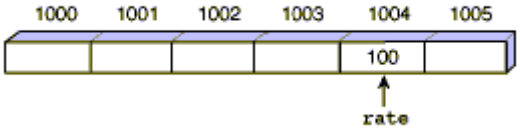
Pointer Kavramı (işaretçiler)

Pointer Nedir?

Bilgisayarın ana belleği (RAM) sıralı kaydetme bölgelerinden(gözlerinden) oluşmuştur. Her göze bir adres atanmıştır. Bu adreslerin değerleri 0 ila RAMe bağlı olarak MAX arasında değerler alabilir.

Programlama dillerinde bir değişken tanımlandığında, o değişkene tipine bağlı olarak RAM den bir bölge ayrılır. Örneğin `rate` adlı bir değişken tanımladığımızı varsayalım. Bu değişken bellekte Şekil 1 deki bir yere yazılır.



Şekil 1 : *rate adlı değişkenin bellekteki adresi*

Pointer(işaretçi), bellek alanındaki bir gözün adresinin saklandığı değişkendir. Pointerlara, veriler(yani değişkenlerin içeriği) değil de, o verilerin bellekte saklı olduğu bellek gözlerinin başlangıç adresleri atanır. Bir pointer, diğer değişkenler gibi, sayısal bir değişkendir. Bu sebeple kullanılmadan önde program içinde bildirilmelidir. Pointer değişkenler şöyle tanımlanır:

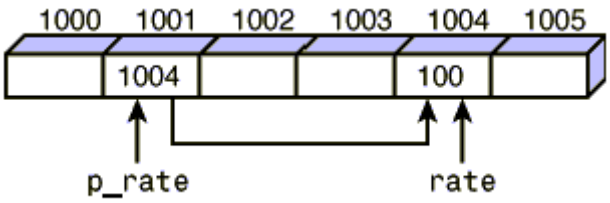
```
tipadı *ptradı;
```

Burada tipadı herhangi bir C tip adı olabilir. Değişkenin önüne konan `*` karakteri indirection operatörü olarak adlandırılır ve `tipadı` ile bildirilen `ptradı` değişkenini işaret eder. Örneğin:

```
char *kr1;          /* karakter için */
int *x;             /* tamsayı için */
float *deger,sonuc; /* deger pointer tipinde sonuc sıradan bir
reel değişkenler */
```

Yukarıda bildirilen pointer değişkenlerden; `kr` bir karakterin, `x` bir tamsayının ve `deger` bir gerçel sayının bellekte saklı olduğu yerlerin adreslerini tutar. Fakat `sonuc` adlı float değişken sıradan bir gerçel sayıdır.

Bir pointera, bir değişkenin adresini atamak için & adres-operatörü kullanılır. Bu operatör bir değişkenin önüne konursa, o değişkenin içeriği ile değilde adresi ile ilgileniliyor anlamına gelir. Örneğin karakter tipindeki `rate` adlı bir değişken ve `p_rate` ise pointer tipinde, `rate` değişkenini işaret eden bir değişken olsun. Bu durum Şekil 2 de gösterilmektedir:



Şekil 2 : *rate adlı sıradan bir değişken ve onu işaret eden p_rate adlı pionter değişkeninin bellekteki dizilimi*

Derleyicide şöyle ifade edilir:

```
char rate;
char *p_rate;
...
p_rate = &rate;
```

Yani `p_rate` pointeri rate değişkeninin saklandığı adresi işaret etmektedir. Program 1 pointer kavramını anlamak için oldukça önemli bir örnektir.

Program 1 : *ilk Pointer programı*

```
/*ptr01.c*/
1: main()
2: {
3:      int x, *isaret;
4:
5:      x = 888;
6:      işaret = &x;
7:
8:      printf("x in degeri = %d\n",x);
9:      printf("x in adresi = %x\n",isaret);
10:     printf("x in değeri = %d\n",*isaret);
11:     printf("x in adresi = %x\n",&x);
12: }
x in degeri = 888
x in adresi = ffde
x in degeri = 888
x in adresi = ffde
```

Özetle, eğer `ptr` adında bir pointera, `deg` adlı bir değişkene sahipseniz ve `ptr = & deg;` şeklinde bir atama yapmışsanız:

- `*ptr` ve `deg`, deg adlı değişkenin içeriği ile ilgilidir.
- `ptr` ve `& deg`, deg adlı değişkenin adresi ile ilgilidir.
- * indirection opreratörü olarak adlandırılır.
- & adres operatörüdür.

