BLM 112- Programlama Dilleri II

Hafta 4
İşaretçiler (Pointers)

Hafıza Yapısı

- Bir değişken tanımlandığında arka planda bilgisayarın hafızasında bir konuma yerleştirilir.
- Hafıza küçük hücrelerden oluşmuş bir blok olarak düşünülebilir.
- Bir değişken tanımlandığında bellek bloğundan gerekli miktarda hücre ilgili değişkene aktarılır.
- Hafızada değişken için ne kadar hücre ayrılacağı değişkenin tipine göre değişir.

Hafıza Yapısı


```
#include <stdio.h>
#include <limits.h>
#include <float.h>

int main(void)
{
  printf("char -> byte: %d bit: %d min: %d max: %d\n", sizeof(char), sizeof(char)*8, CHAR_MIN, CHAR_MAX);
  printf("unsigned char -> byte: %d bit: %d min: %d max: %d\n", sizeof(unsigned char), sizeof(unsigned char)*8, (unsigned char) 0,
UCHAR_MAX);
  printf("signed char -> byte: %d bit: %d min: %d max: %d\n\n", sizeof(signed char), sizeof(signed char)*8, SCHAR_MIN, SCHAR_MAX);

  printf("int -> byte: %d bit: %d min: %d max: %d\n", sizeof(int), sizeof(int)*8, INT_MIN, INT_MAX);
  printf("unsigned int -> byte: %d bit: %d min: %u max: %u\n", sizeof(unsigned int), sizeof(unsigned int)*8, (unsigned int) 0, UINT_MAX);
  printf("short int -> byte: %d bit: %d min: %hd max: %hd\n", sizeof(short int), sizeof(short int)*8, SHRT_MIN, SHRT_MAX);
  printf("unsigned short int -> byte: %d bit: %d min: %hu max: %hu\n\n", sizeof(unsigned short int), sizeof(unsigned short int)*8, 0,
USHRT_MAX);

  printf("long int -> byte: %d bit: %d min: %ld max: %ld\n", sizeof(long int), sizeof(long int)*8, LONG_MIN, LONG_MAX);
  printf("long long int -> byte: %d bit: %d min: %lld max: %lld\n", sizeof(long long int), sizeof(long long int)*8, LLONG_MIN, LLONG_MAX);
  printf("unsigned long int -> byte: %d bit: %d min: %lu max: %lu\n", sizeof(unsigned long int), sizeof(unsigned long int)*8, (unsigned long int)
0, ULONG_MAX);
  printf("unsigned long long int -> byte: %d bit: %d min: %llu max: %llu\n\n", sizeof(unsigned long long int), sizeof(unsigned long long int)*8,
(unsigned long long int) 0, ULLONG_MAX);

  printf("float -> byte: %d bit: %d min: %g max: %g\n", sizeof(float), sizeof(float)*8, -FLT_MAX, FLT_MAX);
  printf("double -> byte: %d bit: %d min: %lg max: %lg\n", sizeof(double), sizeof(double)*8, -DBL_MAX, DBL_MAX);
  printf("long double -> byte: %d bit: %d min: %Lg max: %Lg\n", sizeof(long double), sizeof(long double)*8, -LDBL_MAX, LDBL_MAX);

  return 0;
}
```

Hafıza Yapısı


```
char -> byte: 1 bit: 8 min: -128 max: 127
unsigned char -> byte: 1 bit: 8 min: 0 max: 255
signed char -> byte: 1 bit: 8 min: -128 max: 127

int -> byte: 4 bit: 32 min: -2147483648 max: 2147483647
unsigned int -> byte: 4 bit: 32 min: 0 max: 4294967295
short int -> byte: 2 bit: 16 min: -32768 max: 32767
unsigned short int -> byte: 2 bit: 16 min: 0 max: 65535

long int -> byte: 4 bit: 32 min: -2147483648 max: 2147483647
long long int -> byte: 8 bit: 64 min: -9223372036854775808 max: 9223372036854775807
unsigned long int -> byte: 4 bit: 32 min: 0 max: 4294967295
unsigned long long int -> byte: 8 bit: 64 min: 0 max: 18446744073709551615

float -> byte: 4 bit: 32 min: -3.40282e+038 max: 3.40282e+038
double -> byte: 8 bit: 64 min: -1.79769e+308 max: 1.79769e+308
long double -> byte: 12 bit: 96 min: 3.4E-4932 max: 1.1E+4932
```

