

Fonksiyonlar

- Fonksiyon belli bir gorevi yerine getirmek icin dizayn edilmiş,kendi icinde yeterli, program parcalarina denir.
- Neden: Sizi tekrardan kurtarir, bir islemi defalarca yapacaksanız fonksiyon kullanin. Ana program icinde istediginiz kadar cagirin. Baska programlar icinde de kullanabilirsiniz.
- Fonksiyonlar kara kutular olarak dusunulebilirler ve kendilerine gonderilen ve kendilerinden alinan bilgiler ile tanimlanirlar. Iclerinde neyin nasıl yapıldığı fonksiyonu ilgilendirir. Örneğin printf() fonksiyonu: biz sadece nasıl kullanılacağını biliyoruz, icinde neler olduğu bizi fazla alakadar etmiyor.

Fonksiyonlar

- Ana programi anlamlı iş parçalarına ayırın: her bir parça bir işi icra etsin.
- Her bir parçanın program ile olan ilişkisini tespit edin (programdan ne alıyor, programa ne geri veriyor)
- Fonksiyon nasıl tanımlanır, nasıl çağrılır, birbirleriyle nasıl haberleşirler.

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

```
long kup(long x);
```

```
long giris, cevap;
```

```
main()
```

```
{
```

```
printf("Bir tam sayi girin: ");
```

```
scanf("%d", &giris);
```

```
cevap = kup(giris);
```

```
printf("\n %ld sayisinin kupu=%ld.\n", giris, cevap);
```

```
return 0;
```

```
}
```

```
long kup(long x)
```

```
{
```

```
long x_kup;
```

```
x_kup = x * x * x;
```

```
return x_kup;
```

```
}
```

Analizi

- Fonsiyon prototipi: Compilar a ne tip bir fonksiyon oldugunu soyler
- Fonksiyonun kullanimi: fonksiyon cagrilir ve sonucu alinir
- Fonksiyonun tanimlanmasi: ne is yapacagi tanimlanir.
- Degiskenlerde oldugu gibi, her fonksiyonunda bir turu olamalidir.

long kup(long x);

- () parantezler bunun bir fonksiyon oldugunu belirtir/

Ornegin Analizi

- İlk kullanılan **long** fonksiyonun turunu ifade eder
- Parantez icindeki **long** fonksiyonun bir arguman aldığını ve türünün long olduğunu ifade eder.
- ; ise fonksiyonun deklare edildigini ifade eder (tanimlandigini degil)
- Fonksiyon deklerasyonu main() den hemen once gelir. Main() icine de, degiskenlerin tanimlandigi yer, olabilirler.

Ornegin Analizi

- main() icinde fonksiyonlar adlari ile cagrilirlar,
kup(giris);
- Fonksiyon icra edilir ve cagrildigi yere geri doner, isminden sonraki satirdan program devam eder.
- Ana program ve fonksiyon ayni file icinde olabilirler, bu durumda compile etmek daha kolaydir.
- Farkli dosyalar icinde olabilirler, bu durumda da fonksiyonu farkli programlarin kullanmasi kolaydir.

Fonksiyon Argumanlari

```
void dubs(int x, int y, int z) ;
```

3 tane arguman aliyor: x,y,z

```
void show_n_char(char ch, int num);
```

```
void show_n_char(char, int);
```

Kullanimi:

```
show_n_char(SPACE, 12);
```

Argumanlari; SPACE ve 12

Fonksiyonun deger geri dondurmеси

- **Int fonksiyon (float a, float b)**

Int: dondurulen degerin turu, fonksiyon iki float data aliyor ve bir integer datayi ana programa geri donduruyor.


```
#include <stdio.h>

int imin(int, int);

int main(void) {
    int sayi1, sayi2;
    printf("Bir cift sayi giriniz (q cikis):\n");
    while (scanf("%d %d", &sayi1, &sayi2) == 2)
    {
        printf("Iki sayidan %d ve %d kucugu %d.\n", sayi1, sayi2,
            imin(sayi1,sayi2));
        printf(" Bir cift sayi giriniz (q cikis):\n ");
    }
    return 0;
}

int imin(int n,int m) {
    int min;
    if (n < m)
        min = n;
    else min = m;
    return min;
}
```

Fonksiyon

- `kucuk=imin(sayi1,sayi2); evet`
- `Kucuk=min; hayir`
- `y=2+2*imin(sayi1,sayi2)+25; evet`
- Fonksiyon ilk return gordugunde cagrildigi yere geri doner. Birden fazla return kullanabilirsiniz.

Fonksiyon

- Fonksiyonlar geri dondurdukları tür ile aynı tür olarak deklare edilmeliler
- Fonksiyon geri bir şey dondurmuyorsa void olarak tanımlanmalı
- `int imax(int, int);` evet
- `int imax(int a, int b);` evet
- Argüman kullanılmıyorsa void yazılmalı

```
#include <stdio.h>
int imax(int, int); /* prototip, deklarasyon */
int main(void) {
    printf("İki sayinin %d ve %d buyugu %d.\n", 3, 5,
        imax(3.0, 5.0));
    return 0;
}
int imax(int n, int m) {
    int max;
    if (n > m)
max = n;
else max = m;
    return max;
}
```

Fonksiyon

- Kısa fonksiyonlarda deklarasyon yerine fonksiyon tanımı konulabilir:

```
int imax(int a, int b) { return a > b ? a : b; }  
int main()  
{  
    ...  
    z = imax(x, 50);  
    ...  
}
```

Fonksiyon içindeki değişkenler fonksiyona aittir.

```
#include <stdio.h>
int x = 1, y = 2;
void demo(void);
main(){
    printf("\n demo() dan once, x = %d ve y = %d.", x, y);
    demo();
    printf("\n demo() dan sonra, x = %d ve y = %d\n.", x, y);
    return 0;
}
```

```
void demo(void){
    int x = 88, y = 99;
    printf("\n demo() nun icinde, x = %d ve y = %d.", x, y);
}
```

Birden fazla return olabilir:

```
#include <stdio.h>
int x, y, z;
int büyük( int , int );
main()
{
    puts("iki tam sayi lutfen: ");
    scanf("%d%d", &x, &y);
    z = büyük(x,y);
    printf("\n büyük olan sayi= %d\n.", z);
    return 0;
}
int büyük( int a, int b)
{
    if (a > b)
return a;
    else
return b;
}
```

Ruh, pardon, Fonksiyon Çağırma

- `printf("%d nin yarisi %d.", x, yari(x));`
- `y = yari (x) + yari (z);`
 - `a = yari (x);`
 - `b = yari (z);`
 - `y = a + b;`
- `x = yari (third(square(yari(y))));`
 - `if (yari(x) > 10)`

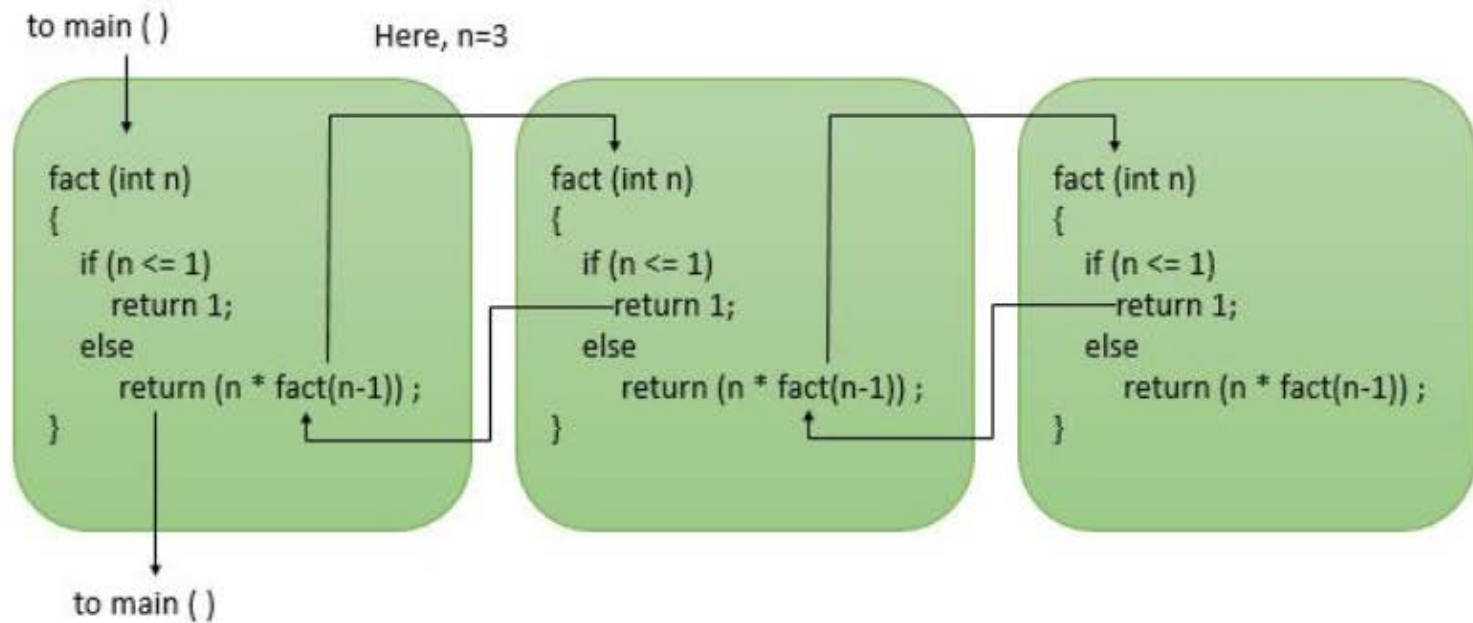
Recursion

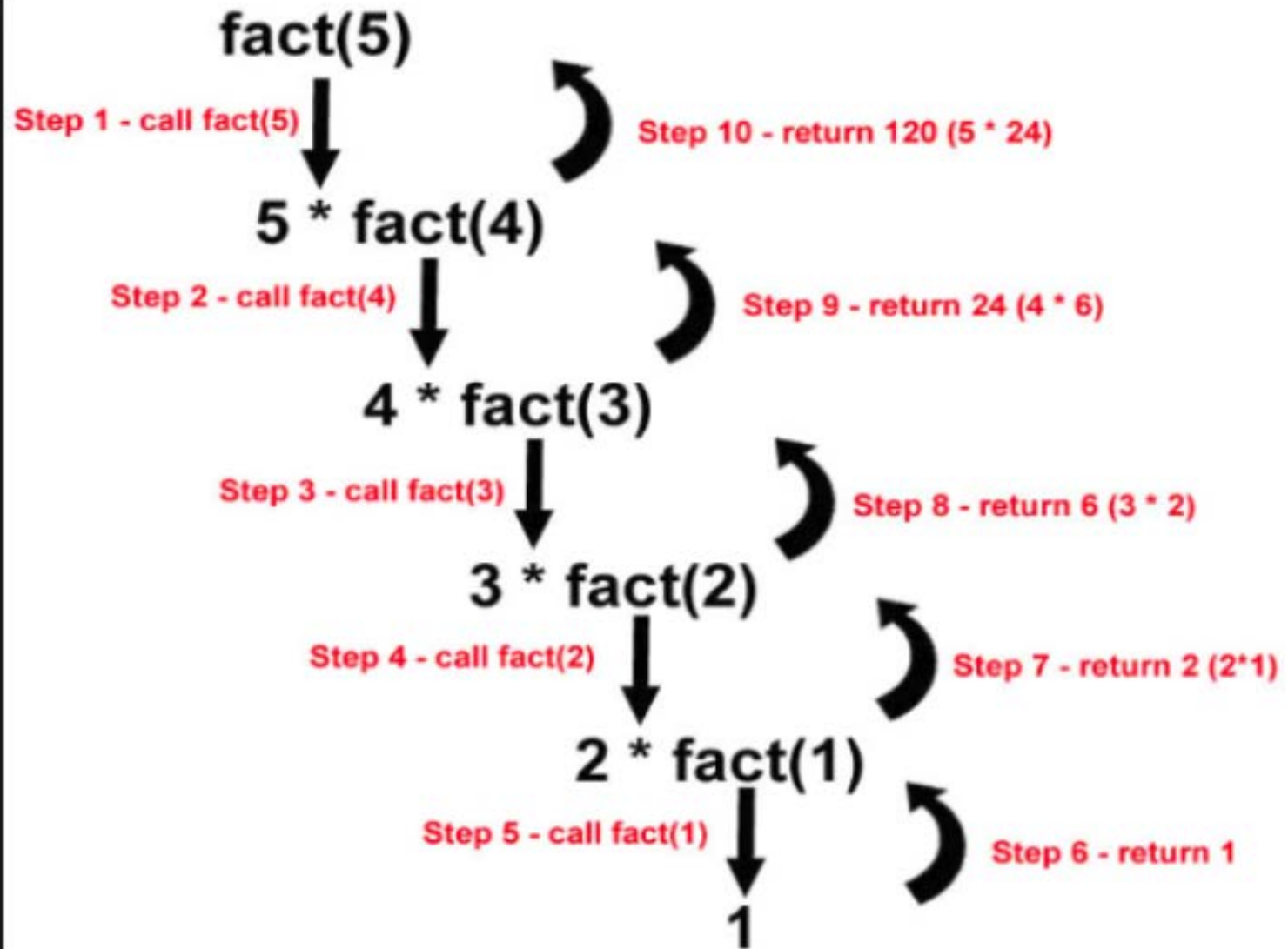
```
#include <stdio.h>
int main ()
{
    int i;
    for ( i = 1; i <=5; i++ )
        printf("%d! = %d\n",i, fact(i) );
    return 0;
}
long int fact( int n )
{
    if ( n <= 1 )
        return 1;
    else
        return ( n * fact (n-1) );
}
```

//recursive step

//end factorial

```
return ( n * fact (n-1) );
```





////////////////////////////////////Sum numbers////////////////////////////////////

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

```
int sum(int n);
```

```
int main() {
```

```
    int number, result;
```

```
    printf("Enter a positive integer: ");
```

```
    scanf("%d", &number);
```

```
    result = sum(number);
```

```
    printf("sum = %d", result);
```

```
    return 0;
```

```
}
```

```
int sum(int num)
```

```
{
```

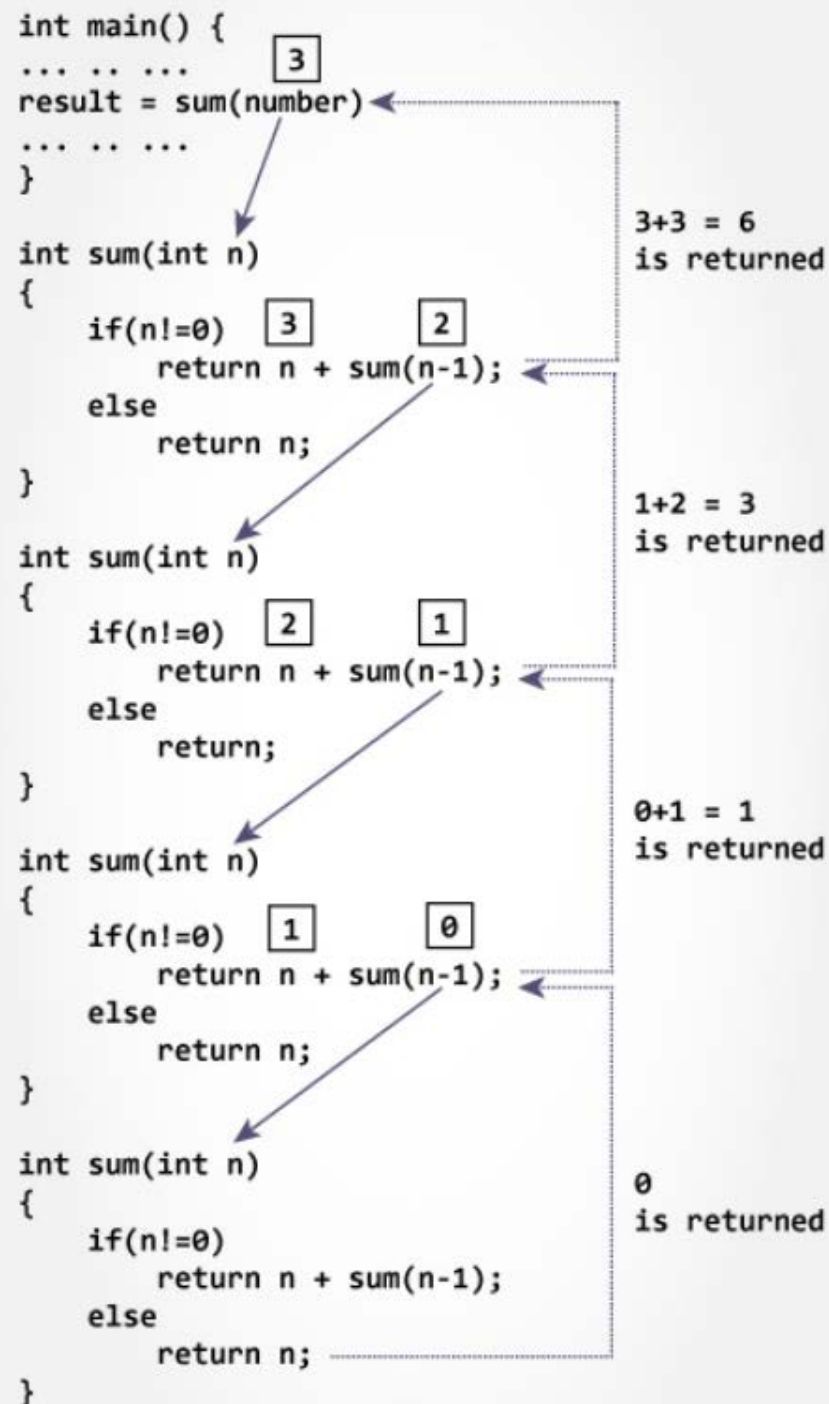
```
    if (num!=0)
```

```
        return num + sum(num-1);    // sum() function calls itself
```

```
    else
```

```
        return num;
```

```
}
```



- **Given that you can use either a loop or recursion to code a function, which should you use? Normally, the loop is the better choice.**
-
- **First, because each recursive call gets its own set of variables, recursion uses more memory; each recursive call places a new set of variables on the stack.**
- **Second, recursion is slower because each function call takes time.**
- **Recursion is worth understanding because in some cases, there is no simple loop alternative.**

```
long fact(int n)    // loop-based function
```

```
{
```

```
    long ans;
```

```
    for (ans = 1; n > 1; n--)
```

```
        ans *= n;
```

```
    return ans;
```

```
}
```

```
long rfact(int n)    // recursive version
```

```
{
```

```
    long ans;
```

```
    if (n > 0)
```

```
        ans= n * rfact(n-1);
```

```
    else
```

```
        ans = 1;
```

```
    return ans;
```

```
}
```

Eğlence

- 1 Integer sayıları recursive function kullanarak binary sayılara çeviren bir program yazınız.
- 2 Fibonacci numbers , örneğin:1, 1, 2, 3, 5, 8, 13... kendinden önceki iki sayının toplamından oluşur. Recursive function kullanarak, girilen bir tam sayıya kadar olan Fibonacci sayılarını yazan bir program yazınız.
- 3 Girilen bir tamsayının basamaklarının sırasının tersinin oluşturduğu sayıyı veren programı fonksiyon kullanarak yazınız.

Fonksiyonda değer değişimi

```
#include <stdio.h>

void interchange(int u, int v);

int main(void){
    int x = 5, y = 10;
    printf("Originally x = %d and y = %d.\n", x , y);
    interchange(x, y);
    printf("Now x = %d and y = %d.\n", x, y);
    return 0;
}

void interchange(int u, int v){
    int temp;
    printf("Originally u = %d and v = %d.\n", u , v);
    temp = u;
    u = v;
    v = temp;
    printf("Now u = %d and v = %d.\n", u, v);
}
```

Değişkenlerin adresleri

- Unary operatörü & değişkenlerin depolandığı adresleri verir.

```
#include <stdio.h>
void fonk1(int);          /* fonksiyon deklarasyonu */
int main(void)
{
    int deg1 = 2, deg2 = 5;      /* main() içinde yerel */
    printf("main() icinde, deg1 = %d and &deg1 = %p\n", deg1, &deg1);
    printf("main() icinde, deg2 = %d and &deg2 = %p\n", deg2, &deg2);
    fonk1(deg1);
    return 0;
}

void fonk1(int deg2)          /* fonksiyon tanımı */
{
    int deg1 = 10;            /* fonk1() içinde yerel*/
    printf("fonk1() icinde, deg1 = %d and &deg1 = %p\n", deg1, &deg1);
    printf("fonk1() icinde, deg2 = %d and &deg2 = %p\n", deg2, &deg2);
}
```

Dizi ve Pointer

Dizi:

- Aynı tür ve isimde verinin blok halinde saklandığı yer.

yaş **alp**[35]

- yaş:=türü,
- Alp= ismi
- 35=eleman sayısı (harcadığı yıllar)
- `int powers[8] = {1,2,4,6,8,16,32,64};`
- `int days[MONTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};`

Dizi

- Index: dizi elemanlarına ulaşmak ve onların yerlerini belirlemek için kullanılır.

Index=0,1,2,...,34: (35 eleman)

Çok boyutlu ise

alp[1][2]

Boyut sayısı konusunda sınır yok

dizinin kullandığı hafıza konusunda var

Dizi

```
float expenses[100];
```

```
int a[10];
```

```
expenses[i] = 100;
```

```
expenses[2 + 3] = 100; /* = expenses[5] */
```

```
expenses[a[2]] = 100; /* a[] bir integer dizi */
```

```
#include <stdio.h>
float aylik_gider[13];
int count;
main()
{
    for (count = 1; count < 13; count++)
    {
        printf("Aylik giderler %d: ", count);
        scanf("%f", &aylik_gider [count]);
    }

    for (count = 1; count < 13; count++)
    {
        printf("Ay %d = YTL %.2f\n", count, ylik_gider[count]);
    }
    return 0;
}
```

```
#include <stdio.h>
#define MAX_NOT 100
#define OGRENCI 10
int not[OGRENCI];
int idx;
int toplam = 0;
main() {
    for( idx=0;idx< OGRENCI;idx++)
    {
        printf( "%d inci ogrencinin notu: ", idx +1);
        scanf( "%d", &not[idx] );
        while ( not[idx] > MAX_NOT )
        {
            printf( "\nen yuksek not %d",MAX_NOT );
            printf( "\nDogru not lutfen: " );
            scanf( "%d", &not[idx] );
        }
        toplam += not[idx];
    }
    printf( "\n\n Ortalama not= %d\n", ( toplam / OGRENCI) );
    return (0);
}
```

Diziye değer atama

- `İnt alp[4]={1,3,5,7};`
- `İnt alp[4]={1,3,5};`
- Eksik sayıda eleman olabilir fakat fazla olamaz.
- `İnt alp[4][3]={1,2,3,4,5,6,7,8,9,10,11,12}`

Burada: `alp[0][0]=1 alp[0][1]=2 alp[0][2]=3`
`alp[1][0]=4 alp[1][1]=5 alp[1][2]=6`


```
int a[3][4];
```



`a[0][0]`

`a[0][1]`

`a[0][2]`

`a[0][3]`



`a[1][0]`

`a[1][1]`

`a[1][2]`

`a[1][3]`



`a[2][0]`

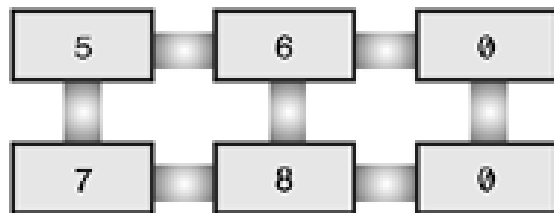
`a[2][1]`

`a[2][2]`

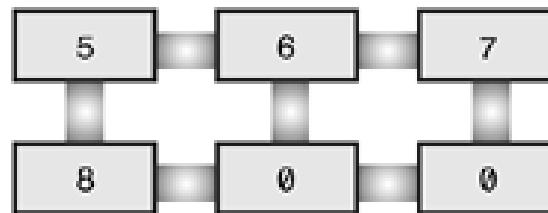
`a[2][3]`

Diziye değer atatma

- `int alp[4][3]={ {1,2,3},
 {4,5,6},
 {7,8,9},
 {10,11,12}
 };`

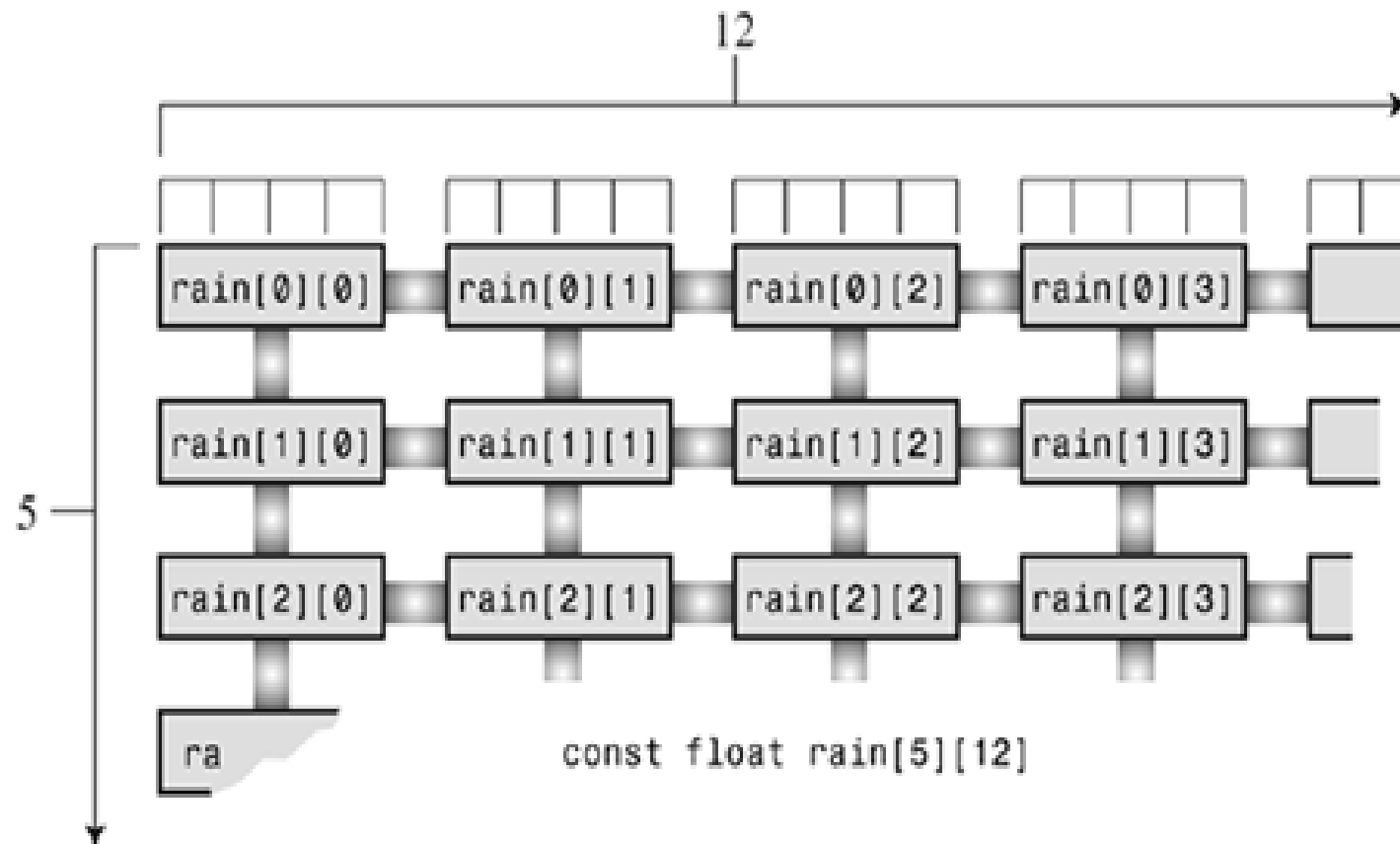


```
int sq[2][3] = {  
                {5,6},  
                {7,8}  
};
```



```
int sq[2][3]={5,6,7, 8};
```

