C'ye Giriş

ILK C PROGRAMINIZ

En basit C programi:

```
main()
{
}
```

Bu bir programdir, ve bunu kisaltmanin, basitlestirmenin bir yolu yoktur.Isin kotu tarafi, bu program birsey yapmaz. Buradaki en onemli kelime, main() sozcugudur. Her programda olmasi gereken bu sozcuk, daha sonra gorecegimiz gibi, ilk satirda olmak zorunda degildir, fakat bir giris noktasi saglamasi nedeni ile gereklidir. Bunu takiben, iki tane parantez vardir. Bunlar da, main'in bir fonksiyon oldugunu belirtir. (Bir fonksiyonun tam olarak nelerden olustugunu daha sonra gorecegiz) Programin kendisi ise, iki kume isareti arasinda yer alir.

BIRSEYLER YAPAN BIR PROGRAM:

Daha ilginc bir program:

```
main()
{
printf("Bu bir satirlik yazidir.");
}
```

Bu programin, ayni diger program gibi, main, ve kume isaretleri vardir. Icinde yer alan fonksiyonun, bir satiri ekrana getirmesi icin, satiri " " isaretleri arasina aliyoruz. Ayrica

fonksiyonun parametresi oldugunu belirtmek icin de, cevresine parantez koyuyoruz

Satirin sonundaki noktali virgule dikkatinizi cekerim: Bir satirin bittigini derleyiciye bildirmek icin, C dilinde ; noktali virgul kullanilir.

DAHA COK SEY YAZAN BIR PROGRAM

```
main()
{
printf("Bu bir satirlik yazidir.\n");
printf("Bu bir baska ");
printf(" satirdir.\n");
printf("Bu ucuncu satirdir.\n");
}
```

Bu programda, 4 tane islenecek komut vardir. Satirlar bu sirada islenir. Ilk satirin sonundaki tuhaf ters bolu isareti, ondan sonra gelecek karakterin bir kontrol karakteri oldugunu belirtiyor. Bu durumda n harfi, yeni bir satir istegini belirtir. Yani, cursor, ekranin sol basina, ve bir satir asagiya kayar. Katarin herhangi bir yerinde yeni bir satir isteyince, "\n" komutunu verebilirsiniz. Hatta bir kelimenin ortasina bile koyup, kelimeyi iki satira bolebilirsiniz.

Ilk komut, metini ekrana yazar, ve bir satir asagi iner. Ikinci komut, yazdiktan sonra, yeni satir yapmadan, ucuncu komutun icindekileri ekrana yazar. Bu komutun sonunda, yeni satira gecilir. Dorduncu komut ise, ucuncu satiri yazar, ve bir return karakteri sunar.

RAKAMLAR YAZALIM

```
main()
{
```

```
int index;
index = 13;
printf("Indexin degeri simdi %d\n",index);
index = 27;
printf("Indexin degeri simdi %d\n",index);
index = 10;
printf("Indexin degeri simdi %d\n",index);
}
```

Bu programda ise, ilk defa olarak bir degisken kullaniyoruz. main() ve { isaretlerine artik alismis olmalisiniz. Bunun altinda "int index" diye bir satir yer aliyor. Bu satir, "index" isimli bir tamsayi degiskenini tanimliyor. Cogu mikrobilgisayarlar icin, 'int' tipi bir degiskenin alabilecegi degerler, -32768 ila 32767 dir. 'index' ismi ise, TANIMLAYICILAR da bahsettigimiz kurallara uyan herhangi birsey olabilir. Bu satirin sonunda da, satirin bittigini belirten ; noktali virgul yer alir.

Bir satirda birden fazla tam sayi tanimlanabilir, fakat henuz biz bununla ortaligi karistirmayacagiz.

Programi incelerken, uc tane atama satiri oldugunu, ve bu satirlarin altinda da degerlerin yazildigini goruyoruz. Once 13 atanir, ve ekrana yazilir, sonra 27 ve 10.

RAKAMLARI NASIL YAZARIZ

Sozumuze sadik kalmak icin, tekrar printf komutuna donelim, ve nasil calistigini gorelim. Gordugunuz gibi, butun satirlar, birbiri ile ayni, ve diger orneklerden farki, icindeki % isareti. Bu harf, printf'e ekrana yazmayi durdurup, ozel birsey yapmasini soyler. % isaretinden sonra gelen harf, d, bir tamsayi yazilacagini belirtir. Bundan sonra, yeni satira geciren tanidik \n isaretini goruyoruz.

Den-denler arasinda kalan butun harfler, printf komutu ile ekrana cikacaklari tanimlar. Bundan sonraki virgul ve "index" sozcugu yer alir. printf komutu buradan degiskenin degerlerini okur. Daha fazla %d ekleyerek ,ve bunlari yine virgul ile birbirine ekleyerek, birden fazla degiskenin de bu komut ile goruntulenmesini saglayabiliriz. Hatirlamaniz gereken onemli bir nokta, saha tanimlayici %d ile gecirdiginiz degisken miktari, ayni kalmalidir, yoksa bir runtime hatasi verir.

BILGI SATIRLARI NASIL EKLENIR

```
/* Bu satiri, derleyici kullanmaz */

main() /* Bir satir daha */
{
printf("Bilgi satirlarinin nasil eklenecegini ");
/* Bilgi satirlari,
bir satirdan uzun olabilir.
*/
printf("goruyoruz.\n");
}

/* Ve programin sonu... */
```

Programa aciklik katmak icin, eklenebilecek bilgiler, derleyici tarafindan uzerinden atlanir. Lutfen yukaridaki programi, iyi bir ornek olarak almayin. Son derece daginik bir sekilde katilmis bilgi satirlari, sadece kullanimini gostermek amaci iledir. Bilgi satirlari, /* isaretleri ile baslar, ve */ isareti ile sona erir.

Dikkat etmeniz gereken bir nokta, birkac satirdan olusan bilgi satirlarinda bulunan program komutlarinin, isleme konmayacagidir.

Bilgi satirlari, programin nasil calistigini gostermesi bakimindan cok onemlidir. Yazdiginiz bir programin, bir baskasi tarafindan okunabilmesi, yada siz nasil calistigini unuttuktan sonra hatirlayabilmeniz icin, mumkun oldugu kadar cok bilgi satiri eklemekte fayda vardir.