Hafıza Yapısı

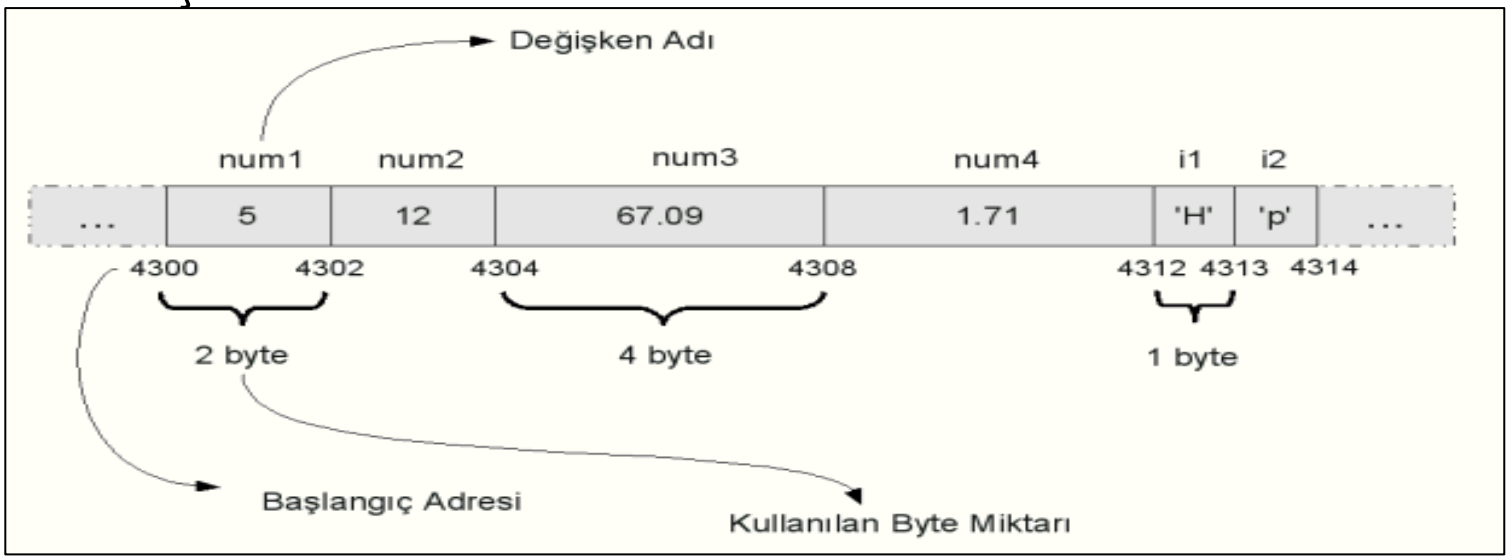

```c
#include <stdio.h>

int main(void)
{
    // Degiskenler tanımlanıyor:
    int num1, num2;
    float num3, num4;
    char i1, i2;
    // Degiskenlere atama yapiliyor:
    num1 = 5;
    num2 = 12;
    num3 = 67.09;
    num4 = 1.71;
    i1 = 'H';
    i2 = 'p';
    return 0;
}
```

Hafıza Yapısı

- Hücrelerden oluşan bellek yapısını verilen kod parçası için uygularsak.
  - int veri tipi 2 bayt, float tipinin 4 bayt, char tipinin de 1 bayt yer kapladığını varsayalım.
  - Her bir hücre 1 bayt alanı temsil etsin.
  - Değişkenler için ayrılan hafıza alanı 4300 adresinden başlasın.

```
// Değişkenlere
num1 = 5;
num2 = 12;
num3 = 67.09;
num4 = 1.71;
i1 = 'H';
i2 = 'p';
```



Hafıza Yapısı

- Bir değişken tanımlandığında hafızada onun için gereken alan rezerve edilir.
- Örn. int num1 tanımlaması, bellekte uygun bir yerde 2 bayt alanın num1 değişkeni için ayrılmasını sağlar.
- Daha sonra num1 değişkenine 5 değeri atandığında ayrılan hafıza alanına 5 değeri kaydediliyor.
- Aslında, num1 ile ilgili yapacağınız bütün işlemler, 4300 adresiyle 4302 adresi arasındaki bellek hücrelerinin değişmesiyle alakalıdır.
- Değişken dediğimiz; uygun bir bellek alanının, bir isme rezerv edilip, kullanılmasından ibarettir.

