

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;

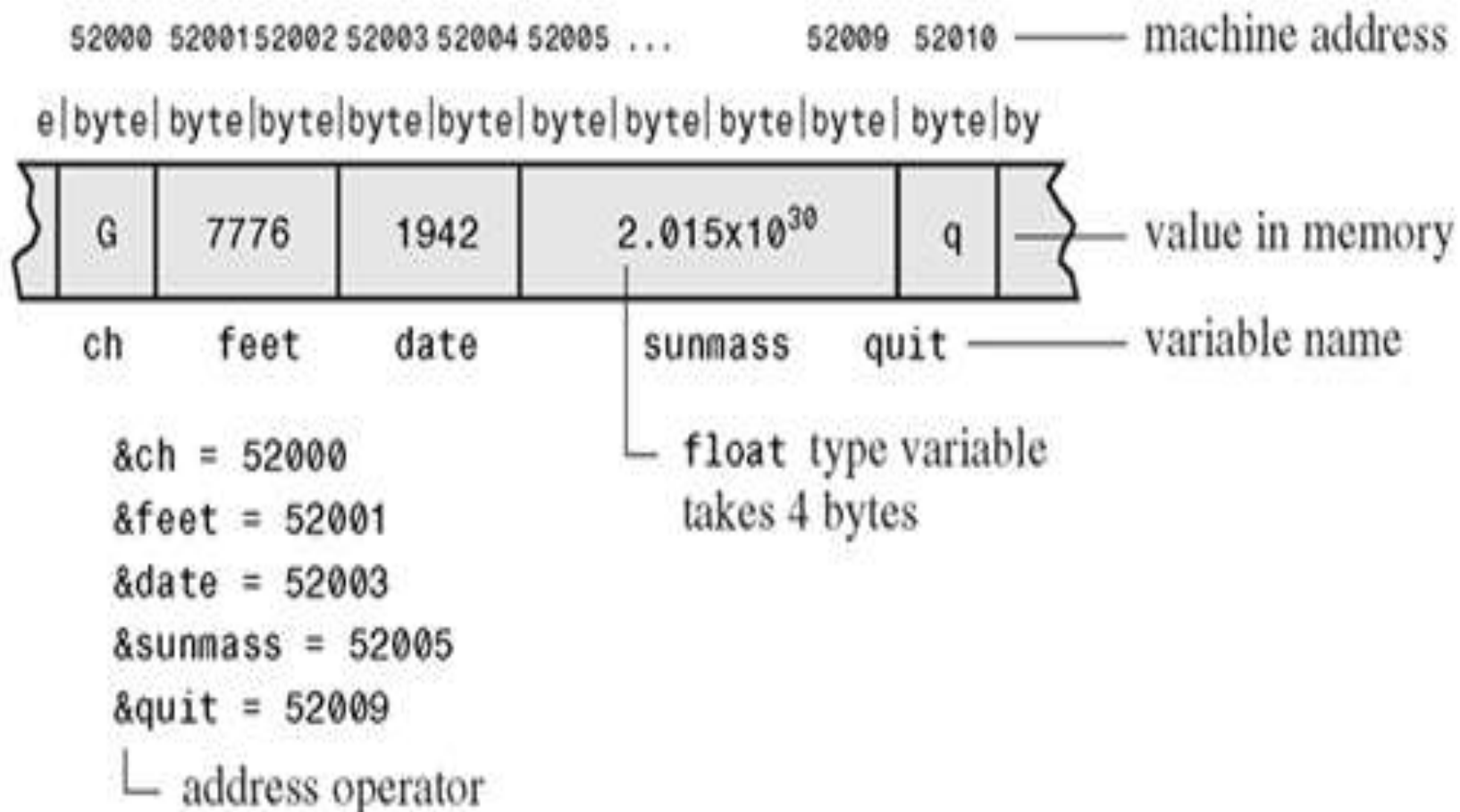
Pointers

- 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**).