- `Const float rain[5][12]`



Initialization of a 2d array

```
// Different ways to initialize two-dimensional array
```

```
int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};
```

```
int c[][3] = {{1, 3, 0}, {-1, 5, 9}};
```

```
int c[2][3] = {1, 3, 0, -1, 5, 9};
```

```
#include <stdio.h>

const int CITY = 2;    const int WEEK = 7;

int main(){
    int temperature[CITY][WEEK];
        for (int i = 0; i < CITY; ++i) {
            for (int j = 0; j < WEEK; ++j) {
                printf("City %d, Day %d: ", i + 1, j + 1);
                scanf("%d", &temperature[i][j]);
            }
        }
        for (int i = 0; i < CITY; ++i) {
            for (int j = 0; j < WEEK; ++j) {
                printf("City %d, Day %d = %d\n", i + 1, j + 1, temperature[i][j]);
            }
        }
    return 0;
}
```

```
#include <stdio.h>
int main() {
    // declare an array
    int a[2][2];
    // perform a few operations
    a[0][0] = 10;
    a[0][1] = a[0][0] * 10;
    a[1][0] = a[0][1] / 5;
    a[1][1] = a[0][1] + a[1][0];
    // print the array
    printf("%d %d\n", a[0][0], a[0][1]);
    printf("%d %d\n", a[1][0], a[1][1]);
    return 0;
}
```

// C program to find the sum of two matrices of order 2*2

#include <stdio.h>

int main(){

float a[2][2], b[2][2], result[2][2];

printf("Enter elements of 1st matrix\n");

for (int i = 0; i < 2; ++i)

for (int j = 0; j < 2; ++j) {

printf("Enter a%d%d: ", i + 1, j + 1);

scanf("%f", &a[i][j]);

}

printf("Enter elements of 2nd matrix\n");

for (int i = 0; i < 2; ++i)

for (int j = 0; j < 2; ++j) {

printf("Enter b%d%d: ", i + 1, j + 1);

scanf("%f", &b[i][j]);

}

for (int i = 0; i < 2; ++i)

for (int j = 0; j < 2; ++j){

result[i][j] = a[i][j] + b[i][j];

}

printf("\nSum Of Matrix:");

for (int i = 0; i < 2; ++i)

for (int j = 0; j < 2; ++j) {

printf("%.1f\t", result[i][j]);

if (j == 1)

printf("\n");

}

return 0;}

```
#include <stdio.h>

int main() {
// declare a 3x2x2 3D //array
with initializer list
int a[3][2][2] = {
{
    {10, 20},
    {30, 40},
},
{
    {1, 2},
    {4, 5}
},
{
    {3, 5},
    {7, 11}
}
};
```

```
// declare loop variable
int i, j, k;
for (i = 0; i < 3; i++) {
    for (j = 0; j < 2; j++) {
        for (k = 0; k < 2; k++) {
            printf("a[%d][%d][%d] = %d\n",
                i, j, k, a[i][j][k]);
        }
    }
}
return 0;
}
```



```
#include<stdio.h>  // ARRAY

void main(){
    void read(int *,int);
    void dis(int *,int);
    int a[5],i,sum=0;
    printf("Enter the elements of array
    \n");
    read(a,5);    /*read the array*/
    printf("The array elements are \n");
    dis(a,5);    //Show the array/
}
```

```
void read(int c[],int i)
{
    int j;
    for(j=0;j<i;j++)
        scanf("%d",&c[j]);
    fflush(stdin);
}
```

```
void dis(int d[],int i)
{
    int j;
    for(j=0;j<i;j++)
        printf("%d ",d[j]);
    printf("\n");
}
```

`/* no_data.c -- uninitialized array */`

```
#include <stdio.h>  
#define SIZE 4  
int main(void)  
{  
  int no_data[SIZE];      /* uninitialized array */  
  int i;  
  printf("%s%s\n", "i", "no_data[i]);  
    for (i = 0; i < SIZE; i++)  
      printf("%d%d\n", i, no_data[i]);  
return 0;  
}
```

/* some_data.c -- partially initialized array */

#include <stdio.h>

#define SIZE 4

int main(void)

{

int some_data[SIZE] = {1492, 1066};

int i;

printf("%s%s\n", "i", "some_data[i]);

for (i = 0; i < SIZE; i++)

printf("%d%d\n", i, some_data[i]);

return 0;

}

```
#include <stdio.h>      #include <stdlib.h>
int random_array[10][10][10];    int a, b, c;
main() {
    for (a = 0; a < 10; a++){
        for (b = 0; b < 10; b++) {
            for (c = 0; c < 10; c++){
                random_array[a][b][c] = rand();
            }
        }
    }

    for (a = 0; a < 10; a++) {
        for (b = 0; b < 10; b++){
            for (c = 0; c < 10; c++) {
                printf("\nrandom_array[%d][%d][%d] = ", a, b, c);
                printf("%d", random_array[a][b][c]);
            }
            printf("\nDevam icin Enter, CTRL-C cikis.");
            getchar(); //her bir boyutu görmek için duraklat
        }
    }

    return 0;
}
```

**// The first element of first list is added to the first element of the second list, and
//the result of the addition is the first element of the third list.**

```
#include<stdio.h>
#include<conio.h>
void main() {
    void read(int *,int);
    void dis(int *,int);
    void add(int *,int *,int *,int);
    int a[5],b[5],c[5],i;
        printf("Enter the elements of first list \n");
        read(a,5);                                /*read the first list*/
        printf("The elements of first list are \n");
        dis(a,5);                                  /*Display the first list*/
        printf("Enter the elements of second list \n");
        read(b,5);                                  /*read the second list*/
        printf("The elements of second list are \n");
        dis(b,5);                                  /*Display the second list*/
            add(a,b,c,i);
            printf("The resultant list is \n");
            dis(c,5);
    }
```

```
void add(int a[],int b[],int c[],int i) {  
    for(i=0;i<5;i++)  
    {  
        c[i]=a[i]+b[i];  
    }  
}  
  
    void read(int c[],int i) {  
        int j;  
        for(j=0;j<i;j++)  
            scanf("%d",&c[j]);  
        fflush(stdin);  
    }  
  
    void dis(int d[],int i) {  
        int j;  
        for(j=0;j<i;j++)  
            printf("%d ",d[j]);  
            printf("\n");  
    }  
  
}
```

//The following program makes a reverse version of the list.

```
#include<stdio.h>
#include<conio.h>
void main(){
    void read(int *,int);
    void dis(int *,int);
    void inverse(int *,int *,int);
    int a[5],b[5];
    read(a,5);
    dis(a,5);
    inverse(a,b,5);
    dis(b,5);
}

    void read(int c[],int i){
        int j;
        printf("Enter the list \n");
        for(j=0;j<i;j++)
            scanf("%d",&c[j]);
        fflush(stdin);
    }
```

```
void dis(int d[],int i){
    int j;
    printf("The list is \n");
        for(j=0;j<i;j++)
            printf("%d ",d[j]);
    printf("\n");
}

void inverse(int a[],int inverse_b[],int j){
    int i,k;
    k=j-1;
        for(i=0;i<j;i++)
        {
            inverse_b[i]=a[k];
            k--;
        }
}
```


- **MERGING OF TWO SORTED LISTS**

Suppose the first list is 10 20 25 50 63, and the second list is 12 16 62 68 80. The sorted lists are 63 50 25 20 10 and 80 68 62 16 12.

The first element of the first list is 63, which is smaller than 80, so the first element of the resultant list is 80. Now, 63 is compared with 68; again it is smaller, so the second element in the resultant list is 68. Next, 63 is compared with 50. In this case it is greater, so the third element of the resultant list is 63.

Repeat this process for all the elements of the first list and the second list. The resultant list is 80 68 63 62 50 25 20 16 12 10

```
#include<stdio.h>

void main()
{
    void read(int *,int);    void dis(int *,int);    void sort(int *,int);    void merge(int *,int *,int
        *,int);
    int a[5],b[5],c[10];
    printf("Enter the elements of first list \n");        read(a,5);    /*read the list*/
    printf("The elements of first list are \n");        dis(a,5);    /*Display the first list*/
    printf("Enter the elements of second list \n");        read(b,5);    /*read the list*/
    printf("The elements of second list are \n");        dis(b,5);    /*Display the second list*/
    sort(a,5);
    printf("The sorted list a is:\n");
    dis(a,5);
    sort(b,5);
    printf("The sorted list b is:\n");
    dis(b,5);
    merge(a,b,c,5);
    printf("The elements of merged list are \n");
    dis(c,10);        /*Display the merged list*/
}
```

```
void read(int c[],int i)
{   int j;
    for(j=0;j<i;j++)
        scanf("%d",&c[j]);   fflush(stdin);
}

void dis(int d[],int i)
{   int j;
    for(j=0;j<i;j++)
        printf("%d ",d[j]);   printf("\n");
}

void sort(int arr[],int k)
{   int temp;   int i,j;
    for(i=0;i<k;i++)
    {
        for(j=0;j<k-i-1;j++)
        {
            if(arr[j]<arr[j+1])
            {
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
}
```

```
void merge(int a[],int b[],int c[],int k)
{
    int ptra=0,ptrb=0,ptrc=0;
    while(ptra<k && ptrb<k)
    {
        if(a[ptra] > b[ptrb])
        {
            c[ptrc]=a[ptra];
            ptra++;
        }
        else
        {
            c[ptrc]=b[ptrb];
            ptrb++;
        }
        ptrc++;
    }
    while(ptra<k)
    {
        c[ptrc]=a[ptra];
        ptra++;ptrc++;
    }
    while(ptrb<k)
    {
        c[ptrc]=b[ptrb];
        ptrb++; ptrc++;
    }
}
```

Dizi

- Dizi 64KB ten fazla yer kaplamasın. (Bu tabi ki aşılabilir bir limit)

Adress of Operatörü

&

değişken=değer (5,-2.8,g,i,...) tutar.

&değişken=değrin saklandığı adresi

**& operatörü değişken adresine
yönlendirir.**

Pointers

- Pointer nedir anlayabilmek için bilgisayar bilgiyi nasıl saklar bilmek lazım: RAM ard arda dizilmiş binlerce depolama ünitesinden oluşur ve her bir ünitenin kendine ait bir adı (adresi) vardır. Hafıza adresi 0 dan başlar maximuma (ne kadar hafıza, RAM, yüklü ise) kadar gider.
- Bir değişken deklere ettiğinizde compiler değişkenin türüne göre hafızada bir yer ayırır; adresi bilinen bir yer. Bu adres değişkenin adı ile ilişkilendirilir. Program değişkenin adını kullandığında otomatik olarak adresi alır ve orada depolanmış bilgiye ulaşır.
- Adress bu şekilde kullanılır fakat biz farkında olmayız.

Pointers

- Adres bir tam sayıdır ve C dilinde tam sayılarla neler yapılabilirse adres ile aynıları yapılır.
- Eğer bir değişkenin adresini biliyorsanız, ikinci bir değişken tanımlayarak ilk değişkenin adresini burada saklayabilirsiniz.
- Pointer diğer değişkenler gibi önce deklere edilir:

tür *pointer_ismi;

Burada tür değişkenin türüdür (pointerin işaret ettiği değişken)

Burada asteriks (*) ise pointer_ismi 'nin bir pointer olduğunu (bir değişken olmadığını) belirtir.

Pointers

Pointer da diğer değişkenler gibi tanımlanır:

char *alp1,*alp2, ch, alp3;

float sayi1, *sayi2;

Pointer değeri atama:

- Pointer deklere edildikten sonra bir değer almalı ki o değeri temsil etsin, işaret, etsin. Pointer değer ataması yapılmaz ise değişken gibi hafızadaki rastgele bir değeri alır .

Pointers

- Bir değişkenin adresi address-of ismiyle bilinen C karakteri “&” ile alınır.
- & karakteri bir değişkenin önüne getirilirse o değişkenin adresini geri döndürür:

Pointer=&değişken;

Ör:

float oran, *p_oran;

oran=2.8;

p_oran=&oran;

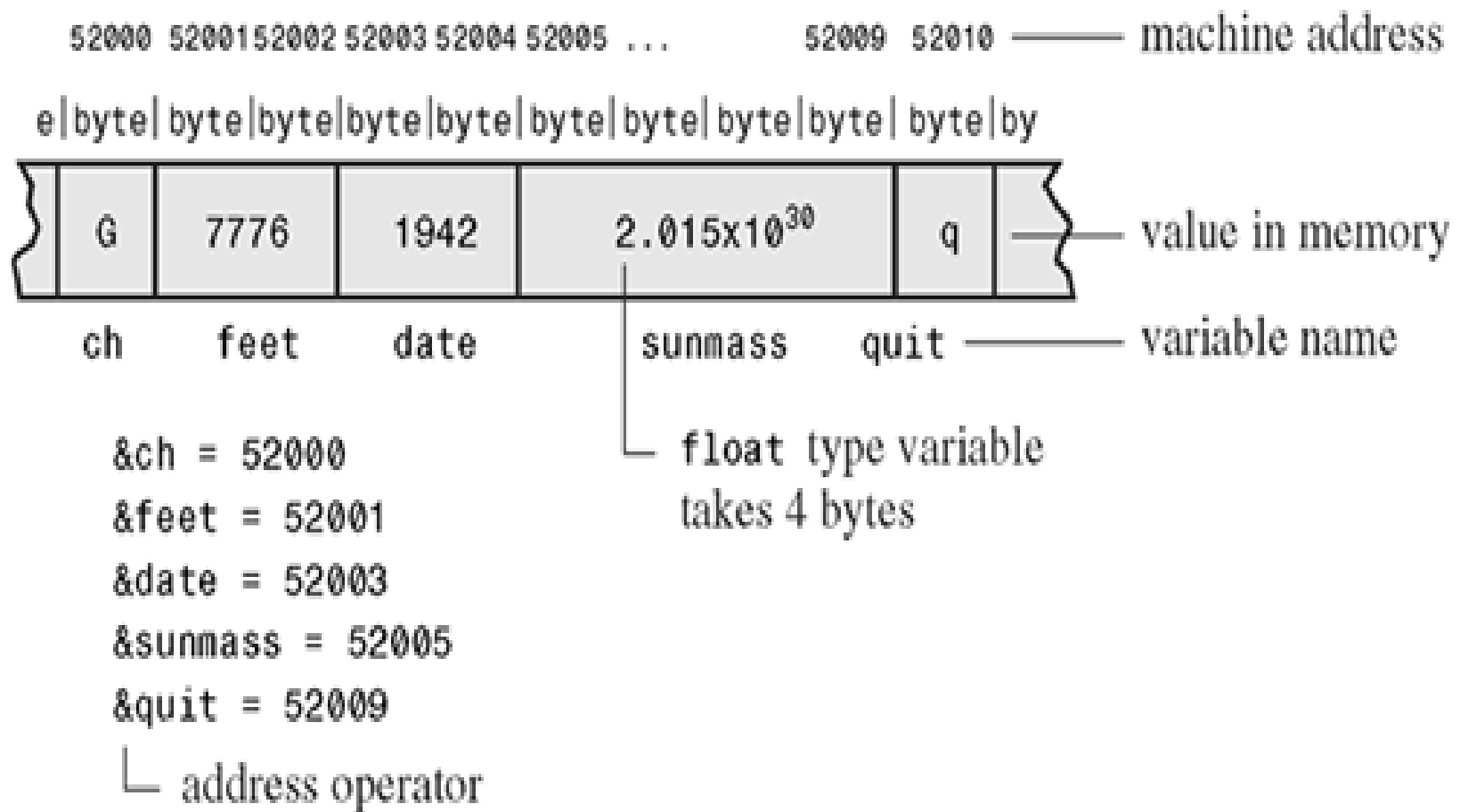
Pointers

- Nasıl Kullanırız, Ne işimize yarar:
- Eğer “ * ” karakteri pointer önüne konursa pointerin işaret ettiği değişkeni anlaşılır.

Printf(“%f”, oran);

Printf(“%f”, *p_oran);

- İki printf() aynı sonucu verir. Birincisi değişkene direkt olarak adı ile ulaşıyor. İkinci print() ise değişkene pointer ile (***p_oran=oran**).



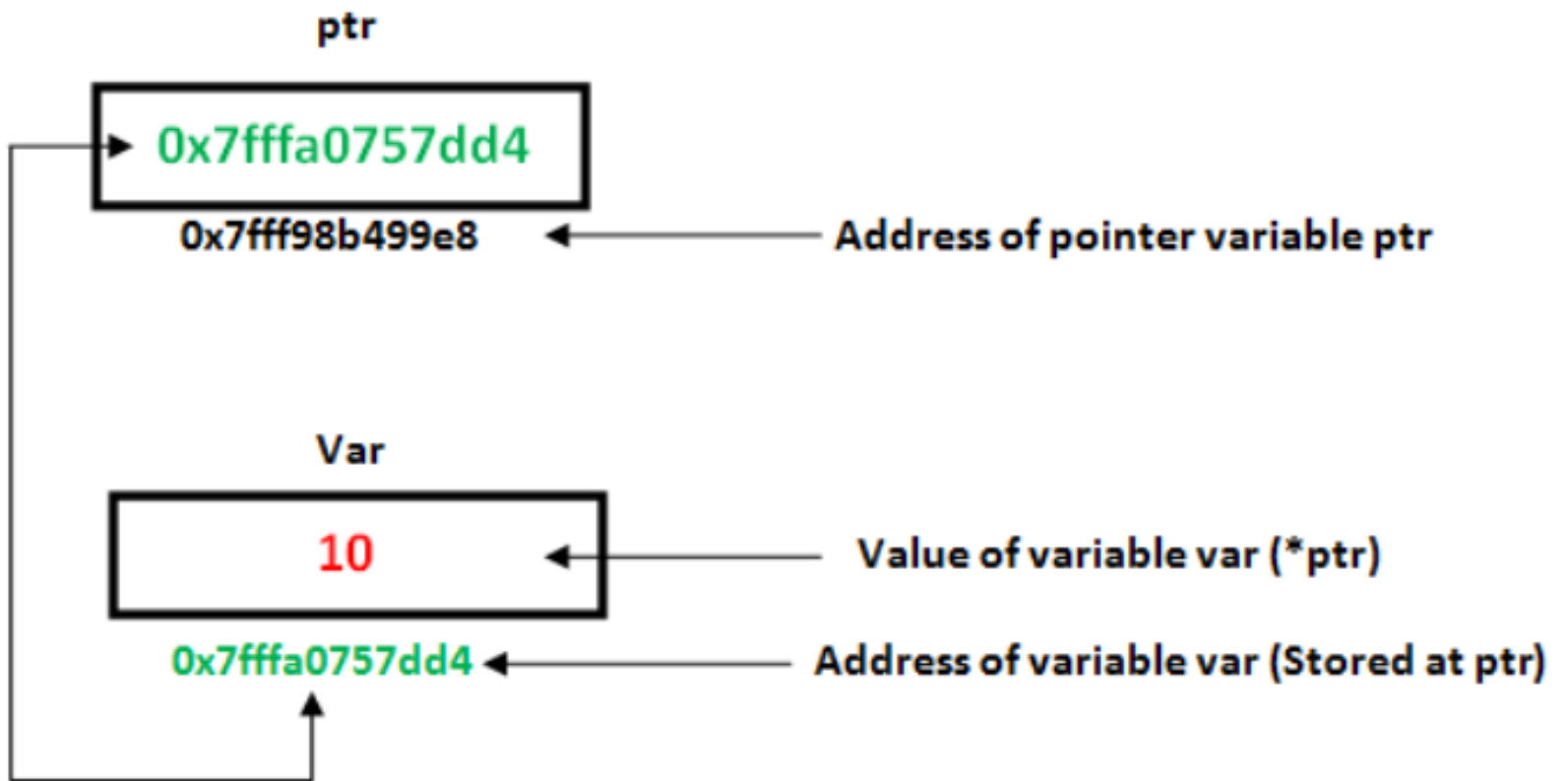
```
#include <stdio.h>

int var = 1;                /* int var deklere et ve değer ata */
int *ptr;                   /* int işaret etmek için pointer deklere et */

main()
{
    /* var adresini ptr ye yükle (pointer ilk değerini ata) */
    ptr = &var;

    /* var değişkenine direkt ve indirekt olarak ulaş */
    printf("\n Direkt, var = %d", var);
    printf("\n Indirekt, var = %d", *ptr);

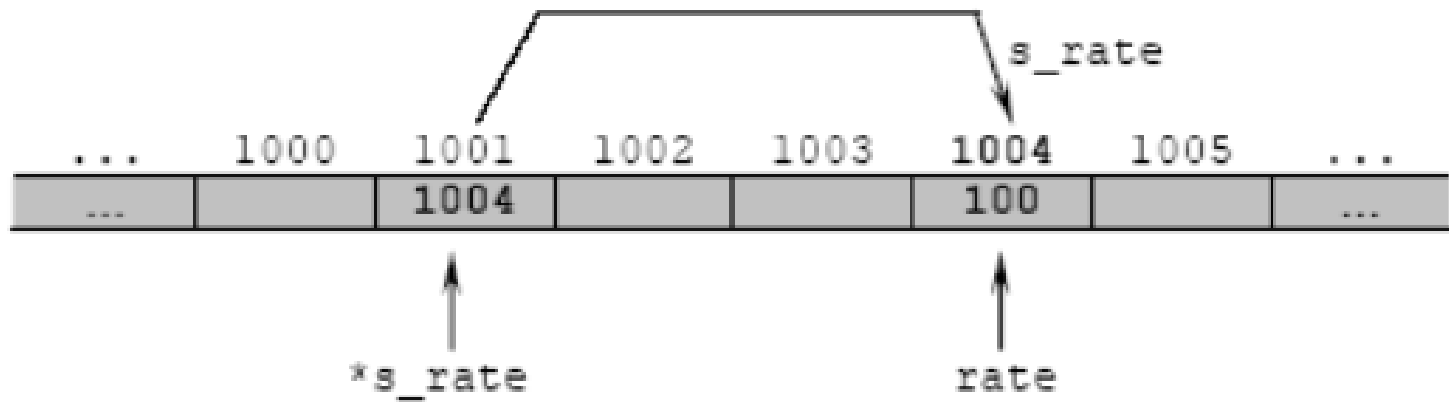
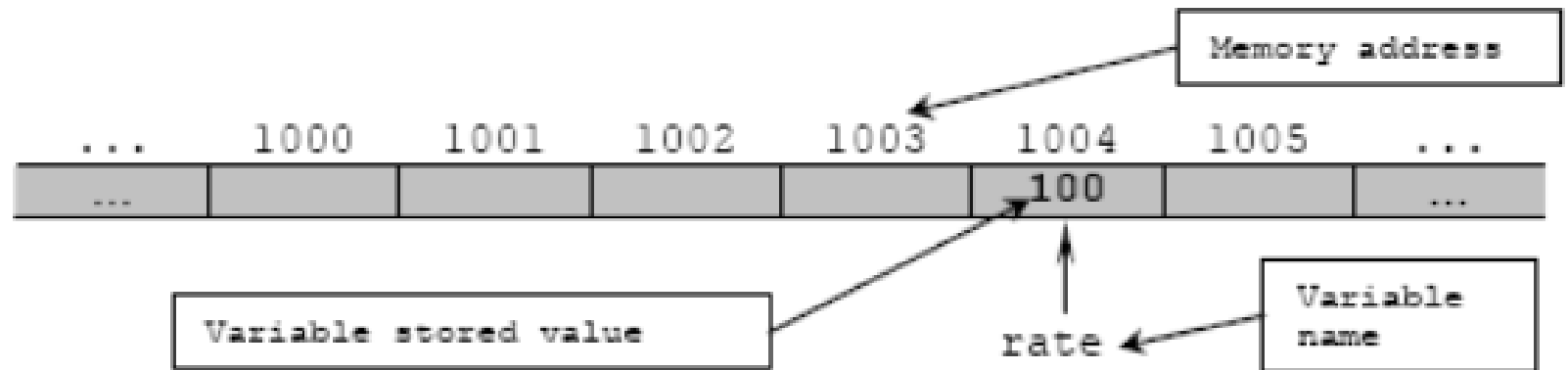
    /* değişken adresinin iki yollan yazılması */
    printf("\n\n var adresi = %d", &var);
    printf("\n var adresi = %d\n", ptr);
    return 0;
}
```



Pointers

- Pointer ve veri türü: Pointer ilk byte adresini işaret eder ve türe göre verinin hangi adresleri işgal ettiği belirlenir; int ise 2 bytes, float ise 4 bytes gibi.

```
int rate = 100;
```



/* swap3.c -- using pointers to make swapping work */

#include <stdio.h>

void interchange(int * u, int * v);

int main(void)

{

int x = 5, y = 10;

printf("Originally x = %d and y = %d.\n", x, y);

interchange(&x, &y); /* send addresses to function */

printf("Now x = %d and y = %d.\n", x, y);

return 0;

}

void interchange(int * u, int * v)

{

int temp;

temp = *u; /* temp gets value that u points to */

****u = *v;***

****v = temp;***

}


```
#include <stdio.h>

#define MAX 10

void display_array(int*,const int);

int main(){
    int i;    int a[MAX];    int* pa;

    pa = a;
    for(i = 0; i < MAX; i++) {
        *(pa + i) = rand() % 100;
    }
    display_array(pa,MAX); /* display array via pointer */
    display_array(a,MAX); /* display array via array name */
    return 0;
}

void display_array(int* p,const int size){
    int i;
    for(i = 0; i < size; i++)    {
        printf("%d ",p[i]);
    }
    printf("\n");}
```



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

```
main(){  
int a= 3, b = 5;  
int *c = multiply (&a,&b);  
printf("Product = %d",*c);  
}
```

Pointers as Function Return

```
int* multiply(int *a, int *b)  
{  
    int c = *a * *b;  
    return &c;  
}
```

Pointers ve Diziler

- Köşeli parantez olmadan dizi ismi kullanılırsa dizinin ilk elemanının adresi elde edilir.

```
int Dizi[10];
```

```
dizi==&dizi[0]
```

```
int *p_dizi;
```

```
p_dizi=dizi;
```

İlk örnekte dizi ismi aynı zamanda diziyi işaret eden (ilk elemanı) bir pointer. Ancak Bu pointer bir sabittir ve değiştirilemez.(dizi yeri hafızada sabit)

Pointers ve Diziler

- İkinci örnekte ise bir pointer değişkeni tanımlandı ve değeri dizinin ilk elemanı olarak atandı. Bu pointerin işaret ettiği adres her zaman program içinde değiştirilebilir. Örneğin dizinin diğer elemanlarını işaret edebilir.
- Dizinin ilk elemanının adresi 1000 (bu bir int dizisi) ise ikinci elemanın adresi 1002 dir. (hatırlanırsa int 2 byte)
- Float bir dizi için ilk adres 1000 ise ikinci elemanın adresi 1004 (float 4 bytes)

```
int *p;
```

```
p = arr;
```

```
// or,
```

```
p = &arr[0];
```