Pointer Tanımlama

- Bir veri bloğunun bellekte bulunduğu adresi içeren (gösteren) veri tipidir.

```
veri_tipi  *p;
```

- p değişkeni <veri_tipi> ile belirtilen tipte bir verinin bellekte saklandığı adresi içerir.

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

- Dikkat edilmesi gereken tek nokta; pointer'ı işaret edeceği değişken tipine uygun tanımlamaktır.
- Yani float bir değişkeni, int bir pointer ile işaretlemeye çalışmak yanlıştır!

Pointer Tanımlama

- Bir pointer'ın var olan bir değişkenin bulunduğu adresi göstermesi için değişkenin adresinin pointer'a atanması gerekir.
- Bunun için değişkenin hafızada tutulduğu adresin bilinmesi gerekir.
- Bu da adres operatörü (&) ile mümkündür.
  - &y → y değişkeninin adresini verir.

```
int y = 5;
int *yPtr;
yPtr = &y;
```

Pointer Tanımlama

- Pointer bir değişkenin adresinin gösterir şekilde atandıktan sonra o pointer, ilgili değişkeni işaret eder.

- Eğer bahsettiğimiz değişkenin sahip olduğu değeri pointer ile göstermek veya değişken değerini değiştirmek isterseniz, pointer başına '*' getirerek işlemlerinizi yapabilirsiniz.

- Pointer başına '*' getirerek yapacağınız her atama işlemi, değişkeni de etkileyecektir.

Pointer Tanımlama


```c
#include <stdio.h>

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

   printf("i adresi    %p\n", &i);
   printf("iptr degeri %p\n", iptr);

   printf("i degeri    %d\n", i);
   printf("*iptr degeri %d\n", *iptr);

   getchar();
   return 0;
}
```

Değişkene Pointer ile Erişme

- Pointer kullanarak, değişkenlerin sakladığı değerleri de değiştirebiliriz.
- Bir işaretçinin gösterdiği adresteki veriye erişmek için işaretçi değişkeninin önüne * karakteri eklenir.

```
1  #include<stdio.h>
2  int main()
3  {
4      int i;
5      int *iptr;
6      iptr = &i;
7      *iptr = 8;
8      printf("i değişkeninin değeri %d\n", i);
9      printf("iptr adresinin içeriği %d\n", *iptr);
10
11     getchar();
12     return 0;
13 }
```

Değişkenleri Pointer ile İlişkilendirme


```c
#include<stdio.h>
int main( void )
{
    // int tipinde değişken tanımlıyoruz:
    int xyz = 10, k;
    // int tipinde pointer tanımlıyoruz:
    int *p;

    // xyz değişkeninin adresini pointer'a atıyoruz.
    // Bir değişken adresini '&' işaretiyle alırız.
    p = &xyz;

    // k değişkenine xyz'nin değeri atanır. Pointer'lar değer tutmaz.
    // değer tutan değişkenleri işaret eder.
    //Başına '*' koyulduğunda, işaret ettiği değişkenin değerini gösterir.
    k = *p;

    return 0;
}
```

Değişkene Pointer ile Erişme

- Bir pointer'in işaret ettiği değişkeni program boyunca sürekli değiştirebilirsiniz.

```c
#include<stdio.h>
int main( void )
{
    int x, y, z;
    int *int_addr;
    x = 41;
    y = 12;
    // int_addr x degiskenini isaret ediyor.
    int_addr = &x;
    // int_addr'in isaret ettigi degiskenin sakladigi deger aliniyor. (yani x'in degeri)
    z = *int_addr;
    printf( "z: %d\n", z );
    // int_addr, artik y degiskenini isaret ediyor.
    int_addr = &y;
    // int_addr'in isaret ettigi degiskenin sakladigi deger aliniyor. (yani y'nin degeri)
    z = *int_addr;
    printf( "z: %d\n" ,z );

    return 0;
}
```

Pointer Tanımlama

- Bir pointer'ın boş bir veri bloğunu göstermesi için malloc fonksiyonu kullanılır.

