CME 112- Programming Languages II

Week 4
Pointers

Assist. Prof. Dr. Caner Özcan

Kindness is the golden chain by which society is bound together. ~Goethe

Memory Structure

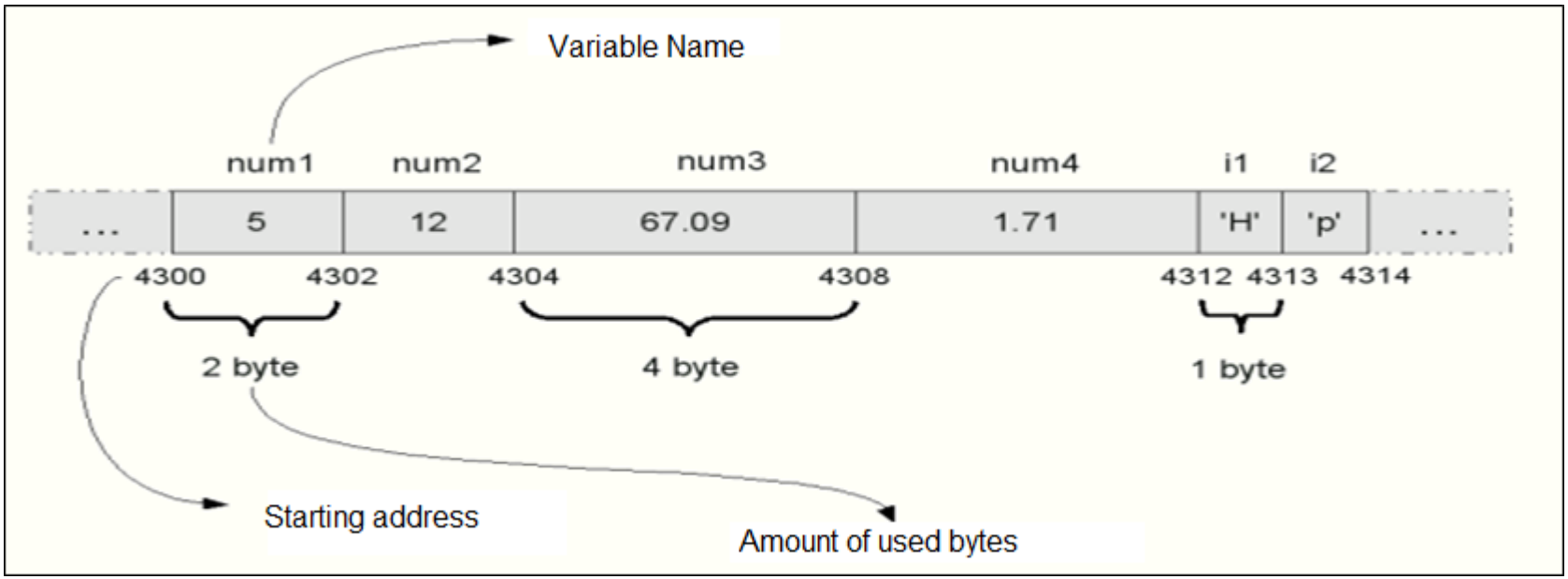
- When a variable defined it is stored somewhere in memory.
- Memory can be thought as block consist of cells.
- When a variable defined, required number of cell from memory is allocated for the variable.
- How many cell will be reserved for the variable depends on the type of variable.

Memory Structure


```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;
}
```

Memory Structure

- If we illustrate the structure of memory after the code in previous slide.
  - Assume that size of int is 2 byte, size of float is 4 byte and size of char is byte.
  - Each cell represents 1 byte space.
  - Memory portion for defined variables starts from the address 4300.



Memory Structure

- When a variable is defined, a space required for the variable is reserved in the memory.
- E.g. definition int num1 reserves 2 byte space for variable num1.
- After that if the value 5 is assigned on variable num1, 5 is stored in memory location allocated for that variable.
- Actually, all operations taken on variable num1 is the modification of cells in the memory location between 4300 and 4302.
- Variable is actually a memory location reserved for a particular label.