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

```
int main()
```

```
{
```

```
    int i;
```

```
    int a[5] = {1, 2, 3, 4, 5};
```

```
    int *p = a;    // same as int*p = &a[0]
```

```
    for (i = 0; i < 5; i++)
```

```
    {
```

```
        printf("%d", *p);
```

```
        p++;
```

```
    }
```

```
    return 0;
```

```
}
```

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

```
int main(){
```

```
int x;
```

```
int *ptr_p;
```

```
x = 5;
```

```
ptr_p = &x;
```

```
*ptr_p = 10;
```

```
printf("%d\n", x);
```

```
return 0;
```

```
}
```



```
#include <stdio.h>
int i[10], x;
float f[10];
double d[10];
main()
{
    printf("\t\tInteger\t\tFloat\t\tDouble");
    printf("\n=====");
    printf("=====");
    /* Dizinin her bir elemanının adresi. */
    for (x = 0; x < 10; x++)
        printf("\nEleman %d:\t%d\t%d\t\t%d", x, &i[x], &f[x], &d[x]);
    printf("\n=====");
    printf("=====\n");
    return 0;
}
```

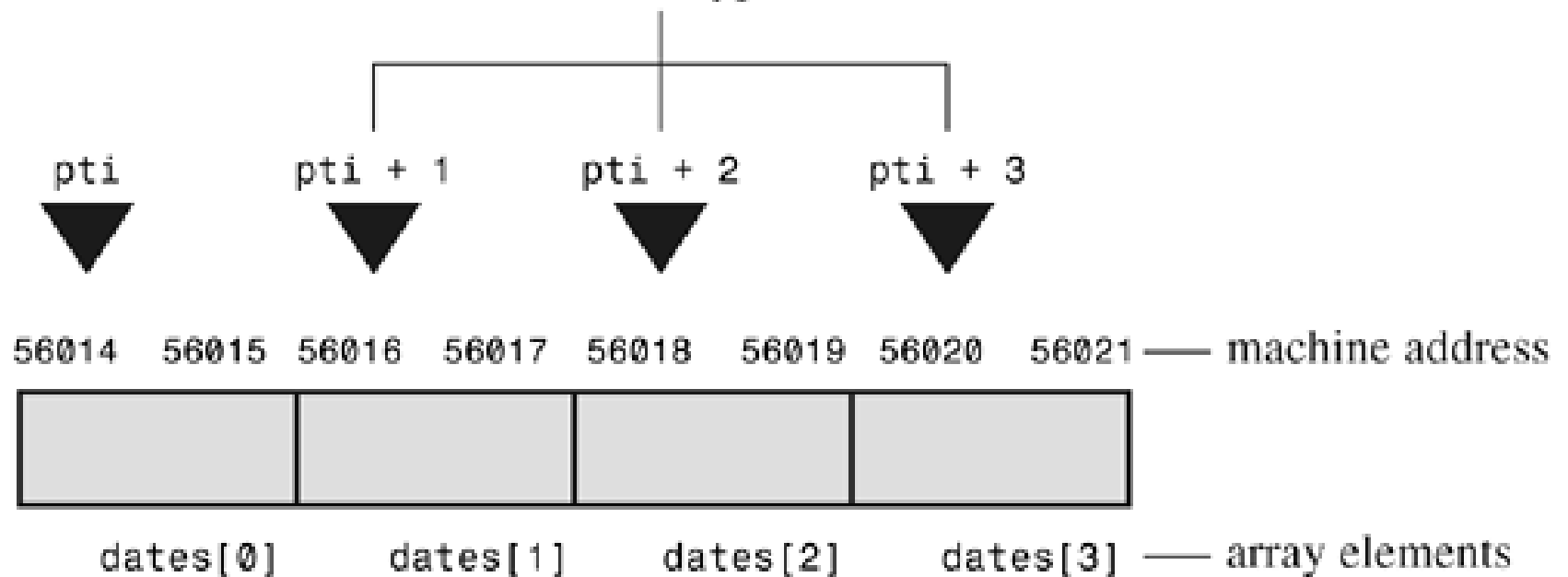
Pointer Aritmatik

- Pointer üzerinde aritmatik işlemler yapılabilir; pointer değeri üzerinde.
- Pointer için artırma veya azaltma +1 veya -1 ile yapılır; ama gerçekte C artırma veya eksiltmeyi veri türüne göre gerçekleştirir;
- Örneğin int için 2 byte artırılır pointer adresi.
- `Int_pointer++` (adres 2 byte artar)
- `Float_pointer--` (adres 4 byte azalır)

Pointer Aritmatik

- `Int_pointer+=4;` adres $4 \times 2\text{Bytes} = 8\text{bytes}$ artar.
- `Float_pointer+10;` ???
- Dikkat: C pointer başlangıç veya bitiş adresleri hatırlamaz; artırma veya eksiltme yaparken dizi dışına çıkabilirsiniz.

pointer addition increases by 2
since `pti` is type `int`



```
#include <stdio.h>
#define MAX 10
int i_array[MAX] = { 0,1,2,3,4,5,6,7,8,9 };
int *i_ptr, count;          /* pointer ve deęişken deklere et. */
float f_array[MAX] = { .0, .1, .2, .3, .4, .5, .6, .7, .8, .9 };
float *f_ptr;               /* float türünde pointer deklere et. */

main()
{
/* pointer başlangıç deęerlerini, adreslerini, ata */
i_ptr = i_array;
f_ptr = f_array;
/* Dizileri yazdır*/
for (count = 0; count < MAX; count++)
printf("%d\t%f\n", *i_ptr++, *f_ptr++);
return 0;
}
```

Pointer Aritmatik

- Diğer aritmatik işlemler de yapılabilir; Kritik nokta: birden fazla pointer ile işlem yapıyorsanız bunlar aynı dizi için tanımlanmış olmalılar. Farklı diziler için tanımlanmış iki pointer ile işlem yapamazsınız.

- İşlemler: `pnt1-pnt2;`
`pnt1<pnt2`

`Pnt1==pnt2;`

`!=, >, <, >=,`

Bölme ve çarpma ya izin yok

```
#include <stdio.h>

const int MAX = 3;

int main () {
    int var[] = {10, 100, 200};
    int i, *ptr;
    ptr = var;
    for ( i = 0; i < MAX; i++) {
        printf("Address of var[%d] = %x\n", i, ptr );
        printf("Value of var[%d] = %d\n", i, *ptr );
        /* move to the next location */
        ptr++;
    }
    return 0;}

```

```
Address of var[0] = bf882b30
Value of var[0] = 10
Address of var[1] = bf882b34
Value of var[1] = 100
Address of var[2] = bf882b38
Value of var[2] = 200

```

```
#include <stdio.h>

const int MAX = 3;

int main () {
    int var[] = {10, 100, 200};
    int i, *ptr;
    ptr = var;
    i = 0;
    while ( ptr <= &var[MAX - 1] ) {
        printf("Address of var[%d] = %x\n", i, ptr );
        printf("Value of var[%d] = %d\n", i, *ptr );
        ptr++;
        i++;
    }
    return 0;
}
```

```
Address of var[0] = bfdbcb20
Value of var[0] = 10
Address of var[1] = bfdbcb24
Value of var[1] = 100
Address of var[2] = bfdbcb28
Value of var[2] = 200
```


- `ptr_to_int++;`
- `ptr_to_float++;`
- `ptr_to_int += 4;` (increases the value stored in `ptr_to_int` by 8 (assuming that an integer is 2 bytes), so it points four array elements ahead)
- `ptr_to_float += 10;` (increases the value stored in `ptr_to_float` by 40 (assuming that a float is 4 bytes), so it points 10 array elements ahead)

Önemli

- Dikkat:

Bir pointer tanımladınız

```
İnt *ptr;
```

Ve ilk değerini, hangi adresi gösterdiğini tanımlamadınız;

Ve şer bu pointer tarafından işaret edilen adrese bir değer atarsanız

```
*ptr=25;
```

Pointer tarafından işaret edilen (ve tarafımızdan bilinmeyen, biz adres göstermedik) yere, adrese, 25 sayısını sakladık. Bu adres hafızada herhangi bir yer olabilir. Program kodlarının saklandığı yerde olabilir, işletim sistemi tarafından kullanılan bir yerde.

25 değeri o adresteki değerin üzerine yazılır ve sonuç ilginç program hataları da olabilir, bilgisayarın çökmesine de neden olabilir.

```
#include <stdio.h>

int main() {
    int i, x[6], sum = 0;
    printf("Enter 6 numbers: ");
    for(i = 0; i < 6; ++i) {
        // Equivalent to scanf("%d", &x[i]);
        scanf("%d", x+i);

        // Equivalent to sum += x[i]
        sum += *(x+i);
    }
    printf("Sum = %d", sum);
    return 0;
}
```

```
Enter 6 numbers:  2
3
4
4
12
4
Sum = 29
```

Dizi ve pointer

- Dizi ismi dizinin ilk elemanının adresini verir.

- `İnt dizi[10];`

`dizi` =ilk elemanın adresi

`dizi=&dizi[0];`

`*dizi=dizinin ilk elemanı=dizi[0]`

`*(dizi+1)=dizi[1]`

`*(dizi+2)=dizi[2]`

- `*(array) == array[0]`
- `*(array + 1) == array[1]`
- `*(array + 2) == array[2]`
- ...
- `*(array + n) == array[n]`

```
dates +2 == &date[2]
```

```
*(dates +2)
```

```
*(dates + 2) == dates[2]
```

```
*dates +2
```

```
/* day_mon3.c -- uses pointer notation */
#include <stdio.h>
#define MONTHS 12
int main(void)
{
int days[MONTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};
int index;

    for (index = 0; index < MONTHS; index++)
        printf("Month %2d has %d days.\n", index + 1,
            *(days + index)); // same as days[index]
    return 0;
}
```

```
#include <stdio.h>
const int MAX = 3;
int main () {
    int var[] = {10, 100, 200};
    int i;
    for (i = 0; i < MAX; i++) {
        printf("Value of var[%d] = %d\n", i, var[i] );
    }
    return 0;
}
```

Value of var[0] = 10

Value of var[1] = 100

Value of var[2] = 200

```
#include <stdio.h>
const int MAX = 3;
int main () {
    int var[] = {10, 100, 200};
    int i, *ptr[MAX];
    for ( i = 0; i < MAX; i++) {
        ptr[i] = &var[i]; /* assign the address of integer. */
    }
    for ( i = 0; i < MAX; i++) {
        printf("Value of var[%d] = %d\n", i, *ptr[i] );
    }
    return 0;
}
```

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

```
int main () {
```

```
    /* an array with 5 elements */
```

```
    double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

```
    double *p;
```

```
    int i;
```

```
    p = balance;
```

```
    /* output each array element's value using pointer */
```

```
    for ( i = 0; i < 5; i++ ) {
```

```
        printf("(p + %d) : %f\n", i, *(p + i) );
```

```
    }
```

```
    printf( "Array values using balance as address\n");
```

```
        for ( i = 0; i < 5; i++ ) {
```

```
            printf("(balance + %d) : %f\n", i, *(balance + i) );
```

```
        }
```

```
    return 0;
```

```
}
```



```
#include <stdio.h>
const int MAX = 4;
int main () {
    char *names[] = {
        "Zara Ali",
        "Hina Ali",
        "Nuha Ali",
        "Sara Ali"
    };
    int i = 0;
    for ( i = 0; i < MAX; i++) {
        printf("Value of names[%d] = %s\n", i, names[i] );
    }
    return 0;
}
```

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

```
int main()
```

```
{
```

```
    int x[5] = {1, 2, 3, 4, 5};
```

```
    int *ptr;
```

```
    ptr = &x[2];
```

```
    printf("*ptr = %d \n", *ptr);
```

```
    printf("*ptr+1 = %d \n", *ptr+1);
```

```
    printf("*ptr-1 = %d", *ptr-1);
```

```
    return 0;
```

```
}
```

```
*ptr = 3
```

```
*ptr+1 = 4
```

```
*ptr-1 = 2
```

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

```
void greatestOfAll( int *p){
```

```
    int max = *p;
```

```
    for(int i=0; i < 5; i++){
```

```
        if(*(p+i) > max)
```

```
            max = *(p+i);
```

```
    }
```

```
    printf("The largest element is %d\n",max);
```

```
}
```

```
main(){
```

```
    int myNumbers[5] = { 34, 65, -456, 0, 3455};
```

```
    greatestOfAll(myNumbers);
```

```
}
```

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

```
#include <string.h>
```

```
void wish(char *p){  
    printf("Have a nice day, %s",p);  
}
```

```
main(){  
    printf("Enter your name : \n");  
    char name[20];  
    gets(name);  
    wish(name);  
}
```

Dizi ve fonksiyonlar

- Daha önce gördük; fonksiyon argümanları farklı türlerde (int, float, char,...) olabilir fakat tek matematiksel değer olabiliyorlar.
- Dizinin elemanlarını tek tek fonksiyon argümanı olarak kullanabiliriz, fakat, dizinin tamamını fonksiyona yollamak istiyorsak?
- Dizi işlemleri yapan bir fonksiyon yazmak durumundasınız ve dizi boyutu her kullanımda değişiyor, nasıl bir fonksiyon?

Dizi ve fonksiyonlar

- Fonksiyona dizinin ilk elemanının gösteren pointer yollarız (pointer tek nümerik bir sayı) böylece fonksiyon adresi kullanarak diziye ulaşır ve işlemlerini yapar.
- Dizi boyutu değişken ise fonksiyon dizi sonunu (boyutunu) nasıl bilecek:
- Dizinin son elemanını özel bir değer seçersiniz ve fonksiyon dizi işlemleri yaparken bu elemana ulaştığında dizinin sonuna geldiğini anlar.
- Bu metodun dezavantajı dizi sonu belirteci saklamanız ve her dizi elemanı için bunu kontrol etmeniz. Pek kullanışlı değil.

Dizi ve fonksiyonlar

- Kullanışlı olan metod: Dizi pointer ve boyutunu fonksiyon argümanı olarak fonksiyona yollamak.

```

#include <stdio.h>
#define MAX 10
int array[MAX], count;
int buyuk(int x[], int y); // veya: int buyuk(int *x, int y)
main()
{
    for (count = 0; count < MAX; count++) //klavyeden max değerleri gir
    {
        printf("turuncu int olan sayi giriniz: ");
        scanf("%d", &array[count]);
    }
    /* Fonksiyonu çağır ve değeri yazdır. */
    printf("\n\n en buyuk degeri= %d\n", buyuk(array, MAX));
    return 0;
}

```

```

int buyuk (int x[], int y)
{
    int count, enbuyuk = -12000;
    for ( count = 0; count < y; count++)
    {
        if (x[count] > enbuyuk)
            enbuyuk = x[count];
    }
    return enbuyuk;
}

```



```
#include <stdio.h>
#define SIZE 10
int sum(int ar[], int n);
int main(void)
{
    int marbles[SIZE] = {20,10,5,39,4,16,19,26,31,20};
    long answer;
    answer = sum(marbles, SIZE);
    printf("mermer toplami is %ld.\n", answer);
    printf("dizi buyuklugu %u bytes.\n", sizeof marbles);
    return 0;
}
int sum(int ar[], int n)
{
    int i; int total = 0;
    for( i = 0; i < n; i++)
        total += ar[i];
    printf("ar 'in buyuklugu %u bytes.\n", sizeof ar);
    return total;
}
```

```
#include <stdio.h>
#define SIZE 10
int sump(int * start, int * end);
int main(void)
{
int marbles[SIZE] = {20,10,5,39,4,16,19,26,31,20};
long answer;
answer = sump(marbles, marbles + SIZE);
printf("mermer toplami %ld.\n", answer);
return 0;
}
int sump(int * start, int * end)
{
int total = 0;
while (start < end)
{
total += *start;
start++;
}
return total;
}

// total += *start++; Nasıl çalışır
```

```
/* order.c -- precedence in pointer operations */
#include <stdio.h>
int data[2] = {100, 200};
int moredata[2] = {300, 400};
int main(void)
{
    int * p1, * p2, * p3;
    p1 = p2 = data;
    p3 = moredata;
    printf(" *p1 = %d, *p2 = %d, *p3 = %d\n",
           *p1, *p2, *p3);

    printf("*p1++ = %d, *++p2 = %d, (*p3)++ = %d\n",
           *p1++, *++p2, (*p3)++);

    printf(" *p1 = %d, *p2 = %d, *p3 = %d\n",
           *p1, *p2, *p3);

    return 0;
}
```

```
int add_array (int *a, int num_elements);  
int main() {  
    int Tab[5] = {100, 220, 37, 16, 98};  
    printf("Total summation is %d\n", add_array(Tab, 5));  
    return 0;  
}
```

```
int add_array (int *p, int size) {  
    int total = 0;  
    int k;  
    for (k = 0; k < size; k++) {  
        total += p[k]; /* it is equivalent to total +=*p ;p++; */  
    }  
    return (total);  
}
```

```

// ptr_ops.c -- pointer operations
#include <stdio.h>
int main(void)
{
    int urn[5] = {100,200,300,400,500};
    int * ptr1, * ptr2, *ptr3;
    ptr1 = urn;      // assign an address to a pointer
    ptr2 = &urn[2];   // ditto
                    // dereference a pointer and take
                    // the address of a pointer
    printf("pointer value, dereferenced pointer, pointer address:\n");
    printf("ptr1 = %p, *ptr1 = %d, &ptr1 = %p\n",
        ptr1, *ptr1, &ptr1);
    ptr3 = ptr1 + 4; // pointer addition

    printf("\nadding an int to a pointer:\n");
    printf("ptr1 + 4 = %p, *(ptr1 + 3) = %d\n",
        ptr1 + 4, *(ptr1 + 3));
    ptr1++;          // increment a pointer
    printf("\nvalues after ptr1++:\n");
    printf("ptr1 = %p, *ptr1 = %d, &ptr1 = %p\n",
        ptr1, *ptr1, &ptr1);
    ptr2--;          // decrement a pointer
    printf("\nvalues after --ptr2:\n");
    printf("ptr2 = %p, *ptr2 = %d, &ptr2 = %p\n",
        ptr2, *ptr2, &ptr2);
    --ptr1;          // restore to original value
    ++ptr2;          // restore to original value
    printf("\nPointers reset to original values:\n");
    printf("ptr1 = %p, ptr2 = %p\n", ptr1, ptr2);
                    // subtract one pointer from another
    printf("\nsubtracting one pointer from another:\n");
    printf("ptr2 = %p, ptr1 = %p, ptr2 - ptr1 = %d\n",
        ptr2, ptr1, ptr2 - ptr1);
                    // subtract an integer from a pointer
    printf("\nsubtracting an int from a pointer:\n");
    printf("ptr3 = %p, ptr3 - 2 = %p\n",
        ptr3, ptr3 - 2);
}

```

extra

- Eğer dizinin elemanlarının modifiye (değiştirme) edilmesini istemiyorsanız:

Const char aylar[12]={0cak,.....,aralık}

- Read-only dizi oluştu
- Eğer dizi elemanlarını atamazsanız rastgele program hafızadaki değerleri atar.
- Eğer dizinin elemanlarının bir kısmını atarsanız, geri kalanlar 0 olarak atanır.
- Dizi boyutuna uygun counter (sayıcı) ayarlamak sizin göreviniz

Lab Sorusu1

- Aşağıda yıllar ve ayları içeren bir dizi verilmiştir. Bu dizinin içeriği verilen yıl ve ayda düşen yağmur miktarıdır.
- `const float rain[YEARS][MONTHS] =`
`{`
`{4.3,4.3,4.3,3.0,2.0,1.2,0.2,0.2,0.4,2.4,3.5,6.6},`
`{8.5,8.2,1.2,1.6,2.4,0.0,5.2,0.9,0.3,0.9,1.4,7.3},`
`{9.1,8.5,6.7,4.3,2.1,0.8,0.2,0.2,1.1,2.3,6.1,8.4},`
`{7.2,9.9,8.4,3.3,1.2,0.8,0.4,0.0,0.6,1.7,4.3,6.2},`
`{7.6,5.6,3.8,2.8,3.8,0.2,0.0,0.0,0.0,1.3,2.6,5.2}`
`};`

Buna göre

Eğlence

- 1) Yıl.....Yağmur
2000.....?
.....
2004.....?
 - 2) Yıllık averajı
 - 3) Aylık yağmur ortalamasını
- Bulan bir program yazınız.

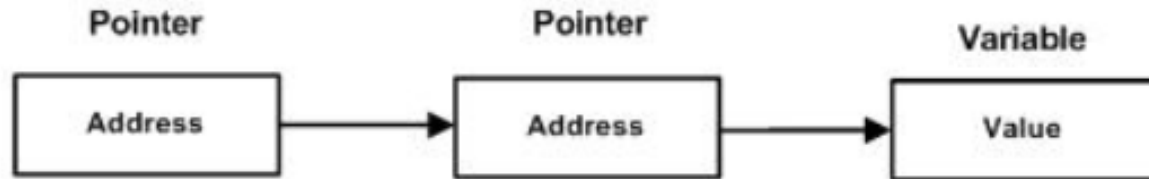
2) 20 elemanlık, elemanları random atanmış bir dizi oluşturun. Bir fonksiyon yazın: fonksiyon dizinin en büyük elemanının indeksini bulup yazsın. İkinci bir fonksiyon dizinin en büyük elemanı ile en küçüğünün farkını bulup yazsın.

```
#include <stdio.h>
#define SIZE 5
void show_array(const double ar[], int n);
void mult_array(double ar[], int n, double mult);
int main(void)
{
    double dip[SIZE] = {20.0, 17.66, 8.2, 15.3, 22.22};
    printf("Orijinal dip dizisi:\n");
    show_array(dip, SIZE);
    mult_array(dip, SIZE, 2.5);
    printf("mult_array() fonksiyonu isletildikten sonra dip  
dizisi=:\n");
    show_array(dip, SIZE);
    return 0;
}
```

```
void show_array(const double ar[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%8.3f ", ar[i]);
    putchar('\n');
}
```

```
void mult_array(double ar[], int n, double mult)
{
    int i;
    for (i = 0; i < n; i++) ar[i] *= mult;
}
```

Pointers to Pointers



```
int *ptr; //pointer deklere et
ptr = &x; //değişkenin adresini ata
x = 12;
*ptr = 12;
int x = 12;
int *ptr = &x;
int **ptr_to_ptr = &ptr; // pointer to pointer
**ptr_to_ptr = 12;
printf("%d", **ptr_to_ptr);
```

Double Pointer



```
#include <stdio.h>

int main(void)
{
    int x, *p, **q;
    x = 10;
    p = &x;
    q = &p;
    printf("%d", **q);    /* print the value of x */
    return 0;
}
```

```
#include <stdio.h>

int main () {
    int var;
    int *ptr;
    int **pptr;
    var = 3000;
    ptr = &var; /* take the address of var */
    /* take the address of ptr using address of operator & */
    pptr = &ptr;
    printf("Value of var = %d\n", var );
    printf("Value available at *ptr = %d\n", *ptr );
    printf("Value available at **pptr = %d\n", **pptr);
    return 0;
}
```

```
void alloc2(int** p) {  
    *p = (int*)malloc(sizeof(int));  
    **p = 10;  
}  
  
void alloc1(int* p) {  
    p = (int*)malloc(sizeof(int));  
    *p = 10;  
}
```

```
int main(){  
    int *p = NULL;  
    alloc1(p);  
    //printf("%d ",*p);//undefined  
    alloc2(&p);  
    printf("%d ",*p);//will print 10  
    free(p);  
    return 0;  
}
```

The reason it occurs like this is that in alloc1 the pointer is passed in by value. So, when it is reassigned to the result of the malloc call inside of alloc1, the change does not pertain to code in a different scope.

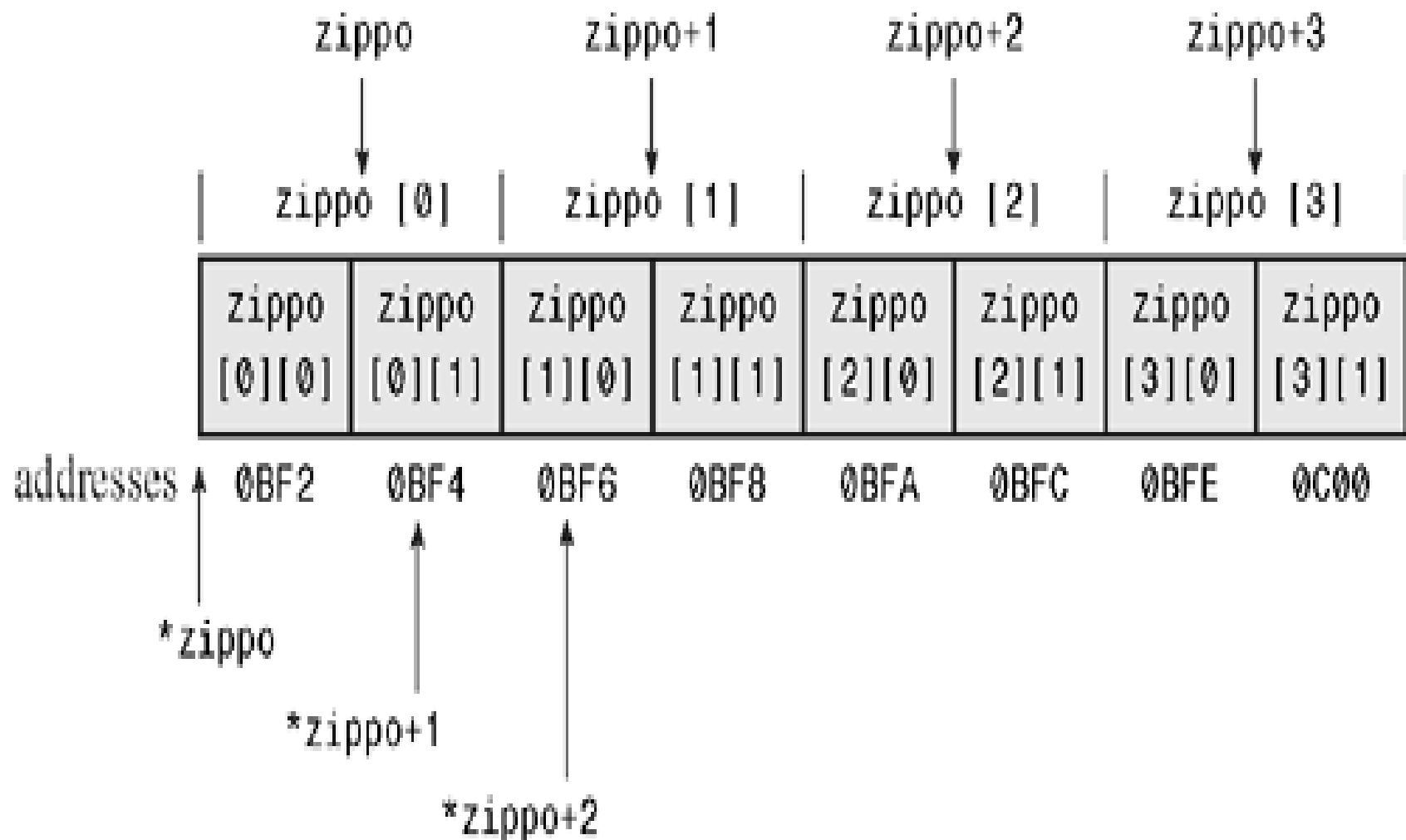

```
void func(int **pointer)  
{  
    ...  
}
```

```
int main(void)  
{  
    int *pointer;  
    func(&pointer);  
    ...  
}
```

```
void foo( char ** ptr)  
{  
    *ptr = malloc(255); // allocate some memory  
    strcpy( *ptr, "Hello World");  
}  
int main()  
{  
    char *ptr = 0;  
    // call function with a pointer to pointer  
    foo( &ptr );  
    printf("%s\n", ptr);  
    // free up the memory  
    free(ptr);  
    return 0;  
}
```

Çok Boyutlu Dizi ve Pointer

- `int zippo[4][2]; /* int türünde iki boyutlu dizi */`
zippo dizinin ismi ve dizinin ilk elemanının adresi dir.
- `zippo` int türü iki elemanlı bir dizinin adresidir.
- `zippo=&zippo[0]`
- `zippo[0]` kendisi int türü bir elemanlı dizinin adresi
- `zippo[0]=&zippo[0][0]`



- Pointere veya adrese 1 eklemek işaret edilen tür kadar bir artırımını ifade eder.
- Zippo ve zippo[0] bu bağlamda farklıdırlar: zippo iki int türü bir elemanı işaret ederken, zippo[0] ise int türü bir elemanı işaret eder.
- Zippo+1 ve zippo[0]+1 farklı değerleri gösterir.
- Pointerin gösterdiği değişkene ulaşmak için “ * ” operatörü kullanılır.
- Zippo[0] dizinin ilk elemanı zippo[0][0] değişkeninin adresidir ve *(zippo[0][0]) ise zippo[0][0] adresindeki değişkeni ifade eder.
- *zippo ise ilk elemanın kendisini ifade eder bu da zippo[0] olur.
- Zippo[0] zaten kendisi bir adrestir. (zippo[0][0] ın adresi) Bu durumda *zippo=&zippo[0][0].
- Bu durumda dereference işlemi iki kez yapılmalıdır.
- **zippo==*& zippo[0][0]= zippo[0][0]
- Kısaca: zippo bir adresin adresidir ve iki kez * kullanılmalıdır.

