A

BLM102

PROGRAMLAMA DİLLERİ II

Yrd. Doç. Dr. Baha ŞEN

baha.sen@karabuk.edu.tr

KBUZEM
Karabük Üniversitesi
Uzaktan Eğitim Araştırma ve Uygulama Merkezi

1. Pointer (gösterici, işaretçi)

1.1. Tanımlanması ve Kullanımı

Bir veri bloğunun bellekte bulunduğu adresi içeren (gösteren) veri tipidir. Tanımlama biçimi:

```
veri tipi  *p;
```

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

```
int *iptr;
float *fptr;
```

Bu kadar tanımla sonucunda bellekte p değişkeni mevcuttur. Ancak işaret ettiği veri bloğu yoktur. Bunun için iki yol vardır. Birincisi kullanılan herhangi bir değişkeni işaret etmek, ikincisi ise veri bloğunu boş belleği kullanarak oluşturmak.

1. İşaretçi değişkenin var olan bir değişkenin bulunduğu adresi göstermesi.

Bu işlemi yapabilmek için var olan değişkenin adresinin bilinmesi gerekmektedir.

& operatörü : Bir değişkenin adresinin belirlenmesi için kullanılır. Kullanım biçimi:

&değişken

&i : i değişkenin adresini verir.

```
int main()
{
  int i;
  int *iptr;
  i = 5;
  iptr = &i;
  printf("i değişkeninin adresi    %p\n", &i);
  printf("iptr değişkeninin değeri %p\n", iptr);
  return 0;
}
```

Çıktı:

i değişkeninin adresi 0028FF18

iptr değişkeninin değeri 0028FF18

2. Veri bloğunu boş belleği kullanarak oluşturmak.

Bu yolla veriler için dinamik yer ayırılır. Bunun için malloc işlevi kullanılır.(C++ new deyimini desteklemektedir.)

void *malloc(n) : Boş bellekten n byte yer ayırıp başlangıç adresini döndürür.

```
iptr = (*int) malloc(sizeof(int));
```

ya da

```
iptr = (*int) malloc(2);
```

```
iprt = new int; //C++ da kullanımı bu şekilde yapılabilir.
```

Veriye işaretçi değişken yoluyla erişim

Bir işaretçinin gösterdiği adresteki veriye erişmek için işaretçi değişkeninin önüne * karakteri eklenir.

```
int main()
{
  int i;
  int *iptr;
  iptr = &i;
  *iptr = 8;
  printf("i değişkeninin değeri %d\n", i);
  printf("iptr adresinin içeriği %d\n", *iptr);
return 0;
}
```

Ekranda çıktı :

i değişkeninin değeri 8

iptr adresinin içeriği 8

!!! İşaretçi değişkenin gösterdiği adresin içeriği değişken ilişkilendirilmeden kullanılmamalıdır.

1.2. İşaretçi 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.

```
int *iptr, i;
...
iptr = &i;              i değişkenin adresinin 1000 olduğunu varsayalım.
iptr nin değeri 1000 dir.

iptr++;                 iptr nin değeri 1002 olur. (  int değeri işaret ettiği için)
```

aynı örneği double için yaparsak

```
double *iptr, i;
...
iptr = &i;              i değişkenin adresinin 1000 olduğunu varsayalım.
iptr nin değeri 1000 dir.

iptr++;                 iptr nin değeri 1008 olur. (  double değeri işaret ettiği
için)

int *iptr, i, j;
...
iptr = &i;              i değişkenin adresinin 1000 olduğunu varsayalım.
iptr nin değeri 1000 dir.

*(iptr+4)=2;            1008 adresinin içeriğini 2 yapar.
```

!!! Arttırma işaret edilen veri bloğuna göre yapılır Yani bir sonraki veri bloğunun gösterilmesi sağlanır.