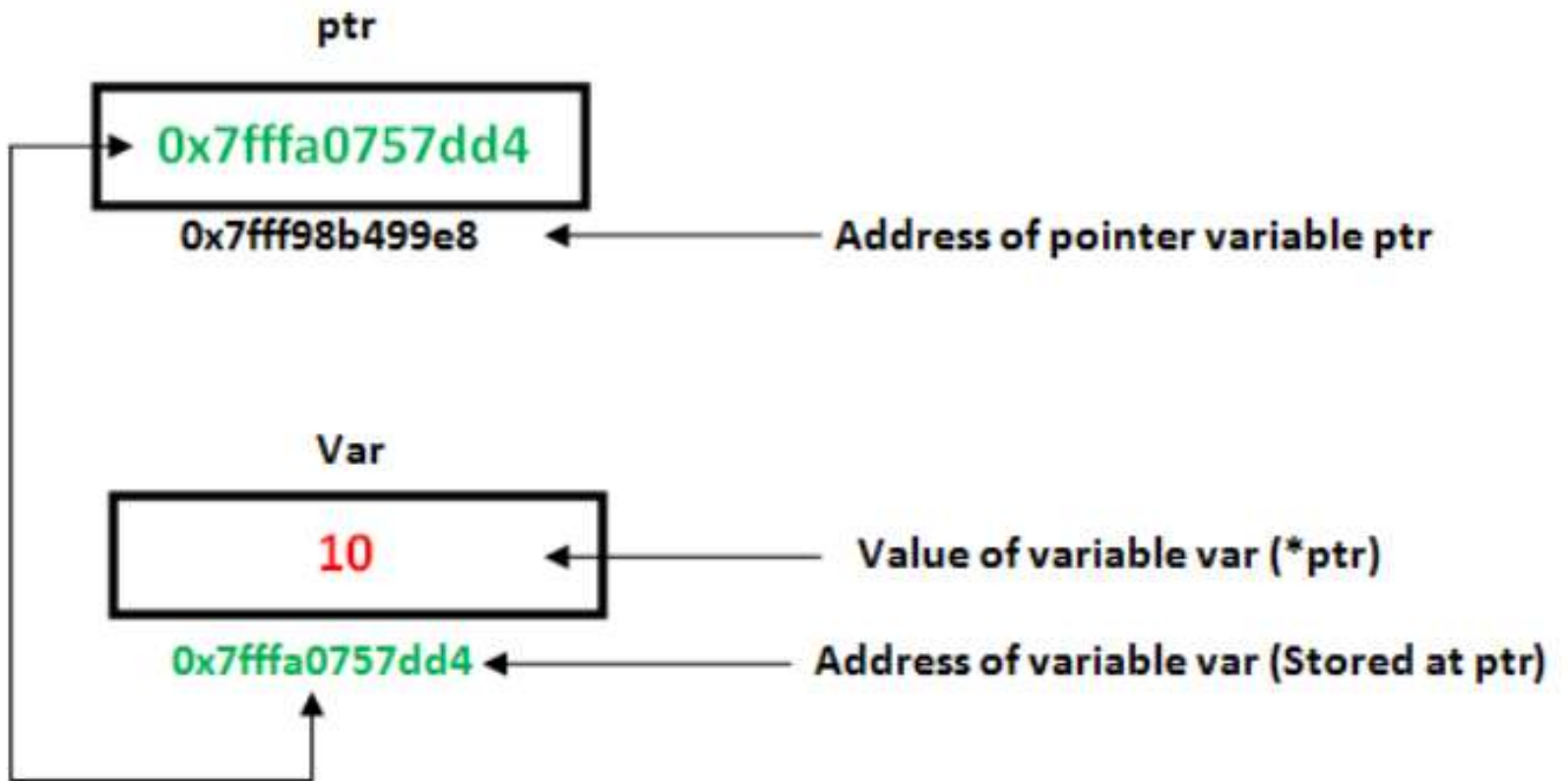
```
#include <stdio.h>

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);

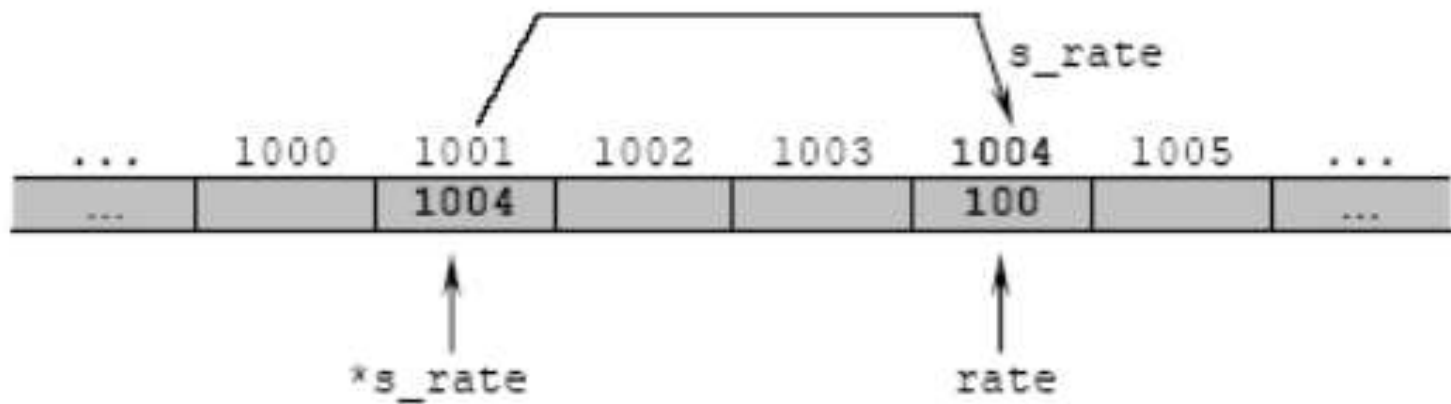
