GÖSTERİCİLER

Gösterici nedir? Niçin İhtiyaç Duyarız?

Şu ana kadar gördüğümüz `char`, `int`, `long`, `float`, `double` tipli değişkenler ve bunlara ait diziler, değer tipler (value type) olarak adlandırılır. Çünkü kimliklendirilen değişken, tutulacak değeri içerir. Değer tiplerin yanı sıra değişkenlerin adreslerini tutan değişkenlere de ihtiyaç duyarız. İşte adres tutan bu değişkenlere gösterici tipler (pointer type) adını veririz. Verilerin tipine göre gösterici tanımlandığından göstericiler, türetilmiş tiplerdir (derived type).

Gösterici tipler, gösterdiği yerdeki verileri değiştirmek için de kullanılır. Bu nedenle gösterici tipler tanımlanırken gösterdiği yerdeki verinin tipine göre tanımlanırlar. Bütün gösterici tipler bellekte aynı miktarda yer kaplar. Çünkü hepsi adres tutar. Ancak işaret ettikleri yerdeki veriler, verinin tipine bağlı olarak bellekte farklı miktarda yer kaplar. Bütün bu tanımlamaların ışığında göstericileri, vatandaşların ikametlerini gösteren adres numaraları olarak düşünebiliriz.

Gösterici tiplere ihtiyaç duyulmasının sebebi hız ve esnekliktir. Genel olarak birkaç madde ile sıralayacak olursak;

- Belleğin istenilen bölgesine erişim sağlamak için göstericiler kullanılır. Günümüz modern dillerinde referans tipler kullanılır, ancak referans tiplerde gösterilen adreste bir nesnenin bulunduğu garanti edilir göstericiler gibi her bellek bölgesine erişilemez.
- Bazen çalıştırma anında yeni bellek gölgelerine ihtiyaç duyarız bu durumda göstericiler gereklidir. Dinamik veri yapıları (dynamic data structure) göstericiler yardımıyla oluşturulup yönetilir.
- Fonksiyonlarda yerel değişkenler ve argümanlar yığın belleğe itilir ve fonksiyondan geri dönüldüğünde bu değişkenler eski haline yığın bellekten geri çekilerek çevrilir. Fonksiyonun yerel değişken ya da argümanlardan birini değiştirmesi istendiğinde fonksiyona argüman olarak değişkenin adresi verilir. Böylece fonksiyon içinde gösterici olarak işlem gören argümanlar fonksiyondan geri döndüğünde değerleri değişmiş olur.

Göstericilere olan ihtiyacı bir başka örnekle de açıklayabiliriz; Bir ormandaki ağaçları boylarına göre sıralamaya kalkarsak her birini yer değiştirme yöntemiyle sıralamak oldukça külfetli ve zaman alana bir işlemdir. Halbuki ağaçlara numara vererek ve bu numaraların karşısına uzunluklarını yazacağımız bir liste oluşturulduğunda sadece numaraları sıralamak oldukça kolay ve zahmetsiz bir işlemdir. İşte buradaki numaraları tutan değişkenler göstericilerdir.

Gösterici Tanımlama

Göstericiler, gösterdiği yerdeki verinin tipine göre tanımlanırlar. Gösterici tanımlanırken veri tipi izleyen başvuru kaldırma işleci (dereference operator) olan yıldız (`*`) kullanılır. Başvuru kaldırma terimi, gösterici tarafından tutulan bellek adresinde saklanan değere erişmeyi de ifade eder.

```
veritipi* gostericikimligi;
```

Göstericilere değer atatama aşağıdaki şekilde yapılır;

```
gostericikimligi = &degiskenkimliği;
```