iptr++ ; bir sonraki veri bloğunu göster

(*iptr)++; iptr değişkeninin gösterdiği adresteki değeri 1 arttır

1.3. İşaretçiler ve Diziler

İşarteçiler üzerinde geçerli aritmetik işlemler yardımıyla, dizilere işaretçi değişkenler ile erişmek mümkündür.

```
#include <stdio.h>
int main()
{
  int i[10], j;
  int *iptr;
  for (j=0; j<10; j++)
       i[j]=j;
  /* Dizinin başlangıç adresine erişmek için ilk
elemanın adresi kullanılabilir  &i[0]  veya doğrudan
i */
  iptr = i;
  for (j=0; j<10; j++) {
      printf("%d ", *iptr);
      iptr++;
  }
  printf("\n");
/* iptr artık dizinin başını göstermez */
  iptr = i;
  for (j=0; j<10; j++)
      printf("%d ", *(iptr+j));
  printf("\n");
/* iptr hala dizinin başını gösterir */
  getch();
     return 0;
}
```

Örnek : İşaretçi ve dizgi kullanımı.

```
#include <stdio.h>
#include <conio.h>
int main()
{
  char *a="1234567890";
  char b[11];
  char *p1, *p2;
  printf("%s\n", a);
  p1 = a;
  p2 = b;
  while (*p1 != '\0') {
      *p2 = *p1;
      p1++;
      p2++;
  }
  printf("%s\n", b);
  getch();
  return 0;
}
```

1.4. İşlevleri Referans Yoluyla Çağırma

Şu ana yazdığımız işlevlerde gönderilen parametrelerin (diziler hariç) değerlerinin değiştirilmesi mümkün değil idi. İşlev çağırıldığı zaman parametrelerin bir kopyası çıkartılıp işleve gönderiliyordu. Bir işlevin birden fazla değer gönderebilmesi için işaretçilere gereksinimiz vardır.

```
void arttir(int);
main()
{
  int i;
  i = 5;
  printf("öncesi %d\n", i);
  arttir(i);
  printf("sonrası %d\n", i);
  getch();
}

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

Çıktı :

öncesi 5

sonrası 5

Gönderilen parametrenin kopyası işleve gönderildiği için işlev içerisinde yapılan değişiklikler işlevin çağırıldığı yeri etkilemez. Eğer parametredeki değişikliklerin işlevin çağırıldığı yerde de geçerli olmasını istiyorsak işleve parametrenin adresini göndermek gerekir.

```
void arttir(int*);
main()
{
  int i;
  i = 5;
  printf("öncesi %d\n", i);
  arttir(&i);
  printf("sonrası %d\n", i);
  getch();
}

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

Çıktı :

öncesi 5

sonrası 6

Örnek : İşleve gönderilen dizinin işlev içerisinde işaretçi olarak kullanımı.

```
#include <stdio.h>
#include <conio.h>
#define N 5
float ort (int *);
main()
{
  int s[N];
  int i, k;

  for (i=0; i<N; i++) {
     s[i] = rand() % 100;
     printf("%4d",s[i]);
  }
  printf("\n");
  getch();
}
float ort (int *a)
{
 int i;
 float t = 0;
 for (i=0; i<N; i++)
    t = t + *(a+i);
 return t/N;
}
```

1.5. Dinamik Bellek Kullanımı

void malloc(n): En az n byte uzunluğunda bellekten yer ayırır. İşlevin değeri

>0 ise bloğun bellekteki yeri, NULL yer yok demektir.

int *i;

i = (int *) malloc(2000) ; 2000 byte yer ayırıp bloğun başlangıç adresini i 'ye atar

(1000 elemanlı int dizisi)

double *x;

x = (double *) malloc(8*2000); 2000 elemanlı double dizi

sizeof(n) : n ifadesinin/tipinin byte olarak uzunluğunu verir.

i = (int *) malloc(1000*sizeof(int)) ; 1000 tane int değer içerecek bellek uzunluğu

x = (double *) malloc(2000*sizeof(double)); 2000 elemanlı double dizi

void free (void *block) : malloc işlevinin tersi. Block değişkenin tuttuğu yeri boş belleğe gönderir.