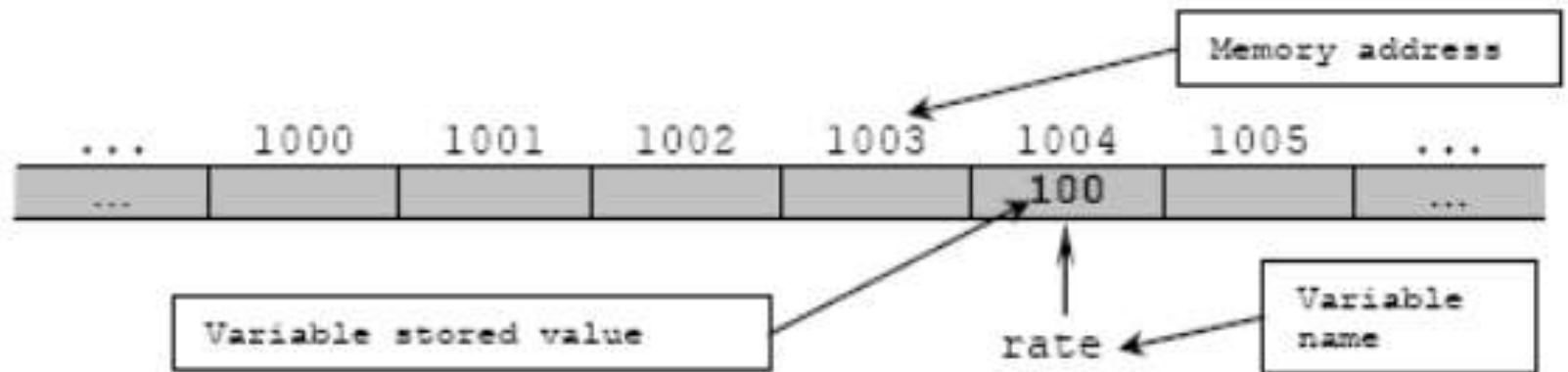
    /* değişken adresinin iki yollan yazılması */
    printf("\n\n var adresi = %d", &var);
    printf("\n var adresi = %d\n", ptr);
    return 0;
}
```