- Bu yolla veriler için dinamik yer ayrılır.
  - **malloc(n)** → Boş bellekten n bayt yer ayırıp başlangıç adresini döndürür.
  - **iptr = (int*) malloc(sizeof(int));**
  - ya da **iptr = (int*) malloc(4);**

Pointer Boyutu

- Pointer'lar genelde sabit boyutta yer kaplar. Örneğin 32 bit bir sistemde genellikle pointer'lar 32 bit olur.

```c
#include<stdio.h>
int main()
{
    double i;
    double *iptr;

    iptr = &i;
    printf("i boyutu: %d\n", sizeof(i));
    printf("iptr boyutu: %d", sizeof(iptr));

    getchar();
    return 0;
}
```

Pointer Tutan Pointer’lar

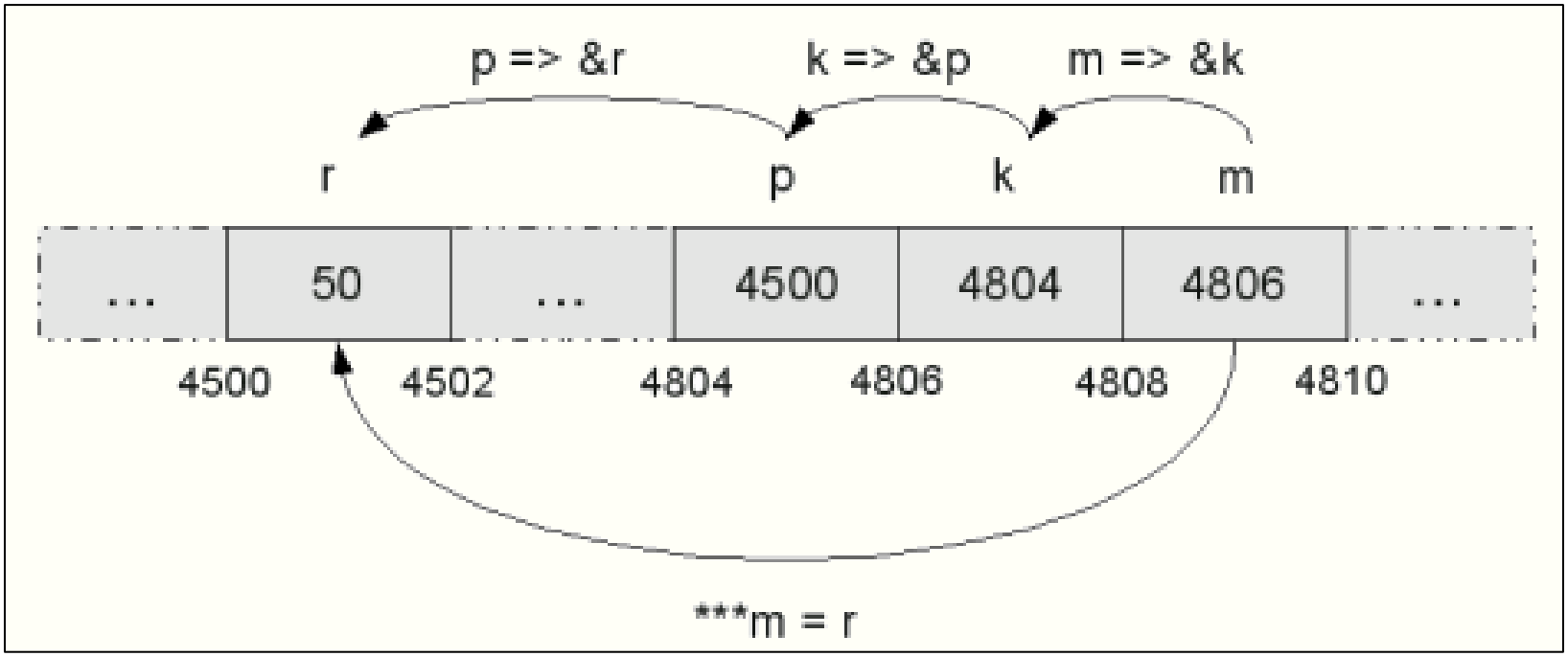
- Pointer'lar, gördüğümüz gibi değişkenleri işaret ederler.