Bir değişkenin adresinin adres işleci (`&`) ile elde edilebileceği *Tekli İşleçler* başlığında anlatılmıştı. Aşağıda göstericilere ilişkin kod örnekleri verilmiştir;

```
char* ptrToChar;  /*tuttuğu adreste char tipinde veri olan ptrToChar kimlikli göstericisi
tanımlandı. */

int i=10;
int* ptrToInt=&i; /* tuttuğu adreste int tipinde veri olan ptrToInt kimlikli göstericisi
tanımlandı ve ilk değer olarak i değişkeninin adresi atanıyor */
*ptrToInt =20; /* ptrToInt göstericisinin gösterdiği adresteki tamsayı değeri 20 yaptık.
ptrToInt göstericisi, i değişkeninin adresini tuttuğundan i değişkeninin değeri de 20 olmuştur
```

```
*/
```

Bazı durumlarda göstericilerin veya gösterdiği değerlerin sabit olması gerekebilir. Üç durum ortaya çıkabilir;

– Gösterilen değerin sabit olması: Bu durumda gösterici tanımlaması aşağıdaki gibi yapılır;

```
int main() {
   int i=10;
   const int* ptrToi=&i; // değerin sabit olması durumunda gösterici tanımı

   *ptrToi=20; /* HATA! pi göstericisinin tuttuğu adresteki tamsayı
    değeri 20 yapılamaz. */

   return 0;
}
```

– Göstericinin işaret ettiği adresin sabit olması: Kısaca göstericinin sabit olması durumunda tanımlama aşağıdaki gibi yapılır;

```
int main() {
   int i=10;
   int* const ptrToi=&i; // göstericinin sabit olması durumunda gösterici tanımı
   /* bu tür tanımlamada adres ilk değer (initialize) olarak mutlaka verilmelidir. */

   int j=30;
   *ptrToi=20; // pi göstericisinin tuttuğu adresteki tamsayı değeri 20 olur.
   ptrToi =&j; //HATA! : pi göstericisinin tuttuğu adres değiştirilemez!

   return 0;
}
```

– Hem göstericinin hem de değerin sabit olması durumu:

```
int main() {
   int j=30;
   const int* const ptr=&j; /* Hem gösterici, hem de gösterilen veri sabit */
   /* bu tür tanımlamada da adres ilk değer (initialize) olarak mutlaka verilmelidir. */
   return 0;
}
```

Atanmış kesin bir adresiniz yoksa, bir gösterici değişkenine `NULL` değeri atamak her zaman iyi bir uygulamadır. İlk değeri olmayan gösterici tanımlamak yerine, ilk değeri sıfır olan göstericiler tanımlanmak doru bir yöntemdir. Bu durumda ilk değer `NULL` olarak atanır. Bu durumda gösterici, NULL gösterici (NULL pointer) adı verilir. Aşağıdaki gibi kullanılır;

```
veritipi* gostericikimligi = NULL;
gostericikimligi = NULL;
```

Örnek kod aşağıda verilebilir;

```
#include <stdio.h>
int main(){

 int* ptr = NULL;
 printf("ptr göstericisinin tuttuğu adres: %p\n", ptr);

 int agirlik=80;
 ptr=&agirlik;
 printf("ptr göstericisinin tuttuğu adres: %p ve değeri: %d\n", ptr, *ptr);

 return 0;
}
/* Program Çıktısı:
ptr göstericisinin tuttuğu adres: (nil)
ptr göstericisinin tuttuğu adres: 0x7ffdedc33dec ve değeri: 80
```

```
...Program finished with exit code 0
*/
```