```
#include <stdio.h>

int main(void)
{
int zippo[4][2] = { {2,4}, {6,8}, {1,3}, {5, 7} };
printf(" zippo = %p, zippo + 1 = %p\n", zippo, zippo + 1);

printf("zippo[0] = %p, zippo[0] + 1 = %p\n", zippo[0], zippo[0] + 1);

printf(" *zippo = %p, *zippo + 1 = %p\n", *zippo, *zippo + 1);

printf("zippo[0][0] = %d\n", zippo[0][0]); printf(" *zippo[0] = %d\n",
    *zippo[0]);

printf(" **zippo = %d\n", **zippo);

printf(" zippo[2][1] = %d\n", zippo[2][1]); printf("*(*(zippo+2) + 1) =
    %d\n", *(*(zippo+2) + 1));
return 0;
}
```

- `zippo`

İlk iki-int elemanın adresi

- `zippo+2`

üçüncü iki-int elemanın adresi

- `*(zippo+2)`

üçüncü eleman,iki-int eleman, ilkinin adresi

- `*(zippo+2) + 1`

ikinci elemanın adresi (dizide üçüncü eleman)

- `*(*(zippo+2) + 1)`

(dizide üçüncü elemanın) ikinci değişkenin kendisi

`(zippo[2][1])`

Pointer ve çok boyutlu dizi

- `İnt *ptr_say1; // tek nümerik değer,`
- `İnt *ptr_dizi[2]; // iki elemanlı bir dizi için pointer kullanımı`
- `İnt *(ptr_dizi)[2]; // iki boyutlu bir dizi için pointer kullanımı`


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

```
int main(void)
```

```
{
```

```
int zippo[4][2] = { {2,4}, {6,8}, {1,3}, {5, 7} };
```

```
int (*pz)[2];
```

```
pz = zippo;
```

```
printf(" pz = %p, pz + 1 = %p\n", pz, pz + 1);
```

```
printf("pz[0] = %p, pz[0] + 1 = %p\n", pz[0], pz[0] +  
1);
```

```
printf(" *pz = %p, *pz + 1 = %p\n", *pz, *pz + 1);
```

```
printf("pz[0][0] = %d\n", pz[0][0]);
```

```
printf(" *pz[0] = %d\n", *pz[0]);
```

```
printf(" **pz = %d\n", **pz);
```

```
printf(" pz[2][1] = %d\n", pz[2][1]);
```

```
printf("*(*(pz+2) + 1) = %d\n", *(*(pz+2) + 1));
```

```
return 0;
```

```
}
```

Sorular, sınava dikkat

- `*ptr` ve `*(ptr + 2)` değerleri nedir?
`int *ptr;`
`int torf[2][2] = {12, 14, 16};`
`ptr = torf[0];`
- `**ptr` ve `** (ptr + 1)` değerleri nedir?
`int (*ptr)[2];`
`int torf[2][2] = {12, 14, 16};`
`ptr = torf;`

Pointers to Function

type (*ptr_to_func)(parameter_list);

int (*func1)(int x);

void (*func2)(double y, double z);

char (*func3)(char *p[]);

void (*func4)();

Initializing:

float square(float x);

/* The function prototype. */

float (*p)(float x);

/* The pointer declaration. */

float square(float x)

/* The function definition. */

{

return x * x;

}

p = square; */*initialize*/*

answer = p(x); */*call the function*/*

```
#include <stdio.h>  
  
double square(double x);    /* The function prototype. */  
double (*p)(double x);    /* The pointer declaration. */  
  
main() {  
    p = square;            /* Initialize p to point to square(). */  
/* Call square() two ways. */  
    printf("%f %f\n", square(6.6), p(6.6));  
    return(0);  
}
```

```
double square(double x) {  
    return x * x;  
}
```

- A function name without parentheses is a pointer to the function

```
int sum (int num1, int num2){
    return num1+num2;
}

int main(){
int (*f2p) (int, int);
    f2p = sum;

    //Calling function using function pointer
    int op1 = f2p(10, 13);

    //Calling function in normal way using function name
    int op2 = sum(10, 13);

    printf("Output1: Call using function pointer: %d",op1);
    printf("\nOutput2: Call using function name: %d", op2);

    return 0;
}
```

```
void fun(int a)  
{  
    printf("Value of a is %d\n", a);  
}
```

```
int main()  
{  
    void (*fun_ptr)(int) = fun;  
  
    fun_ptr(10);  
  
    return 0;  
}
```

/*Using a pointer to a function to call different functions*/

#include <stdio.h>

void func1(int x); /* The function prototypes. */

void one(void);

void two(void);

void other(void);

main() {

int a;

for (;;)

{

puts("\nEnter an integer between 1 and 10, 0 to exit: ");

scanf("%d", &a);

if (a == 0)

break;

func1(a);

}

return(0);

}

```
void func1(int x)
{
    void (*ptr)(void);           /* The pointer to function. */
    if (x == 1)
        ptr = one;
    else if (x == 2)
        ptr = two;
    else
        ptr = other;
    ptr();
}

void one(void) {
    puts("You entered 1.");
}

void two(void) {
    puts("You entered 2.");
}

void other(void) {
    puts("You entered something other than 1 or 2.");
}
```



```
#include <stdio.h>

void add(int a, int b){
    printf("Addition is %d\n", a+b);
}

    void subtract(int a, int b) {
        printf("Subtraction is %d\n", a-b);
    }

        void multiply(int a, int b) {
            printf("Multiplication is %d\n", a*b);
        }

int main() {
    // fun_ptr_arr is an array of function pointers
    void (*fun_ptr_arr[])(int, int) = {add, subtract, multiply};
    unsigned int ch, a = 15, b = 10;
    printf("Enter Choice: 0 for add, 1 for subtract and 2 for multiply\n");
    scanf("%d", &ch);
    if (ch > 2) return 0;
    (*fun_ptr_arr[ch])(a, b);
    return 0;
}
```

- `int *func(int, int);` // this function returns a pointer to int
- `double *func(int, int);` // this function returns a pointer to double

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

```
int *return_pointer(int *, int); // this function returns a pointer of type int
```

```
int main(){
```

```
    int i, *ptr;
```

```
    int arr[] = {11, 22, 33, 44, 55};
```

```
    i = 4;
```

```
    printf("Address of arr = %u\n", arr);
```

```
    ptr = return_pointer(arr, i);
```

```
    printf("\nAfter incrementing arr by 4 \n\n");
```

```
    printf("Address of ptr = %u\n\n" , ptr);
```

```
    printf("Value at %u is %d\n", ptr, *ptr);
```

```
return 0;
```

```
}
```

```
int *return_pointer(int *p, int n){
```

```
    p = p + n;
```

```
    return p;
```

```
}
```

1	Address of arr = 2686736
2	
3	After incrementing arr by 4
4	
5	Address of ptr = 2686752
6	
7	Value at 2686752 is 55

Pointer declaration	Description
<code>int *x</code>	x is a pointer to int data type.
<code>int *x[10]</code>	x is an array[10] of pointer to int data type.
<code>int *(x[10])</code>	x is an array[10] of pointer to int data type.
<code>int **x</code>	x is a pointer to a pointer to an int data type – double pointers.
<code>int (*x)[10]</code>	x is a pointer to an array[10] of int data type.
<code>int *funct()</code>	funct is a function returning an integer pointer.
<code>int (*funct)()</code>	funct is a pointer to a function returning int data type – quite familiar constructs.
<code>int (*(funct())[10])()</code>	funct is a function returning pointer to an array[10] of pointers to functions returning int.
<code>int ((*x[4])())[5]</code>	x is an array[4] of pointers to functions returning pointers to array[5] of int.

Eğlence

- Float türünde bir dizi tanımlayın (5 elemanlı olabilir). Bu dizinin elemanlarını iki farklı diziye kopyalamak için iki fonksiyon yazınız. Birinci fonksiyon dizi ismini ikinci fonksiyon pointer kullanarak çalışsınlar.
- Bir fonksiyon yazınız ve bu fonksiyon bir dizinin en büyük elemanın indexini geri döndürsün

```
#include <stdio.h>
#define ROWS 3
#define COLS 4
void sum_rows(int ar[][COLS], int rows);
void sum_cols(int[][COLS], int );
int sum2d(int (*ar)[COLS], int rows);
int main(void)
{
int junk[ROWS][COLS] = { {2,4,6,8}, {3,5,7,9}, {12,10,8,6} };
sum_rows(junk, ROWS);
sum_cols(junk, ROWS);
printf("Sum of all elements = %d\n", sum2d(junk, ROWS));
return 0;
}
```

```
void sum_rows(int ar[][COLS], int rows)
{
    int r;
    int c;
    int tot;
    for (r = 0; r < rows; r++)
    {
        tot = 0;
        for (c = 0; c < COLS; c++)
            tot += ar[r][c];
        printf("row %d: sum = %d\n", r, tot);
    }
}
```

```
void sum_cols(int ar[][COLS], int rows)
{
    int r;
int c;
    int tot;
    for (c = 0; c < COLS; c++)
    {
        tot = 0;
for (r = 0; r < rows; r++)
        tot += ar[r][c];
        printf("col %d: sum = %d\n", c, tot);
    }
}
```

```
int sum2d(int ar[][COLS], int rows)
{
    int r;
    int c;
    int tot = 0;
    for (r = 0; r < rows; r++)
        for (c = 0; c < COLS; c++)
            tot += ar[r][c];
    return tot;
}
```


Karakter ve karakter dizileri (strings)

- `char a, b, c; /* char değişkeni deklere et */`
 - `char code = 'x'; /* code isimli char değişkeni deklere et ve ilk değerini ata, x değeri code değişkeninin saklandı */`
- `code = '!'; /* ! değerini code isimli değişkende depola */`
- `#define EX 'x'`
- `char code = EX; /* code eşittir 'x' */`
- `const char A = 'Z';`

Karakter serisi, dizisi

- `char string[10];` // 10 elemanlı char türünde bir dizi
//deklere eder, yalnız 9 karakter
//tutar?! (çünkü string `\0` =null
//karakterleri ile sonlandırılır, compiler
//tarafından.)

`char string[10]={‘A’, ‘L’, ‘P’, ‘A’, ‘S’, ‘L’, ‘A’, ‘N’, ‘\0’};`

`char string[10] = “ALPASLAN”;`

`char string[] = “ALPASLAN”;`

Strings ve Pointers

`char *mesaj;` // char türü için bir pointer deklarasyonu

`char *mesaj = "Ali'nin hayaleti!";`
`//Hafızada bir yerlerde bu mesaj kaydedildi ve biz`
`//mesajın ilk harfinin adresini biliyoruz.`
`char mesaj[] = "Ali'nin hayaleti!";`

Aynı şey?

`Char *fruit[3];` pointer dizisi

A	p	p	l	e	\0	\0
---	---	---	---	---	----	----

P	e	a	r	\0	\0	\0
---	---	---	---	----	----	----

O	r	a	n	g	e	\0
---	---	---	---	---	---	----

```
char fruit[3][7]=  
{"Apple",  
"Pear",  
"Orange"  
};
```

deklerasyon farkı

A	p	p	l	e	\0
---	---	---	---	---	----

P	e	a	r	\0
---	---	---	---	----

O	r	a	n	g	e	\0
---	---	---	---	---	---	----

```
char* fruit[3]=  
{"Apple",  
"Pear",  
"Orange"  
};
```

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

```
int main(void){
```

```
    const char * mesg = "Deli olma!";
```

```
    const char * copy;
```

```
    copy = mesg;
```

```
    printf("%s\n", copy);
```

```
    printf("mesg = %s; &mesg = %p; degeri = %p\n",
```

```
        mesg, &mesg, mesg);
```

```
    printf("copy = %s; &copy = %p; degeri = %p\n",
```

```
        copy, &copy, copy);
```

```
    return 0;
```

```
}
```

String girişi

`char *name;`

`scanf("%s", name);` // isim girişi yapılır, fakat kaç harften oluştuğu belli değil, name pointer'ı ile gösterilen yazılır. İlk harften başlar ilk boşluğa kadar okur.

`scanf("%ns", name);` // İlk n karakteri okur.

`scanf("%s%s%s", s1, s2, s3);` // birden fazla giriş olabilir.

`char name[81];` // şimdi sınırladık ve bizim tanımladığımız diziye yazılır isim.

.

```
/* Demonstrates using scanf() to input numeric and text data. */  
#include <stdio.h>  
char lname[81], fname[81];  
int count, id_num;  
main()  
{  
    puts("Enter last name, first name, ID number separated");  
    puts("by spaces, then press Enter.");  
    /* Input the three data items. */  
    count = scanf("%s%s%d", lname, fname, &id_num);  
    /* Display the data. */  
    printf("%d items entered: %s %s %d \n", count, fname, lname,  
        id_num);  
    return 0;  
}
```

gets()

Enter tuşuna basılıncaya kadar olan bütün karakterleri alır, sonuna null ekler

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

```
#define MAX 81
```

```
int main(void)
```

```
{
```

```
    char name[MAX];
```

```
    printf("Merhaba isminiz nedir?\n");
```

```
    gets(name);
```

```
    printf("Ne guzel isim, %s.\n", name);
```

```
    return 0;
```

```
}
```


**//gets() kelimelerin depolandığı yerin adresini geri
//döndürür.**

#include <stdio.h>

#define MAX 81

int main(void)

{

char name[MAX];

char * ptr;

printf("Merhaba isminiz nedir?\n");

ptr = gets(name);

printf("%s? ne isim ama! %s!\n", name, ptr);

return 0;

}

while (gets(name) != NULL)

// eğer her şey yolunda ise gets() fonksiyonu girişi okur ve name adresine kaydeder. Bu adres return ile geri döndürülür. File sonu ise veya okunacak bir şey yoksa null pointer geri döndürülür. (NULL karakteri, boşluk, girilinceye kadar while döngüsü içinde kalırsınız.)

Gets() fonksiyonunun dezavantajı string büyüklüğünü kontrol etmeyişiştir. Yeterli alan var mı bilemez. (ekstra karakterler bitişikteki hafıza ünitelerine yazılır, kontrolümüz dışında.)

```
#include <stdio.h>  
char input[81], *ptr;
```

```
main()  
{
```

```
    puts("satir satir yazinizi giriniz, Enter\' a basiniz.");  
    puts("bitirdiginizde bos satir birakiniz.");
```

```
    while ( *(ptr = gets(input)) != NULL)  
        printf("girisiniz: %s\n", input);
```

```
    puts("herhalde bitti byeeee\n");
```

```
    return 0;  
}
```

fgets()

- Kaç karakter alınacağı bildirilir: \0 karakterini okur ve nereden okuma yapılacağı bildirilir.(stdin (standard input) klavye girişi için)

```
#include <stdio.h>
#define MAX 81
int main(void)
{
    char name[MAX];
    char * ptr;

    printf("Merhaba isminiz nedir?\n");
    ptr = fgets(name, MAX, stdin);

    printf("%s ne isim ama! %s\n", name, ptr);
    return 0;
}
```

scanf()

Scanf() daha çok kelime girişi için kullanılır (giriş boşluk, tab, yeni satır olduğunda sonlanır). %s ile kullanılır.

Alınacak karakter sayısı sınırlandırılabilir. (%10s= sadece 10 karakter al.)

Gets() ise enter e basılıncaya kadar yazılanları alır.

```
#include <stdio.h>
int main(void)
{
    char name1[11], name2[11];

    int count;
    printf("iki isim giriniz.\n");

    count = scanf("%5s %10s",name1, name2);

    printf(" %d isim girildi bunlar:%s ve %s.\n", count, name1, name2);

    return 0;
}
```

String çıkışı

puts()

```
#include <stdio.h>
#define DEF "#define ile tanımlanmış string"
int main(void)
{
    char str1[80] = "dizi ile verilen string.";
    const char * str2 = "pointer ile verileni.";
    puts("Buraya ne yazarsan ekrana yazarım abi.");
    puts(DEF);
    puts(str1);
    puts(str2);
    puts(&str1[5]);
    puts(str2+4);
    return 0;
}
```



```
#include <stdio.h>
char *message1 = "C";
char *message2 = "is the";
char *message3 = "best";
char *message4 = "programming";
char *message5 = "language!!";
main()
{
    puts(message1);
    puts(message2);
    puts(message3);
    puts(message4);
    puts(message5);
    return 0;
}
```

eğlence

- Kullanıcıdan satırlar halinde yazı girişi alan ve girişleri satırların ilk kelimelerini dikkate alarak alfabetik olarak sıraya koyup yazan bir program yazınız.

Ek Uygulamalar:

Kendisine gönderilen bir sayının faktoriyelini hesaplayana ve bu değeri parametre olarak gönderilen adrese kopyalayan program

```
void factorial(int n, long *p);
#include <stdio.h>
int main()
{
    long a;
    factorial(7, &a);
    printf ("%d! = %ld", 7, a);
    return 0;
}

void factorial(int n, long *p);
{
    if (n == 0 || n == 1)
        *p = 1;
    for (*p = 1; n > 0; n--)
        *p *= n;
}
```

- n elemanlı **int** türden bir dizinin aritmetik ortalamasını bul

```
#include <stdio.h>
double getavg(int *p, int size);
int main()
{
    int s[10] = {1, 23, 45, -4, 67, 12, 22, 90, -3, 44};
    double average;
    average = getavg(s, 10);
    printf("dizinin aritmetik ortalaması = %lf\n", average);
    return 0;
}
double getavg(int *p, int size)
{
    int i;
    double total = 0;
    for (i = 0; i < size; ++i)
        total += p[i];
    return total / size;
}
```

- Bir dizinin en büyük elemanını bulup bu elemanın değerini döndüren

```
#include <stdio.h>
int *getmax(int *p, int size);
int main()
{
    int s[10] = {1, 23, 45, -4, 67, 12, 22, 90, -3, 44};
    printf("%d\n", *getmax(s, 10));
    return 0;
}
int *getmax(int *p, int size)
{
    int *pmax, i;
    pmax = p;
    for (i = 1; i < size; ++i)
        if (*pmax < p[i])
            pmax = p + i;
    return pmax;
}
```


Structures

- Birden fazla değişkenin, daha kolay manipülasyon için, bir araya getirildiği yapılara denir. Bu değişkenler (dizilerdekilerin aksine) farklı veri türlerinden olabilirler.
- Her C veri türü içerebilir, başka structure da, dizi de
- Structure içindeki her değişkene üye (member) denir.

Structures tanım ve deklarasyon

- Bir grafik programı için x ve y değişkenleri kullanacaksınız.
- Structure tanımı:

```
struct coord {  
                                int x;  
                                int y;  
                                };
```

Struct: structure tanımı başlıyor

Coord: structure türü ismi

{ } içinde değişkenler

Structures tanım ve deklarasyon

- Deklarasyon:

```
struct coord {  
    int x;  
    int y;  
} first, second;
```

Structure türü **coord** tanımlandı ve iki structure (coord türünde) **first ve second** olarak tanımlandı.

Structures tanım ve deklarasyon

Structure tanımı önce yapıp, daha sonra başka bir kod alanında structure deklarasyonu yapılabilir:

```
struct coord {  
    int x;  
    int y;  
};
```

```
/* diğer kod parçaları */
```

```
struct coord first, second;
```

Strucure elemanlarına nasıl ulaşırız?

- Structure değişkenlerine üye operatörü ile ulaşılır: “.”

`first.x = 10;`

`first.y = 30;`

Nasıl print ederiz, ikinci deklarasyonu kullanarak:

`printf("%d,%d", second.x, second.y);`

- Nasıl kopyalarız

`first = second;`

Aşağıdaki işleme eşittir

`first.x = second.x;`

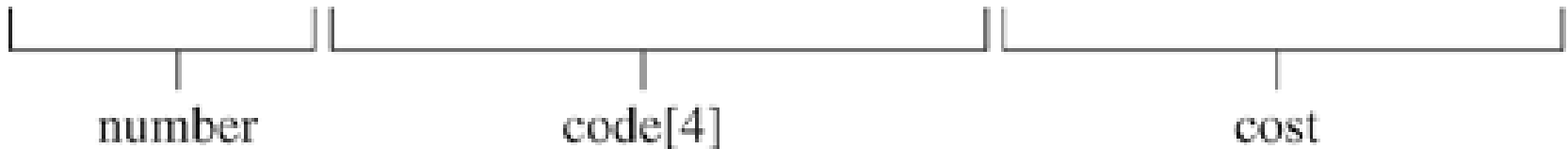
`first.y = second.y;`

Hafıza durumu

```
struct stuff {  
    int number;  
    char code[4];  
    float cost;  
};
```



code[0] - - - - - code[3]



Nasıl ilk değer ataması yapılır

- Hatırlayınız

```
int count = 0;
```

```
int dizi[7] = {0,1,1,2,3,5,8};
```

```
struct kitap {  
    char isim[MAXISIM];  
    char yazar[MAXYAZAR];  
    float eder;  
} kutuphane;
```

```
struct kitap kutuphane = {  
    "C ogreniyorum",  
    "alp",  
    5.00  
};
```

Nasıl ilk değer ataması yapılır

```
struct kitap kutuphane = {  
    .eder = 5.00,  
    .yazar = "alp",  
    .isim = "Zor ogrenirsiniz"  
};
```

Structure dizileri

- ***Bu kadar kolay olacağını sanmadınız herhalde***
- Bir structure 3-5 değişken tutar, ama biz structure dizisi oluşturup yüzlerce değişkene sahip olabiliriz.
- Nasıl?

struct kitap kutuphane[500];

- 500 tane kitap türünde kütüphane structure tanımlandı. Kutuphane[0] birinci kitap structure, Kutuphane[1] ikinci kitap structure
- **Kutuphane[7].yazar="alpaslan";**
- **Kutuphane[250].eder=12.99; //YTL**

```
#include <stdio.h>
#include <stdio.h>
#define MAXISIM 40
#define MAXYAZAR 40
#define MAXKITAP 100
struct kitap {
    char isim[MAXISIM];
    char yazar[MAXYAZAR];
    float eder;
};
int main(void)
{
    struct kitap kutuphane[MAXKITAP];
    int count = 0;
    int index;
    printf("Lutfen kitap ismini giriniz.\n");
    printf("cikis icin satir basinda enter'e basiniz.\n");
    while (count < MAXKITAP && gets(kutuphane[count].isim) !=
        NULL
            && kutuphane[count].isim[0] != '\0')
    {
        printf("Yazar ismini girin.\n");
```



```
gets(kutuphane[count].yazar);
    printf("kitap ederini, degerini,fiyatini, girin.\n");
    scanf("%f", &kutuphane[count++].eder);
    while (getchar() != '\n')
        continue;      /* giriş sistemini temizle */
    if (count < MAXKITAP)
        printf("Bir sonraki kitap ismini girin.\n");
}
if (count > 0)
{
    printf("Iste kitaplarin listesi:\n");
    for (index = 0; index < count; index++)
        printf("%s by %s: $%.2f\n", kutuphane[index].isim,
            kutuphane[index].yazar, kutuphane[index].eder);
}
else
    printf("Hic kitap yok mu, ne kotu.\n");
return 0;
}
```

Structure içinde structure

```
#include <stdio.h>
#define LEN 20
const char * msgs[5] =
{
    "    Bu guzel akşam için teşekkürler, ",
    "sunu ispatladın ki sen ",
    "özel birisin. Böyle günleri tekrar tekrar yapmalıyız",
    "yemekler bir harikaydı ",
    "memnun oldum, çok, hemde çok"
};

    struct isim {
        char ilk[LEN];
        char soyad[LEN];
    };

        struct kisi {
            struct isim ad;
            char favoriyemek[LEN];
            char isi[LEN];
            float geliri;
        };
```

```
int main(void)
{
    struct kisi uye = {
        { "alp", "duysak" },
        "tarhana corbasi",
        "Universite hocasi",
        16000.00
    };
    printf("Sayin %s, \n\n", uye.ad.ilk);
    printf("%s%s.\n", msgs[0], uye.ad.ilk);
    printf("%s%s\n", msgs[1], uye.isi);
    printf("%s\n", msgs[2]);
    printf("%s%s%s", msgs[3], uye.favoriyemek, msgs[4]);
        if (uye.geliri > 75000.0)
            puts("!");
        else
            puts(".");
    printf("\n%40s%s\n", " ", "Gorusmek dilegiyle,");
        printf("%40s%s\n", " ", "Byeee");
    return 0;
}
```

```
#include <stdio.h>
int length, width;
long area;
    struct coord{
        int x;
        int y;
    };
        struct rectangle{
            struct coord topleft;
            struct coord bottomrt;
        } mybox;
main()
{
    printf("\nEnter the top left x coordinate: ");
    scanf("%d", &mybox.topleft.x);
    printf("\nEnter the top left y coordinate: ");
    scanf("%d", &mybox.topleft.y);
    printf("\nEnter the bottom right x coordinate: ");
    scanf("%d", &mybox.bottomrt.x);
    printf("\nEnter the bottom right y coordinate: ");
    scanf("%d", &mybox.bottomrt.y);
        width = mybox.bottomrt.x - mybox.topleft.x;
        length = mybox.bottomrt.y - mybox.topleft.y;
        area = width * length;
        printf("\nThe area is %ld units.\n", area);
    return 0;
}
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

struct giris {
    char adi[20];
    char soyadi[20];
    char tel[10];
};

    struct giris liste[4];
    int i;

main()
{
    for (i = 0; i < 4; i++)
    {
        printf("\nllk adi giriniz: ");
        scanf("%s", liste[i].adi);
        printf("Soy adi giriniz: ");
        scanf("%s", liste[i].soyadi);
        printf("telefonu 555-6666 formatinda giriniz: ");
        scanf("%s", liste[i].tel);
    }

    printf("\n\n");
    for (i = 0; i < 4; i++)
    {
        printf("Adi: %s %s", liste[i].adi, liste[i].soyadi);
        printf("\t\tTelefonu: %s\n", liste[i].tel);
    }

    return 0;
}
```

İç içe yapılarda değer atama

```
struct musteri {  
    char firma[20];  
    char kontak[25];  
}  
struct satis {  
    struct musteri alici;  
    char mal[20];  
    float miktar;  
} benimsatisim = {  
    { "DPU", "alpaslan"},  
    "Ogrenciler",  
    30 adet  
};
```

Structures ve Pointers

```
struct giris {  
    char adi[20];  
    char soyadi[20];  
    char tel[10];  
};
```

```
    struct giris liste[4];
```

```
main()  
{
```

```
.....
```

```
    struct giris *personel;  
    personel=&liste[0];
```

```
.....
```

```
        (*personel).adi;  
        personel->adi
```

```
#include <stdio.h>
struct giris {
    char adi[20];
    char soyadi[20];
    char tel[10];
};

    struct giris liste[4];

int i;
main() {
    struct giris *personel;
    personel=&liste[0];
    for (i = 0; i < 4; i++) {
        printf("\nllk adi giriniz: ");
        scanf("%s", liste[i].adi);
        printf("Soy adi giriniz: ");
        scanf("%s", liste[i].soyadi);
        printf("telefonu 555-6666 formatinda giriniz: ");
        scanf("%s", liste[i].tel);
    }
    printf("\n\n");
    for (i = 0; i < 4; i++) {
        printf("adi:%s  %s",personel->adi,personel->soyadi);
        printf("\t\tTelefonu: %s\n", (*personel).tel);
        personel++;
    }

    return 0;
}
```


Structures-Functions-Pointers

- Structure üyeleri diğer değişkenler gibi fonksiyon argumanları olabilirler. Fonksiyon bunların structure üyeleri olduğunu bilmek zorunda değil.