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;
```

```
}      ÖNEMLİ!
```

```
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
```

dizinin ismi ayn zamanda dizinin adresidir.

```
void display_array(int*,const int);
```

```
int main(){
```

```
    int i;    int a[MAX];    int* pa;
```

```
        pa = a;
```

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

pa" dizinin elamanlarini temsil eder. pa+i dizinin i sonrasindaki elamani temsil eder.

```
        *(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");}
```

sorular bu sekilde olacak!!


```
#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;  
}
```

burdaki adresi yukar yolladk

Pointers ve Diziler

- Köşeli parentez 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()
```

```
{
```

dizinin ismini adres atamada direkt kullanabiliriz.

```
    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;
}
```

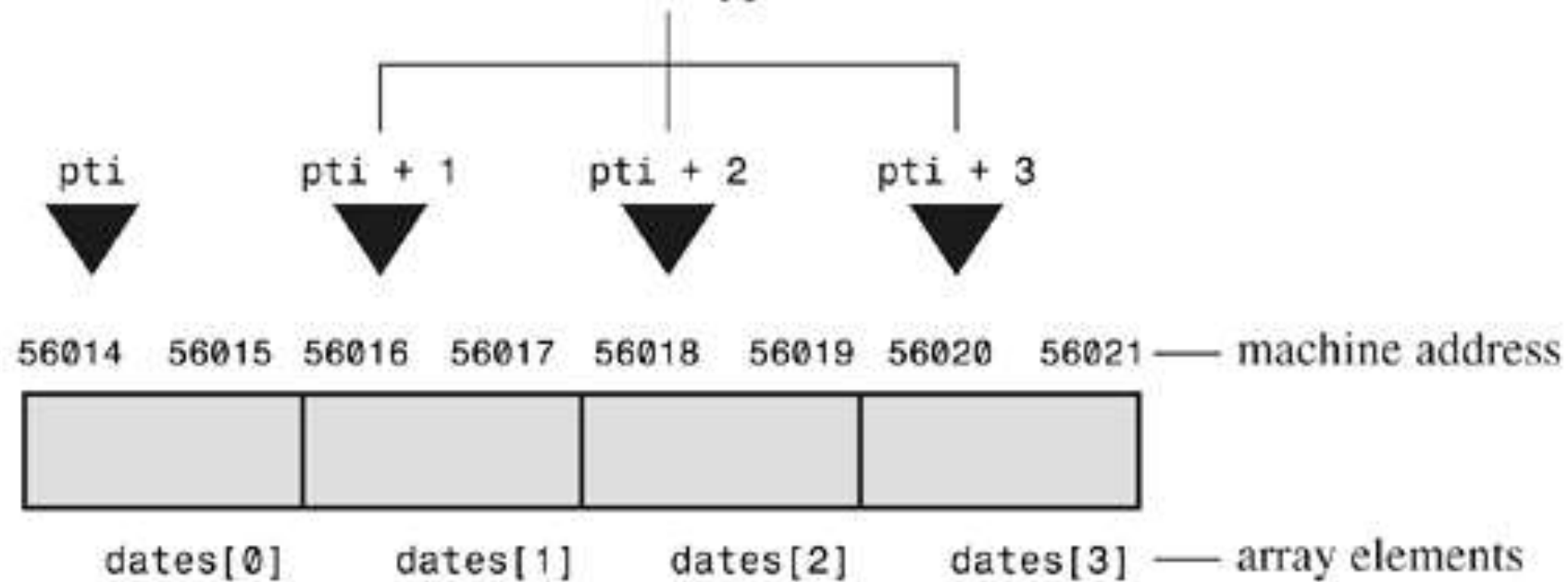
Pointer Aritmatik

- 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

pointerlara sayisal degerler vermiyoruz.

- Dikkat:

Bir pointer tanımladınız

```
Int *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 adress 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]`

pointer aritmetigini ifade

- ...

- `*(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) );
```

```
    }
```

elamanin içeriklerini yazmis olacak

p: i. elaman

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

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

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

```
        }
```

dizinin ismi adresstir!!

```
    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); = printf("*ptr-new = %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 - 2, 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.

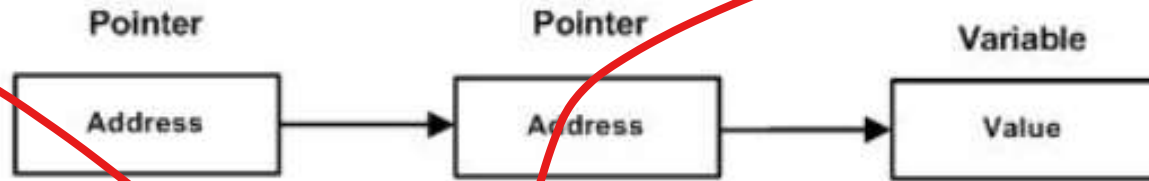
2) 20 elemanlık, elemanları random atanmış bir dizi oluşturun. Bir fonksiyon yazın: fonksiyon dizinin en büyük elemanının indeksini bulup yazsın. İkinci bir fonksiyon dizinin en büyük elemanı ile en küçüğünün farkını bulup yazsın.