Dinamik Diziler

Bir dizi daha önce anlatıldığı gibi, programın başında kaç elemanlı olduğu belirtilerek bildirilirse, derleyici o dizi için gerekli bellek alanını program sonlandırılıncaya kadar saklı tutar ve o alan başka amaçlar için kullanılmaz. Dizilerde dinamik çalışma, programın çaklıştırılması sırasında gerekli bellek alanının işletim sisteminden istenmesi ve işi bittiğinde bu alanın geri verilmesidir. Bu amaçla standart kütüphanede malloc(), calloc(), realloc() ve free() fonksiyonları vardır. Bu şekilde dizileri kullanmak için:

```
          ...
          tip *ptr;
          ...
          ptr = malloc(50); /* herbiri 8 bit olan 50 elemanlık yer
isteniyor */
          .
          .                 /* kullanılıyor */
          .
          free(ptr);        /* geri veiliyor */
          ...
```

Program 2 ile basit olarak dinamik bir dizinin kullanımı gösterilmiştir.

Program 2 : *Dinamik bir dizinin kullanımı*

```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:
4:  main(){
5:          int *x,i,N,toplam=0;
6:          float ortalama;
7:
8:         printf("Eleman sayısı ");
9:         scanf("%d",&N);
10:
11:        x = (int *) malloc(N*sizeof(int)); /* N tane int gözü
12:                                              isteniyor (Nx2 byte) */
13:
14:         puts("Elemanları girin:");
15:         for(i=0;i<N;i++){
16:           printf("%d.eleman = ",i+1);
17:           scanf("%d",&x[i]);  /* x[i] ile *(x+i) aynı anlamda !... */
18:           toplam += x[i];
19:         }
20:
21:         free(x);  /* ayrılan yer boşaltılıyor... */
22:
23:         ortalama = (float) toplam/N;
24:
25:         printf("Bu %d sayının ortlaması %f dir\n",N,ortalama);
26:
27: }
```

5. satırda x değişkeni pointer tipinde tanımlanmıştır. N sayısı 9. satırda klavyeden okutulmuştur. 11. satırda x pointeri için malloc() fonksiyonu ile bellekten N tane `int` tipinde bellek gözü istenmiştir. malloc() fonksiyonu bellekten istenen yerin başlangıç adresini x değişkenine aktarır. N tane yer istendiğine göre, x, artık N elemanlı bir dizidir. Buna göre x[0]

ayrılan bölgenin ilk adresindeki değişkeni temsil eder. Benzer olarak diğer elemanlar da, bellekte peşpeşe sıralanır. Bu yüzden, x 17. satırda bir dizi gibi okutulmuştur. Eğer x bir dizi gibi ifade edilmek istenmiyorsa i. eleman `x[i]`, `*(x+i)` şeklinde de kullanılabilir. x in elemanların toplamı toplam adlı değişkene 18. satırda aktarılmaktadır. 21. satırda x için ayrılan bellek alanı free fonksiyonu ile serbest bırakılmıştır.

Program 2 sıradan bellek kullanımından oldukça farklıdır. Eğer x bir pointer değil de, sıradan bir dizi gibi bildirilseydi, programın başında x in eleman sayısı mutlaka belirtilmesi gerekirdi. Bu bize, pointer kullanımın ne kadar önemli olduğunu gösterir.

Pointer Tipindeki Fonksiyonlar

Bir fonksiyon tanımlanırken, parametreleri pointer olabileceği gibi, tipi de pointer olabilir. Bu, o fonksiyonun *kendisini çağırana bir adres göndereceği* anlamına gelir. Bu tip uygulamalar özellikle string uygulamalarında sık kullanılır.

Program 3 de, geri dönüş değeri int tipinde olan bir pointer fonksiyon örneği verilmiştir. Bu fonksiyon iki sayının toplamının saklandığı toplam adlı değişkenin adresini, çağırılan yere gönderir.

Program 3 : *Geri dönüş değeri, pointer olan bir fonksiyon*

```
1:  #include <stdio.h>
2:
3:  int *topla(int,int);
4:
5:  main(){
6:    int a,b;
7:    int *p;
8:
9:    a = 2;
10:   b = 4;
11:
12:   p = topla(a,b);
13:
14:  printf("Toplam : %d (adresi %p)\n",*p,p);
15: }
16:
17: int *topla(int x,int y)
18: {
19:   int *ptr,toplam;
20:
21:   toplam = x+y;
22:   ptr = &toplam;
23:
24:   return ptr;
25: }
Toplamları 6 (adresi 0xbbb7fc4)
```