- Pointer'da bir değişkendir ve onu da işaret edecek bir pointer yapısı kullanılabilir.

- Pointer değişkenini işaret edecek bir değişken tanımlıyorsanız; başına '**' getirmeniz gerekir.

- Buradaki ‘*’ sayısı değişebilir. Eğer, pointer işaret eden bir pointer'i işaret edecek bir pointer tanımlamak istiyorsanız, üç defa yıldız (***) yazmanız gerekir.

Pointer Tutan Pointer'lar



Pointer 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 i , *iPtr;
iPtr = &i; // iPtr örneğin 1000 nolu adresi gösteriyorsa
iPtr += 2;  // Bu işlemden sonra iPtr'ın yeni değeri 1008 (iPtr+2*4)
```

- Çünkü int tipi hafızada 4 bayt yer kaplıyor.

Pointer Aritmetiği


```c
#include<stdio.h>
int main( void )
{
    int i, *iPtr;
    double y, *yPtr;

    iPtr = &i;
    printf("iPtr gosterdigi adres: %d \n", iPtr);
    iPtr ++;  //int tipi için bir sonraki adres bloğu 4 bayt fazlası.
    printf("iPtr gosterdigi adres: %d \n\n", iPtr);

    yPtr = &y;
    printf("yPtr gosterdigi adres: %d \n", yPtr);
    yPtr ++;  //double tipi için bir sonraki adres bloğu 8 bayt fazlası.
    printf("yPtr gosterdigi adres: %d ", yPtr);

    getchar();
    return 0;
}
```

Pointer Aritmetiği

- **int i , *iPtr;**
- **iPtr = &i;** // iPtr örneğin 1000 nolu adresi gösteriyorsa
- **(*iPtr) ++;** // Bu işlem 1000 nolu adresin içeriğini 1 artırır.
- **iPtr ++;** // Bu işlem iPtr nin 1004 nolu adresi göstermesini sağlar
- **(*iPtr) +=2;** // Bu işlem 1000 nolu adresin içeriğini 2 artırır.
- **(*iPtr) =7;** // Bu işlem 1000 nolu adresin içeriğini 7 yapar.
- ***(iPtr+2) = 5;** //iPtr 1000 nolu adresi gösteriyorsa 1008 nolu adresin içeriğini 5 yapar.

Diziler ile Pointer Arası İlişki

- iPtr örneğin 1000 nolu adresi gösteriyorsa
- Bir dizi adı sabit bir pointer gibi düşünülebilir.
- Diziler ile pointer'lar yakından ilişkilidir.
- Pointer'lar değişkenleri gösterdikleri gibi, dizileri de gösterebilirler.

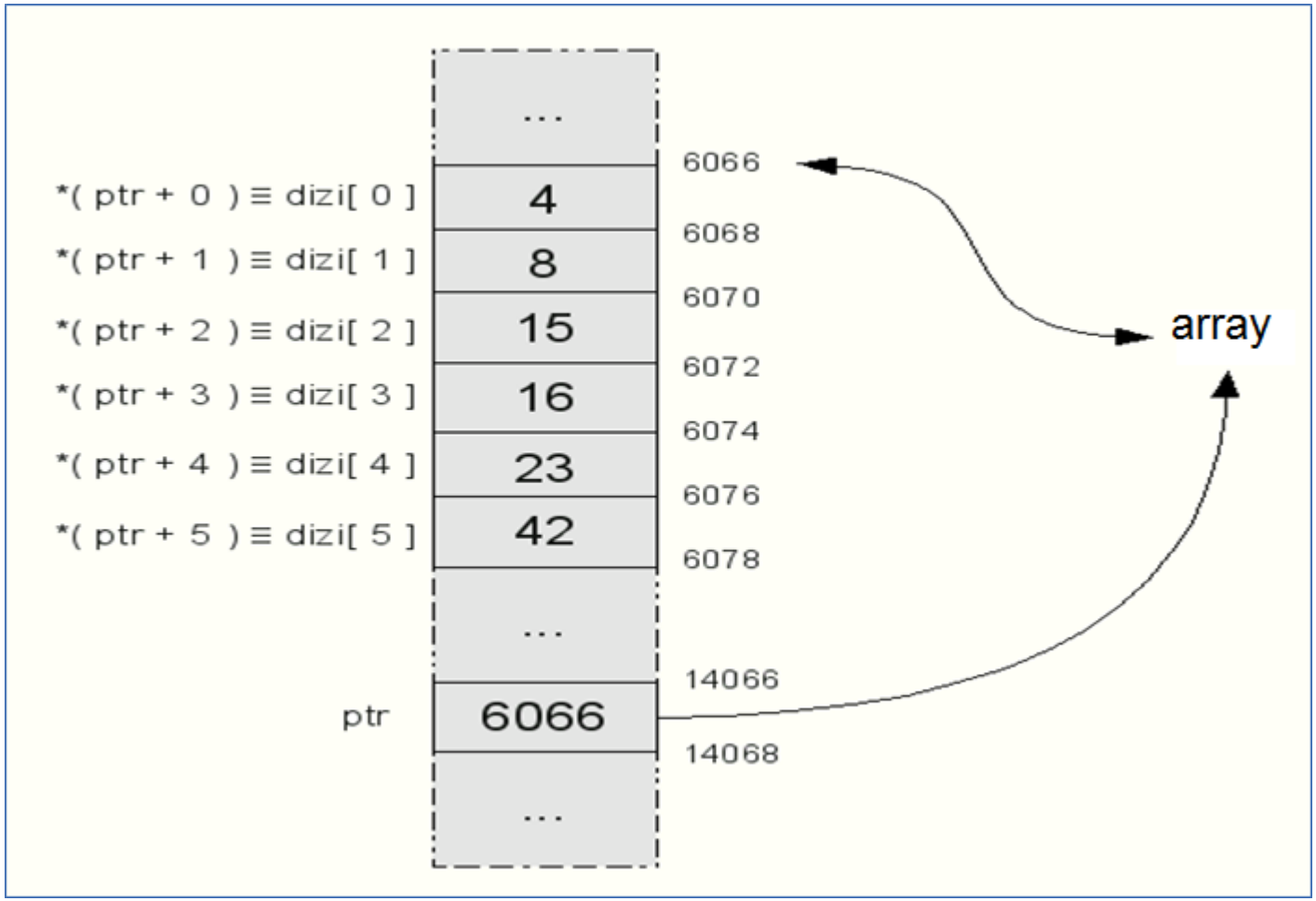
```
int dizi [6];
int *ptr;
```

- Dizi ve pointer'ı eşitlemek için dizinin adı kullanılabilir. Çünkü dizi adı aslında o dizinin ilk elemanının adresidir.