Bazi derleyiciler ic ice bilgi satirlarini kabul ederler, fakat genelde, ic ice bilgi satirlari kabul edilmez.

IYI SAYFA DIZIMI

Yazdiginiz bir program, kolay anlasilir olmalidir. Bunun icin, duzgun bir sekilde programlamak cok onemlidir. C derleyicileri, komutlar arasindaki bosluklari goz onune almaz, ve bu nedenle de programlariniza aciklik katmak icin, dilediginiz gibi bosluk ve bos satir birakabilirsiniz.

Su iki programi karsilastiralim:

```
main()
{
int index;
index = 13;
printf("Indexin degeri simdi %d\n",index);
index = 27;
printf("Indexin degeri simdi %d\n",index);
index = 10;
printf("Indexin degeri simdi %d\n",index);
}
```

ve:

```
main()/* Program buradan basliyor */{printf("iyi yazis,"); printf
("programin anlasilmasini kolaylastirir\n");}
```

Akış Kontrolü ve Döngüler

'WHILE' DÖNGÜSÜ

C dilinde, bircok tip dongu vardir. While dongusu, herhangi bir test, dogru kaldigi surece, bir program parcasini tekrarlar. Bu testin sonucu yalnis cikarsa, while dongusu sona erer, ve program normal akisina devam eder.

```
main() /* while dongusunun bir ornegi */
{
int count;

count = 0;
while (count<6)
{
  printf("count'un degeri: %d oldu. ",count);
  count = count + 1;
}
}
```

Bu programda, count isimli bir degiskeni tanimliyoruz, ve sifira esitliyoruz. while dongusunun kullanimi, gorundugu gibi, 'while' sozcugu, ve parantez icinde bir test'den olusur. Parantezlerin icindeki deyim dogru kaldigi surece, bu program tekrarlar. Bu programda, degiskenin degeri teker teker arttirildigindan, eninde sonunda degeri altiya varacaktir, ve bu durumda program donguden cikacaktir.

Parantezlerin icinde yer alan deyimleri, bundan sonraki konuda isleyecegiz. O zamana kadar,

bunlarin dusundugunuz seyleri yaptigini kabul ediniz.

Birkac onemli nokta:

- Sayet 'count' un baslangic degeri 5 den buyuk birseye atanmis olsa idi, dongunun icindekiler hic yapilmayacakti
- Eger 'count = count + 1' komutu ile degerini bir arttirmasa idik, bu program hic durmazdi.
- Son olarak, eger dongude tek bir komut varsa, kume isaretlerine gerek yoktur.

DO-WHILE DONGUSU

Buna benzeyen bir baska komut ise, 'do-while' komutudur. Su program, daha once gordugumuz programa cok benzer:

```
main() /* Do-While dongusu ornegi */
{
int i;


i=0;
do
{
  printf("I nin degeri simdi: %d oldu.\n",i);
  i = i + 1;
} while (i<5);
}
```

Yegane farkin, dongunun bir 'do' komutu ile yapilmasi ve denkligin kontrolunun sona birakilmasidir. Bu durumda, parantezlerin arasinda deyim dogru kaldigi surece, dongu

tekrarlanir.

Burada onemli noktalar: Kontrol, dongunun sonunda yapildigindan, kume isaretlerinin arasindaki deyimler daima en az bir kere islenir. Ayrica yine, sayet i nin degeri degismez ise, program donguden cikmaz. Son olarak, sayet dongunun icinde bir tek komut varsa, kume isaretlerine gerek yoktur.

Ayrica, dilediginiz miktarda donguyu ic ice de koymaniz mümkündür.

FOR DONGUSU

For dongusu, yeni birsey degildir. Sadece, 'while' dongusunun bir baska seklidir:

```
main() /* Bir for dongusu */
{
int index;
for(index=0;index<6;index = index + 1)
printf("index'in degeri simdi %d oldu.\n",index);
}
```

'For' dongusu, uc parcadan olusmustur. Her kesim birbirinden ; ile ayrilir. Ilk kesimi, baslangic kesimi (initalization) dir. Burada bulunan islemler, dongu baslamadan once, ve bir kere yapilir. Aslinda buraya yazilacak seyler icin bir sinir yoktur, fakat basit tutmakta fayda vardir. Bu kesime birden fazla islem yazilabilir, bunlari da birbirinden ',' virgul ile ayirmak gerekir.

Ikinci kesimde, "index<6" diyen parcada, bu dongunun her turunda kontrol edilmesi gereken deyim yer alir. Bu deyim dogru oldugu surece, dongu devam eder. Dogru yada yalnis sonuc veren herhangi bir deyim, bu kesimde yer alabilir.

Ucuncu kesimde yer alan islemler ise, yine dongunun her turunda yapilir, fakat isleme baslamasi,

dongunun icinde yer alan komutlarin islenmesinden sonra yapilir.

'For' komutundan sonra, ya tek bir komut gelir, yada kume isaretleri icinde, bir komut bloku.. C de hemen heryerde, tek bir komut yerine, bir komut bloku koymaniz mumkundur.

IF KOMUTU

```
/* Bu, if-else komutunun bir ornegidir */

main()
{
int data;
for (data=0;data<10;data = data + 1)
{
if (data==2)
printf("Data simdi %d ye esit.\n",data);

if (data<5)
printf("Data simdi %d. Bu da, 5 den azdir. \n",data);
else
printf("Data simdi %d. Bu da, 4 den buyuktur.\n",data);
}
}
```

Bu programda, ilk once, icinde iki tane if komutu olan bir for dongusu gorunuyor. Bu dongunun on kere tekrarlanacagi, acik bir sekilde goruluyor.

Ilk if satirina bakin: "if" kelimesi ile basliyor, ve sonra bir parantez icinde, sarti goruluyor. Sayet bu parantezin icindeki islemin sonucu dogru (evet) ise, if'den hemen sonra gelen satir islenir.

Sayet cevap yalnis ise, if'den sonra gelen komut, atlanir. Burada da, tek bir komut, kume isaretleri ile, bir komut bloku haline getirilebilir.