Bir gösterici bir başka göstericinin adresini tutabilir. Buna çifte gösterici (double pointer) adı verilir. Aşağıda buna ilişkin bir örnek bulunmaktadır;

```
#include <stdio.h>
int main(){
   int var = 10;
   int* intptr = &var;
   int** ptrptr = &intptr;  //double pointer

   printf("var:%d &var:%p \n",var, &var);
   printf("inttptr: %p *inttptr: %p \n\n", intptr, &intptr);

   printf("var: %d *intptr: %d \n", var, *intptr);
   printf("ptrptr: %p &ptrtptr: %p \n", ptrptr, &ptrptr);
   printf("&intptr: %p *ptrptr : %p \n\n", &intptr, *ptrptr);
   printf("var: %d *intptr: %d **ptrptr: %d", var, *intptr, **ptrptr);

   return 0;
}
/*Program Çıktısı:
var:10 &var:0x7ffd52c80524
inttptr: 0x7ffd52c80524 *inttptr: 0x7ffd52c80528

var: 10 *intptr: 10
ptrptr: 0x7ffd52c80528 &ptrtptr: 0x7ffd52c80530
&intptr: 0x7ffd52c80528 *ptrptr : 0x7ffd52c80524

var: 10 *intptr: 10 **ptrptr: 10

...Program finished with exit code 0
*/
```

Bir tamsayının hangi 4 bayt bellek alanından oluştuğu *Değişken Tanımlama* başlığında anlatılmıştı. Aşağıda tanımlanan bir tamsayının hangi baytlardan oluştuğunu gösteren program verilmiştir.

```
#include <stdio.h>
int main() {
    int i=0x10203040;
    char* p= (char*) &i;
    /* int gösteren adres,
       bayt/char içeriyor diye
       (char*) ifadesiyle tip dönüşümü (cast) yapılıyor.
    */
    printf("Sayı  :%10d-%x\n",i,i);
    printf("1.bayt:%10d-%x\n",*p,*p);
    printf("2.bayt:%10d-%x\n",*(p+1),*(p+1));
    printf("3.bayt:%10d-%x\n",*(p+2),*(p+2));
    printf("4.bayt:%10d-%x\n",*(p+3),*(p+3));
    return 0;
}
/*Program Çıktısı:
Sayı  : 270544960-10203040
1.bayt:        64-40
2.bayt:        48-30
3.bayt:        32-20
4.bayt:        16-10

...Program finished with exit code 0
*/
```

Gösterici Aritmetiği

Bilgisayarların tasarımından ötürü işlemciye bağlı belleklerde her bir bayt ayrı adreslenir. Birkaç bayt (`char`) veri içeren bir bellek bölgesini işaret eden bir gösterici tanımlandığında göstericinin işaret ettiği adres, bir sonraki baytı göstermesi istendiğinde gösterici adresi 1 artar.

```
char dizi1[5]={'I','L','H','A','N'}; /* Bellekte Art arda 5 bayt
                                        yer ayrılmış dizi. */
char* pc=&dizi1[0]; /* pc göstericisi dizinin ilk elemanını
                       gösterecek adres (65FDE0) varsayıldı. */
*pc='X';         //dizi {'X','L','H','A','N'} oldu.
pc++;            //pc göstericisi bir sonraki baytı gösterecek (65FDE1) şekilde artırıldı.
*pc='Y';         // dizi {'X','Y','H','A','N'} oldu.
pc+=2;           //pc göstericisi 2 sonraki baytı gösterecek (65FDE2) şekilde artırıldı.

*pc='Z';         // dizi {'X','Y','H','A','Z'} oldu.
```

Benzer şekilde birkaç tamsayı (`int`) veri içeren bir bellek bölgesini işaret eden bir gösterici tanımlandığında göstericinin işaret ettiği adres, bir sonraki tamsayıyı göstermesi istendiğinde gösterici adresi 4 artar. Çünkü tamsayılar bellekte 4 bayt yer kaplar.

```
int dizi2[5]={2,0,3,10,2}; // Bellekte art arda 5 tamsayı (5x4bayt) yer ayrılmış dizi.
int* pi=&dizi2[0];  /*pi göstericisi dizinin ilk elemanını
                      gösterecek adres (65FDE0) varsayıldı. */
*pi=-1;          //dizi {-1,0,3,10,2} şekline döndü.
pi++;            //pi göstericisi ikinci elemanını gösterecek (65FDE4) şekilde artırıldı.
*pi=-2;          //dizi {-1,-2,3,10,2} şekline döndü.
pi+=2;           //pi göstericisi 2 sonraki elemanı gösterecek (65FDEC) şekilde artırıldı.
*pi=-3;          //dizi {-1,-2,3,10,-3} şekline döndü.
```