```
int sum(int a, int b);
```

```
Struct sayilar{
```

```
    int x;
```

```
    int y;
```

```
    } top;
```

```
main()
```

```
{
```

```
.....
```

```
c=sum(top.x,top.y)
```

```
.....
```

```
#include <stdio.h>
#define BOY 50
    struct banka {
        char  bank[BOY];
        double bank_depozit;
        char  save[BOY];
        double save_depozit;
    };
double sum(double, double); // üyeler argüman
int main(void) {
    struct banka alp = {
        "Ver-alma Bank",
        10.500,
        "gecmis olsun",
        12.500
    };
    printf("Alpaslanin toplam %f YTL si var\n",
        sum(alp. bank_depozit, alp.save_depozit));
    return 0;
}
double sum(double x, double y) {
    return(x+y);
}
```

```
#include <stdio.h>
#define BOY 50
    struct banka {
        char  bank[BOY];
        double bank_depozit;
        char  save[BOY];
        double save_depozit;
    };
double sum(const struct banka *); /* argüman bir pointer */
int main(void) {
    struct banka alp = {
        "Ver-alma Bank",
        10.500,
        "gecmis olsun",
        12.500
    };
    printf("Alpaslanin toplam %f YTL si var\n", sum(&alp));
    return 0;
}

double sum(const struct banka * para) {
    return(para->bank_depozit + para->save_depozit);
}
```

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

```
#define BOY 50
```

```
struct banka {  
    char  bank[BOY];  
    double bank_depozit;  
    char  save[BOY];  
    double save_depozit;  
};
```

```
double sum(const struct banka kendisi); // structere kedisi gidiyor
```

```
int main(void) {  
    struct banka alp = {  
        "Ver-alma Bank",  
        10.500,  
        "gecmis olsun",  
        12.500  
    };  
    printf("Alpaslanin toplam %f YTL si var\n", sum(alp));  
    return 0;  
}
```

```
double sum(const struct banka kendisi) {  
    return(kendisi.bank_depozit + kendisi.save_depozit);  
}
```

```
#include <stdio.h>

struct veri{
    float miktar;
    char ilk_ismi[30];
    char soy_adi[30];
} toplam;

void print_toplam(struct veri x);

main() {
    printf("bagis yapanin ilk ismi ve soyadi,\n");
    printf("aralarinda bosluk birakarak : ");
    scanf("%s %s", toplam.ilk_ismi, toplam.soy_adi);
    printf("\nVerilen miktar: ");
    scanf("%f", &toplam.miktar);
    print_toplam( toplam );
    return 0;
}

void print_toplam(struct veri x) {
    printf("\nVeren %s %s, miktar $%f.\n", x.ilk_ismi, x.soy_adi,
x.miktar);
}
```

```
#include <stdio.h>
struct giris {
    char adi[20];
    char soyadi[20];
    char tel[10];
};

    struct giris liste[4];
    int i;
    void yaz(const struct giris liste[]);
main() {
    for (i = 0; i < 4; i++) {
        printf("\nllk adi giriniz: ");
        scanf("%s", liste[i].adi);
        printf("Soy adi giriniz: ");
        scanf("%s", liste[i].soyadi);
        printf("telefonu 555-6666 formatinda giriniz: ");
        scanf("%s", liste[i].tel);
    }
    yaz(liste);
    return 0;
}//////////Foksiyon//////////
void yaz(const struct giris liste[]) {
    printf("\n\n");
    for (i = 0; i < 4; i++) {
        printf("Adi: %s %s", liste[i].adi, liste[i].soyadi);
        printf("\t\tTelefonu: %s\n", liste[i].tel);
    }
}
```

```
#include <stdio.h>
struct giris {
    char adi[20];
    char soyadi[20];
    char tel[10];
};

    struct giris liste[4];
    int i;
    void olustur(struct giris liste[]);
    void yaz(const struct giris liste[]);

main() {
    olustur(liste);
    yaz(liste);
    return 0;
}//////////Fonksiyon//////////

void olustur(struct giris liste[]) {
    for (i = 0; i < 4; i++) {
        printf("\nllk adi giriniz: ");
        scanf("%s", liste[i].adi);
        printf("Soy adi giriniz: ");
        scanf("%s", liste[i].soyadi);
        printf("telefonu 555-6666 formatinda giriniz: ");
        scanf("%s", liste[i].tel);
    }
}

void yaz(const struct giris liste[]) {
    printf("\n\n");
    for (i = 0; i < 4; i++) {
        printf("Adi: %s %s", liste[i].adi, liste[i].soyadi);
        printf("\t\tTelefonu: %s\n", liste[i].tel);
    }
}
```

Çoklu kaynak dosyası ,le çalışma den1.h

```
struct giris {  
char adi[20];  
char soyadi[20];  
char tel[10];  
};  
  
    struct giris liste[4];  
    int i;  
    void olustur(struct giris liste[]);  
    void yaz(const struct giris liste[]);
```


Den1main.c

```
#include "den1.h"  
main() {  
    olustur(liste);  
    yaz(liste);  
    return 0;  
}
```

Olustur.c

```
#include "den1.h"

void olustur(struct giris liste[]) {
    for (int i = 0; i < 4; i++)    {
        printf("\nilk adi giriniz: ");
        scanf("%s", liste[i].adi);
        printf("Soy adi giriniz: ");
        scanf("%s", liste[i].soyadi);
        printf("telefonu 555-6666 formatinda
giriniz: ");
        scanf("%s", liste[i].tel);
    }
```

Yaz.c

```
#include "den1.h"

void yaz(const struct giris liste[]) {
    printf("\n\n");
    for (int i = 0; i < 4; i++) {
        printf("Adi: %s %s", liste[i].adi,
liste[i].soyadi);
        printf("\t\tTelefonu: %s\n",
liste[i].tel);
    }
}
```

Structure üyesi Pointers

```
Struct veri{  
    İnt *ptrx;  
    İnt *ptry;  
}Kordinat;
```

```
Kordinat.ptrx=&xdeğişkeni;  
Kordinat.ptry=&ydeğişkeni;
```

.....

```
*Kordinat.ptrx= xdeğişkeni  
*Kordinat.ptry= ydeğişkeni
```

Dosya Kullanımı

- FILE *fopen(const char **filename*, const char **mode*);

<i>mode</i>	Meaning
r	Opens the file for reading. If the file doesn't exist, fopen() returns NULL.
w	Opens the file for writing. If a file of the specified name doesn't exist, it is created. If a file of the specified name does exist, it is deleted without warning, and a new, empty file is created.
a	Opens the file for appending. If a file of the specified name doesn't exist, it is created. If the file does exist, new data is appended to the end of the file.
r+	Opens the file for reading and writing. If a file of the specified name doesn't exist, it is created. If the file does exist, new data is added to the beginning of the file, overwriting existing data.
w+	Opens the file for reading and writing. If a file of the specified name doesn't exist, it is created. If the file does exist, it is overwritten.
a+	Opens a file for reading and appending. If a file of the specified name doesn't exist, it is created. If the file does exist, new data is appended to the end of the file.

```
#include <stdlib.h>
#include <stdio.h>
main()
{
    FILE *fp;
    char filename[40], mode[4];
    while (1)
    {
        printf("\nDosya adini giriniz: ");
        gets(filename);
        printf("\nEnter a mode (max 3 karekter): ");
        gets(mode);
        if ( (fp = fopen( filename, mode )) != NULL )
        {
            printf("\nDosya %s acimi %s modunda basarili.\n", filename, mode);
            fclose(fp);
            puts("Cikis icin x , devam için herhangi bir tusa basiniz.");
            if ( (getc(stdin)) == 'x')
                break;
            else
                continue;
        }
    }
}
```

```
else
{
fprintf(stderr, "\nDosya %s aciminda hata, mod; %s.\n", filename, mode);
puts("Cikis icin x , devam için herhangi bir tusa basiniz.");
if ( (getc(stdin)) == 'x')
break;
else
continue;
}
}
}
```

Dosya Kullanımı

- Dosya yazımı veya dosyadan okuma 3 farklı yolla yapılabilir:
 - Formatlı okuma-yazma; text dosyaları
 - Karakter okuma-yazma; text dosyaları
 - Direkt okuma-yazma; binary data
- C dili bunlar üzerinde katı değildir, akıllı programcı birini veya birkaçını istediği şekilde kullanabilir. Bilgi nasıl yazıldıysa öyle okunmalıdır ama buda değiştirilebilir.

Formatlı yazma

- `int fprintf(FILE *fp, char *fmt, ...);`
`#include <stdlib.h>`
`#include <stdio.h>`
`void clear_kb(void);`
`main()`
`{`
`FILE *fp;`
`float data[5];`
`int count;`
`char filename[20];`
`puts("5 tam olmayan sayi giriniz.");`

```
for (count = 0; count < 5; count++)
    scanf("%f", &data[count]);
    clear_kb();
puts("File ismini girin.");
gets(filename);
if ( (fp = fopen(filename, "w")) == NULL)
{
    fprintf(stderr, "File olusturmada hata %s.", filename);
    exit(1);
}
for (count = 0; count < 5; count++)
{
    fprintf(fp, "\ndata[%d] = %f", count, data[count]);
    fprintf(stdout, "\ndata[%d] = %f", count, data[count]);
}
fclose(fp);
printf("\n");
return(0);
}
void clear_kb(void)
{
    char junk[80];
    gets(junk);
}
```

Formatlı okuma

- **int fscanf(FILE *fp, const char *fmt, ...);**
- Note defteri (text editör) ile aşağıdaki veriyi **deneme.txt** olarak kaydedin;

12.6 8.8

10.5

0.6 1.5

Sonra aşağıdaki programı kullanarak bu veriyi okuyun.

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

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

```
main()
```

```
{
```

```
float f1, f2, f3, f4, f5;
```

```
FILE *fp;
```

```
if ( (fp = fopen("deneme.txt", "r")) == NULL)
```

```
{
```

```
fprintf(stderr, "Dosya aciminda hata.\n");
```

```
exit(1);
```

```
}
```

```
fscanf(fp, "%f %f %f %f %f", &f1, &f2, &f3, &f4, &f5);
```

```
printf("Dosyadaki degerler %f, %f, %f, %f, and %f\n.",  
f1, f2, f3, f4, f5);
```

```
fclose(fp);
```

```
return(0);
```

```
}
```

Karakter okuma

getc() ve fgetc() :tek karakter

fgets(): satırlar halinde karakter

Getc() ve fgetc() aynıdır ve birbirlerinin yerine kullanılabilirler. Prototip tanımı

int getc(FILE *fp);

Fp fopen tarafından döndürülen pointer.

char *fgets(char *str, int n, FILE *fp);

Str: girişin depolanacağı yer, **n** max karakter sayısı, **fp** pointer fopen ile gelen

Karakter yazma

- `int putc(int ch, FILE *fp);` tek karakter
- `char fputs(char *str, FILE *fp);` satır

Direkt Okuma-Yazma

- Binary modu için kullanılır, formatı
`int fwrite(void *buf, int size, int count, FILE *fp);`
- **Buf:** hafızada verinin bulunduğu yer adresi,
- **size:** byte olarak verinin büyüklüğü,
- **Count:** yazılacak eleman sayısı,
- **Fp:** açtığınız dosya için adres

- Double x=5.235; bir dosyaya nasıl yazılır:

fwrite(&x, sizeof(double), 1, fp);

Array data[]: 50 elemanlı bir structure(tür address) :

fwrite(data, sizeof(address), 50, fp);

fwrite(data, sizeof(data), 1, fp);

Okuma:

int fread(void **buf*, int *size*, int *count*, FILE *fp);

```
#include <stdlib.h>
#include <stdio.h>
#define SIZE 20
main()
{
    int count, array1[SIZE], array2[SIZE];
    FILE *fp;
    for (count = 0; count < SIZE; count++)
        array1[count] = 2 * count;
    if ( (fp = fopen("direct.txt", "wb")) == NULL)
    {
        fprintf(stderr, "Dosya acmada hata.");
        exit(1);
    }
    if (fwrite(array1, sizeof(int), SIZE, fp) != SIZE)
    {
        fprintf(stderr, "Dosya yazımında hata.");
        exit(1);
    }
    fclose(fp);
}
```



```
if ( (fp = fopen("direct.txt", "rb")) == NULL)
{
    fprintf(stderr, "Dosya acmada hata.");
    exit(1);
}
if (fread(array2, sizeof(int), SIZE, fp) != SIZE)
{
    fprintf(stderr, "Dosya okumada hata.");
    exit(1);
}
fclose(fp);
for (count = 0; count < SIZE; count++)
    printf("%d\t%d\n", array1[count], array2[count]);

return(0);
}
```

Dosya kontrol

- void **rewind(FILE *fp);** Pozisyon gösterici dosyanın başına gider
- long **ftell(FILE *fp);** Pozisyon göstericinin yerini verir (0. byttan itibaren)

```
#include <stdlib.h>
#include <stdio.h>
#define BUFLLEN 6
char msg[] = "abcdefghijklmnopqrstuvwxyz";
main()
{
FILE *fp;
char buf[BUFLLEN];
if ( (fp = fopen("TEXT.TXT", "w")) == NULL)
{
fprintf(stderr, "Dosya acmada hata.");
exit(1);
}
if (fputs(msg, fp) == EOF)
{
fprintf(stderr, "Dosyaya yazımda hata.");
exit(1); }
fclose(fp);
```

```
if ( (fp = fopen("TEXT.TXT", "r")) == NULL)
{
    fprintf(stderr, "Dosya acmada hata.");
    exit(1);
}
printf("\nDosya acildikten hemen sonra, pozisyon = %ld", ftell(fp));
fgets(buf, BUFLen, fp);
printf("\n bazi karakterleri %s okuduktan sonra, pozisyon = %ld",
    buf, ftell(fp));
fgets(buf, BUFLen, fp);
printf("\n\n Sonraki 5 karakter %s, ve simdiki pozisyon = %ld",
    buf, ftell(fp));
rewind(fp);
printf("\n\nGeri sarmadan sonra pozisyon %ld",
    ftell(fp));
fgets(buf, BUFLen, fp);
printf("\nve yeniden bastan okuma %s\n", buf);
fclose(fp);
return(0);
}
```

Dosya sonunun tespiti

- EOF (-1) tespiti ile:

```
while ( (c = fgetc( fp )) != EOF )
```

Veya

`int feof(FILE *fp):` Dosya sonu değilse 0 geri döndürür, dosya sonuna gelinmişse sıfırdan farklı bir sayı. Hem text hem binary için kullanılabilir.

Bir text dosyası oluşturup aşağıdaki program ile okutun

```
#include <stdlib.h>
#include <stdio.h>
#define BUFSIZE 100
main()
{
char buf[BUFSIZE];
char filename[60];
FILE *fp;
puts("Dosya adi lutfen: ");
gets(filename);
if ( (fp = fopen(filename, "r")) == NULL)
{
fprintf(stderr, "Dosya acmada hata.");
exit(1);
}
while ( !feof(fp) )
{
fgets(buf, BUFSIZE, fp);
printf("%s",buf);
}
fclose(fp);
return(0); }
```

Dosya silme

- **int remove(const char **filename*);**
- Silinecek dosyanın pointeri, remove() dosyayı silerse 0 geri döndürür. Silemezse -1 geri döndürür.

```
#include <stdio.h>

main()
{
    char filename[80];
    printf("Silinecek dosyanin ismi: ");
    gets(filename);
    if ( remove(filename) == 0)
        printf("Dosya %s silindi efendim.\n", filename);
    else
        fprintf(stderr, "Dosya silmede hata olustu %s.\n", filename);
    return(0);
}
```

Dosya ismi deęiřtirme

- **int rename(const char **oldname*, const char **newname*);**

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

```
main()
```

```
{
```

```
    char oldname[80], newname[80];
```

```
    printf("degistirmek istediginiz dosyanin ismi: ");
```

```
    gets(oldname);
```

```
    printf("Yeni isim: ");
```

```
    gets(newname);
```

```
    if ( rename( oldname, newname ) == 0 )
```

```
        printf("%s dosyasi %s olarak degistirildi.\n", oldname, newname);
```

```
    else
```

```
        fprintf(stderr, "dosya %s ismi degistirmede hata olustu.\n", oldname);
```

```
    return(0);
```

```
}
```


Kelime karşılaştırma

```
#include <stdio.h>
#include <string.h>
#define CEVAP "alpaslan"
#define MAX 40
int main(void)
{
    char dene[MAX];
    puts("Alemin kirali kim, cabuk soyle?");
    gets(dene);
    while (strcmp(dene,CEVAP) != 0)
    {
        puts("Yanlis, kafani koparmadan dogruyu bul, cabuk.");
        gets(dene);
    }
    puts("Simdi oldu, yasmana izin verildi!");
    return 0;
}
```

Ör: structure kaydı

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

```
struct giris {
    char adi[20];
    char soyadi[20];
    char tel[10];
};
```

```
struct giris liste[4];
int i;
void olustur(struct giris liste[]);
void yaz(const struct giris liste[]);
```

```
main()
{
    FILE *fp;
    struct giris liste2[4];
    olustur(liste);
```

```
if ( (fp = fopen("dosya.dat", "a+b")) == NULL) {  
    fprintf(stderr, "Dosya acmada hata.");  
    exit(1);  
}  
if (fwrite(liste, sizeof(struct giris), 4, fp) != 4) {  
    fprintf(stderr, "Dosya yazımında hata.");  
    exit(1);  
}  
fclose(fp);  
    if ( (fp = fopen("dosya.dat", "rb")) == NULL) {  
        fprintf(stderr, "Dosya acmada hata.");  
        exit(1);  
    }  
    if (fread(liste2, sizeof(struct giris), 4, fp) != 4) {  
        fprintf(stderr, "Dosya okumada hata.");  
        exit(1);  
    }  
    yaz(liste2);  
    fclose(fp);  
return 0;  
}
```

```
void olustur(struct giris liste[])
{
    for (i = 0; i < 4; i++)
    {
        printf("\nilk adi giriniz: ");
        scanf("%s", liste[i].adi);
        printf("Soy adi giriniz: ");
        scanf("%s", liste[i].soyadi);
        printf("telefonu 555-6666 formatinda giriniz: ");
        scanf("%s", liste[i].tel);
    }
}

void yaz(const struct giris liste2[])
{
    printf("\n\n");
    for (i = 0; i < 4; i++)
    {
        printf("Adi: %s %s", liste2[i].adi, liste2[i].soyadi);
        printf("\t\tTelefonu: %s\n", liste2[i].tel);
    }
}
```

Unions

- Veri saklamamın bir başka yolu

```
union hold {  
    int digit;  
    double sayi;  
    char harf;  
};
```

Union hold tut;

Üyeleri aynı hafıza bölgesini kullanır, bu sebeple aynı anda bir üyesi aktif olarak kullanılabilir. Tanım ve kullanımı structur ile aynıdır.

```
#include <stdio.h>
#define CH 'C'
#define INT 'I'
#define FLO 'F'

    struct veri{
        char type;
        union data {
            char c;
            int i;
            float f;
        } ortak;
    };

void print_function( struct veri bilgi );
main()
{
    struct veri var;
    var.type = CH;
    var.ortak.c = '$';
    print_function( var );
    var.type = FLO;
    var.ortak.f = (float) 12345.67890;
    print_function( var );
    var.type = 'x';
    var.ortak.i = 111;
    print_function( var );

return 0;
}
```

```
void print_function( struct veri bilgi )
{
    printf("\n\nThe bilgi degeri...");
    switch( bilgi.type )
    {
        case CH: printf("%c", bilgi.ortak.c);
        break;
        case INT: printf("%d", bilgi.ortak.i);
        break;
        case FLO: printf("%f", bilgi.ortak.f);
        break;
        default: printf("tur bilinmiyor:
%c\n",
        bilgi.type);
        break;
    }
}
```

Enumerated Types

- İnt sabitler için sembolik isimler kullanılmasına olanak verir. Böylece programın okunurluğunu geliştirilir.

```
enum spectrum {red, orange, yellow, green, blue, violet};
```

```
enum spectrum color;
```

Spectrum=yeni bir tür

Color=değişken

Bu değişkenin alabileceği değerler {içinde verilmiş }

```
printf("red = %d, orange = %d\n", red, orange);
```

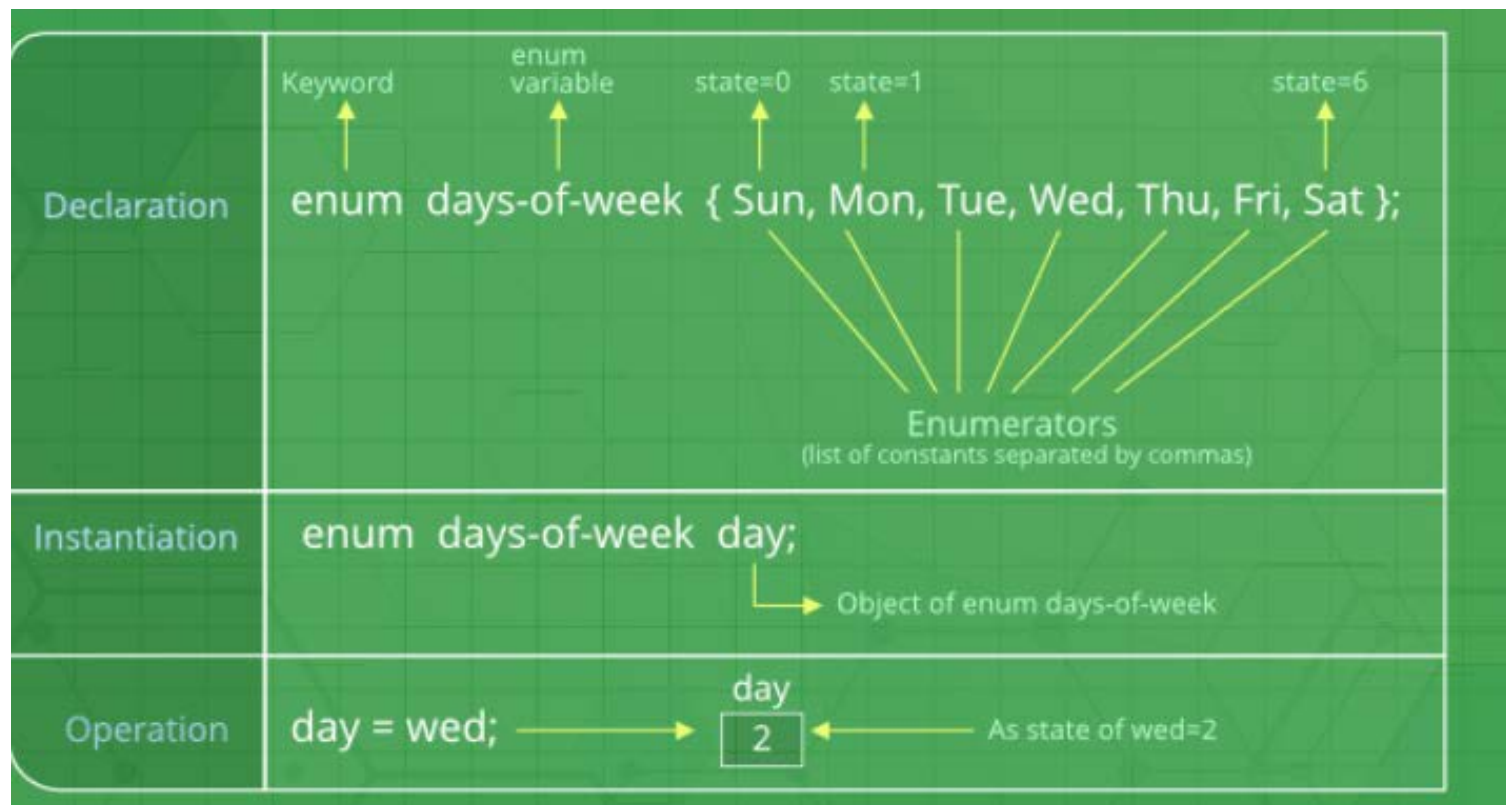
yazılabilir çünkü bunlar gerçekte int sabitlerdir.

Çıktı red=0, orange=1 dir.

Ne oldu, red kelimesi 0'ı temsil eden bir sabit oldu.

Enumerated Types

- **İnt değerleri {0,1,2,...} olarak otomatik atanır**
veya
- kullanıcı tarafından belirtilir:
**enum levels {low = 100, medium = 500,
high = 2000};**



```
enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};
```

```
int main()  
{  
    enum week day;  
    day = Wed;  
    printf("%d",day);  
    return 0;  
}
```

Output:

2

```
enum year{Jan, Feb, Mar, Apr, May, Jun, Jul,  
          Aug, Sep, Oct, Nov, Dec};
```

```
int main()  
{  
    int i;  
    for (i=Jan; i<=Dec; i++)  
        printf("%d ", i);  
  
    return 0;  
}
```

Output:

```
0 1 2 3 4 5 6 7 8 9 10 11
```

```
enum day {sunday = 1, monday, tuesday = 5,  
          wednesday, thursday = 10, friday, saturday};
```

```
int main()  
{  
    printf("%d %d %d %d %d %d %d", sunday, monday, tuesday,  
          wednesday, thursday, friday, saturday);  
    return 0;  
}
```

Output:

```
1 2 5 6 10 11 12
```

typedef : var olan türlere kendi adını ver

Typedef: var olan türler için yeni isimler belirlemek için kullanılır, okunurluğu ve bağımsızlığı artırır.