```
#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;
}
```

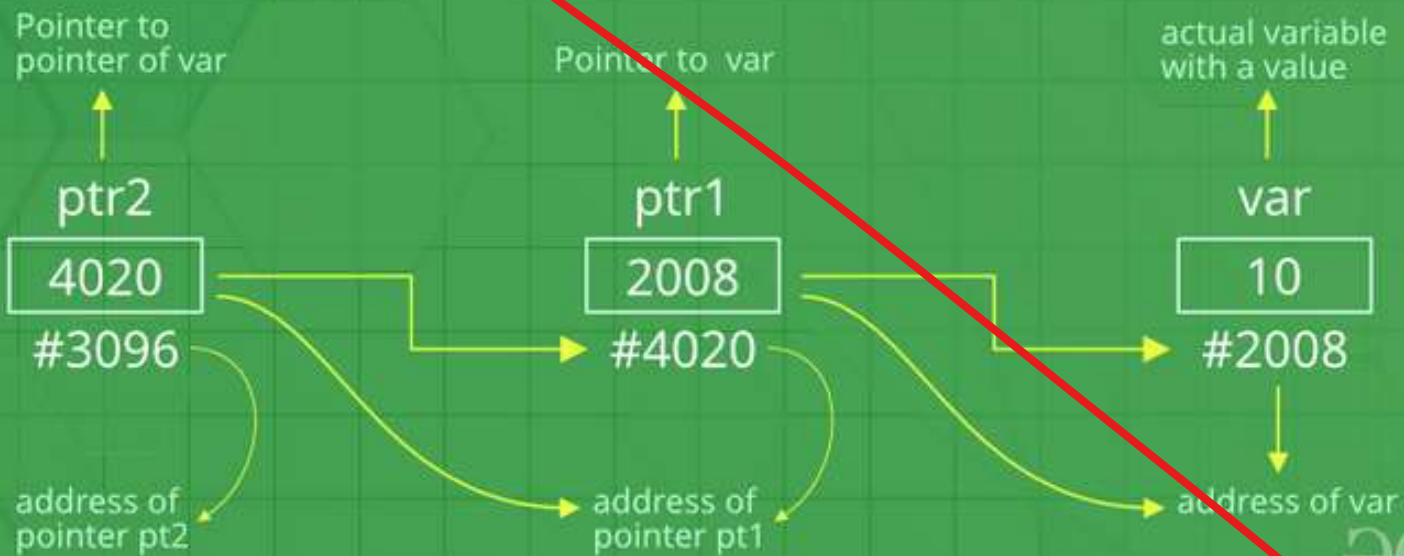
Pointers to Pointers



```
int *ptr; //pointer deklere et
ptr = &x; //değişkenin adresini ata
x = 12;
*ptr = 12;

