Bir değişkenin adresini bulmak

- Her değişkenin hafızada bulundugu yeri gosteren özel bir adresi vardır. Bu adresi almak için ampersand (&) operatörü kullanılır.

```
long int j;       //        j , adresi 62FE4C olan long int bir degisken
ptr = &j;         //        ptr bu adresi saklayan bir degisken
```

```
#include <stdio.h>
void main() {
                int j=1;
                printf( "The value of j is: %d\n" , j );
                printf( "The address of j is: %p\n" , &j );
                //%p adresi portable ve hex olarak yazmayi saglar
}
```

The value of j is: 1

The address of j is: 000000000062FE4C

- & işareti solda kullanılamaz. Bir değişkenin adresi değiştirilemez.

```
&x = 1000; /* ILLEGAL */
```

- ptr = &j; // ptr 'in tipi ne olmalıdır?

Adres tutmak için pointer adı verilen ozel bir tip kullanılır.

```
long *ptr;       /* Long, ptr'nin gosterdiği adreste bulunan verinin tipini gosterir*/
long long_var;
ptr = &long_var;       /*LEGAL*/
```

```
long *ptr;
float float_var;
ptr = &float_var;       /* ILLEGAL - ptr sadece bir long int'in adresini tutabilir */
```

-

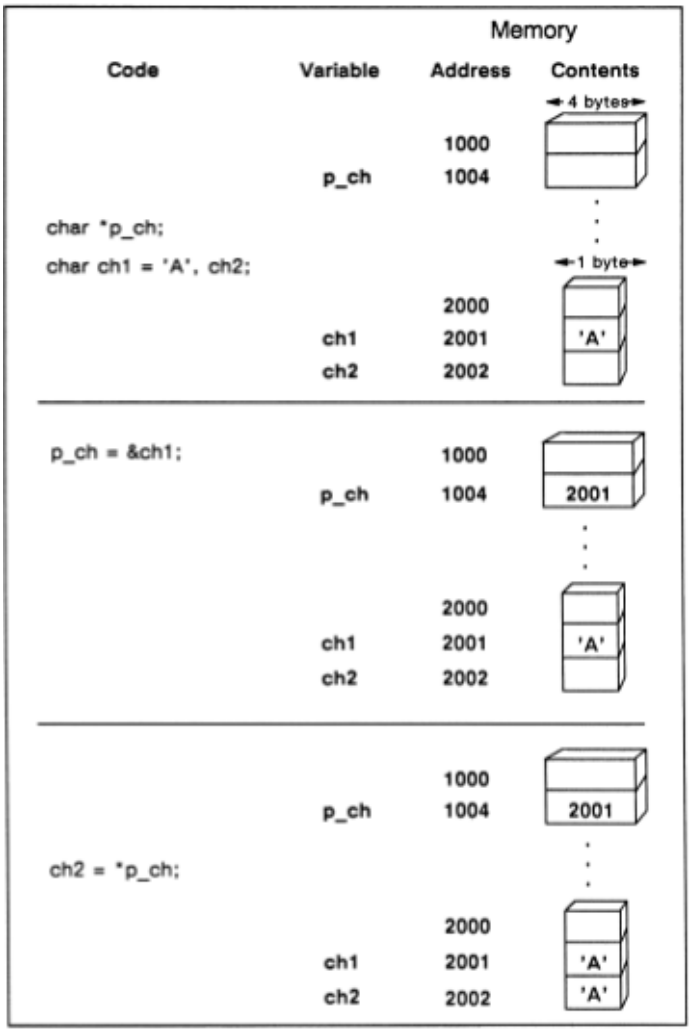
```
#include <stdio.h>
int main () {
        int j=1;
         int *pj;
        pj = &j;
        printf("The value of j is: %d\n", j);
        printf("The adres of j is: %p\n", pj);
        return 0;
        }
```

The result is:
The value of j is: 1

The address of j is: 000000000062FE4C

Dereferencing

- (*) Operatörü hem pointer tanımlamak hem de o pointer'in gosterdigi adreste saklanan veriye ulasmak icin kullanır.




```
#include <stdio.h>
int main ()
{
char *p_ch;
char ch1 = 'A', ch2;
printf ( "The address of p_ch is %p\n", &p_ch );
p_ch = &ch1;
printf ( "The value stored at p_ch is %p\n", p_ch );
printf( "The dereferenced value of p_ch is %c\n" , *p_ch );
ch2 = *p_ch;
printf ( "The value stored at ch2 is %c\n", ch2 );
```

```
return  0;
}
```

The address of p_ch is 000000000062FE40

The value stored at p_ch is 000000000062FE3F

The dereferenced value of p_ch is A

The value stored at ch2 is A

- Pointer initialization:

```
int j;                                  int j;
int *ptr_to_j = &j;    //LEGAL          int *ptr_to_j = &j;   //ILLEGAL
```

Pointer Arithmetic

- C dili pointer ile integer'lari isleme sokmaya izin verir. p bir pointer olmak uzere,

p+3 legal bir işlemdir.

- p bir adres olduğu için p ile yapılan aritmetik işlem de adres üretir. Ancak basitçe p'nin değerine 3 eklemektense SCALING yapılır. p hangi tipte verinin adresini tutarsa tutsun, p+3 p'den 3 nesne sonrayı işeret eder

```
char *p;
char c='A';
p=&c;                   //p=1000 olsun
p=p+3;                  // p = 1001 olur
```

- 32 bitlik bir sistemde p int* olarak tanımlansaydı;

```
p=p+3                   //p= 1000+ (4*3) = 100C
```

Dizi elemanlarına pointer uzerinden ulasma

- short ar[4];

```
short *p;
p = &ar[0] ;
*(p+3) // ar[3]
```

- Dizi isimleri tek başlarına kullanıldıklarında pointer olarak işleme alınırlar

ar <-> &ar[0]

ar[n] <-> *(ar + n)

Fonksiyonlar

- Fonksiyonlar C dilinin temelidir ve tüm işlemin gerçekleştiği yerlerdir.
- Her fonksiyon, kendi başına bir kod bloğudur. Bu yüzden fonksiyonlar kendi tanım alanlarını oluştururlar.
- Fonksiyon bloğunun yapısı;

```
dönüş_tipi fonksiyon_adi(parametre listesi)
{
        işlem bloğu
}
```

- dönüş_tipi, fonksiyon sonlandiginda fonksiyonun çağırıldığı noktaya gidecek olan değerin tipini belirtir.
- Fonksiyonlar aynı anda bir deger dondurebilir. Deger dondurmeyebilir (void)
- Parametre listesi, fonksiyon çağırılırken kullanılması gereken değişkenler ve onların veri tiplerinden olusur. Fonksiyon parametre kullanmadan da çağırılabilir.
- Fonksiyon bir programda üç şekilde görünebilir: (Prototip tanımı, Fonksiyon gövdesi ve Fonksiyonun çağırılması)
- C'de fonksiyona parametre gönderme işlemi 2 ayrı şekilde yapılır: Değer ile ve referans ile çağırma.
- Fonksiyona gönderilen parametrelerde fonksiyon tarafından yapılan değişikliklerin ana fonksiyona yansımasını sağlamak için refernas ile çağırma yapılır.
- Toplama, çarpma, kare alma ve swap işlemleri yapan fonksiyonlar yazalım.