```
typedef int tam;  
tam count=2;  
typedef struct {  
    int x;  
    int y;  
} coord;  
coord alt, ust;
```

Karmaşık deklarasyonlar!?

* : pointer için

() : fonksiyon

[] : dizi

[] ve () önceliği aynı ve * dan yüksek.

ÖR: **int *alp[10]** ??? Bu bir pointer dizisi (bir diziyi işaret eden pointer değil)

int (* alp)[10]; pointer to dizi alp

C Fonksiyonları

<i>Function</i>	<i>Prototype</i>	<i>Description</i>
acos()	double acos(double x)	Returns the arccosine of its argument. The argument must be in the range $-1 \leq x \leq 1$, and the return value is in the range $0 \leq \text{acos} \leq \pi$.
asin()	double asin(double x)	Returns the arcsine of its argument. The argument must be in the range $-1 \leq x \leq 1$, and the return value is in the range $-\pi/2 \leq \text{asin} \leq \pi/2$.
atan()	double atan(double x)	Returns the arctangent of its argument. The return value is in the range $-\pi/2 \leq \text{atan} \leq \pi/2$.
atan2()	double atan2(double x, double y)	Returns the arctangent of x/y . The value returned is in the range $-\pi \leq \text{atan2} \leq \pi$.
cos()	double cos(double x)	Returns the cosine of its argument.
sin()	double sin(double x)	Returns the sine of its argument.
tan()	double tan(double x)	Returns the tangent of its argument.

<i>Function</i>	<i>Prototype</i>	<i>Description</i>
exp()	double exp(double x)	Returns the natural exponent of its argument, that is, e^x where e equals 2.7182818284590452354.
log()	double log(double x)	Returns the natural logarithm of its argument. The argument must be greater than 0.
log10()	double log10(double x)	Returns the base-10 logarithm of its argument. The argument must be greater than 0.
frexp()	double frexp(double x, int *y)	The function calculates the normalized fraction representing the value x . The function's return value r is a fraction in the range $0.5 \leq r \leq 1.0$. The function assigns to y an integer exponent such that $x = r * 2^y$. If the value passed to the function is 0, both r and y are 0.
ldexp()	double ldexp(double x, int y)	Returns $x * 2^y$.

<i>Function</i>	<i>Prototype</i>	<i>Description</i>
cosh()	double cosh(double x)	Returns the hyperbolic cosine of its argument.
sinh()	double sinh(double x)	Returns the hyperbolic sine of its argument.
tanh()	double tanh(double x)	Returns the hyperbolic tangent of its argument.

<i>Function</i>	<i>Prototype</i>	<i>Description</i>
sqrt()	double sqrt(double x)	Returns the square root of its argument. The argument must be zero or greater.
ceil()	double ceil(double x)	Returns the smallest integer not less than its argument. For example, ceil(4.5) returns 5.0, and ceil(-4.5) returns -4.0. Although ceil() returns an integer value, it is returned as a type double.
abs()	int abs(int x)	Returns the absolute
labs()	long labs(long x)	value of their arguments.
floor()	double floor(double x)	Returns the largest integer not greater than its argument. For example, floor(4.5) returns 4.0, and floor(-4.5) returns -5.0.
modf()	double modf(double x, double *y)	Splits x into integral and fractional parts, each with the same sign as x. The fractional part is returned by the function, and the integral part is assigned to *y.
pow()	double pow(double x, double y)	Returns x^y . An error occurs if $x == 0$ and $y \leq 0$, or if $x < 0$ and y is not an integer.
fmod()	double fmod(double x, double y)	Returns the floating-point remainder of x/y , with the same sign as x. The function returns 0 if $x == 0$.


```
#include <stdio.h>
#include <math.h>
main() {
    double x;

    printf("Bir sayi lutfen ");
    scanf( "%lf", &x);

    printf("\n\nOrjinal sayiniz %lf", x);

    printf("\nCeil: %lf", ceil(x));
    printf("\nFloor: %lf", floor(x));
    if( x >= 0 )
        printf("\nkarekoku %lf", sqrt(x) );
    else
        printf("\nNegatif sayi" );

    printf("\nCosine: %lf\n", cos(x));
    return(0);
}
```

Hafıza ile çalışma

malloc()

void *malloc(size_t *num*);

Kullanımı:

double * ptd;

ptd = (double *) malloc(30 * sizeof(double));

Syntax of malloc()


```
1. ptr = (castType*) malloc(size);
```

Example

```
1. ptr = (int*) malloc(100 * sizeof(float));
```

Malloc()

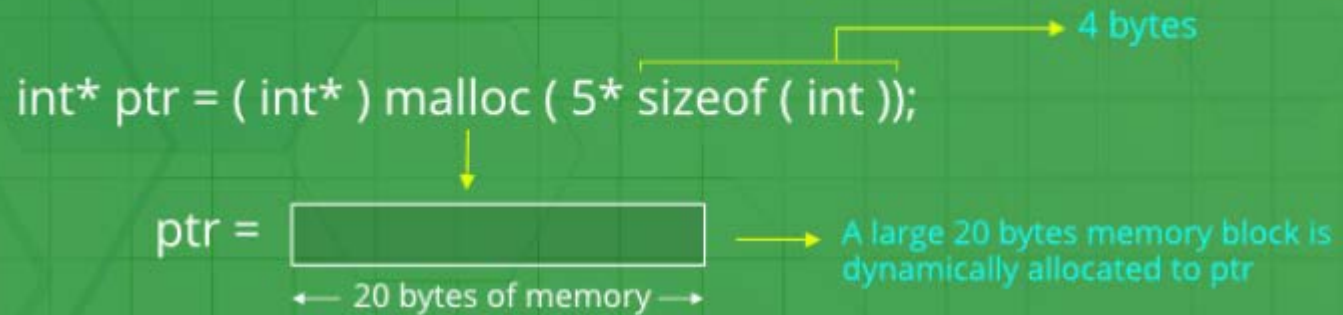
```
int* ptr = ( int* ) malloc ( 5* sizeof ( int ) );
```

ptr = 

← 20 bytes of memory →

4 bytes

A large 20 bytes memory block is dynamically allocated to ptr



```
#include <stdio.h>  #include <stdlib.h>
int main(void) {
    double * ptd;    int max, sayi;    int i = 0;

    puts("Kac sayi kullanacaksiniz, max sayi?");
    scanf("%d", &max);
    ptd = (double *) malloc(max * sizeof (double));

    if (ptd == NULL) {
        puts("Hafiza da yer ayirma basarisiz.");
        exit(EXIT_FAILURE);
    }
    puts("Sayilari girin (cikis icin q):");
    while (i < max && scanf("%lf", &ptd[i]) == 1)
        ++i;
    printf("Girdiginiz sayilar %d bunlar:\n", sayi = i);
    for (i = 0; i < sayi; i++) {
        printf("%7.2f ", ptd[i]);
        if (i % 7 == 6)
            putchar('\n');
    }
    free(ptd);
    return 0;
}
```

```
struct rec {  
    int i;  
    float PI;  
    char A;  
};  
  
int main() {  
    struct rec *ptr_one;  
    ptr_one = (struct rec *) malloc(sizeof(struct rec));  
    ptr_one->i = 10;  
    ptr_one->PI = 3.14;  
    ptr_one->A = 'a';  
    printf("First value: %d\n", ptr_one->i);  
    printf("Second value: %f\n", ptr_one->PI);  
    printf("Third value: %c\n", ptr_one->A);  
    free(ptr_one);  
    return 0;}
```

Hafıza ile çalışma

calloc()

```
long * new;
```

```
new = (long *)calloc(100, sizeof (long));
```

İlk argüman kaç hafıza ünitesi, ikincisi hafıza ünitesinin boyutu

free() calloc() ile de kullanılır.

Syntax:

```
ptr = (cast-type*)calloc(n, element-size);
```

Calloc()

```
int* ptr = ( int* ) calloc ( 5, sizeof ( int ) );
```



Free()

```
int* ptr = ( int* ) calloc ( 5, sizeof ( int ) );
```



operation on ptr

free(ptr)



The memory of ptr is released



```
#include <stdlib.h>
#include <stdio.h>
main()
{
    unsigned num;
    int *ptr;

    printf("ne kadar int turunde hafiza istiyorsunuz ");
    scanf("%d", &num);

    ptr = (int*)calloc(num, sizeof(int));

    if (ptr != NULL)
        puts("Hafiza ayirma basarili.");
    else
        puts("Basarisiz.");
    return(0);
}
```



```

#include <stdio.h>  #include <stdlib.h>
int main() {
    // This pointer will hold the    // base address of the block created
    int *ptr, *ptr1;
    int n, i, sum = 0;
    n = 5; // Get the number of elements for the array
    printf("Enter number of elements: %d\n", n);
    ptr = (int*)malloc(n * sizeof(int)); // Dynamically allocate memory using malloc()
    ptr1 = (int*)calloc(n, sizeof(int)); // Dynamically allocate memory using calloc()
    // Check if the memory has been successfully    // allocated by malloc or not
    if (ptr == NULL || ptr1 == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {
        printf("Memory successfully allocated using malloc.\n"); // Memory has been successfully allocated
        free(ptr); // Free the memory

        printf("Malloc Memory successfully freed.\n");
        printf("\nMemory successfully allocated using calloc.\n"); // Memory has been successfully allocated
        free(ptr1); // Free the memory
        printf("Calloc Memory successfully freed.\n");
    }
    return 0;
}

```

void *realloc(void *ptr, size_t size);

Malloc() veya calloc() ile ayrılmış olan hafızanın boyutunu değiştirir.

*ptr hafıza bloğunun adresi, size ise yeni boyut.

void *memcpy(void *dest, void *src, size_t count);

Src= kopyalanacak adres,

Dest=yeni yer,

Count= kopyalanacak byte sayısı

Dest geri döndürülür, artık bilginin kopyası bu pointer ile işaret edilir.

İki hafıza çakışıyorsa bu fonksiyon güvenli çalışmaz.

void *memmove(void *dest, void *src, size_t count);

Memmove() memcpy() yaptığı her şeyi yapar, çakışmayı kolayca halleder.

Realloc()

```
int* ptr = ( int* ) malloc ( 5* sizeof ( int ));
```

4 bytes

ptr = 

← 20 bytes of memory →

A large 20 bytes memory block is dynamically allocated to ptr

```
ptr = realloc ( ptr, 10* sizeof( int ));
```

ptr = 

← 40 bytes of memory →

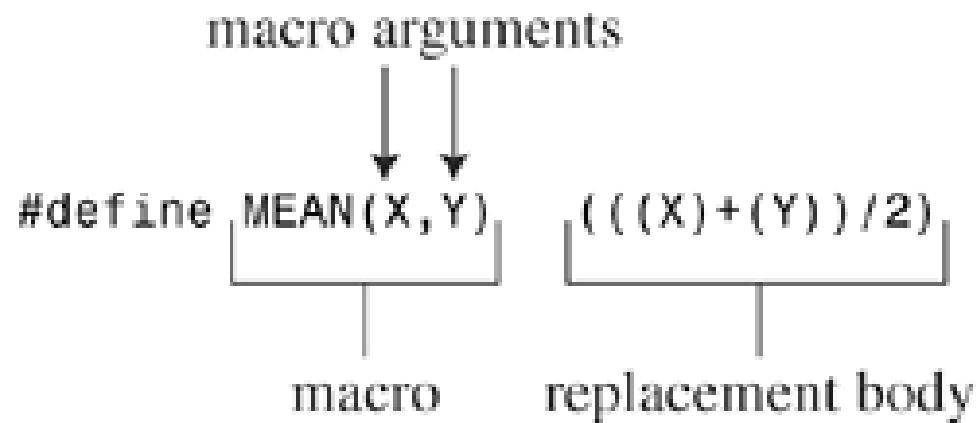
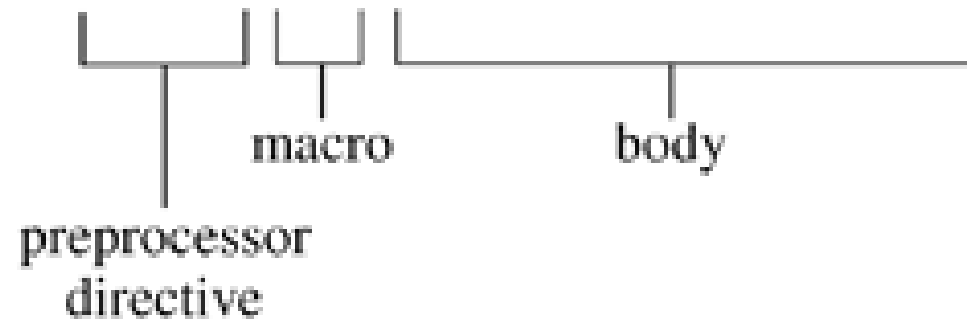
The size of ptr is changed from 20 bytes to 40 bytes dynamically

Realloc() kullanımı

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *ptr, i, n1, n2;
    printf("Enter size of array: ");
    scanf("%d", &n1);
    ptr = (int*)malloc(n1 * sizeof(int));
    printf("Addresses of previously allocated memory: ");
    for (i = 0; i < n1; ++i)
        printf("%u\n", ptr + i);
    printf("\nEnter new size of array: ");
    scanf("%d", &n2);
    ptr = realloc(ptr, n2 * sizeof(int));
    printf("Addresses of newly allocated memory: ");
    for (i = 0; i < n2; ++i)
        printf("%u\n", ptr + i);
    return 0; }
```

```
#include <stdio.h>  ///// #define kullanımı ////////////////////////////////////
#define TWO 2
#define OW "kararlilik"
#define FOUR TWO*TWO
#define PX printf("X is %d.\n", x)
#define FMT "X is %d.\n"
int main(void)
{
    int x = TWO;
    PX;
    x = FOUR;
    printf(FMT, x);
    printf("%s\n", OW);
    printf("TWO: OW\n");
    return 0;}
```

```
#define PX printf("x is %d.\n",x)
```



```
#include <stdio.h>

#define SQUARE(X) X*X
#define PR(X)  printf("The result is %d.\n", X)

int main(void)
{
    int x = 4;          int z;
    printf("x = %d\n", x);
    z = SQUARE(x);
    printf("Evaluating SQUARE(x): ");
    PR(z);
    printf("Evaluating SQUARE(x+2): ");
    PR(SQUARE(x+2));
    printf("Evaluating 100/SQUARE(2): ");
    PR(100/SQUARE(2));
    printf("x is %d.\n", x);
    printf("Evaluating SQUARE(++x): ");
    PR(SQUARE(++x));
    return 0;
}
```

```
#include <stdio.h>

#define PSQR(x) printf("The square of " #x " is %d.\n",((x)*(x)))

int main(void)
{
    int y = 5;

    PSQR(y);

    PSQR(2 + 4);

    return 0;
}
```



```
// names_st.h -- names_st structure header file  
#define SLEN 32  
struct names_st  
{  
    char first[SLEN];  
    char last[SLEN];  
};  
typedef struct names_st names;  
void get_names(names *);  
void show_names(const names *);
```

```
#include <stdio.h>
#include "names_st.h"    // include the header file
void get_names(names * pn)
{
    int i;
    printf("Please enter your first name: ");
    fgets(pn->first, SLEN, stdin);
    i = 0;
    while (pn->first[i] != '\n' && pn->first[i] != '\0')
        i++;
    if (pn->first[i] == '\n')
        pn->first[i] = '\0';
    else
        while (getchar() != '\n')
            continue;
    printf("Please enter your last name: ");
    fgets(pn->last, SLEN, stdin);
    i = 0;
    while (pn->last[i] != '\n' && pn->last[i] != '\0')
        i++;
    if (pn->last[i] == '\n')
        pn->last[i] = '\0';
    else
        while (getchar() != '\n')
            continue;
}
void show_names(const names * pn)
{
    printf("%s %s", pn->first, pn->last);
}
```

- **The `#undef` Directive**
- The `#undef` directive "undefines" a given `#define`.
- That is, suppose you have this definition:
 - **`#define LIMIT 400`**
Then the directive
 - **`#undef LIMIT`**

Conditional Compilation

- You can use the other directives mentioned to set up conditional compilations. That is, you can use them to tell the compiler to accept or ignore blocks of information or code according to conditions at the time of compilation.
- **The #ifdef, #else, and #endif Directives**
- A short example will clarify what conditional compilation does. Consider the following:

#ifdef MAVIS

#include "horse.h" // gets done if MAVIS is #defined

#define STABLES 5

#else

#include "cow.h" // gets done if MAVIS isn't #defined

#define STABLES 15

#endif

The **#ifndef** Directive

- The **#ifndef** directive can be used with **#else** and **#endif** in the same way that **#ifdef** is. The **#ifndef** asks whether the following identifier is not defined; **#ifndef** is the negative of **#ifdef**. This directive is often used to define a constant if it is not already defined. Here's an example:

- ```
/* arrays.h */
#ifndef SIZE
#define SIZE 100
#endif
```

## The #if and #elif Directives

- The #if directive is more like the regular C if. It is followed by a constant integer expression that is considered true if nonzero, and you can use C's relational and logical operators with it:

```
#if SYS == 1
#include "ibm.h"
#endif
```

- You can use the #elif directive (not available in some older implementations) to extend an if-else sequence. For example, you could do this:

```
#if SYS == 1
#include "ibmpc.h"
#elif SYS == 2
#include "vax.h"
#elif SYS == 3
#include "mac.h"
#else
#include "general.h"
#endif
```

## Predefined Macros

| Macro                         | Meaning                                                                                     |
|-------------------------------|---------------------------------------------------------------------------------------------|
| <code>__DATE__</code>         | A character string literal in the form "Mmm dd yyyy" representing the date of preprocessing |
| <code>__FILE__</code>         | A character string literal representing the name of the current source code file            |
| <code>__LINE__</code>         | An integer constant representing the line number in the current source code file            |
| <code>__STDC__</code>         | Set to 1 to indicate the implementation conforms to the C Standard                          |
| <code>__STDC_HOSTED__</code>  | Set to 1 for a hosted environment; 0 otherwise                                              |
| <code>__STDC_VERSION__</code> | For C99, set to 199901L                                                                     |
| <code>__TIME__</code>         | The time of translation in the form "hh:mm:ss"                                              |

## Inline Functions

- Normally, a function call has overhead. That means it **takes execution time to set up the call, pass arguments, jump to the function code, and return**. Reducing that time is one of the justifications for using function-like macros. C99 has an alternative, inline functions. The C99 standard has this to say: "Making a function an inline function suggests that calls to the function be as fast as possible. The extent to which such suggestions are effective is implementation-defined." So making a function an inline function may shortcut the usual function call mechanism, or it may do nothing at all.
- The way to create an inline function is to use the function specifier inline in the function declaration. Usually, inline functions are defined before the first use in a file, so the definition also acts as a prototype. That is, the code would look like this:

```
#include <stdio.h>
inline void eatline() // inline definition/prototype
{
 while (getchar() != '\n')
 continue;
}
int main()
{
 ...
 eatline(); // function call
 ...
}
```



# ANSI C Standard Math Functions

| Prototype                                     | Description                                                                      |
|-----------------------------------------------|----------------------------------------------------------------------------------|
| <code>double acos(double x)</code>            | Returns the angle (0 to $\pi$ radians) whose cosine is <code>x</code>            |
| <code>double asin(double x)</code>            | Returns the angle ( $-\pi/2$ to $\pi/2$ radians) whose sine is <code>x</code>    |
| <code>double atan(double x)</code>            | Returns the angle ( $-\pi/2$ to $\pi/2$ radians) whose tangent is <code>x</code> |
| <code>double atan2(double y, double x)</code> | Returns the angle ( $-\pi$ to $\pi$ radians) whose tangent is <code>y / x</code> |
| <code>double cos(double x)</code>             | Returns the cosine of <code>x</code> ( <code>x</code> in radians)                |
| <code>double sin(double x)</code>             | Returns the sine of <code>x</code> ( <code>x</code> in radians)                  |
| <code>double tan(double x)</code>             | Returns the tangent of <code>x</code> ( <code>x</code> in radians)               |
| <code>double exp(double x)</code>             | Returns the exponential function of <code>x</code> ( $e^x$ )                     |
| <code>double log(double x)</code>             | Returns the natural logarithm of <code>x</code>                                  |
| <code>double log10(double x)</code>           | Returns the base 10 logarithm of <code>x</code>                                  |
| <code>double pow(double x, double y)</code>   | Returns <code>x</code> to the <code>y</code> power                               |
| <code>double sqrt(double x)</code>            | Returns the square root of <code>x</code>                                        |
| <code>double ceil(double x)</code>            | Returns the smallest integral value not less than <code>x</code>                 |
| <code>double fabs(double x)</code>            | Returns the absolute value of <code>x</code>                                     |
| <code>double floor(double x)</code>           | Returns the largest integral value not greater than <code>x</code>               |

## **memcpy() and memmove() from the string.h Library**

- You can't assign one array to another, so we've been using loops to copy one array to another, element by element. The one exception is that we've used the strcpy() and strncpy() functions for character arrays. The memcpy() and memmove() functions offer you almost the same convenience for other kinds of arrays. Here are the prototypes for these two functions:

- **void \*memcpy(void \* restrict s1, const void \* restrict s2, size\_t n);**  
**void \*memmove(void \*s1, const void \*s2, size\_t n);**

Both of these functions copy n bytes from the location pointed to by s2 to the location pointed to by s1, and both return the value of s1. The difference between the two, as indicated by the keyword restrict, is that memcpy() is free to assume that there is no overlap between the two memory ranges. The memmove() function doesn't make that assumption, so copying takes place as if all the bytes are first copied to a temporary buffer before being copied to the final destination. What if you use memcpy() when there are overlapping ranges? The behavior is undefined, meaning it might work or it might not. The compiler won't stop you from using the memcpy() function when you shouldn't, so it's your responsibility to make sure the ranges aren't overlapping when you use it. It's just another part of the programmer's burden.

```

#include <stdio.h> #include <string.h> #include <stdlib.h> #define SIZE 10

void show_array(const int ar[], int n);

int main()
{
 int values[SIZE] = {1,2,3,4,5,6,7,8,9,10}; int target[SIZE]; double curious[SIZE / 2] = {1.0, 2.0, 3.0, 4.0, 5.0};
 puts("memcpy() used:"); puts("values (original data): "); show_array(values, SIZE);
 memcpy(target, values, SIZE * sizeof(int));
 puts("target (copy of values):");
 show_array(target, SIZE);
 puts("\nUsing memmove() with overlapping ranges:");
 memmove(values + 2, values, 5 * sizeof(int));
 puts("values -- elements 0-5 copied to 2-7:");
 show_array(values, SIZE);
 puts("\nUsing memcpy() to copy double to int:");
 memcpy(target, curious, (SIZE / 2) * sizeof(double));
 puts("target -- 5 doubles into 10 int positions:");
 show_array(target, SIZE);
 return 0;
}

void show_array(const int ar[], int n)
{
 int i;
 for (i = 0; i < n; i++)
 printf("%d ", ar[i]);
 putchar('\n');
}

```

There are 4 types of storage class:

1. automatic
2. external
3. static
4. register

## Storage classes in C

| Storage Specifier | Storage      | Initial value | Scope                    | Life                |
|-------------------|--------------|---------------|--------------------------|---------------------|
| auto              | stack        | Garbage       | Within block             | End of block        |
| extern            | Data segment | Zero          | global<br>Multiple files | Till end of program |
| static            | Data segment | Zero          | Within block             | Till end of program |
| register          | CPU Register | Garbage       | Within block             | End of block        |

Syntax:

```
static data_type var_name = var_value;
```

- Static variables have a property of preserving their value even after they are out of their scope! Hence, static variables preserve their previous value in their previous scope and are not initialized again in the new scope.

```
#include<stdio.h>
int fun()
{
 static int count = 0;
 count++;
 return count;
}

int main()
{
 printf("%d ", fun());
 printf("%d ", fun());
 return 0;
}
```

Output:

```
1 2
```

```
#include<stdio.h>
int fun()
{
 int count = 0;
 count++;
 return count;
}

int main()
{
 printf("%d ", fun());
 printf("%d ", fun());
 return 0;
}
```

Output:

```
1 1
```

- Static variables (like global variables) are initialized as 0 if not initialized explicitly. For example in the below program, value of x is printed as 0, while value of y is something garbage.

```
#include <stdio.h>
int main()
{
 static int x;
 int y;
 printf("%d \n %d", x, y);
}
```

Output:

```
0
[some_garbage_value]
```

Static variables should not be declared inside structure. The reason is C compiler requires the entire structure elements to be placed together (i.e.) memory allocation for structure members should be contiguous.

- Unlike global functions in C, access to static functions is restricted to the file where they are declared. Therefore, when we want to restrict access to functions, we make them static. Another reason for making functions static can be reuse of the same function name in other files.

```
static int fun(void)
{
 printf("I am a static function ");
}
```

# “extern” keyword in C

- External variables are also known as global variables. These variables are defined outside the function. These variables are available globally throughout the function execution. The value of global variables can be modified by the functions. “extern” keyword is used to declare and define the external variables.
- **Scope** – They are not bound by any function. They are everywhere in the program i.e. global.
- **Default value** – Default initialized value of global variables are Zero.
- **Lifetime** – Till the end of the execution of the program.



- Here are some important points about extern keyword in C language,
- External variables can be declared number of times but defined only once.
- “extern” keyword is used to extend the visibility of function or variable.
- By default the functions are visible throughout the program, there is no need to declare or define extern functions. It just increase the redundancy.
- Variables with “extern” keyword are only declared not defined.
- Initialization of extern variable is considered as the definition of the extern variable.

```
extern int var = 0;
int main(void)
{
 var = 10;
 return 0;
}
```

## Register Variable

- You can use the register storage class when you want to store local variables within functions or blocks in CPU registers instead of RAM to have quick access to these variables. For example, "counters" are a good candidate to be stored in the register.
- The keyword **register** is used to declare a register storage class. The variables declared using register storage class has lifespan throughout the program.
- It's compiler's choice to put it in a register or not. Generally, compilers themselves do optimizations and put the variables in register.