Burada data==2 islemi, data degiskeninin degerinin ikiye esit olup olmadigini kontrol eder. (Sayet data = 2 olsa idi, tumuyle ayri birsey gerceklesirdi.)

IF-ELSE

Ikinci "if", yine birincisine benziyor. Fakat, ek olarak "else" isimli bir kesimi de iceriyor. Bu da, sayet parantezlerin icindeki islem dogru (EVET) sonuc verirse, "if" den sonra gelen satir islenecektir, sayet yalnis (HAYIR) sonucu verirse, "else" den sonra gelen komut islenecektir. Bu nedenle, iki satirdan biri muhakkak islenecektir.

BREAK ve CONTINUE

```
main()
{
int xx;

for (xx=5;xx<15; xx=xx+1)
{
  if (xx==8)
  break;
  printf("Break dongusunun icinde, xx in degeri simdi %d\n",xx);
}

for (xx=5;xx<15;xx=xx+1)
{
```

```
  if (xx==8)
  continue;
  printf("Continue dongusunun icinde, xx in degeri simdi %d\n",xx);
}
}
```

Bu programda gordugunuz gibi, sayet xx in degeri 8 e esit ise, break isimli komutu cagiran bir if komutu goruyorsunuz. Break komutu, bizi donguden cikarip, programi dongunun hemen altindaki satirdan devam etmesini saglar.

Bu komut, ornegin dongunun icinde hesaplanan bir degere gore, donguden cikmak istediginizde cok ise yarar. Ornekte, xx in degeri sekize ulasinca, program donguden cikar, ve ekrana yazilmis en son deger, yedi olur.

Programin ikinci parcasindaki dongude ise, Continue komutunu goruyoruz. Burada ise, deger 8 e ulasinca, program donguden cikmaz, fakat program dongunun en son satirina atlayip, aradaki printf satirini islemez.

SWITCH

```
main()
{
int kamyon;

for (kamyon = 3;kamyon<13;kamyon = kamyon + 1)
{
switch (kamyon)
{
case 3: printf("Degeri simdi uc.\n");
```

```
break;
case 4: printf("Degeri simdi dort.\n");
break;
case 5:
case 6:
case 7:
case 8: printf("Degeri simdi 5 le 8 arasinda.\n");
break;
case 11:printf("Degeri simdi onbir.\n");
break;
default:printf("Tanimsiz degerlerden biri.\n");
break;
}
}
}
```

Simdiye kadar gordugumuz en buyuk komut olan "switch", aslinda kullanimi kolaydir. Ilk once, "switch" kelimesi ile baslar. Bunun arkasindan, parantez icinde bir deyim gelir. Bundan sonra, dilediginiz kadar 'case' komutlari, kume isaretleri arasinda yer alir. Her degeri sembolize eden 'case' satirlari, degiskenin degeri, iki nokta ust uste, ve bununla ilgili komutlardan olusur.

Bizim ornegimizde, "kamyon" degiskeninin degeri 3 oldugunda, printf satiri, 'Degeri simdi uc' satirinin ekrana yazilmasini, saglar. Daha sonra yer alan 'break' komutu ise, switch in icinde yer alan diger komutlari islenmeden, switch den cikilmasini saglar.

Bir giris noktasi bulunduktan sonra, satirlar bir 'break' komutuna rastlayincaya kadar, yada switch'in son kume isaretine varincaya kadar komutlar siradan islenir.

"Kamyon" un degeri 5 e esit ise, program, case 5,6,7 den gecerek 8'de bulunan printf ve break

komutlarini isler. Break komutu da, programi son kume isaretine getirir. Sayet degiskenin bir degerine karsilik gelen bir case yoksa, 'default:' isimli secenek secilir.

GOTO KOMUTU

```
main()
{
goto leave
printf("Bu satir hic yazilmayacak.\n");
leave:
}
```

Goto komutunu kullanmak icin, "goto" isminin yanina, atlamak istediginiz yerin sembolik ismini yazmaniz yeterlidir. "goto" ile bir dongunun icine atlamaniza izin yoktur, fakat bir dongunun disina atlayabilirsiniz. Ayrica bir fonksiyondan otekine de "goto" ile gecemezsiniz.

Bazi kimseler, goto nun hicbir yerde kullanilmamasi gerektigini belirtiyorlar. Sayet, goto kullanimi ile, diger metodlara gore daha anlasilir bir program olusacaksa, goto yu kullanmaktan cekinmeyin.

SONUNDA - ISE YARAYAN BIR PROGRAM

```
main() /* Santigrad'dan Fahrenheite */
{
int count; /* for degiskeni */
int fahr; /* fahrenheit degerini tutar */
int cen; /* Santigrat degerini tutar */

printf("Santigrad -> Fahrenheit karsilik tablosu\n\n");
```

```
for (count=-2;count<=12;count=count+1)
{
  cen = 10 * count;
  fahr = 32 + (cen * 9) / 5;
  printf(" C = %4d F = %4d ",cen,fahr);
  if (cen == 0)
    printf(" Suyun donma noktasi");
  if (cen == 100)
    printf(" Suyun kaynama noktasi");
    printf("\n");
}
}
```

Bu program, santigrad ve fahrenheit derecelerin tablosunu yaratmaktadir. Birden fazla degisken kullanilan ilk programimizdir bu. Degisken taniminda, uc ayri satir kullanilmasi sayesinde, degiskenlerin yanina ne ise yaradiklarini da yazabiliriz.

TAM SAYI ATAMA

```
main()
{
int a,b,c;

a = 12;
b = 3;

c = a+b;
c = a-b;
```

```
c = a*b;
c = a/b;
c = a%b;

c = 12*a+b/2-a*b*2/(a*c+b*2);

a = a + 1; /* arttirma islemleri */
b = b * 5;

a = b = c = 20; /* Coklu atamalar */
a = b = c = 12*13/4;
}
```

Bu programda uc tam sayi degiskeni tanimliyoruz (a,b,c), ve bunlara degerler atiyoruz. Ilk iki satirda a ve b ye sayisal degerler veriyoruz. Daha sonraki dort satirda, basit islemler goruyorsunuz.

Besinci satirda ise, modulo operatorunu goruyorsunuz. Modulo, iki degisken birbirine bolundugunde, kalan degeri verir. Modulo, sadece integer ve char degisken tipleri ile kullanilabilir.