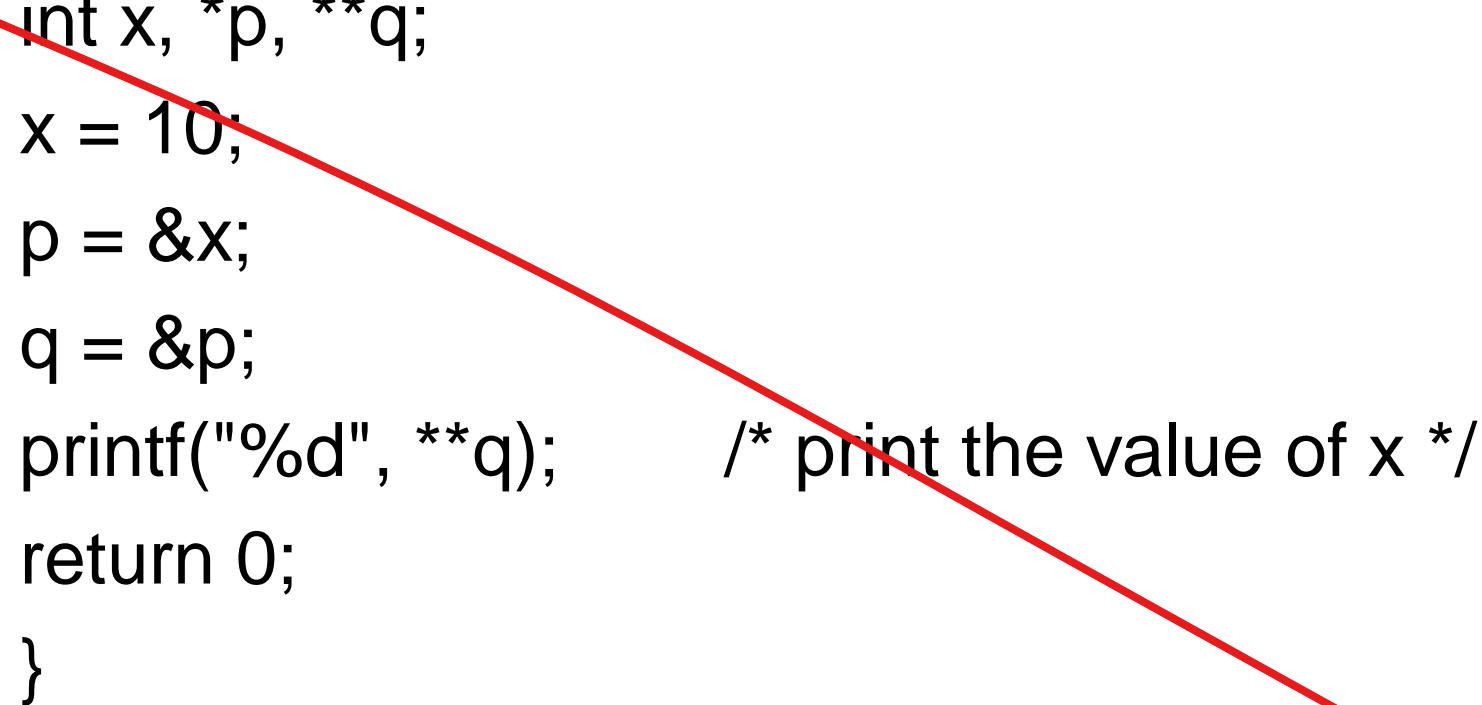
int x = 12;
int *ptr = &x;
int **ptr_to_ptr = &ptr; // pointer to pointer
**ptr_to_ptr = 12;
printf("%d", **ptr_to_ptr);
```

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>

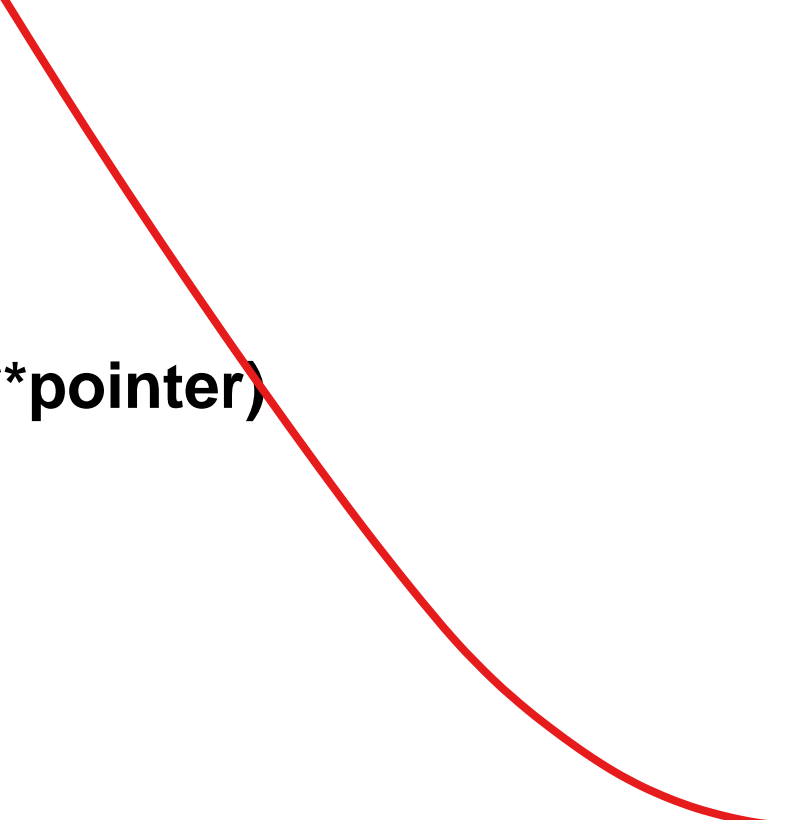
int main () {
    int var;
    int *ptr;
    int **pptr;
    var = 3000;