Görüldüğü üzere göstericilerin üzerinde işleçler (operator) işlem yaparken göstericinin işaret ettiği verinin tipine göre farklı işlem yaparlar. Göstericiler ile aritmetik ve atama işleçleri (`++`, `--`, `+`, `-`, `+=` ve `-=`) ile karşılaştırma işleçleri (`<`, `>` ve `==`) çalışır, diğer işleçler çalışmaz.

Gösterici Dizileri

Göstericiler de dizi olarak tanımlanabilir. Aşağıda buna ilişkin bir örnek verilmiştir;

```
#include <stdio.h>
int main() {
  int i = 10, j=20;
  float f=1.0;
  int k=30, l=40;

  int* ptr[4]; /* int gösteren 4 gösteri içeren bir dizi tanımlandı */

  /* Göstericilere ilk değer veriliyor*/
  ptr[0] = &i; // Birinci eleman i değişkeninin adresini
  ptr[1] = &j; // İkini eleman j değişkeninin adresini
  ptr[2] = &l; // Üçüncü eleman l değişkeninin adresini
  ptr[3] = &k; // Dördüncü eleman k değişkeninin adresini

  // Değerlere Ulaşma
  int indis;
  for (indis = 0; indis < 4; indis++)
    printf("ptr[%d] ile gösterilen değer: %d\n", indis, *ptr[indis]);

  return 0;
}
/*Program Çıktısı:
ptr[0] ile gösterilen değer: 10
ptr[1] ile gösterilen değer: 20
```

```
ptr[2] ile gösterilen değer: 40
ptr[3] ile gösterilen değer: 30

...Program finished with exit code 0
*/
```

Parametre Olarak Göstericiler

Fonksiyonlar çağrılırken argüman olarak giren değerlerin değişmeyeceği *Fonksiyon Çağırma Süreci* başlığında anlatılmıştı. Fonksiyona parametre olarak giren değerler, fonksiyon bloğu içinde değişse bile fonksiyondan geri dönülürken aynı çağrı ortamını sağlamak için kaybolurlar.

```
#include <stdio.h>
void degistir(int,int);
int main(){
   int a = 10, b = 20;
   degistir(a, b);
   printf("1- a:%d, b:%d\n", a, b); //Çıktı: a:10, b:20
}
void degistir(int pX, int pY) {
   int z = pX;
   pX=pY;
   pY=z;
}
```

Fonksiyona giren argümanları gösterici olarak tanımlarsak çağrı ortamına geri dönüldüğünde yerel değişkenler de değişmiş olur. Çünkü fonksiyona değişken adresleri değer olarak girmiştir. Buna ilişkin örnek aşağıda verilmiştir;

```
#include <stdio.h>
void degistir(int *pX, int *pY);
int main(){

   int a = 10, b = 20;
   degistir(&a, &b);
   printf("2- a:%d, b:%d\n", a, b); //Çıktı: a:20, b:10
}
void degistir2(int* pX, int* pY){
   int z = *pX;
   *pX=*pY;
   *pY=z;
}
```

Göstericilerin Dizileri İşaret Etmesi

Çoğu durumda, bir dizi yardımıyla gerçekleştirdiğimiz işleri gösterici ile de gerçekleştirilebiliriz. Genellikle dizilerin kendisi yerine ilk elemanlarını işaret eden göstericiler kullanırız.

```
#include <stdio.h>
int main () {
   int dizi[5] = {10, 20, 30, 40, 50};
   int* ptr = dizi;  // int* ptr=& dizi[0]; olarak da kodlanabilir.

   int indis;
   printf("Dizi elemanlarına gösterici ile erişim: \n");

   for(i = 0; i < 5; i++) {
      printf("dizi[%d]: %d\n", indis, *(ptr+indis));
   }

   return 0;
}
```