Daha sonra gelen iki arttirma islemleri ise, bu sekilde derleyici tarafindan kabul edilir, fakat bunlari yazmanin daha kestirme bir sekli vardir - bunu daha sonra gorecegiz.

Son iki satira gelince, bunlar cok tuhaf gorunebilir goze. C derleyicisi, atama satirlarini, sagdan sola dogru okur. Bunun sayesinde, coklu atamalar gibi, cok faydali islemler yapilabilir. Bu ornekte, derleyici, yirmiyi alip, c ye atiyor. Sola dogru devam ederken, b yi gorup, en son elde edilen sonucu (20) b ye atiyor. Ayni sekilde a ya da, b nin degeri veriliyor.

Bu programi derleyip, calistirmak son derece SIKICI olabilir. Bu programin hicbir ciktisi yoktur. Dilerseniz, ogrendiginiz printf fonksiyonu ile, programin yaptiklarini daha yakindan inceleyebilirsiniz.

C de veri tanimlari, program bloku icinde, islenecek komutlardan once gelir. Sayet tanimlari programin ortasina yerlestirmeye calisirsaniz, derleyici bir hata verecektir.

VERi TiPLERi ve KARŞILAŞTIRMA

Veri Tipleri

```
main()
{
int a,b,c; /* -32767 den 32767 ye - tamsayi olarak */
char x,y,z; /* 0 dan 255 e - tamsayi olarak */
float num,toy,thing; /* 10e-38 den 10e+38 e - ondalikli olarak */

a = b = c = -27;
x = y = z = 'A';
num = toy = thing = 3.6792;

a = y; /* a nin degeri simdi 65 (karakter A) */
x = b; /* x simdi tuhaf bir sayi olacak */
num = b; /* num simdi -27.00 olacak */
a = toy /* a simdi 3 olacak */
}
```

Gordugunuz gibi, birkac integer daha tanimladik. Fakat, bundan baska, iki yeni tip daha kattik.

"Char" ve "float".

"Char" tipi, nerdeyse integer ile ayni manada. Fakat, sadece 0 ila 255 arasindaki sayilari alabilir, ve genellikle hafizada bir bytelik bir yerde saklanir. Bu tip veri, genellikle kelime katarlari saklamak icin kullanilir.

DATA TIPLERININ KARISTIRILMASI

Bu anda, C nin "int" ve "char" i nasil kullandigini gormenin tam zamani. C deki "int" tipi ile calisan cogu fonksiyonlar, karakter tip veri ile de ayni sekilde calisabilir, cunku karakter tipi, bir cins integer'dir. "char" ve "int" tiplerini neredeyse istediginiz gibi karistirmak mumkundur. Derleyicinin akli karismaz, ama sizin karisabilir. Bunun icin dogru tip veriyi kullanmakta fayda vardir.

FLOAT

Ikinci yeni tip veri, "float" tipidir. Kayar nokta da denilen bu tipin sinirlari cok genistir. Cogu bilgisayarlarda, float tipi 10e-38 den 10e+38 e kadardir.

YENI VERI TIPLERINI NASIL KULLANALIM

Bu programin ilk uc satirinda, dokuz tane degiskene deger ataniyor.

- Daha once gordugumuz gibi, "char" tipi, aslinda bir "integer" tipi oldugundan, bir "char" in "int" e cevrilmesinde hicbir sorun yoktur.
- Fakat, bir integer'i "char" a cevirmek icin, bir standart yoktur. Bu nedenle, sayet tamsayi degiskeninin degeri, "char" sahasindan buyukse, cikan sonuc cok sasirtici olabilir.

- Ucuncu satirda ise, bir tamsayiyi, "float" a atiyoruz. Bu durumda, derleyici, bu ceviriyi bizim icin yapar.
- Fakat tersini yapmak ise, biraz daha karisiktir. Derleyici sayet varsa, degiskenin ondalik degerini ne yapacagina karar vermek zorundadir. Genellikle de, ondalik kesimi gozardi eder.

Bu programin da hicbir ciktisi yok. Hem zaten karakter ve float tiplerinin nasil ekrana yazilabilecegini gormedik.. Bundan sonraki programa kadar sabir.

```
main()
{
int a; /* basit tamsayi tipi */
long int b; /* uzun tamsayi tipi */
short int c; /* kisa tamsayi tipi */
unsigned int d; /* isaretsiz (+ - siz) tamsayi */
char e; /* karakter tipi */
float f; /* kayar nokta tipi */
double g; /* cift hassasiyet kayar nokta */

a = 1023;
b = 2222;
c = 123;
d = 1234;
e = 'X';
f = 3.14159;
g = 3.1415926535898;

printf("a = %d\n",a); /* desimal */
```

```
printf("a = %o\n",a); /* oktal */
printf("a = %x\n",a); /* heksadesimal */
printf("b = %ld\n",b); /* uzun desimal */
printf("c = %d\n",c); /* kisa desimal */
printf("d = %u\n",d); /* isaretsiz */
printf("e = %c\n",e); /* karakter */
printf("f = %f\n",f); /* kayar nokta */
printf("g = %f\n",g); /* cift hassasiyet k.n */
printf("\n");
printf("a = %d\n",a); /* basit 'int' cikti */
printf("a = %7d\n",a); /* 7 uzunlukta bir saha kullan*/
printf("a = %-7d\n",a); /* sola dayali 7 lik saha */
printf("\n");
printf("f = %f\n",f); /* basit kayan nokta */
printf("f = %12f\n",f); /* 12 lik bir saha kullan*/
printf("f = %12.3f\n",f); /* noktadan sonra 3 hane */
printf("f = %12.5f\n",f); /* noktadan sonra 5 hane */
printf("f = %-12.5f\n",f); /* sola yapisik 12 hane */
}
```

Bu program, C dilinde bulunan butun standart basit veri tiplerini kapsiyor. Baska tiplerde var, fakat bunlar basit tiplerin bir araya gelmesi ile olusurlar. Bunlardan daha sonra bahsedecegiz.