Örnek : Bir dizinin elemanlarının işaretçi olması.

```
main()
{
  char *aylar[] = {"", "Ocak", "Şubat", "Mart", "Nisan",
                   "Mayıs", "Haziran", "Temmuz", "Ağustos",
                   "Eylül", "Ekim", "Kasım", "Aralık"};
  int i;
  printf("Ayın sırasını gir "); scanf("%d", &i);
  if (i>0 && i<13)
     printf("%s\n", aylar[i]);
  getch();
}
```

Benzer şekilde

```
        float *a[100];
```

tanımlaması her bir elemanı bellekte bir 'float' sayıyı gösteren işaretçi olan 100 elemanlı bir dizidir.

Örnek : Bir işaretçinin adresini içeren işaretçiler.

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

  printf("  i  ve  &i  : %d  %p\n", i, &i);
  printf(" *iptr ve  iptr  : %d  %p\n", *iptr, iptr);
  printf("*iptrptr ve  iptrptr :  %p  %p\n", *iptrptr, iptrptr);
  getch();
}
```

Çıktı:

```
   i   ve   &i   : 5   0028FF18
 *iptr  ve   iptr   : 5   0028FF18
*iptrptr ve  iptrptr :  0028FF18   0028FF14
```

1.6. İşaretçiler ve Yapılar

Bir işaretçi işleve parametre olarak gönderildiğinde basit değişken gibi değişkenin kopyası alınıp gönderiliyordu. Yapının büyük olduğu durumlarda bu da sorun çıkartır. İşaretçinin bir yapı verisini göstermesi.

```
struct ogrenci{
  char no[10];
  int  notu;
};
struct ogrenci *a
```

Tanımlamasında a değişkenini oluşturan alanlara erişmek için, bilinen yol:

```
*a.notu=56;
strcpy((*a).no, "95001");
```

Bunun farklı kullanımı:

```
a->notu=56;
strcpy(a->no, "95001");
```

Örnek : Yapının adresinin işleve gönderilmesi.

```
#include <stdio.h>
#include <conio.h>
typedef struct {
   char adi[35];
   char adres1[40];
   char adres2[40];
   char tel[15];
   float borc;
} kisiler;
void yaz(kisiler *z);
int main()
{
  kisiler  a;

  printf("Adını gir  : "); gets(a.adi);
  printf("Adres-1    : "); gets(a.adres1);
  printf("Adres-2    : "); gets(a.adres2);
  printf("Telefonu   : "); gets(a.tel);
  printf("Borcu      : "); scanf("%f", &(a.borc));
  yaz(&a);
  getch();
  return 0;
}
void yaz(kisiler *z)
{

 printf("Adı        : "); puts(z->adi);
 printf("Adresi     : "); puts(z->adres1);
 printf("           : "); puts(z->adres2);
 printf("Telefonu   : "); puts(z->tel);
 printf("Borcu      : "); printf("%.0f\n", z->borc);
}
```

1.7. Dizilerde Dinamik Bellek Kullanımı

Üç boyutlu dizi tanımı ve kullanımı için aşağıdaki parametreleri
Üç boyut --> Gerilim - Akım - Zaman

Örnek : Tek boyutlu diziyi üç boyutlu gibi kullanma.

```
/* Üç boyutlu dinamik dizi kullanımı. Her boyut farklı uzunlukta
   Dizi tek boyutlu gözüküyor. Ancak indis, hesaplanarak bulunuyor
   Yani
      *(a + i*y*z + j*z +k)
            ^      ^
            ^      ^ 3. boyutun uzunluğu
            ^2. boyutun uzunluğu
   şeklinde kullanılabilir.
*/
#include <stdio.h>
#include <conio.h>
#define x    4
#define y    5
#define z    9
void matris_yaz(int *);
int main()
{
  int *a;
  int i, j, k;
  a=(int *) malloc(x * y * z * sizeof(int));// eleman sayisi kadar yer ac

  for (i=0; i<x; i++) {
     printf("i = %d  \n", i);
     for (j=0; j<y; j++) {
        for (k=0; k<z; k++) {
           *(a + i*y*z + j*z +k) = i*j*k;
           printf("%5d ",*(a + i*y*z + j*z +k));
        }
        printf("\n");
```