Defining Pointer

- Pointer is a data type that shows the memory address of a data block.

```
data_type  *p;
```

- Variable p stores the address of a variable which is in **<data_type> type**

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

- The only thing that we should pay attention is defining pointer suitable for the data type it points.
- A float variable must only be pointed by a float type pointer.

Defining Pointer

- To make a pointer show the address of a variable, address of the variable should be assigned to the pointer.
- For this purpose we should know the address of the memory location used for the variable.
- It is possible with address operator (&).
  - &y → gives the address of variable y.

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

Defining Pointer

- After assigning the address of a variable to a pointer, pointer starts to show the address of related variable.

- If we want to access or modify the value of a variable with pointer, we should use * character in the beginning of pointer name.

- All modifications done with * character in the beginning of pointer name effects the original variable.

Defining Pointer


```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;
}
```

Accessing Variables by Pointers

- Using pointers, we can change the values of stored variables.
- For accessing the value of a variable with pointer, we should use * character in the beginning of pointer name.

```c
#include<stdio.h>
int main()
{
    int i;
    int *iptr;
    iptr = &i;
    *iptr = 8;
    printf("i değişkeninin değeri %d\n", i);
    printf("iptr adresinin içeriği %d\n", *iptr);

    getchar();
    return 0;
}
```

Associating Variables with Pointers


```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;
}
```

Defining Pointer

- You can change the variable that pointer shows constantly throughout the program.

```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;
}
```

Defining Pointer

- Malloc function is used to show a pointer to an empty block of data.

- Thus, space for data is allocated dynamically.
  - **malloc(n)** → Takes the n byte place from empty memory and returns the starting address.
  - **iptr = (int*) malloc(sizeof(int));**
  - else **iptr = (int*) malloc(4);**

Size of Pointer

- Pointers generally have a fixed size, for example on a 32-bit system they're usually 32-bit.

```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;
}
```

Pointers that point other Pointers

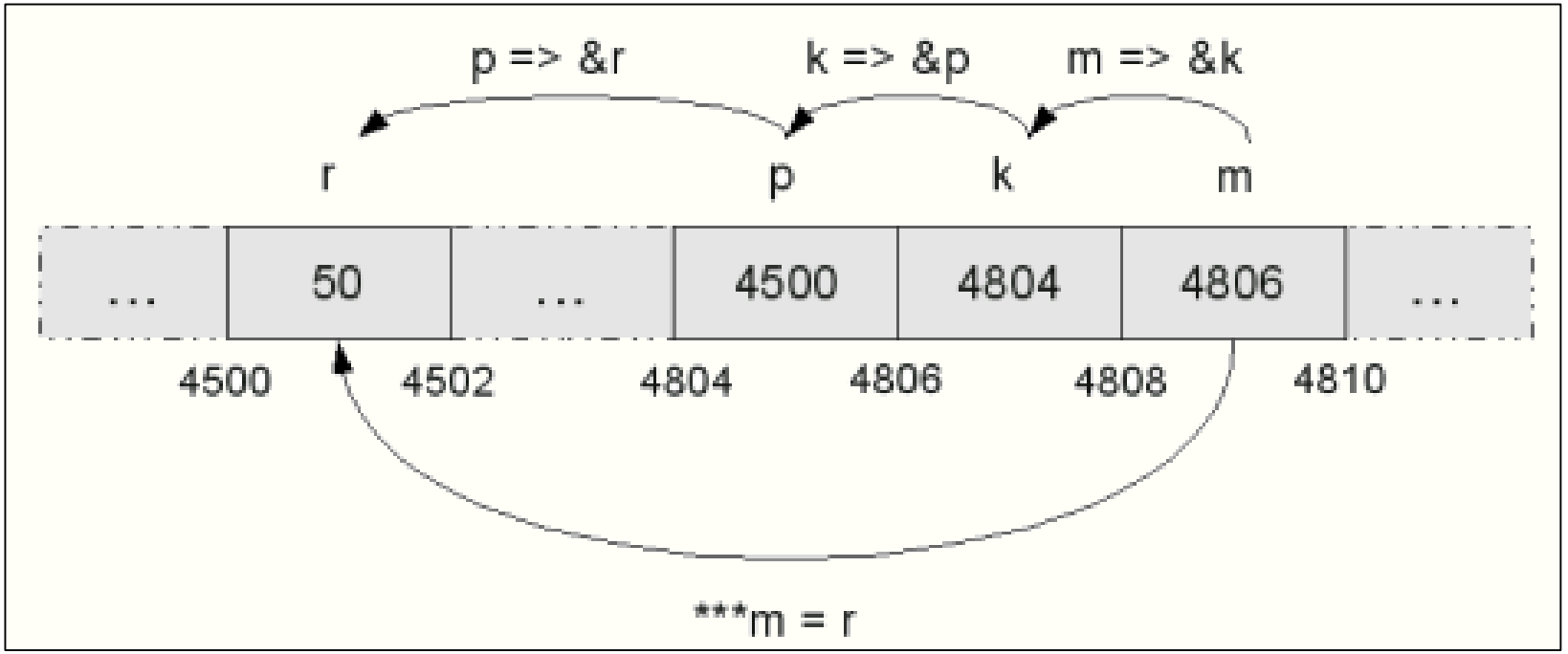
- As seen that pointers store the memory addresses of variables.