Programi inceleyin. Ilk once basit 'int', sonra 'long int' ve 'short int' gorunuyor. 'unsigned' tipi, yine integer kadar bir sahada saklanir, fakat arti yada eksi isareti tasimadigindan, genellikle siniri 0 - 65535 dir. (Sayet long, short, yada unsigned deyimi kullanilmissa, sonuna 'int' yazilmasi gereksizdir.)

Daha once char ve float u gormustuk. Bunlar disinda kalan 'double' tipi, 'float' a nazaran daha

buyuk bir sahada saklanir, ve daha hassas sonuclar verebilir.

Cogu derleyicilerin matematik fonksiyonlari, float tipini kullanmaz, double tipini kullanir. Bu nedenle verdiginiz float degeri, size transparan olarak double'a cevirir.

PRINTF'in CEVİRİM KARAKTERLERİ

Printf fonksiyonunda kullanilan karakterler sunlardir:

- d desimal
- o oktal
- x heksadesimal
- u unsigned (isaretsiz)
- c karakter
- s string (karakter katari)
- s string (karakter katari)

Bu harfler, bir yuzde isaretinden sonra kullanirlar. Bu iki harf arasina sunlar ilave edilebilir:

- - sahasinin icinde sola dayanmis (n) minimum saha uzunlugunu belirler
- . n ile m yi birbirinden ayirir (m) float tipi icin noktadan sonraki hane sayisi
- l 'long' tipi oldugunu belirtmek icin

Bu programi derleyip sonuclarini inceleyin. Dilediginiz gibi degistirerek, sonuclari inceleyin.

MANTIKSAL KARSILASTIRMALAR

```
main() /* Bir suru karsilastirma */
{
int x = 11,y = 11,z = 11;
char a = 40,b = 40,c = 40;
float r = 12.987,s = 12.987,t = 12.987;

/* Birinci grup */

if (x == y) z = -13; /* z = -13 olacak */
if (x > z) a = 'A'; /* a = 65 olacak */
if (!(x > z)) a = 'B'; /* bu hicbir sey yapmayacak */
if (b <= c) r = 0.0; /* r = 0.0 olacak */
if (r != s) t = c/2; /* t = 20 olacak */

/* Ikinci grup */

if (x = (r != s)) z = 1000; /* x pozitif olacak, ve
z = 1000 olacak */
if (x = y) z = 222; /* bu, x = y, and z = 222 yapar */
if (x != 0) z = 333; /* z = 333 olacak */
if (x) z = 444; /* z = 444 olacak */

/* Ucuncu grup */

x = y = z = 77;
if ((x == y) && (x == 77)) z = 33; /* z = 33 olur */
if ((x > y) || (z > 12)) z = 22; /* z = 22 olacak */
```

```
if (x && y && z) z = 11; /* z = 11 olur */
if ((x = 1) && (y = 2) && (z = 3)) r = 12.00; /* Bu ise,
x = 1, y = 2, z = 3, r = 12.00 yapar */
if ((x == 2) && (y = 3) && (z = 4)) r = 14.56; /* Birsey degistiremez */


/* Dorducu grup */

if (x == x); z = 27.345; /* z daima deger degistirir */
if (x != x) z = 27.345; /* Hicbirsey degismez */
if (x = 0) z = 27.345; /* x = 0 olur, z degismez */
}
```

Karsilas.C isimli programa lutfen bakin. Ilk basinda dokuz tane degisken hazirliyoruz. Daha once yapmadigimiz sekilde, bunlari hem tanimlayip, hem ilk degerlerini veriyoruz.

Gordugunuz gibi if ile komutlar arasinda bir satir birakmamiz gerekmiyor. Programin daha okunabilir olmasi icin arada satir birakmak sart degildir.

Birinci gruptaki karsilastirmalar, iki degiskeni karsilastirdiklari icin, en basit olanlari. Ilk satirda, x in y ye esit olup olmadigina bakiyoruz. Burada iki esit isareti yerine (==) tek esit de kullanilabilirdi, fakat manasi degisirdi.

Ucuncu satirda, NOT isaretini goruyorsunuz. Bu unlem isareti, herhangi bir karsilastirmanin sonucunu degistirmek icin kullanilabilir.

DAHA ZOR KARSILASTIRMALAR

Ikinci grupta yer alan karsilastirmalar daha zor. Ilk once parantezler arasinda tuhaf bir ifade yer

aliyor.. Bunu anlamak icin C dilindeki 'EVET' ve 'HAYIR' kavramlarini bilmemiz gerekiyor. C de 'HAYIR', 0 degerindedir. 'EVET' ise, sifirdan degisik herhangi birseydir. Bir EVET/HAYIR testinin sonucu herhangi bir integer yada karakter degiskenine atanabilir.

Ilk ornege bakin: r!=s deyimi, r nin degeri 0.0 a atandigindan, 'EVET' bir sonuc verecektir. Bu sonuc, sifirdan degisik bir rakam, ve herhalde 1 olacaktir. Olusan bu sonuc, x degiskenine atanir. Sayet x den sonra iki esit isareti olsa idi (x == (r!=s) gibi) bu durumda bu 1 degeri, x ile karsilastirilirdi. Fakat tek bir isaret oldugundan, r ile s yi karsilastirmanin sonucu, x e atanir. Ayrica bu atama isleminin sonucu da sifirdan degisik oldugundan, z de 1000 e esitlenir.

Ikinci ornekte ise, x degiskeni, y nin degerini alir, cunku arada tek esit isareti vardir. Ayrica sonuc 11 oldugundan, z de 222 ye esitlenir.

Ikinci grubun ucuncusunde, x i sifira karsilastiriyoruz. Sayet sonuc 'EVET' ise, yani x sifir degilse, z ye 333 degerini atiyoruz. Bu grubun en son orneginde ise, sayet x in degeri sifir degil ise, z ye 444 atiyoruz. Yani ucuncu ve dorduncu ornekler, birbirine esdirler.

Ucuncu gruptaki karsilastirmalar, yeni deyimler sunuyor. Yani 'AND' ve 'OR' deyimleri. Ilk once 3 degiskene de 77 degerini atiyoruz ki, islemlere bilinen degerlerle basliyabilelim. Buradaki ilk ornekte, yeni kontrol isaretimiz '&&' i goruyoruz. Bu satirin okunusu ise: 'Sayet x, y ye esit ise, vede x, 77 ye esit ise, z nin degerini 33 yap.' Yani, AND operandi icin, iki taraftaki islemlerin EVET (TRUE) sonuc vermesi gereklidir.