- If you use & operator with a register variable then compiler may give an error or warning.
- *register* keyword can be used with pointer variables. Obviously, a register can have address of a memory location.
- Register is a storage class, and C doesn't allow multiple storage class specifiers for a variable. So, *register* can not be used with *static* .

```
#include<stdio.h>

int main()
{
 register int i = 10;
 int* a = &i;
 printf("%d", *a);
 getchar();
 return 0;
}
```

```
#include<stdio.h>

int main()
{
 int i = 10;
 register int* a = &i;
 printf("%d", *a);
 getchar();
 return 0;
}
```

```
#include<stdio.h>

int main()
{
 int i = 10;
 register static int* a = &i;
 printf("%d", *a);
 getchar();
 return 0;
}
```

# Auto

- This is the default storage class for all the variables declared inside a function or a block. Hence, the keyword `auto` is rarely used while writing programs in C language. Auto variables can be only accessed within the block/function they have been declared and not outside them
- The variables which are declared inside a block are known as **automatic** or **local variables**; these variables allocate memory automatically upon entry to that block and free the occupied memory upon exit from that block.

```
int main()
{
 auto int a;
 int b;

 return 0;
}
```

```
#include <stdio.h> // fonksiyon argümanlarının sayısı belli değil////////////////////
```

```
#include <stdarg.h>
```

```
double sum(int, ...);
```

```
int main(void)
```

```
{
```

```
 double s,t;
```

```
 s = sum(3, 1.1, 2.5, 13.3);
```

```
 t = sum(6, 1.1, 2.1, 13.1, 4.1, 5.1, 6.1);
```

```
 printf("return value for “ "sum(3, 1.1, 2.5, 13.3): %g\n", s);
```

```
 printf("return value for “ "sum(6, 1.1, 2.1, 13.1, 4.1, 5.1, 6.1): %g\n", t);
```

```
 return 0;
```

```
}
```

```
double sum(int lim,...)
```

```
{
```

```
 va_list ap; // declare object to hold arguments
```

```
 double tot = 0;
```

```
 int i;
```

```
 va_start(ap, lim); // initialize ap to argument list
```

```
 for (i = 0; i < lim; i++)
```

```
 tot += va_arg(ap, double); // access each item in argument list
```

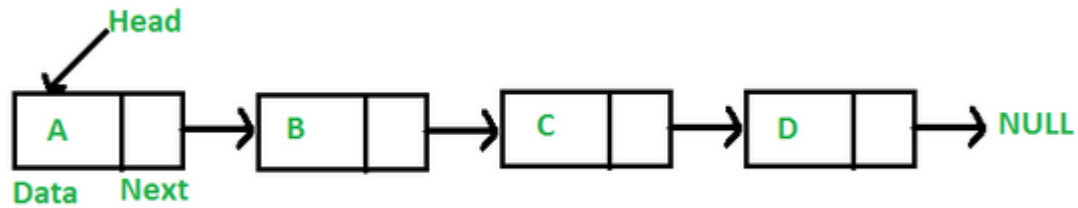
```
 va_end(ap); // clean up
```

```
 return tot;
```

```
}
```

```
// C program to demonstrate working of variable arguments to find //average of multiple numbers.
#include <stdarg.h>
#include <stdio.h>
int average(int num, ...) {
 va_list valist;
 int sum = 0, i;
 va_start(valist, num);
 for (i = 0; i < num; i++)
 sum += va_arg(valist, int);
 va_end(valist);
 return sum / num;
}
int main() {
 printf("Average of {2, 3, 4} = %d\n", average(2, 3, 4));
 printf("Average of {3, 5, 10, 15} = %d\n", average(3, 5, 10, 15));
 return 0;
}
```

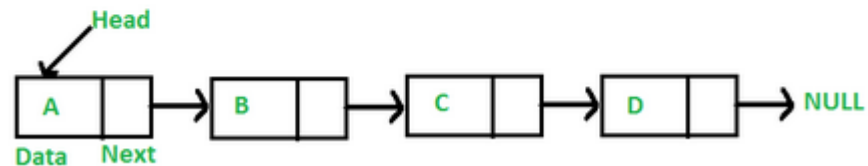
# Linked lists



|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| 40 | 55 | 63 | 17 | 22 | 68 | 89 | 97 | 89 |
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  |

<- Array Indices

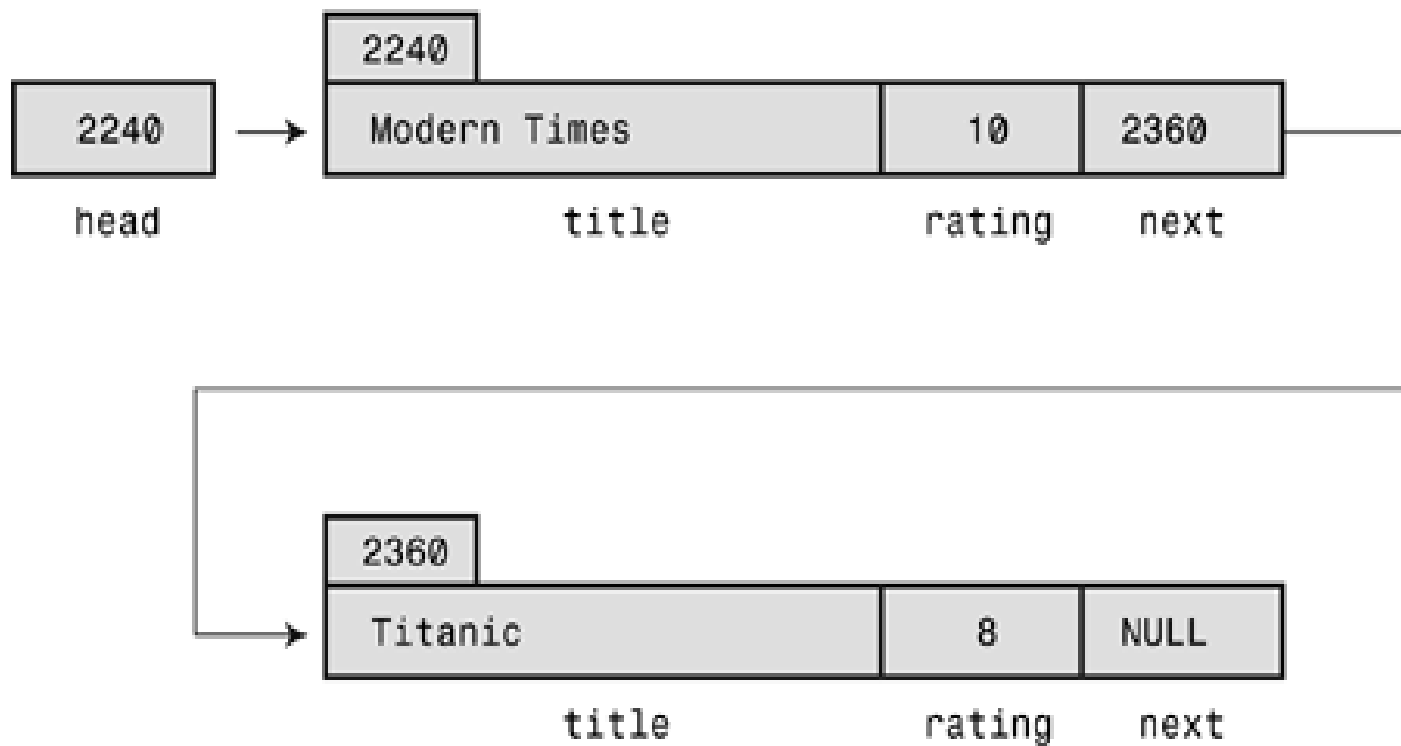
Array Length = 9  
First Index = 0  
Last Index = 8

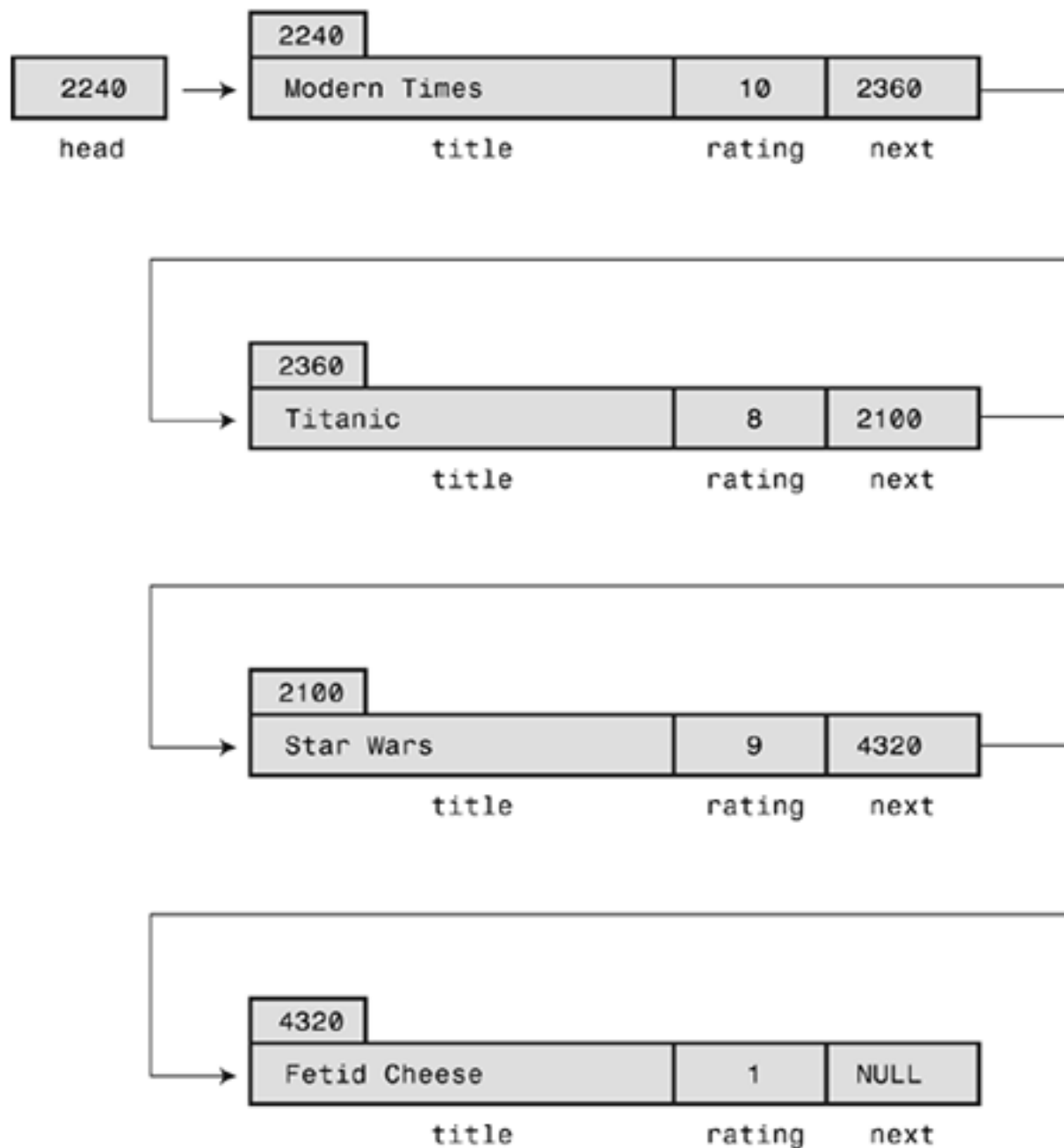


```
#define TSIZE 45
struct film {
 char title[TSIZE]
 int rating;
 struct film * next;
};
struct film * head;
```









# Avantaj & dezavantaj

## Advantages over arrays

- 1) Dynamic size
- 2) Ease of insertion/deletion

## Drawbacks:

- 1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists efficiently with its default implementation. [Read about it here.](#)
- 2) Extra memory space for a pointer is required with each element of the list.
- 3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

C

Java

Python



// A linked list node

**struct** Node

{

int data;

**struct** Node \*next;

};

C

Java

Python



# Node class

**class** Node:

# Function to initialize the node object

**def** \_\_init\_\_(self, data):

self.data = data # Assign data

self.next = None # Initialize  
# next as null

# Linked List class

**class** LinkedList:

# Function to initialize the Linked

# List object

**def** \_\_init\_\_(self):

self.head = None

C

Java

Python

**class** LinkedList

{



Node head; // head of list



/\* Linked list Node\*/

**class** Node

{

int data;

Node next;

// Constructor to create a new node

// Next is by default initialized

// as null

Node(int d) {data = d;}

}

}

# Lists oluşturma

- 3 Aşamalı bir işlemdir:  
**malloc()**
- kullanarak structure için yer ayrılır
- Structure adresi depolanır.
- Eldeki veriler structure kaydedilir

```
struct film {
 char title[TSIZE];
 int rating;
 struct film * next; /* points to next struct in
list */
};
```

Veri girişi var mı?

```
while (gets(input) != NULL && input[0] != '\0')
```

Eğer var ise program hafızada yer açar ve adresi current pointerine yollar:

```
current = (struct film *) malloc(sizeof(struct film));
```

- İlk structure adresi **head** pointerinde saklanır. Sonraki sıradaki structure adresleri **next** pointerinde saklanır.
- Program içinde ilk pointeri ile mi yoksa sonrakiler ile mi işlem yaptığınızı bilmelisiniz. Bunun yolu **head** pointerini NULL yapmak ve program içinde **head** pointerının değerini değiştirmektir.

```
if (head == NULL) /* first structure */
head = current;
else /* subsequent structures */
prev->next = current;
```

**Prev** pointeri önceki ayrılmış structure yerini gösterir.

- Structure üyelerinin değerleri atanır: Next pointerının değeri NULL atanır ki bu structure listedeki son structure. Bundan sonra film ismi ve verilen puanlar girilir.

```
current->next = NULL;
strcpy(current->title, input);
puts("Enter your rating <0-10>:");
scanf("%d", ¤t->rating);
```

- Program ikinci dalga girişler için hazırlanır: Prev pointerı şimdi current pointerına eşitlenir ki yeni giriş yapıldığında bu current bir önceki giriş olacaktır.

```
prev = current;
```



- Program malloc() tarafından kullanılan yeri serbest bırakır.

```
current = head;
while (current != NULL)
{
free(current);
current = current->next;
}
```

```
#include <stdio.h>
#include <stdlib.h> /* has the malloc prototype */
#include <string.h> /* has the strcpy prototype */
#define TSIZE 45 /* size of array to hold title */
```

```
struct film {
 char title[TSIZE];
 int rating;
 struct film * next; /* points to next struct in list */
};
```

```
int main(void)
{
 struct film * head = NULL;
 struct film * prev, * current;
 char input[TSIZE];

 /* Gather and store information */
 puts("Enter first movie title:");
 while (gets(input) != NULL && input[0] != '\0')
 {
 current = (struct film *) malloc(sizeof(struct film));
 if (head == NULL) /* first structure */
 head = current;
 else /* subsequent structures */
 prev->next = current;
 current->next = NULL;
 }
}
```

```
strcpy(current->title, input);
 puts("Enter your rating <0-10>:");
 scanf("%d", ¤t->rating);
 while(getchar() != '\n')
 continue;
 puts("Enter next movie title (empty line to stop):");
 prev = current;
}

/* Show list of movies */
if (head == NULL)
 printf("No data entered. ");
else
 printf ("Here is the movie list:\n");
current = head;
```

```
while (current != NULL)
{
 printf("Movie: %s Rating: %d\n",
 current->title, current->rating);
 current = current->next;
}

/* Program done, so free allocated memory */
current = head;
while (current != NULL)
{
 free(current);
 current = current->next;
}

printf("Bye!\n");
return 0;}
```



// A simple C program for traversal of a linked list

**#include<stdio.h>**

**#include<stdlib.h>**

**struct Node**

**{**

**int data;**

**struct Node \*next;**

**};**

**// This function prints contents of linked list starting from**

**// the given node**

**void printList(struct Node \*n){**

**while (n != NULL)**

**{**

**printf(" %d ", n->data);**

**n = n->next;**

**}**

**}**

```
int main() {
 struct Node* head = NULL; struct Node* second = NULL;
 struct Node* third = NULL;
 // allocate 3 nodes in the heap
 head = (struct Node*)malloc(sizeof(struct Node));
 second = (struct Node*)malloc(sizeof(struct Node));
 third = (struct Node*)malloc(sizeof(struct Node));

 head->data = 1; //assign data in first node
 head->next = second; // Link first node with second

 second->data = 2; //assign data to second node
 second->next = third;
 third->data = 3; //assign data to third node
 third->next = NULL;

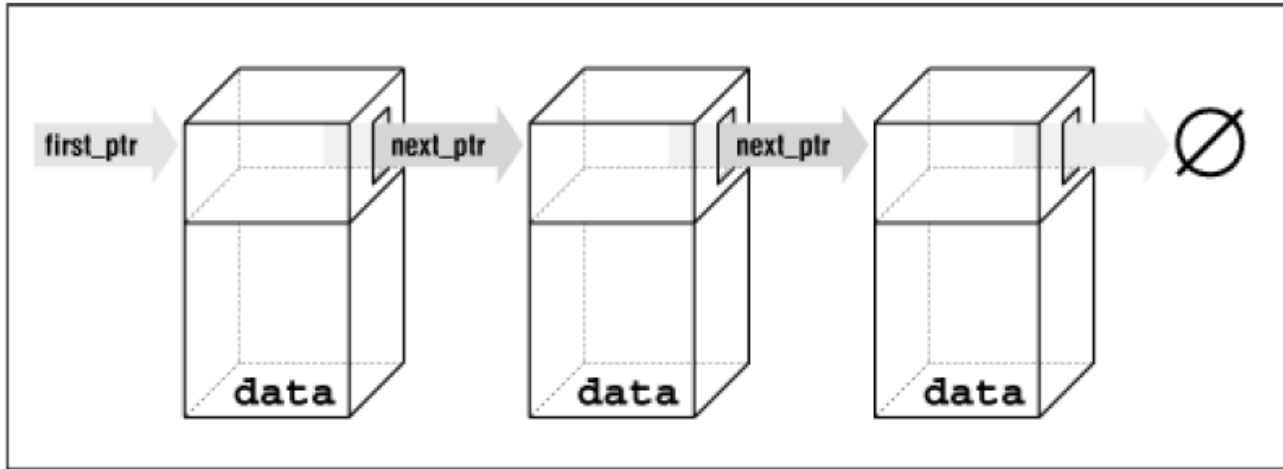
 printList(head);
 return 0;}
```





# Linked Lists (Sıralı listeler)

## Linked list



```
struct linked_list {
```

```
 char data[30];
```

*/\* data in this element \*/*

```
 struct linked_list *next_ptr;
```

*/\* pointer to next element \*/*

```
};
```

```
struct linked_list *first_ptr = NULL;
```

## Liste başına eleman ekleme

1. İlk eleman (veri) için structure oluştur.

```
new_item_ptr = malloc(sizeof(struct linked_list));
```

2. Veriyi yeni oluşturulan elemana depola.

```
(*new_item_ptr).data = item;
```

3. Listenin yeni elamanı eski ilk elamanı işaret etsin

```
(*new_item_ptr).next_ptr = first_ptr;
```

4. Yeni eleman artık ilk eleman

```
first_ptr = new_item_ptr;
```

```
void add_list(char *item)
```

```
{
```

```
 struct linked_list *new_item_ptr; /* pointer to the next item in the list */
```

```
 new_item_ptr = malloc(sizeof(struct linked_list));
```

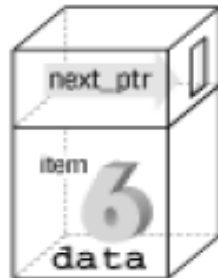
```
 strcpy((*new_item_ptr).data, item);
```

```
 (*new_item_ptr).next_ptr = first_ptr;
```

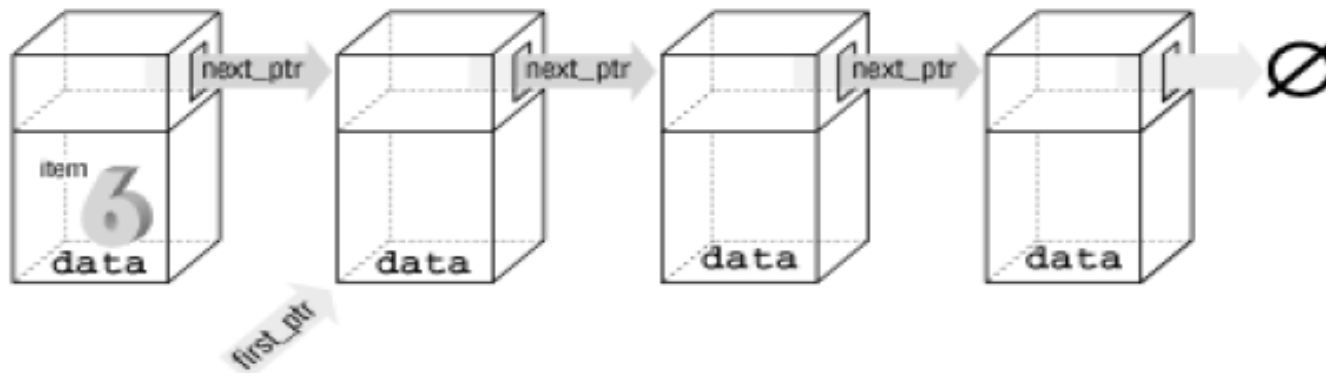
```
 first_ptr = new_item_ptr;
```

```
}
```

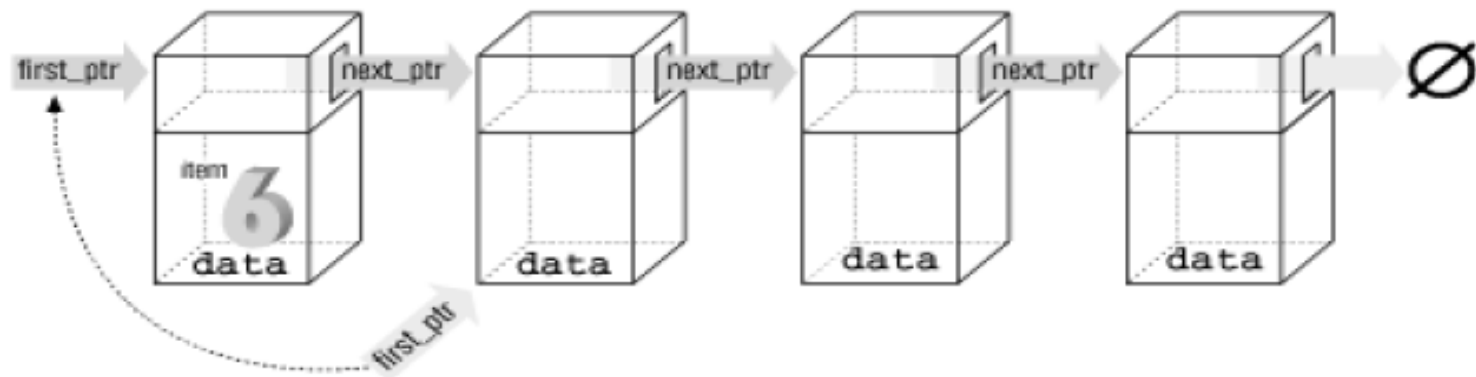
- 1 Create new element.
- 2 Store item in new element.

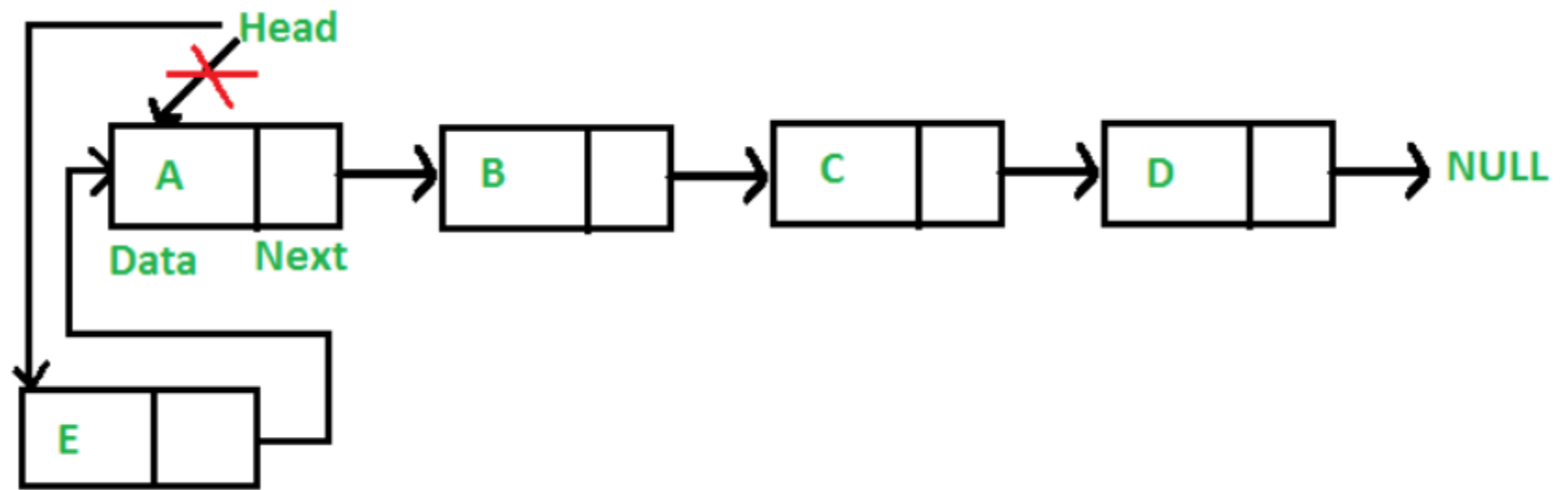


- 3 Make **next\_ptr** point to the first element.



- 4 Change **first\_ptr** to point to the new element, thus breaking the link between **first\_ptr** and the old first element.





**/\* Given a reference (pointer to pointer) to the head of a list and an int, inserts a new node on the front of the list. \*/**

**void push(struct Node\*\* head\_ref, int new\_data)**

**{**

**/\* 1. allocate node \*/**

**struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));**

**/\* 2. put in the data \*/**

**new\_node->data = new\_data;**

**/\* 3. Make next of new node as head \*/**

**new\_node->next = (\*head\_ref);**

**/\* 4. move the head to point to the new node \*/**

**(\*head\_ref) = new\_node;**

**}**

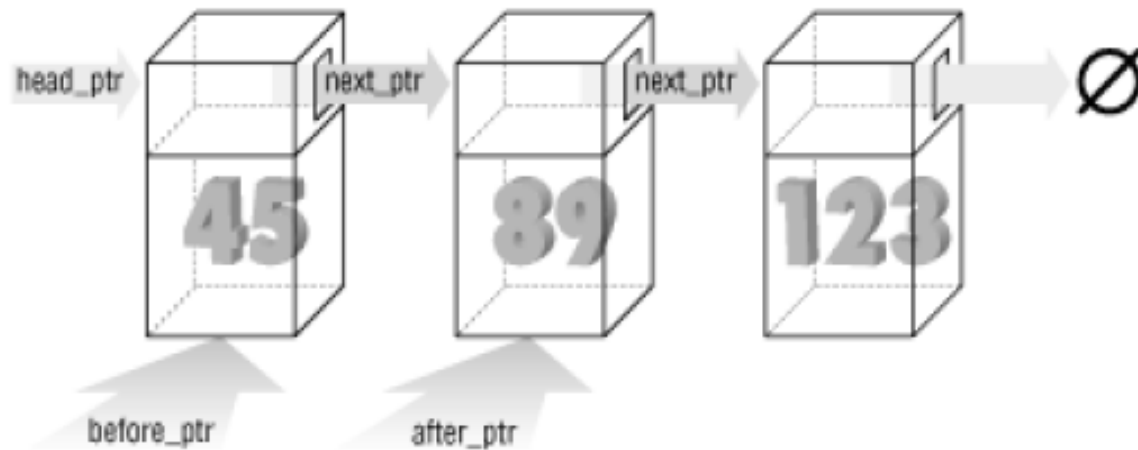
# Listede sıralı olacak şekilde eleman ekleme

```
void enter(struct item *first_ptr, const int value)
{
 struct item *before_ptr; /* Item before this one */
 struct item *after_ptr; /* Item after this one */
 struct item *new_item_ptr; /* Item to add */
 /* Create new item to add to the list */
 before_ptr = first_ptr; /* Start at the beginning */
 after_ptr = before_ptr->next_ptr;
```

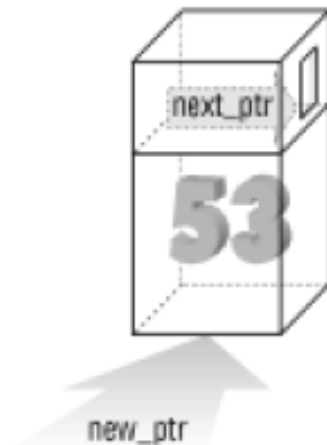
```
while (1) {
 if (after_ptr == NULL)
 break;
 if (after_ptr->value >= value)
 break;
 /* Advance the pointers */
 after_ptr = after_ptr->next_ptr;
 before_ptr = before_ptr->next_ptr;
}
/* Uygun yer bulundu*/
new_item_ptr = malloc(sizeof(struct item));
new_item_ptr->value = value; /* Set value of item */
before_ptr->next_ptr = new_item_ptr;
new_item_ptr->next_ptr = after_ptr;
}
```



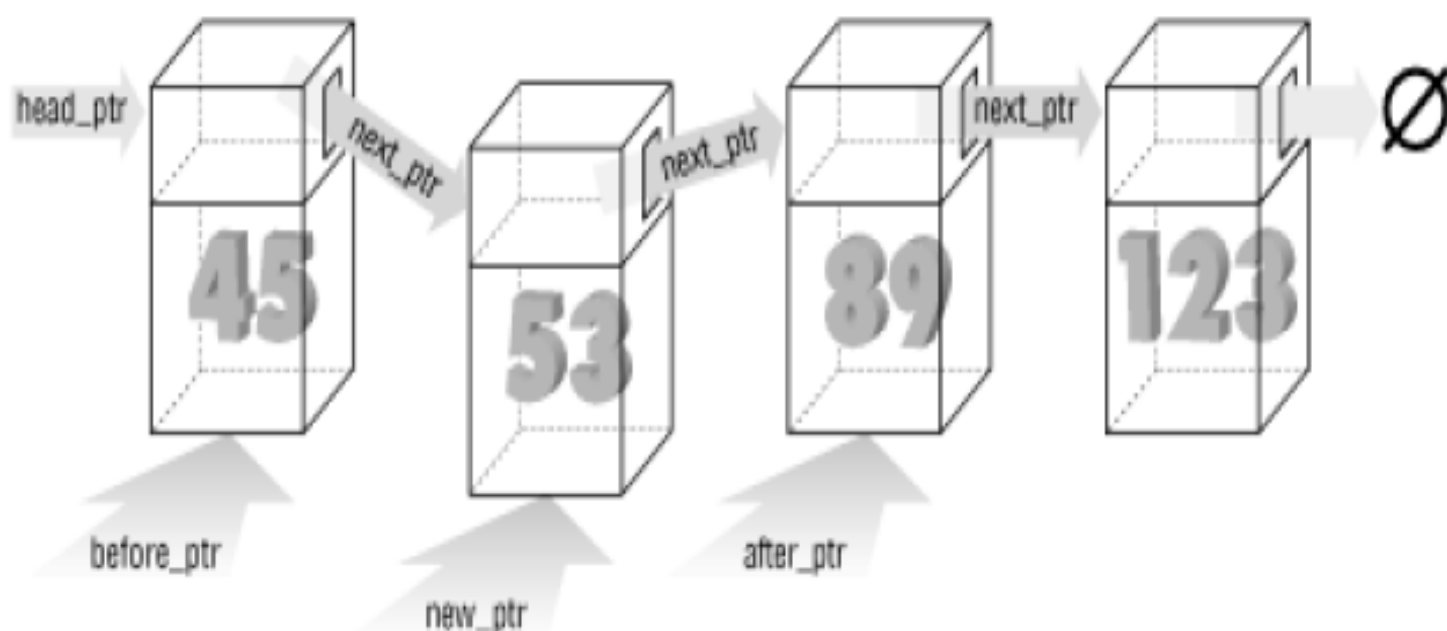
- 1 **before\_ptr** points to the elements before the insertion point, **after\_ptr** points to the element after the insertion point.