    ptr = &var; /* take the address of var */
    /* take the address of ptr using address of operator & */
    pptr = &ptr;
    printf("Value of var = %d\n", var );
    printf("Value available at *ptr = %d\n", *ptr );
    printf("Value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

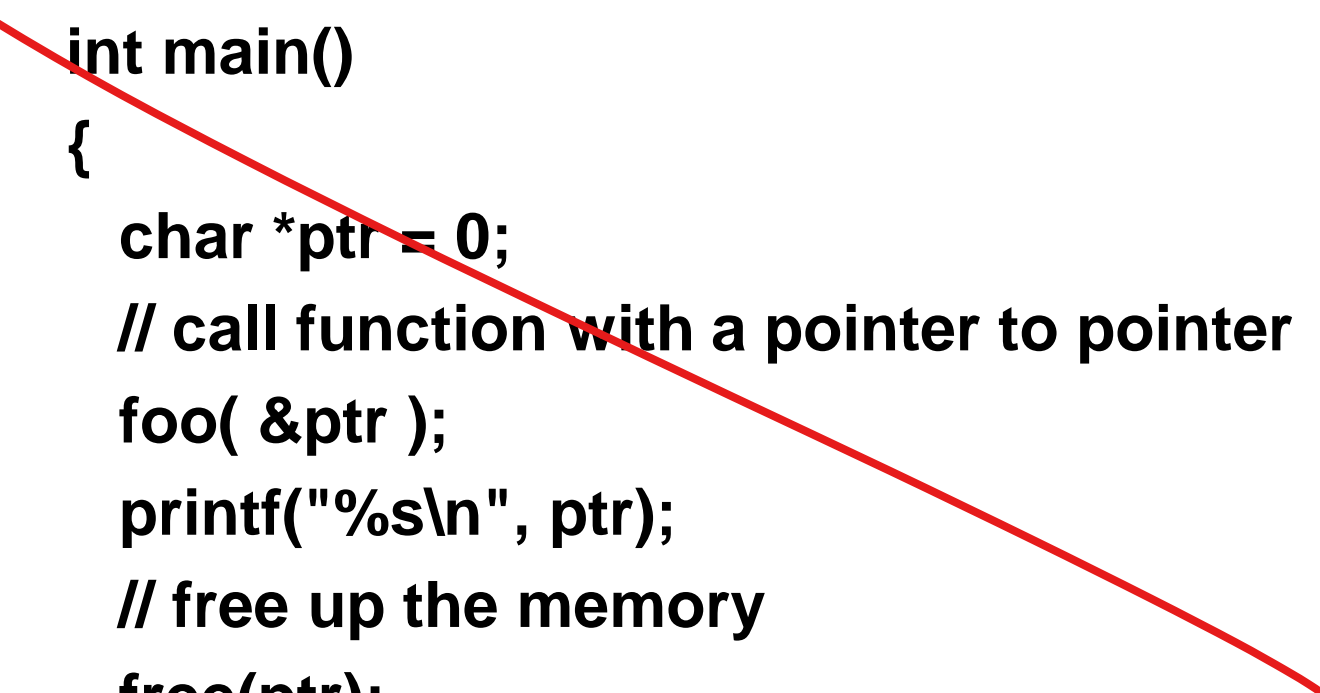


