini ekrana yazdırma*

```
01: /* 10prg05.c: Değer geçerek ve adres geçerek aktarım */
02:
03: #include <stdio.h>
04:
05: void f1(int  );   /* iki fonksiyon */
06: void f2(int *);
07:
08: int main()
09: {
10:    int x = 55;
11:
12:    printf("x in degeri,\n");
13:    printf("Fonksiyonlar cagrilmadan once: %d\n",x);
14:
15:    /* f1 fonksiyonu çağrılıyor...*/
16:    f1(x);
17:    printf("f1 cagirildiktan sonra       : %d\n",x);
18:
19:
20:    /* f2 fonksiyonu çağrılıyor...*/
21:    f2(&x);
22:    printf("f2 cagirildiktan sonra       : %d\n",x);
23:
24:   return 0;
25: }
26:
27:
28: /* Değer geçerek aktarım */
29:    void f1(int n){
30:       n = 66;
31:       printf("f1 fonksiyonu icinde         : %d\n",n);
32:    }
33:
34: /* Adres geçerek aktarım */
35:    void f2(int *n){
36:       *n = 77;
37:       printf("f2 fonksiyonu icinde         : %d\n",*n);
38:    }
```

5. ve 6. satırlada kendine geçilen parametrenin değerini alan `f1` fonksiyonu ve parametrenin adresini alan `f2` adlı iki fonksiyon örneği 11. satırdaki `x` değişkeni 16. ve 21. satırlarda, `f1(x)` ve `f2(&x)` fonksiyonlarına, sırasıyla değer ve adres geçerek aktarılmıştır. `f1` için işlemi ile) değişime uğramış, fakat çağrılma işleminin sonucunda, x'in değeri değişmemiştir. Ancak `f2` içinde x'in (`*n = 77` işlemi çağrıldıktan sonrada korunmuştur. Yani, adres geçerek yaplıan aktarımda, `f2`'ye aktarılan değer değil adres olduğu için, yollanan `x` içinde değişikliğe uğrayacak ve bu değişim çağrıldığı 21. satırdan itibaren devam edecektir.

ÇIKTI

```
x in degeri,
Fonksiyonlar cagrilmadan once: 55
f1 fonksiyonu icinde         : 66
f1 cagirildiktan sonra       : 55
f2 fonksiyonu icinde         : 77
f2 cagirildiktan sonra       : 77
```

Program 11.6'da iki tamsayı değişkeninin nasıl takas (swap) edileceği gösterilmiştir. Bu işlemi C porgramlama dilinde, eğer değişkenle bildirilmemişse, gösterici kullanmadan bu işlemi yapmak imkansızdır.

Program 11.6: *İki tamsayının birbiri ile takas edilmesi*

```
01: /* 10prg06.c: iki sayının birbiri ile takas edilmesi */
02:
03: #include <stdio.h>
04:
05: void takas(int *, int *);
06:
07: int main()
08: {
09:    int a, b;
10:
11:    a=22; b=33;
12:
13:    printf("takas oncesi : a=%d   b=%d\n",a,b);
14:
15:    takas(&a, &b);
16:
17:    printf("takas sonrasi: a=%d   b=%d\n",a,b);
18:
19:
20:  return 0;
21: }
22:
23:
24: void takas(int *x, int *y)
25: {
26:   int z;
27:
28:    z = *x;
29:   *x = *y;
30:   *y =  z;
31: }
```

ÇIKTI

```
takas oncesi : a=22   b=33
takas sonrasi: a=33   b=22
```

11.6 Geri Dönüş Değeri Gösterici Olan Fonksiyonlar

Fonkiyonların geri dönüş değeri bir gösterici olabilir. Bu durumda fonksiyon bir değer değil adres döndürecek demektir.

Program 11.7'da önce bir dizinin indisleri, dizi değerleri ve dizi elemanlarının adresleri ekrana basılır. Daha sonra, `maxAdr();` fonksiy en büyük elemanının adresi döndürülür. *Bu örnek progam, göstericilerin gücünü çok zarif bir biçimde bize sunmaktadır*. Lütfen inceley