```
ptr = dizi; //Artık ptr[0] ve dizi[0] eşittir.
```

- Aynı işlemi ptr= &dizi [0] şeklinde de yapabiliriz.

Diziler ile Pointer Arası İlişki



Diziler ile Pointer Arası İlişki

- Dizi gösteren pointer'lar ile dizinin elemanlarına ulaşmak için:
  - ***(ptr + n)** → n sayısı dizinin n. indisini gösterir.
  - ***(ptr + 4)** → dizinin 4. indisindeki eleman yani dizi[4]
- Diğer alternatif gösterimler
  - **ptr[4]**
  - ***(dizi + 4)**

Diziler ile Pointer Arası İlişki


```c
#include<stdio.h>
int main( void )
{
    int elm;
    int month[ 12 ];
    int *ptr;
    ptr = month; // month[0] 'ın adresini ptr'ye ata
    elm = ptr[ 3 ]; // elm = month[ 3 ]
    ptr = month + 3; // ptr, month[ 3 ] adresini göstersin
    ptr = &month[ 3 ]; // ptr, month[ 3 ] adresini göstersin
    elm = ( ptr+2 )[ 2 ]; // elm = ptr[ 4 ] ( = month[ 7 ] ).
    elm = *( month + 3 );
    ptr = month; // month[0] 'ın adresini ptr'ye ata
    elm = *( ptr + 2 ); // elm = month[ 2 ]
    elm = *( month + 1 ); // elm = month[ 1 ]

    return 0;
}
```

Diziler ile Pointer Arası İlişki


```c
#include <stdio.h>
int main()
{
    int i[10], j;
    int *iptr;
    for (j=0; j<10; j++)
        i[j]=j;

    iptr = i;
    for (j=0; j<10; j++) {
        printf("%d ", *iptr);
        iptr++;
    }
    /* iptr artık dizinin başını göstermez */
    printf("\n%d \n",*(iptr-1));
    iptr = i;
    for (j=0; j<10; j++)
        printf("%d ", *(iptr+j));
    /* iptr hala dizinin başını gösterir */
    printf("\n%d",*iptr);
    getchar();
    return 0;
}
```

Diziler ile Pointer Arası İlişki