Fonksiyonlara dizileri gösteren göstericileri parametre olarak verebiliriz;

– Diziler parametre olarak kullanılırken adres operatörü kullanılmaz. Diziler, gösterici gibi davranırlar. Yani bir dizinin adı, dizinin ilk öğesinin adresi gibi davranır.

```
float notlar[10];
float notlarOrtalamasi(float*); //prototype
//...
float ort=notlarOrtalamasi(notlar); // yada notlarOrtalamasi(&notlar[0]);
//...
float matris[4][3];
float defetminant(float**,int,int); //prototype
//...
float det=defetminant(matris,4,3);
//...
```

– Gösterici gibi davrandıklarından dizi elemanları fonksiyon içinde yine dizi gibi kullanılabilir.

– Gösterici gibi davrandıklarından dizi elemanları fonksiyon içinde değiştirilebilir. Fonksiyondan dönüldüğünde elemanlar değişiklik yapılmış şekliyle işlemler devam eder.

Aşağıda öğrencilerin notlarına ilişkin işlemler yapılan bir program verilmiştir.

```
#include <stdio.h>
#include <math.h>
#define OGRENCISAYISI 10

float notlarOrtalamasi(float*);
void notlariArtir(float*);

int main(){
   float notlar[OGRENCISAYISI]= { 55.0,60.0,70.0,35.0,30.0,
                                  50.0,65.0,90.0,95.0,100.0 };
   float toplam=notlarOrtalamasi(notlar);
   printf("Notlar Ortalaması: %f\n", toplam);
   notlariArtir(notlar);  // Bu fonksiyonla dizi elemanları değiştiriliyor
   int indis;
   for (indis=0; indis<OGRENCISAYISI; indis++)
      printf("Not[%d]=%f\n",indis,notlar[indis]);
   return 0;
}

float notlarOrtalamasi(float* pNotlar){
   int indis;
   float top=0.0;
   for (indis=0; indis<OGRENCISAYISI; indis++)
      top+=pNotlar[indis]; // Gösterici dizi gibi kullanılıyor
   return top/OGRENCISAYISI;
}

void notlariArtir(float* pNotlar) {
   int indis;
   for (indis=0; indis<OGRENCISAYISI; indis++)
      if (pNotlar[indis]<=90) //Göstericinin gösterdiği değerler değiştiriliyor
          pNotlar[indis]+=10.0;

};
/*Program Çıktısı:
Notlar Ortalaması: 65.0
Not[0]=65.0
Not[1]=70.0
Not[2]=80.0
Not[3]=45.0
Not[4]=40.0
Not[5]=60.0
```

```
Not[6]=75.0
Not[7]=100.0
Not[8]=95.0
Not[9]=100.0

...Program finished with exit code 0
*/
```

Fonksiyon Göstericisi

Geri çağırma (callback) fonksiyonları, özellikle olay güdümlü programlamada (event-driven programming) çok kullanışlıdır. Belirli bir olay tetiklendiğinde, bu olaylara yanıt olarak ona eşlenen bir geri çağırma fonksiyonu çalıştırılır. Genellikle grafik ara yüzlü işletim sistemlerinde kullanıcı ara yüzünde bir düğmeye tıklanması gibi bir olay, önceden tanımlanmış bir dizi işlemi başlatabilir. Bu durum, geri çağırma işlevleriyle gerçekleştirilir.