```
    }
  }
  matris_yaz(a);
  getch();
  return 0;
}
void matris_yaz(int *m)
{
  int i, j, k;
  printf("\n");
  getch();
  for (i=0; i<x; i++) {
     printf("i = %d  \n", i);
     for (j=0; j<y; j++) {
        for (k=0; k<z; k++) {
           printf("%5d ",*m);
           *m++;
        }
        printf("\n");
     }
     getch();
  }
}
```

Örnek :

```
/* Uc boyutlu dinamik dizi kullanimi. Her boyut farkli uzunlukta
   Bu yontem ile diziye normal dizi gibi erismek mumkun
   Yani
      a[i][j][k]
   seklinde kullanilabilir.
*/
#include <stdio.h>
#include <conio.h>

#define x    8
#define y    4
#define z   10
main()
{
  double ***a;
  int i,j,k;
  a=(double *) malloc(x*sizeof(double*));/* 1. boyut icin yer ayir
(işaretçiler)   */

  for (i=0; i<x; i++)   /* 2. boyut icin yer ayir.  (işaretçiler)  */
     *(a+i)=(double *) malloc(y*sizeof(double*)); /* 1. boyutun her
                                                        elemani n   */

                                                /* elemanli diziyi gosterir   */

 for (i=0; i<x; i++)   /* 3. boyut icin yer ayir (matrisin elemanlar)    */
     for (j=0; j<y; j++)
        *(*(a+i) + j) = (double *) malloc(z*sizeof(double));

  for (i=0; i<x; i++)
     for (j=0; j<y; j++)
        for (k=0; k<z; k++)
           *(*(*(a + i) + j) + k) = i*j*k;

for (i=0; i<x; i++) {
```

```
      printf("i = %d  \n", i);
      for (j=0; j<y; j++) {
         for (k=0; k<z; k++)
            printf("%4.1f ",a[i][j][k]);
         printf("\n");
      }
      getch();
  }
}
```

1.8. İşaretçilerle İlgili Diğer Konular

1.8.1. İşlev İşaretçileri

İşaretçinin bir işlevin bulunduğu adresi içermesi durumudur. Normal işaretçi gibi işlevin adresini içeren değişken tanımı yapılmalıdır. Örneğin;

```
int (*fnptr) (int, int)
```

fnptr değişkeni iki tane int parametresi olup bir int değer geri gönderen bir işlevin adresini içerebilir.

(int *fnptr (int, int) : iki int parametresi olup int işaretçi geri gönderir)

Örnek : Aynı isim ile farklı iki işlevi çağırma.

```
int kare(int);
int kub(int);
main()
{
  int (*islem)(int);            /* bir int değer alıp geriye int değer gönderen
bir işlevin adresi  */
  int i;
  char c;

  printf("1 / 2 : kare / küb hesabı : ");
  c = getch();
  printf("\nSayıyı gir : ");
  scanf("%d", &i);
  if (c == '1')
     islem = kare;      /* kare işlevinin adresi islem değişkenine kopyalanır
*/
  else
     islem = kub;
  printf("Sonuç = %d\n", islem(i));
  getch();
}
int kare(int s)
{
 return s*s;
}
int kub(int s)
{
 return s*s*s;
}
```

1.8.2. Void İşaretçiler

İşaretçiler void olarak tanımlanabilir. Bu biçimde tanımlanan işaretçilerin gösterdiği adresteki değere erişmek için veri tipi belirtilmelidir.

```
main()
{
  void *a;
  a = (char*) malloc(10);
  strcpy(a,"12345");
  printf("%s\n",  a);
  free(a);
  a = (double*) malloc(sizeof(double));
  *(double*)a = 3.123;              /* değere erişirken veri tipi belirt */
  printf("%f\n", *(double *)a);
  getch();
}
```