- 2 Create new element.

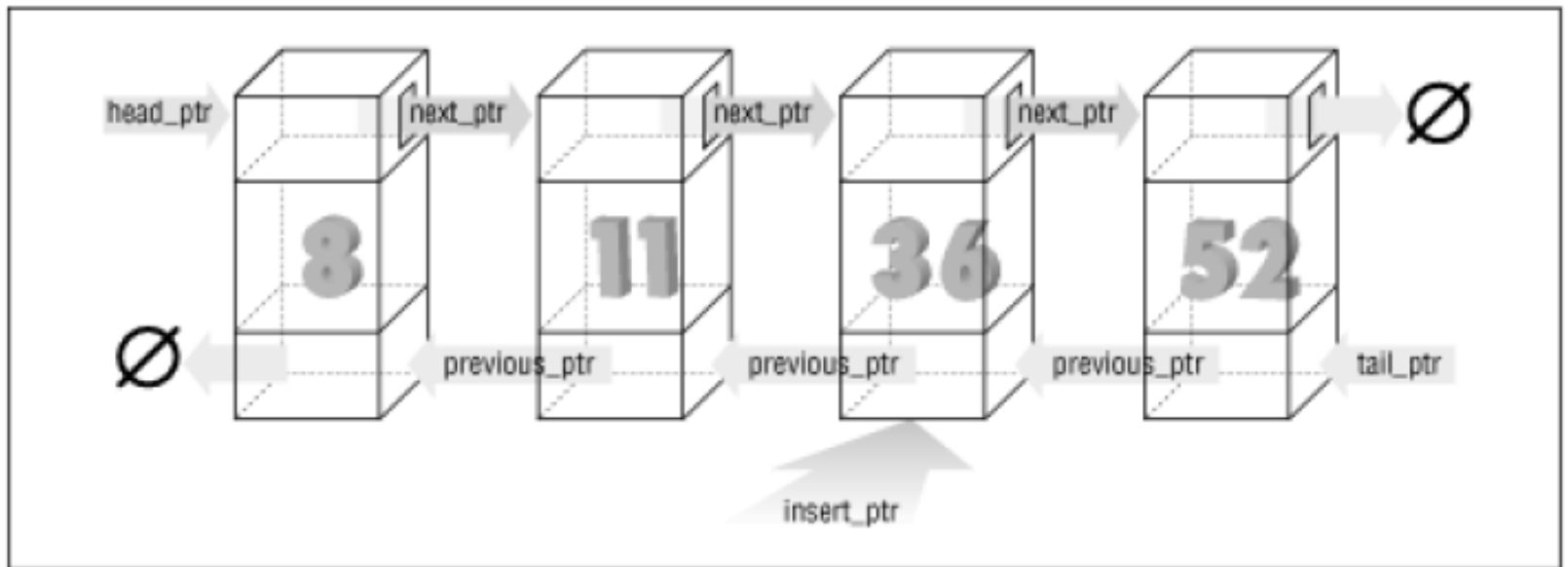


- 3 Make the **next\_ptr** of the new element point to the same element as **after\_ptr**.
- 4 Link the element pointed to by **before\_ptr** to our new element by changing **before\_ptr->next\_ptr**.



# Double-Linked Lists

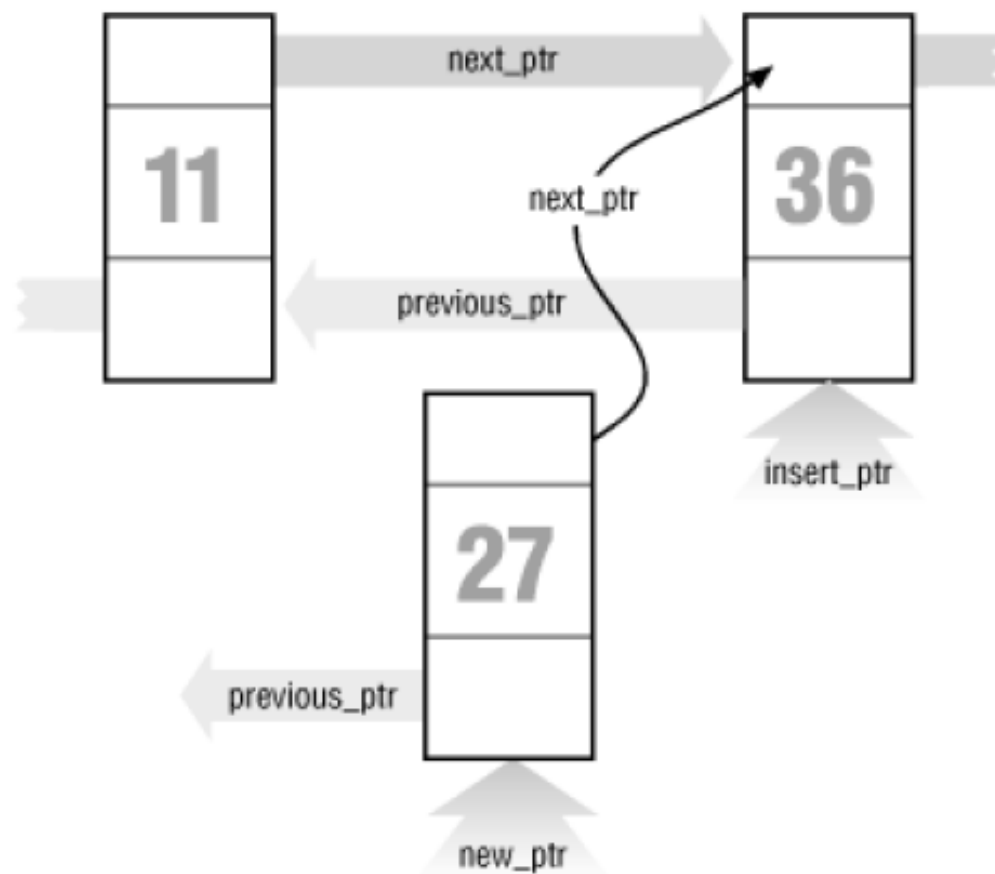
```
struct double_list {
 int data; /* data item */
 struct double_list *next_ptr; /* forward link */
 struct double_list *previous_ptr; /* backward link */
};
```



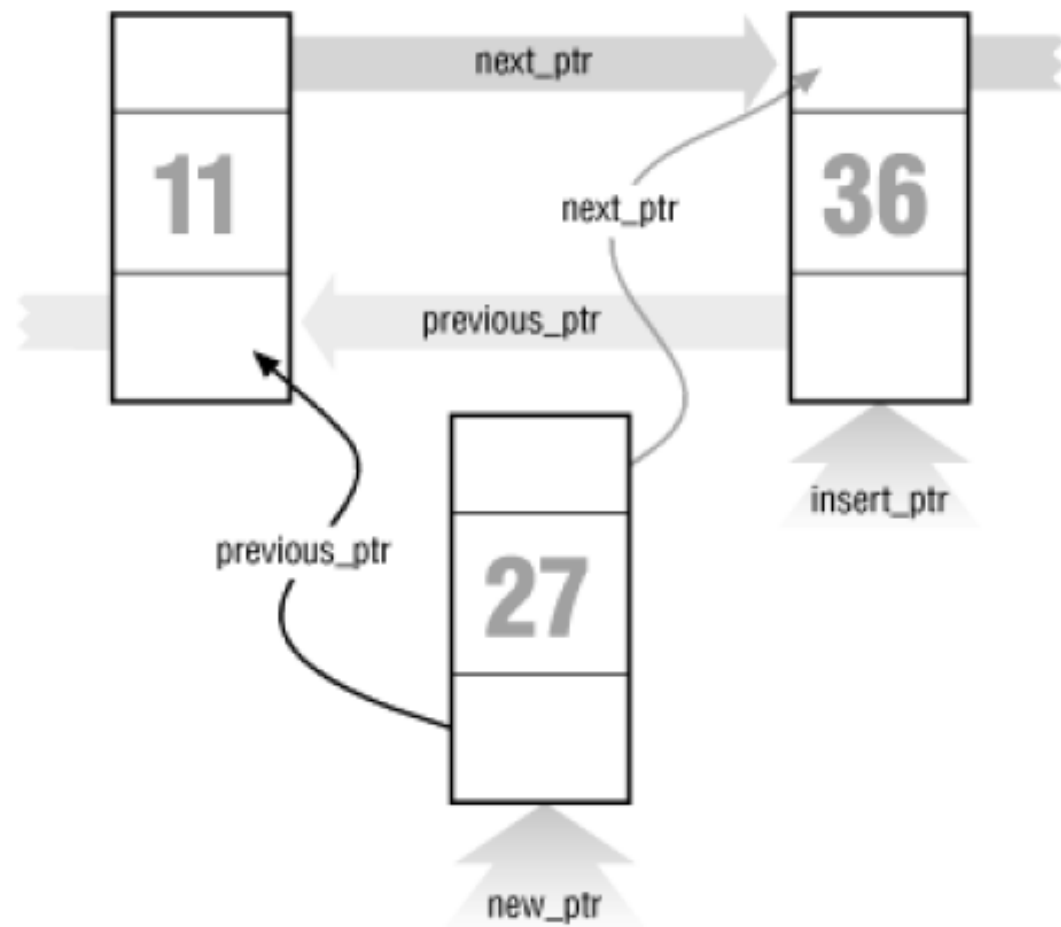
# Eleman ekleme

```
void double_enter(struct double_list *head_ptr, int item)
{
 struct list *insert_ptr; /* insert before this element */
 /****Liste başına ve sonuna eklemeler gözardı edilmiştir***/
 insert_ptr = head_ptr;
 while (1) {
 insert_ptr = insert_ptr->next;
 if (insert_ptr == NULL) /* have we reached the end */
 break;
 if (item >= insert_ptr->data) /* have we reached the right place */
 break;
 }
 new_item_ptr->next_ptr = insert_ptr;
 new_item_ptr->previous_ptr = insert_ptr->previous_ptr;
 insert_ptr->previous_ptr->next_ptr = new_ptr;
 insert_ptr->previous_ptr = new_item_ptr;
```

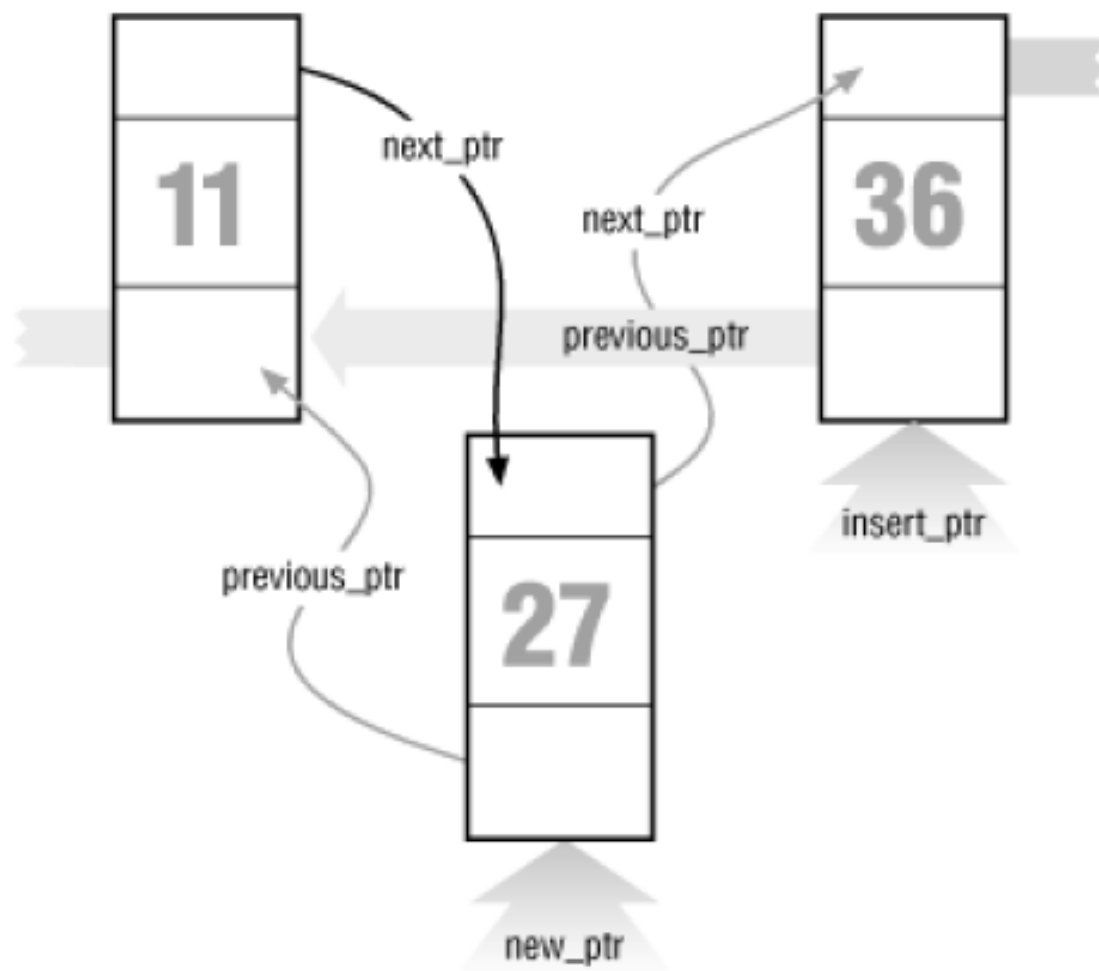
```
new_ptr->next_ptr = insert_ptr;
```



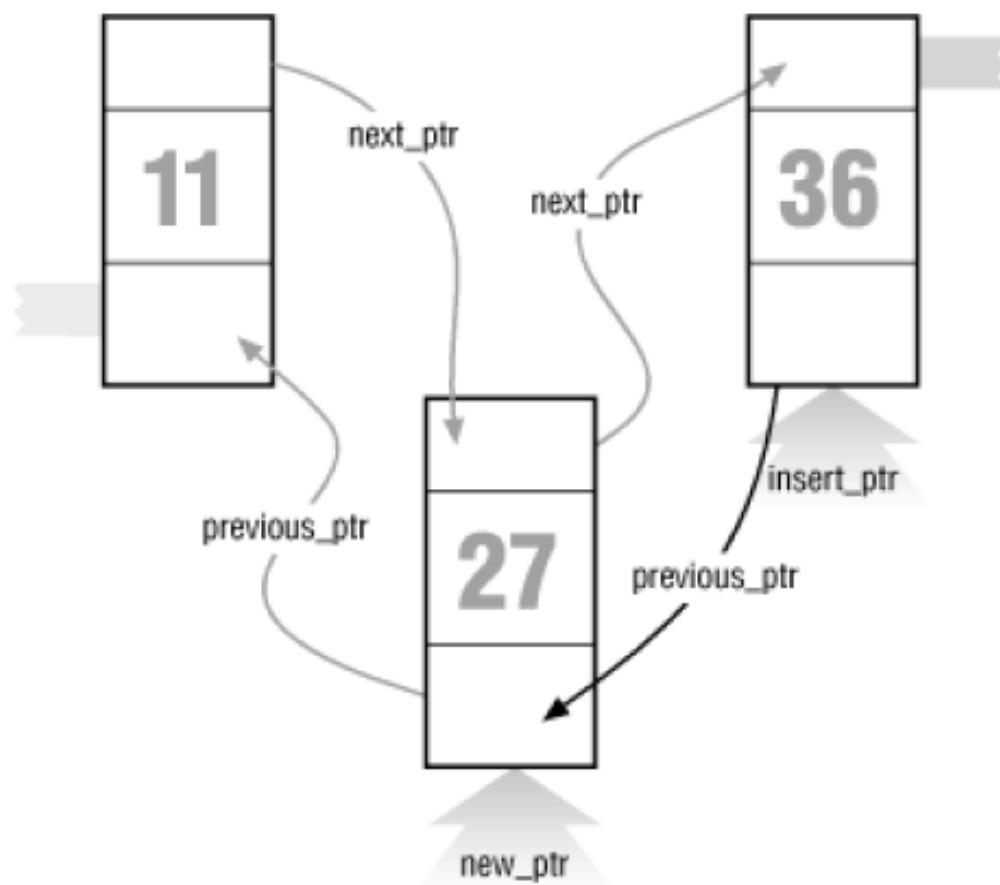
```
new_ptr->previous_ptr = insert_ptr->previous_ptr;
```



```
insert_ptr->previous_ptr->next_ptr = new_ptr;
```



```
insert_ptr->previous_ptr = new_ptr;
```







- **DELETING THE SPECIFIED NODE IN A SINGLY LINKED LIST**

To delete a node, first we determine the node number to be deleted (this is based on the assumption that the nodes of the list are numbered serially from 1 to n). The list is then traversed to get a pointer to the node whose number is given, as well as a pointer to a node that appears before the node to be deleted. Then the link field of the node that appears before the node to be deleted is made to point to the node that appears after the node to be deleted, and the node to be deleted is freed

```
include <stdio.h>
include <stdlib.h>
struct node *delet (struct node *, int);
int length (struct node *);
struct node {
int data;
struct node *link;
};
```

```
struct node *insert(struct node *p, int n) {
struct node *temp;
 if(p==NULL) {
p=(struct node *)malloc(sizeof(struct node));
 if(p==NULL) {
 printf("Error\n");
 exit(0);
 }
p->data = n;
p->link = NULL;
 }

 else {
 temp = p;

 while (temp->link != NULL)
temp = temp->link;
 temp->link = (struct node *)malloc(sizeof(struct node));
 if(temp -> link == NULL) {
 printf("Error\n");
 exit(0);
 }

temp = temp->link; temp->data = n;
temp->link = NULL;
 }

return (p);
}
```

```
void printlist (struct node *p)
{
printf("The data values in the list are\n");
 while (p!= NULL)
 {
 printf("%d\t",p-> data);
 p = p-> link;
 }
}

void main()
{
int n;
int x;
struct node *start = NULL;
printf("Enter the nodes to be created \n");
scanf("%d",&n);
```

```
while (n- > 0)
{
printf("Enter the data values to be placed in a node\n");
scanf("%d",&x);
start = insert (start, x);
}
```

```
printf(" The list before deletion id\n");
printlist (start);
printf("% \n Enter the node no \n");
scanf (" %d",&n);
start = delet (start , n);
printf(" The list after deletion is\n");
printlist (start);
```

```
}
```

```
/* a function to delete the specified node*/
struct node *delet (struct node *p, int node_no)
{
 struct node *prev, *curr ;
 int i;
 if (p == NULL)
 {
 printf("There is no node to be deleted \n");
 }
 else
 {
 if (node_no > length (p))
 {
 printf("Error\n");
 }
 else
 {
 prev = NULL;
 curr = p;
 i = 1 ;
```

```
while (i < node_no)
```

```
{
```

```
prev = curr;
```

```
curr = curr-> link;
```

```
i = i+1;
```

```
}
```

```
if (prev == NULL)
```

```
{
```

```
p = curr -> link;
```

```
free (curr);
```

```
}
```

```
else
```

```
{
```

```
prev -> link = curr -> link ;
```

```
free (curr);
```

```
}
```

```
}
```

```
}
```

```
return(p);
```

```
}
```

/\* a function to compute the length of a linked list \*/

```
int length (struct node *p)
```

```
{
```

```
int count = 0 ;
```

```
while (p != NULL)
```

```
{
```

```
count++;
```

```
p = p->link;
```

```
}
```

```
return (count) ;
```

```
}
```



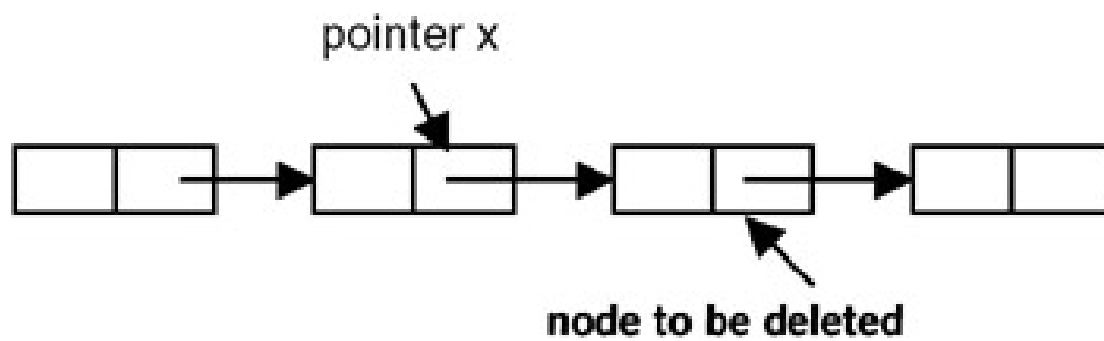
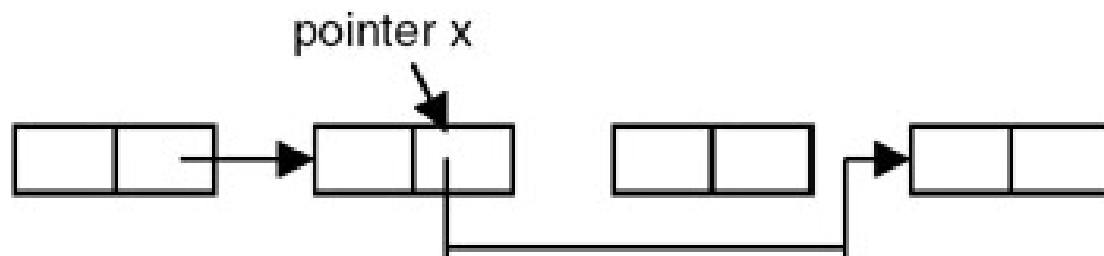


Figure 20.5: Before deletion.





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

struct test_struct{
int val;
struct test_struct *next;
};

struct test_struct *head = NULL;
struct test_struct *curr = NULL;

struct test_struct* create_list(int val){
printf("\n creating list with headnode as [%d]\n", val);
struct test_struct *ptr = (struct test_struct*)malloc(sizeof(struct test_struct));
if (NULL == ptr){
printf("\n Node creation failed \n");
return NULL;
}
ptr->val = val;
ptr->next = NULL;

head = curr = ptr;
return ptr;
}
```

```

struct test_struct* add_to_list(int val, bool add_to_end){
if (NULL == head){
return (create_list(val));
}

 if (add_to_end)
printf("\n Adding node to end of list with value [%d]\n", val);
 else
printf("\n Adding node to beginning of list with value [%d]\n", val);

struct test_struct *ptr = (struct test_struct*)malloc(sizeof(struct test_struct));
 if (NULL == ptr){
 printf("\n Node creation failed \n");
 return NULL; }
ptr->val = val;
ptr->next = NULL;
 if (add_to_end){
curr->next = ptr;
curr = ptr; }
 else{
ptr->next = head;
head = ptr;}

return ptr;}

```

```
struct test_struct* search_in_list(int val, struct test_struct **prev){
 struct test_struct *ptr = head;
 struct test_struct *tmp = NULL;
 bool found = false;
 printf("\n Searching the list for value [%d] \n", val);

 while (ptr != NULL){
 if (ptr->val == val){
 found = true;
 break;
 }
 else{
 tmp = ptr;
 ptr = ptr->next;
 }
 }
 if (true == found){
 if (prev)
 *prev = tmp;
 return ptr;
 }
 else{
 return NULL;
 }
}
```

```
int delete_from_list(int val){
 struct test_struct *prev = NULL;
 struct test_struct *del = NULL;
 printf("\n Deleting value [%d] from list\n", val);

 del = search_in_list(val, &prev);
 if (del == NULL){
 return -1;
 }
 else{
 if (prev != NULL)
 prev->next = del->next;
 if (del == curr){
 curr = prev;
 }
 else if (del == head){
 head = del->next;
 }
 }
 free(del);
 del = NULL;

 return 0;
}
```

```
void print_list(void)
{
 struct test_struct *ptr = head;

 printf("\n -----Printing list Start----- \n");
 while (ptr != NULL)
 {
 printf("\n [%d] \n", ptr->val);
 ptr = ptr->next;
 }
 printf("\n -----Printing list End----- \n");

 return;
}
```

```
int main(void){
int i = 0, ret = 0;
struct test_struct *ptr = NULL;
print_list();
for (i = 5; i<10; i++)
add_to_list(i, true);
print_list();
for (i = 4; i>0; i--)
add_to_list(i, false);
print_list();
for (i = 1; i<10; i += 4){
ptr = search_in_list(i, NULL);
if (NULL == ptr){
printf("\n Search [val = %d] failed, no such element found\n", i);
}
else{
printf("\n Search passed [val = %d]\n", ptr->val);
}
print_list();
ret = delete_from_list(i);
if (ret != 0){
printf("\n delete [val = %d] failed, no such element found\n", i);
}
else{
printf("\n delete [val = %d] passed \n", i);
}
print_list();
}
return 0;}
```