Geri çağırma fonksiyonu, temel olarak, başka bir koda argüman olarak geçirilen ve belirli bir zamanda argümanı geri çağırması beklenen bir icra edilebilir koddur. Geri çağırma mekanizması fonksiyon göstericisine bağlıdır. Fonksiyon göstericisi, bir fonksiyonun bellek adresini depolayan bir değişkendir. Aşağıda buna ilişkin basit bir örnek bulunmaktadır;

```
#include <stdio.h>
void merhaba() {
   printf("Merhaba!\n");
}
void callback(void (*ptr)()) {
   printf("Geri çağırma fonksiyonu çağrılacak!\n");
   (*ptr)();   // Geri çağırma fonksiyonu çağrılıyor
}
main() {
   void (*ptr)() = merhaba;
   callback(ptr);
}
```

Fonksiyonlardan Bir Diziyi Geri Döndürme

C dilinde bir fonksiyonun geri dönüş değeri olarak tüm bir dizi verilemez! Ancak, bir dizinin göstericisini fonksiyondan geri dönüş değeri olarak döndürebilirsiniz. Burada dikkat edilmesi gereken fonksiyondan çıkılınca dizinin bellekten kaldırılmamasıdır. Aşağıda bir tamsayı dizi göstericisini döndüren bir fonksiyon örnekleri verilmiştir;

```
int* tekBoyutluDiziDondur() {
    /*... */
}
int** ikiBoyutluDiziDondur() {
    /*... */
}
```

Yerel değişkenlerin yığın (stack) bellekte tutulduğu ve fonksiyondan geri dönünce fonksiyon yerelindeki değişkenlerin kaybolduğu *Çağrı Kuralı*

Çağrı kuralı (calling convention), bir fonksiyon çağrısıyla karşılaşıldığında argümanların fonksiyona hangi sıraya göre iletileceğini belirtir. İki olasılık vardır; Birincisi argümanlar soldan başlanarak sağa doğru fonksiyona geçirilir. İkincisi ise C dilinde kullanılır ve argümanlar sağdan başlanarak sola doğru fonksiyona geçirilir.

```
fonksiyon(a,b,c,d,e);
```

Yukarıdaki örnek incelendiğinde argümanların hangi sırada fonksiyona geçirildiğinin bir önemi yoktur. Ancak aşağıdaki verilen örnekteki durumu inceleyelim;

```
int a = 1;
```

```
printf("%d %d %d", a, ++a, a++);
```

Bu kodun çıktısı `1 2 3` olarak beklenir ancak çağrı kuralından ötürü çıktı `3 3 1` olur. Bunun nedeni C'nin çağrı kuralının sağdan sola olmasıdır. Bunu şöyle açıklayabiliriz;

6. Fonksiyona sağdan ilk argüman olarak a değişkeni değeri 1 geçirilir.
7. Ardından a++ ifadesi ile a değişkeni 2'ye yükseltilir.
8. Sonra ++a ifadesi ile a değişkeni 3'e yükseltilir.
9. Fonksiyona ikinci argüman olarak 3 geçirilir.
10. Fonksiyona son olarak a'nın son değeri olan 3 geçirilir.

Depolama Sınıfları başlığında anlatılmıştı. Bu nedenle bir fonksiyon içinde tanımlanan bir dizinin göstericisi geri döndürülecek ise `static` olarak tanımlanır.

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

#define BOYUT 10

int* rastgeleOgrenciNotlari( ) {
    static int  notlar[BOYUT];
    for (int i = 0; i < BOYUT; ++i)
        notlar[i] = rand()%101;
    return notlar;
}
void notlariYaz(int* pNotlar,int pUzunluk) {
    printf("Öğrenci Notları:\n");
    for (int i = 0; i < pUzunluk; ++i)
        printf("%4d", pNotlar[i]);
    return;
}
int main () {
   int *notlar;
   srand(time(NULL));
   notlar = rastgeleOgrenciNotlari();
   notlariYaz(notlar,BOYUT);
   return 0;
}
/*Program Çıktısı:
Öğrenci Notları:
  30  31  13  40  41  37  34  83  93  76

...Program finished with exit code 0
*/
```