Bundan sonraki ornek ise, '||' (OR) isaretini gosteriyor. Bu satir ise, 'Sayet x, y den buyuk ise, YADA z, 12 den buyuk ise, z nin degerini 22 yap.' z nin degeri 12 den buyuk oldugu icin, x in y den buyuk olup olmamasi onemli degildir. Cunku OR operandi icin ikisinden birinin EVET olmasi yeterlidir.

Bircok kisimdan olusan bir mantiksal karsilastirma yaparken, karsilastirma soldan saga dogru yapilir, ve sonuc garantilendiginde, bu satirin islenmesi durur. Mesela, bir AND karsilastirmasinda, sayet AND in sol tarafindaki islem HAYIR (FALSE) sonuc verirse, sag tarafindaki islem yapilmaz. Yada, bir OR isleminde, sol tarafindaki islem EVET (TRUE) sonuc verirse, islemin OR dan sonrasina bakilmaz.

Operand'ların İşlem Sırası

Hangi operand ilk once islenir? Bu konuda bircok kural vardir, ve derleyicinin kitabini bunlari uzun uzun anlatir. Fakat, benim tavsiyem, bunlarla ugrasmak yerine, once islenmesini istediginiz kisimin cevresine parantez koymanizdir.

Ucuncu gruptaki orneklere devam ederek, dorduncu ornekte, uc tane basit degiskenin birbiri ile AND edildigini goruyoruz. Ucunun de degerleri sifirdan degisik oldugundan, sonuc EVET oluyor, ve z nin degeri 11'e esitlenir.

Bundan sonraki ornekte ise, uc tane atama islemi gorunuyor. Sayet daha onceki ornekleri anladiysaniz, bu 'if' komutunun dort tane degeri degistirdigini gorebilirsiniz.

BIR HILE

Ucuncu grubun en son orneginde ise, bir hile var. Ilk once, (x==2) nin HAYIR la sonuc verdigini goruyoruz. Ve daha once gordugumuz gibi, C dili, sonuctan emin oluncaya kadar if komutunu isler. Yani, hepsi AND oldugu icin, vede ilk ifade HAYIR (FALSE) oldugu icin, islemi o noktada keser, ve y,z ve r nin degerleri degismez.

Dorduncu gruptaki orneklerin hicbiri calismaz. Bu grup, basinizi derde sokabilecek komutlardir.

ilk ornekte, x == x komutu daima dogrudur, fakat hemen arkasindan gelen noktali virgul yuzunden, bundan sonra gelen z=27.345 komutu ayri bir komut olarak her zaman islenir.

ikincisi daha kolay - x daima x e esit olacagindan, denklem daima yalnis olacaktir. Son olarak, x e sifir degeri atanir, ve parantezin sonucu sifir oldugundan, z ye atama yapilmaz.

C NIN CABUK TARAFLARI

C de 3 tane, bakinca hicbir seye benzemeyen, fakat programlarken hiz saglayan kestirme yol vardir. Bu metodlar iyi C programcilari tarafindan cok SIK kullanildigindan, ogrenmenizde fayda vardir.

```
main()
{
int x = 0,y = 2,z = 1025;
float a = 0.0,b = 3.14159,c = -37.234;

/* Arttirma */
x = x + 1; /* Bu x i bir arttirir */
x++; /* Bu da.. */
++x; /* Bu da.. */
z = y++; /* z = 2, y = 3 */
z = ++y; /* z = 4, y = 4 */

/* Azaltma */
y = y - 1; /* Bu y nin degerini bir azaltir */
y--; /* Bu da.. */
--y; /* Buddah.. */
y = 3;
```

```
z = y--; /* z = 3, y = 2 */
z = --y; /* z = 1, y = 1 */

/* aritmetik islemler */
a = a + 12; /* a ya 12 eklemek */
a += 12; /* 12 daha eklemek.. */
a *= 3.2; /* a yi 3.2 ile carpmak */
a -= b; /* b yi a dan cikarmak */
a /= 10.0; /* a yi ona bolmek */

/* sartli islemler */
a = (b >= 3.0 ? 2.0 : 10.5 ); /* Bu islem....... */

if (b >= 3.0) /* ve bu islemler.. */
a = 2.0; /* birbiri ile aynidir */
else /* ve ayni sonucu */
a = 10.5; /* saglarlar. */

c = (a > b?a:b); /* c, a yada b nin max ini alir */
c = (a > b?b:a); /* c, a yada b nin min ini alir. */
}
```

KESTIRME.C ye bakin. Bu programda, ilk komutta, x in degeri bir tane arttiriliyor. Ikinci ve ucuncu komutlar da ayni seyi yaparlar. Yani, iki tane arti isareti, degiskenin degerini bir arttirir. Ayrica, sayet ++ isareti degiskenin onunde ise, degisken kullanilmadan once degeri arttirilir, sayet ++ isareti degiskenin arkasinda (saginda) ise, kullanildiktan sonra degeri arttirilir.

Dorduncu komutta ise, y nin degeri, z ye atanir, ve daha sonra da y nin degeri bir arttirilir. Bundan sonraki komutta ise, y nin degeri ilk once arttirilir, daha sonra bu deger z ye verilir.

Ikinci grupta, azaltici operatorleri goruyoruz. Ayni arttirici operatorler gibi, bu gruptaki ornekler de bir oncekiler ile aynidir.

Ucuncu grupta, aritmetik kestirme metodlari goruyoruz. ilk ornekte, a ya 12 eklenir. Bunun altindaki satirda ise, tekrar ayni sey yapilir. Yani, += operatoru, soldaki degiskene, sag tarafin sonucunun eklenecegini belirtir. Yine ayni sekilde, bu is carpma, cikarma, ve bolme islemleri icin de yapilabilir.

Dorduncu grupta ise, a ya, karmasik bir degerin atandigini goruyoruz. Bunun hemen altindaki if... satirlari ise, bu tek satir ile es anlamdadir. Bu karsilastirma operatoru, uc parcadan olusmustur. Bu parcalar birbirinden soru, ve iki nokta isaretleri ile ayrilirlar. Ilk once soru isaretinden onceki kisim degerlendirilir, sonuc EVET cikar ise, soru isaretinden hemen sonraki deger, dondurulur, sayet sonuc HAYIR cikar ise, iki nokta isaretinden sonraki deger dondurulur.