Program 11.7: *Bir dizinin en büyük elemanının adresini öğrenmek*

```
01: /* 10prg07.c:  geri donus degeri gosterici olan fonksiyon */
02:
03: #include <stdio.h>
04:
05: double* maxAdr(double a[], int boyut){
06:   double  ebd =  a[0];
07:   double *eba = &a[0];
08:   int i;
09:   for(i=1; i<boyut; i++){
10:     if(a[i]>ebd){
11:       ebd =  a[i]; // en büyük deger
12:       eba = &a[i]; // en büyük adres
13:     }
14:   }
15:   return eba;
16: }
17:
18:
19: int main()
20: {
21:   double x[6] = {1.1, 3.3, 7.1, 5.4, 0.2, -1.5};
22:   double *p;
23:   int k;
24:   // indis, dizi ve adresini ekrana bas
25:   for(k=0; k<6; k++){
26:     printf("%d %lf %p\n", k, x[k], &x[k]);
27:   }
28:
29:   p = maxAdr(x,6);
30:
```

```
31:    printf("En büyük deger: %lf\n", *p);
32:    printf("En büyük adres: %p \n",  p);
33:    printf("En büyük konum: %d \n",  int(p-&x[0]));
34:
35:    return 0;
36: }
```

Dizi elemanları 21. satırda belirlenir. Bu dizinin indisleri, değerleri ve adresleri 26. satırda ekrana basılmıştır. En büyük elemanın adre `p = maxAdr(a,6);` ile `p` göstericisine atanmıştır. 5. satırda bildirilen `maxAdr();` fonksiyonu, en büyük elemanın adresini hesaplayıp gönderir. Burada dikkat edilmesi gereken husus, fonksiyonun dönüş değerinin yerel `eba` göstericisi olmasıdır. `eba` göstericisi 12. satırda fonksiyon parametersi olan dizinin en büyük elemanın adresini tutmaktadır. Son olarak, fonksiyon çağırıldıktan sonra, `p` göstericisin gö tuttuğu adres ve dizinin birinci elemanına göre konumu (indisi) ekrana basılmıştır. Indis hesabı `int(p-&x[0])` işlemi ile yapılabilir. göstericisin tuttuğu adres ile dizinin ilk elemanının adresi arasındaki farktır. Sonuç yine bir adres olduğu için tamsayı değer elde etı takısı kullanılmıştır. Netice itibarıyla bir fonksiyon ile üç şey aynı anda öğrenilmiş olur.

ÇIKTI

```
0 1.100000 0x7fff41b29ec0
1 3.300000 0x7fff41b29ec8
2 7.100000 0x7fff41b29ed0
3 5.400000 0x7fff41b29ed8
4 0.200000 0x7fff41b29ee0
5 -1.500000 0x7fff41b29ee8

En büyük deger: 7.100000
En büyük adres: 0x7fff41b29ed0
En büyük konum: 2
```

11.7 Fonksiyon Göstericileri

Fonksiyon göstericileri, gösterici (pointer) kavramının gücünü gösterin diğer bir uygulama alanıdır. Dizilerde olduğu gibi, fonksiyon ad göstericidir.
Fonksiyon betiğinin (kodlarının) bellekte bir adreste tutulduğu şeklinde düşünebiliriz. Fonksiyon göstericisi basit olarak fonksiyon adın bellek adresini tutan bir göstericidir. *Fonksiyon göstericileri sayesinde fonksiyonlar başka fonksiyonlara parametre olarak aktarılabilm*

Fonksiyon adının bellete yer işgal ettiği şöyle öğrenilebilir:

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

Program 11.8: *Bir fonksiyonun 'adresini' iki yoldan öğrenme*

```
01: /* 10prg08.c:  Bir fonksiyonun 'adresini' öğrenme */
02:
03: #include <stdio.h>
04:
05: int f(int n){
06:   int f=1, i;
07:   for(i=1; i<n; i++)
08:     f*=i;
09:   return f;
10: }
11:
12: int main()
13: {
14:   int (*pf)(int);
15:   pf = &f;
16:
17:   printf("Fonksiyonun adresi = %p\n", &f);
18:   printf("Fonksiyonun adresi = %p\n", pf);
19:
20:   return 0;
21: }
```

ÇIKTI

```
Fonksiyonun adresi = 0x4005b0
Fonksiyonun adresi = 0x4005b0
```

Aşağıdaki ikinci örnekte, bir fonksiyon diğer fonksiyona parametre olarak geçirilmiş ve sayısal türevi hesaplanmıştır. Türev hesaplanır fark yaklaşımı (central difference approximation) yöntemi kullanılmıştır.