```
void func(int **pointer)  
{  
    ...  
}
```

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

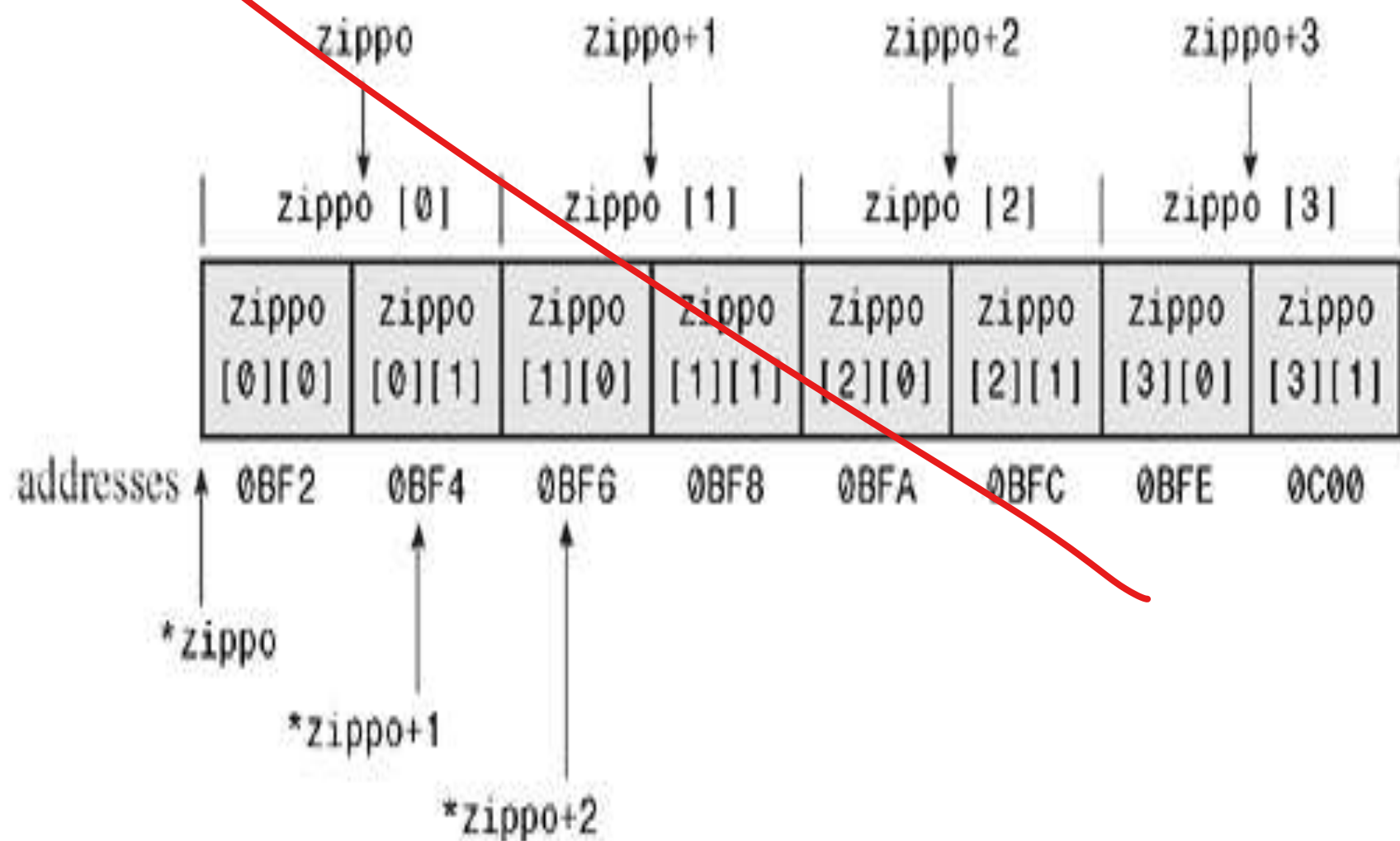
```
void foo( char ** ptr)
{
    *ptr = malloc(255); // allocate some memory
    strcpy( *ptr, "Hello World");
}

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 artırımı 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;
```

bunun tek yildizlisi cikabilir

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
//karakterini 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 girişı

`char *name;`

`scanf("%s", name);` // isim girişı yapılır, fakat kaç harften oluştuđu belli deđil, name pointer'ı ile gösterilen yazılır. İlk harften başlar ilk boşluđa kadar okur.

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

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

`char name[81];` // řimdi sınırlandırdık ve bizim tanımladıđımız 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;
}
```