Bundan sonra ise, bu karsilastirma operatorunun c ye atama yapmakta kullanildigini goruyoruz. Ilk once, a ile b nin hangisinin degeri buyukse, o degere c ye atanir, ve ikincide ise, hangisi daha kucuk ise, o c ye atanir.

FONKSİYONLAR VE DEĞİŞKENLER

Fonksiyonlar

```
int toplam; /* Global degisken */
main()
{
int index;
```

```
baslik(); /* Baslik isimli fonksiyonu cagirir */

for (index = 1;index <= 7;index++)
kare(index); /* Bu, kare fonksiyonunu cagirir. */

bitis(); /* Bu da, bitis isimli fonksiyonu cagirir */
}

baslik() /* Bu fonksiyonun tanimidir */
{
toplam = 0; /* "Toplam" isimli degiskene 0 degeri atanir.. */
printf("Bu, kare programinin basligidir\n\n");
}

kare(rakam) /* Bu, kare fonksiyonunun baslangicidir */
int rakam;
{
int karesi; /* Yerel degisken tanimlaniyor */

karesi = rakam * rakam ; /* Karesini olusturuyor. */
toplam += karesi; /* Bulunan deger, toplama ekleniyor */
printf("%d nin karesi %d dir.\n",rakam,karesi);
}

bitis() /* Bitis fonksiyonu tanimlaniyor. */
{
printf("\nKarelerin toplami: %d dir..\n",toplam);
}
```

KARETOPL.C isimli programa bir bakin. Bu program, fonksiyonlu ilk programimiz. Goreceginiz gibi C de fonksiyon tanimlamak o kadar kolaydir ki, programlarin fonksiyonlara parcalanmasi neredeyse istemeden olur. Aslinda, biz fonksiyonlari kullanip duruyorduk, ornegin kullandigimiz printf komutu, bir fonksiyondur. Printf fonksiyonu, derleyici ile gelen fonksiyon kutuphanesinin bir parcasidir.

Bu programin calisan kismina bir bakin. baslik() isimli bir satir ile basliyor. Iste C de, herhangi bir fonksiyon, bu sekilde cagirilir: ismi, parantez, ve sayet varsa bu fonksiyona gonderilmesi istenen degerler yazilir. Programin calismasi bu satira gelince, baslik isimli fonksiyona atlanir, ve buradaki islemler yapilir. Bitince, program geri doner, ve ana programda kaldigi yerden isleme devam eder, ve "for" dongusune gelir. Burada, yedi kere "kare" isimli bir fonksiyonu cagirir, daha sonra "bitis" fonksiyonunu cagirir ve program sona erer.

FONKSIYONUN TANIMLANMASI

main'den sonra ayni main'in ozelliklerini tasayan bir program goreceksiniz. Sadece bunun ismi "baslik()" olarak tanimlanmistir. Bu basligin ilk satirinda "toplam" degiskeninin degeri 0 a atanir, ve bir baslik satiri yazilir. Dikkat ederseniz, "toplam" degiskenini, fonksiyonlarin disinda, programin basinda tanimlamistik. Bu sekilde tanimlanan bir degisken, o programdaki herhangi bir fonksiyondan cagirilabilir. Bu tip degiskenlere "global" denir.

Bu iki satiri main() in icine de koymamiz mumkundur. Bu ornek sadece fonksiyonlarin kullanimini gostermektedir.

FONKSIYONA DEGER GECIRMEK

Ana programda, "for" dongusunde, "index++" deyimini goruyorsunuz. Ilk olarak gecen konuda ogrendigimiz birer birer arttirma metoduna alismaya bakin, cunku C programlarinda cok karsilasacaksiniz.

"kare" isimli fonksiyonu cagirirken, bir yenilik kattik. Yani, parantez icindeki "index" deyimini. Bu da derleyiciye, o fonksiyona gidince, "index" in o andaki degerini de beraberimizde goturmek istedigimizi belirtir. "Kare" isimli fonksiyonun basligina baktigimizda ise, parantezler icinde bir baska degisken ismi goruyoruz: "rakam." Ana programdan "kare(index)" dedigimizde gelen index'in degerine, bu fonksiyon icinde 'rakam' diyecegimizi belirtiyoruz. Buna rakam demek yerine istedigimiz herhangi bir ismi verebilirdik - C nin degisken isim kurallarina uymasi sarti ile. Fonksiyon, ona ne tip bir deger gecirilecegini bilmesi icinde, hemen alt satirda, "int rakam" diyerek, gelecek bu degerin bir integer olacagini belirtiyoruz.

Kume isaretinden sonra, "int karesi" deyimi ile, sadece bu fonksiyonun icinde tanimli olan bir degisken daha tanimlandigini goruyoruz. Bundan sonra, "karesi" degiskenine 'rakam' in karesini atiyoruz, ve "toplam" degiskenine de "karesi" degiskeninin degerini ekliyoruz.

BIR FONKSIYONA DEGER ATAMA HAKKINDA DAHA BILGI

Aslinda "index" in degerini fonksiyona gecirdigimizde, anlattigimdan biraz daha fazla sey oldu. Gercekte, "index" in degerini gecirmedik bu fonksiyona, o degerin bir kopyasini gecirdik. Bu sayede, "index" in asildegeri, fonksiyon tarafindan kazara zarar goremez. "rakam" isimli

degiskenimizi fonksiyon icinde istedigimiz gibi degistirebilirdik, fakat ana programa geri dondugumuzde, "index" in degeri yine ayni kalirdi.

Boylece, degiskenin degerinin zarar gormesini onlemis oluyoruz, fakat ayni zamanda, ana programa bir deger dondurmemize de mani oluyoruz. Pointers kisimina gelince, cagiran fonkisyona degeri dondurmek icin, iyi tanimli bir metod gorecegiz. O zamana kadar ana programa deger dondurmenin yegane yolu, global degiskenler kullanaraktir. Global degiskenlerden biraz bahsetmistik, bu konu icersinde, daha da bahsedecegiz.