- Pointer is also a variable and an other pointer that shows a pointer can be defined.

- If we define a pointer variable that shows a pointer; we use '**' in the beginning of pointer name.

- Number of * can change. If we define a pointer that points an other pointer that points an other pointer we have to use '***'.

Pointers that point other Pointers



Pointer Arithmetic

- We can use increment (++), decrement (--), addition (+) or subtraction (-) operators with pointers. But this value must be integer.
- When we increment the pointer by 1, pointer shows the next data block.
- New pointer value depends on the data type that pointer shows.

```
int i , *iPtr;
iPtr = &i; // Assume iPtr shows address 1000
iPtr += 2  // After this operation new value of  iPtr is 1008 (iPtr+2*4)
```

- Because int type occupies 4 bytes of memory space.

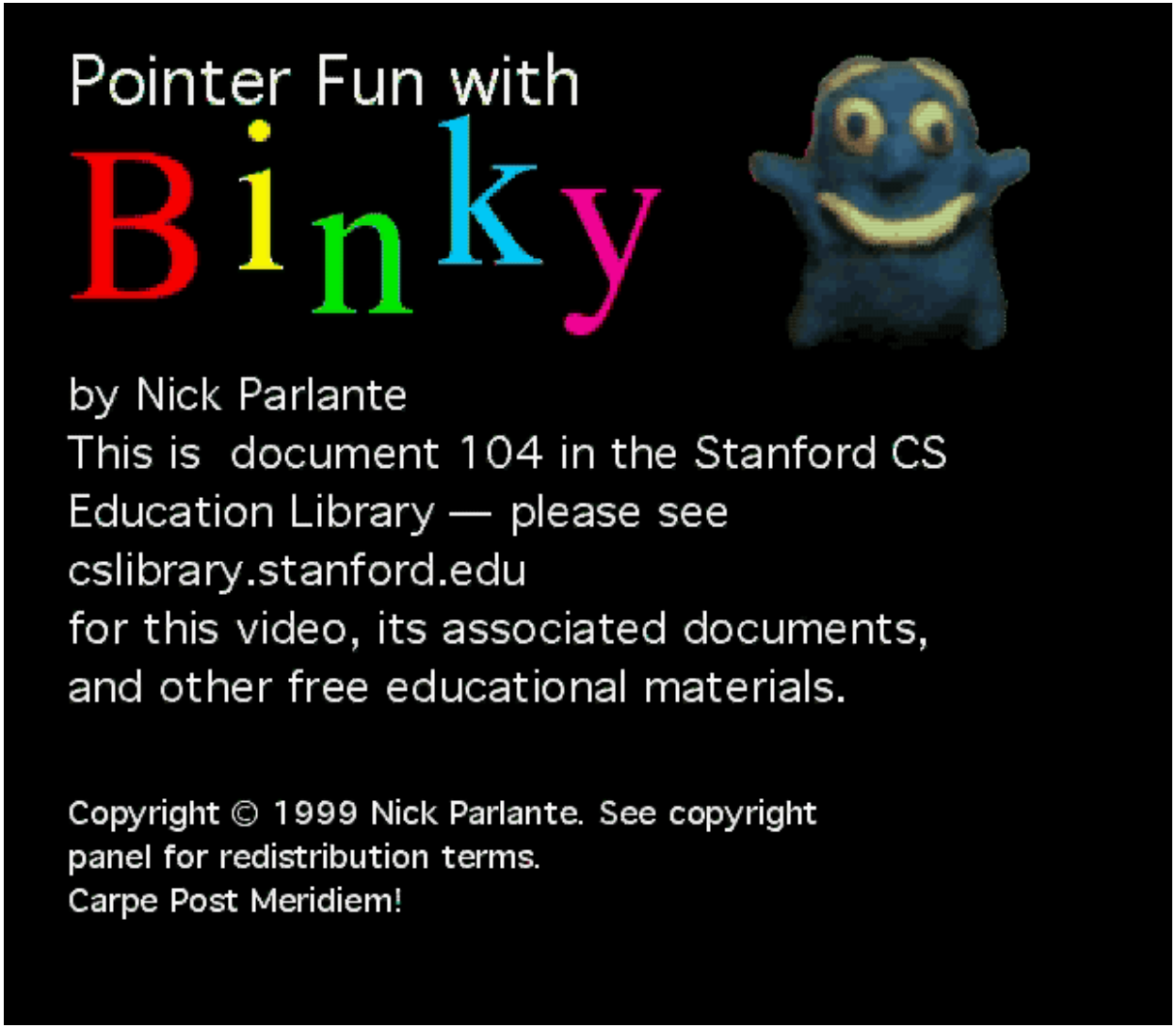
Pointer Arithmetic


```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 Arithmetic

- **int i , *iPtr;**
- **iPtr = &i;** // Assume iPtr shows address 1000