NOT

mfy yönteminde f(*x*) fonksiyonunun (*h* küçük bir değer olmak üzere) Taylor açılımları söyledir:

$$f(x+h) = f(x) + h*f'(x) + h^2*f''(x)/2! + h^3*f'''(x)/3! + \ldots$$
$$f(x-h) = f(x) - h*f'(x) + h^2*f''(x)/2! - h^2*f'''(x)/3! + \ldots$$

-

$$f(x+h) - f(x-h) = 2*h*f'(x) + O(h^3)$$

Burada $O(h^3)$'lü terimler ihmal edilirse birinci türev yaklaşık olarak:

$$f'(x) = [f(x+h) - f(x-h)]/2h$$

formülü ile hesaplanır.

Program 11.9: *Türev alan fonksiyon*

```c
/* 10prg09.c:  Fonksiyon Göstericisi ile türev hesabı */

#include <stdio.h>

double f(double);
double turev( double (*)(double), double);

int main()
{

  double x = 1.1;

  printf("Fonksiyon x = %lf deki degeri = %lf\n", x, f(x));
  printf("Fonksiyon x = %lf deki turevi = %lf\n", x, turev(f, x) );

  return 0;
}


// türevi hesaplanacak fonksiyon
double f(double x){
  return x*x*x - 2*x + 5.;
}
// sayısal türev alan fonksiyon
double turev( double (*fonk)(double x), double x){
  double h = 1.0e-3;
  return (fonk(x+h)-fonk(x-h)) / (2*h);
}
```

ÇIKTI

```
Fonksiyon x = 1.100000 deki degeri = 4.131000
Fonksiyon x = 1.100000 deki turevi = 1.630001
```

11.8 NULL Gösterici

Bir göstericinin bellekte herhangi bir adresi göstermesi, veya öncden göstermiş olduğu adres iptal edilmesi istemirse NULL sabiti kull derleyicide ASCII karakter tablosunun ilk karakteridir ve '\0' ile sembolize edilir.

```c
int *ptr, a = 12;
.
.
ptr = &a;      /* ptr bellekte a değişkenin saklandığı yeri gösteriyor  */
.
.
ptr = NULL;    /* ptr bellekte hiç bir hücreyi göstermiyor */
*ptr = 8       /* hata! NULL göstericinin gösterdiği yere bir değer atanamaz */
```

11.9 void Tipindeki Göstericiler

void göstericiler herhangi bir veri tipine ait olmayan göstericilerdir. Bu özelliğinden dolayı, void gösterici genel gösterici (generic poi adlandırılır.

void göstericiler, void anahtar sözcüğü ile bildirilir. Örneğin:

```c
void *adr;
```

gibi.

void göstericiler yalnızca adres saklamak için kullanılır. Bu yüzden diğer göstericiler arasında atama işlemlerinde kullanılabilir. Örn atamada derleyici bir uyarı veya hata mesajı vermez:

```c
void *v;
char *c;
.
.
.
v = c;    /* sorun yok !*/
```

Program 11.10'de void tipindeki bir göstericinin, program içinde, farklı tipteki verileri nasıl göstereceği ve kullanılacağı örneklenmiştir

Program 11.10: *void gösterici ile farklı tipteki verileri gösterme*

```c
/* 10prg10.c:  void gosterici (generic pointer) uygulamasi */

#include <stdio.h>

int main()
{
  char    kar = 'a';
  int     tam = 66;
  double  ger = 1.2;
  void   *veri;

  veri = &kar;
  printf("veri -> kar: veri  %c  karakter degerini gosteriyor\n", *(char *) veri);

  veri = &tam;
  printf("veri -> tam: simdi veri  %d  tamsayi degerini gosteriyor\n", *(int *) veri);

```

```
18:   veri = &ger;
19:   printf("veri -> ger: simdi de veri  %lf  gercel sayi degerini gosteriyor\n", *(double *) veri);
20:
21:   return 0;
22: }
23:
24:
```

ÇIKTI

```
veri -> kar: veri  a  karakter degerini gosteriyor
veri -> tam: simdi veri  66  tamsayi degerini gosteriyor
veri -> ger: simdi de veri  1.200000  gercel sayi degerini gosteriyor
```

Benzer olarak, fonksiyon parameterelerinin kopyalanması sırasında da bu türden atama işlemleri kullanılabilir. Uygulamada, tipten b işlemlerinin yapıldığı fonksiyonlarda, parametre değişkeni olarak `void` göstericiler kullanılır. Örneğin

```
void free (void *p)
{
    .
    .
    .
}
```

Parametresi `void *p` olan `free` fonksiyonu, herhangi türden gösterici ile çağrılabilir.

Powered by PHP