```c
#include <stdio.h>
int main()
{
      char *a="1234567890";
      char x[10];
      char *p1, *p2;
      printf("%s\n", a);
      p1 = a;
      p2 = x;
      while (*p1 != '\0') {
            *p2 = *p1;
             p1++;
             p2++;
      }
      *p2 = *p1;
      printf("%s\n", x);
      getchar();
      return 0;
}
```

Diziler ile Pointer Arası İlişki

- Diziler pointer içerebilirler.
- Pointer tutan diziler sayesinde birden fazla diziye erişim yapılabilir.
- Pointer tutan diziye dizilerin başlangıç adreslerini atamak yeterlidir.
- Pointer tutan dizi üzerinde yapılan değişiklik orijinal diziyi etkiler.

Diziler ile Pointer Arası İlişki


```c
#include <stdio.h>
int main()
{
    int i,j;
    char * ilkBaharAylar[3] ={"Mart","Nisan","Mayis"};
    char * yazAylar[3] ={"Haziran","Temmuz","Agustos"};
    char * sonBaharAylar[3] ={"Eylul","Ekim","Kasim"};
    char * kisAylar[3] ={"Aralik","Ocak","Subat"};

    char ** table[4];//char pointer(string) tutan dizileri tutan dizi
    table[0] = ilkBaharAylar;
    table[1] = yazAylar;
    table[2] = sonBaharAylar;
    table[3] = kisAylar;

    for(i=0;i<4;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%s\n",table[i][j]);
        }
    }

    getchar();
    return 0;
}
```

ÇALIŞMA SORULARI

1. Karakterden oluşan bir metin aşağıdaki matrisle şifrelenebilir.

A	D	G
B	E	H
C	F	I

Örneğin "ABCAH" metni bu matrise göre 11 21 31 11 23 şeklide şifrelenecektir. (harfin matriste bulunduğu satır indisi *10 + harfin matriste bulunduğu sütun indisi)
Kullanıcıdan şifreleme matrisini ve şifrelenmiş bilgiyi alıp orijinal haline dönüştüren algoritmanın kodunu yazınız.

2. M uzunluğundaki bir sayı dizisinde en az X kere tekrar eden sayıları ve en fazla tekrar eden sayıyı bulup ekrana yazdıran algoritmanın kodunu yazınız. X, M ve sayı dizisi kullanıcı tarafından girilecektir.

3. 10 tabanında verilen bir sayıyı kullanıcının verdiği tabana dönüştüren ve ekrana yazdıran algoritmanın kodunu yazınız. Yeni tabandaki sayı bir dizide saklanmalıdır.

4. Kullanıcının verdiği bir sayı dizisinin varyansını bulan algoritmayı çiziniz.
Varyans, tüm değerlerin ortalama değerden farklarının karelerinin toplamının, değer sayısına bölümüyle $\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2$. bulunur.

5. Bir robotun aldığı görüntü 0 ve 1 lerden oluşan N*N boyutlu bir matris olarak alınmaktadır. Bu görüntüde aşağıdaki şekle (matrise) en çok benzeyen şeklin yerini bulunuz. Benzerlik değerleri aynı olan hücre sayısıyla bulunacaktır.

0	1	0
0	1	0
1	1	1

Gelecek Hafta

- İşaretçiler
- Değer Yoluyla Çağırma
- Referans Yoluyla Çağırma
- Dinamik Bellek Yönetimi

Kaynaklar

- Doç. Dr. Fahri Vatansever, “Algoritma Geliştirme ve Programlamaya Giriş”, Seçkin Yayıncılık, 12. Baskı, 2015.
- Kaan Aslan, “A’dan Z’ye C Klavuzu 8. Basım”, Pusula Yayıncılık, 2002.
- Paul J. Deitel, “C How to Program”, Harvey Deitel.
- “A book on C”, All Kelley, İra Pohl

Dinlediğiniz için teşekkürler

CANER ÖZCAN
canerozcan@karabuk.edu.tr