- **(*iPtr) ++;** // Causes to increment value stored in the address 1000.
- **iPtr ++;** // Causes iPtr to show address 1004 in memory
- **(*iPtr) +=2;** // Increase value by 2 stored in 1000
- **(*iPtr) =7;** // Assign 7 in address 1000.
- ***(iPtr+2) = 5;** // Assign 5 in address 1008.

Pointer Arithmetic



Relationship Between Pointers and Arrays

- An array name can be thought as a constant pointer.
- Arrays and Pointers are closely related.
- Pointers can also point arrays like they point variables.

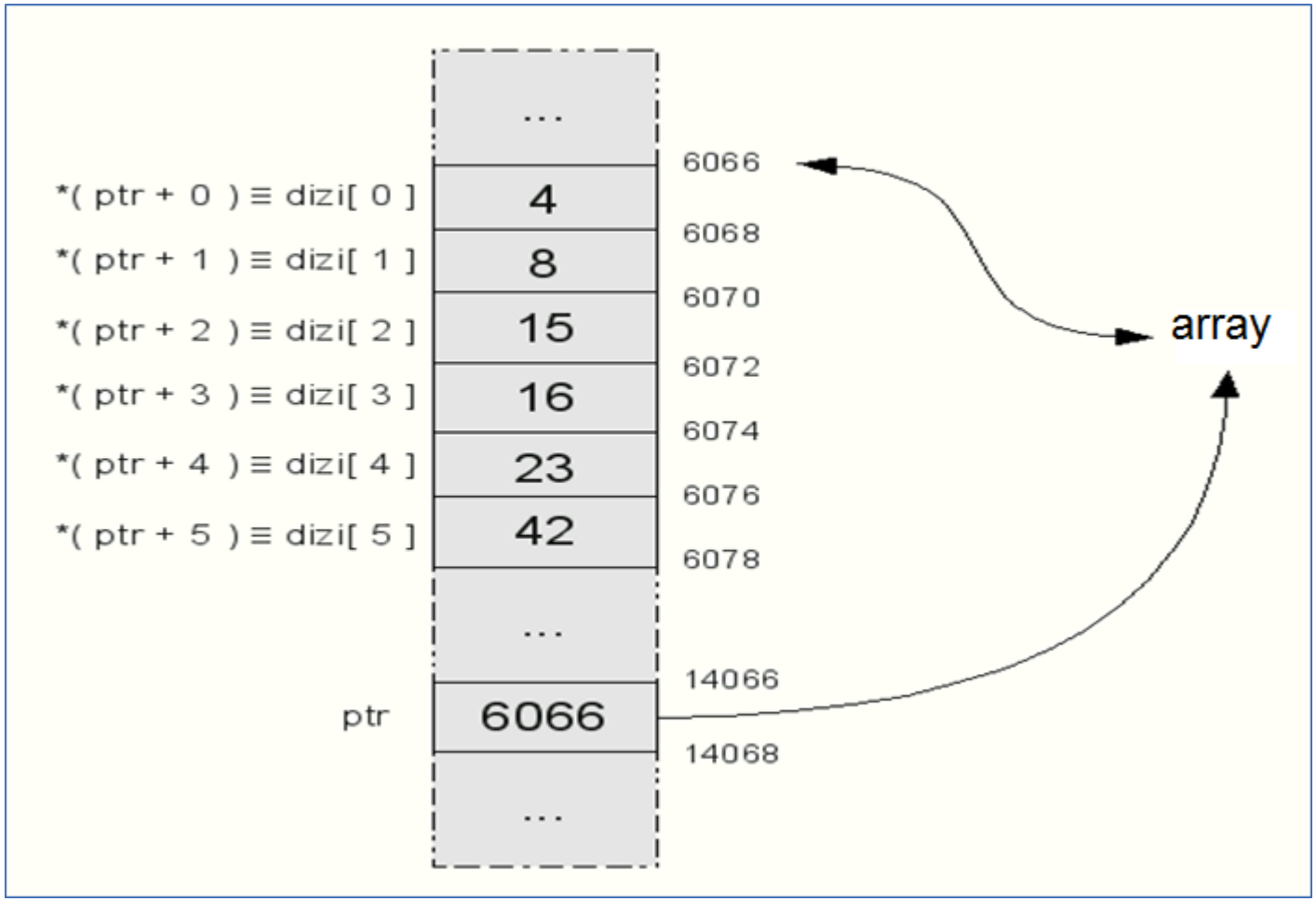
```
int dizi [6];
int *ptr;
```

- Array name can be used to level the arrays and pointers.

```
ptr = dizi; //Now ptr[0] and dizi[0] is same.
```

- To explicitly assign ptr to the address of first element of dizi as ptr = & dizi[0]

Relationship Between Pointers and Arrays



Relationship Between Pointers and Arrays

- To access the elements of the array with pointers that shows array.

  ***(ptr + n)** → where n indicates the index number of element in the array

  ***(ptr + 4)** → gets the value of element dizi[4]

- Other alternatives for dizi[4]

  ptr[4]

  ***(dizi + 4)**

Relationship Between Pointers and Arrays


```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;
}
```

Relationship Between Pointers and Arrays


```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;
}
```

Relationship Between Pointers and Arrays


```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;
}
```

Relationship Between Pointers and Arrays

- Arrays can contain pointer.
- Can access multiple arrays with arrays of pointers.
- We just assign the starting address of arrays to the arrays of pointers.
- Any modification you make on array of pointer will affect the original array.

Relationship Between Pointers and Arrays


```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

Next Week

- Pointers
- Call by Value
- Call by Reference
- Dynamic Memory Allocation

References

- 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

Any Questions ?

Thanks for listening

CANER ÖZCAN
canerozcan@karabuk.edu.tr