Programa devam ederek, bitis() isimli bir fonksiyonun cagirilisina geliyoruz. Bu cagirma da, hicbir yerel degiskeni olmayan fonksiyonu cagirir. "toplam" degiskeninin degerini yazdiktan sonra ana kesime donen program, yapacak baska birsey olmadigini gorunce durur.

UFAK BIR YALANI ITIRAF ETME ZAMANI

Biraz once size bir fonksiyondan bir deger dondurmek icin yegane yolun global degiskenler ile olabilecegini soylemistim. Fakat bir baska metod daha var. Lutfen KARELER.C isimli programa bakin...

```
main() /* Ana program burada. */
{
int x,y;

for(x = 0;x <= 7;x++) {
y = squ(x); /* x*x i hesaplayalim.. */
printf("%d nin karesi %d dir...\n",x,y);
}
```

```
for (x = 0;x <= 7;++x)
printf("%d nin karesi %d dir...\n",x,squ(x));
}

squ(in) /* Bir rakamin karesini bulan fonksiyon */
int in;
{
int kare;

kare = in * in;
return(kare); /* Yeni buldugumuz deger donduruluyor.. */
}
```

Bu program, tek bir deger dondurmenin kolay oldugunu gosteriyor. Fakat, birden fazla deger dondurmek icin, baska metodlara gerek oldugunu hatirlamanizda fayda var.

ana programda, iki tane tamsayi degiskeni tanimliyoruz, ve 8 kere islenen bir "for" dongusu baslatiyoruz. Dongudeki ilk satir, "y = squ(x);", yeni ve tuhaf gorunuslu bir satir. Onceki programlarda gordugumuz gibi, squ(x) kisimi, squ isimli fonksiyonu, x parametresi ile cagirmaktadir. Fonksiyona baktigimizda, bu gecen degiskenin orada 'in' isminde oldugunu, ve kare ismindeki yerel degiskene, gecirdigimiz degerin karesinin atandigini goruyoruz. Daha sonra, yeni "return" komutunu goruyoruz. Parantezler icindeki bu deger, fonksiyonun kendisine atanir, ve ana programa bu deger dondurulur. Yani, "squ(x)" fonksiyonu, x in karesine atanir, ve bu deger, ana programa atanir. Ornegin, x in degeri 4 ise, y nin degeri, "y=squ(x)" satirindan sonra 16 olacaktir.

Bir baska dusunme sekli de, "squ(x)" sozcugunu, "x" in karesi degerinde bir degisken olarak dusunmektir. Bu yeni degisken de, degiskenlerin kullanildigi herhangi bir yerde kullanilabilir.

Baska bir degisken olarak gormeye bir ornek olarak bu programda ikinci bir dongu vardir. Burada, y degiskenine atamak yerine, printf'in icinde, bu fonksiyonu cagiriyoruz.

Bir fonksiyondan donecek degiskenin tipi, derleyiciye bildirilmelidir. Fakat, bizim yaptigimiz gibi sayet belirtmezsek, derleyici donecek degerin tam sayi (integer) olacagini kabul edecektir. Baska tiplerin tanimlanmasini ise, bundan sonraki programda gorecegiz..

KAYAR NOKTA FONKSIYONLARI

```
float z; /* Bu bir global degiskendir */


main()
{
int index;
float x,y,sqr(),glsqr();

for (index = 0;index <= 7;index++){
x = index; /* int'i float yapalim */
y = sqr(x); /* x'in karesini alalim.. */
printf("%d in karesi %10.4f dir.\n",index,y);
}

for (index = 0; index <= 7;index++) {
z = index;
y = glsqr();
printf("%d in karesi %10.4f dir.\n",index,y);
}
}
```

```
float sqr(deger) /* float'in karesini al, float dondur. */
float deger;
{
float karesi;

karesi = deger * deger;
return(karesi);
}

float glsqr() /* float'in karesini al, float dondur. */
{
return(z*z);
}
```

KAYARKAR.C isimli programa bir bakin. Ilk once daha sonra kullanacagimiz bir global degisken tanimlamak ile basliyor. Programin "main" kisiminda, bir tamsayi degiskeni tanimlaniyor. Bunun altinda, iki tani tamsayi degiskeni, iki tane de tuhaf gorunuslu tanimlamalar var. "sqr()" ve "glsqr()" isimli iki fonksiyon gibi gorunuyorlar, ve oyleler. Bu, C dilinde "int" yani tamsayi dan baska birsey dondurecek bir fonksiyonun (float mesela) resmi sekilde tanimlanmasidir. Bu derleyiciye, bu iki fonksiyondan bir deger donunce, bu degerin float olacagini bildiriyor.

Simdi programin ortasinda yer alan "sqr" fonksiyonuna bir bakin. Burada fonksiyonun isminin basinda bir "float" sozcugu goreceksiniz. Bu derleyiciye herhangi bir yerden bu fonksiyon cagirilinca, donecek degerin float olacagini bildiriyor. Simdi bu fonksiyon, ana programdaki cagirmaile uyumludur. Bunun altinda, "float deger" satirini goruyorsunuz. Bu da, bu fonksiyona, cagiran tarafindan gecirilecek degerin, bir "float" yani kayar nokta olacagini bildirir.

Bundan sonraki fonksiyon "glsqr" da, bir kayar nokta donduruyor, fakat o, input icin global bir degikeni (z degiskenini) kullaniyor. Ayrica, yeni bir degisken tanimlamadan, karesini almayi "return" komutunun icinde yapiyor.

Değişken Alanı

```
int say; /* Bu bir global degiskendir. */

main()
{
register int index; /* Bu degisken sadece "main" icinde kullanilabilir */

baslik_1();
baslik_2();
baslik_3();
/* bu programin ana "for" dongusu */
for (index = 8;index > 0;index--)
{
int birsey; /* Bu degisken sadece bu kume isaretleri arasinda tanimli */

for (birsey = 0;birsey <= 6;birsey++)
printf("%d ",birsey);

printf(" index simdi: %d oldu.\n",index);
}
}

int sayac; /* Bu degisken bu noktadan sonra kullanilabilir. */
```

```
baslik_1()
{
int index; /* Bu degisken sadece baslik_1 icinde tanimli */

index = 23;
printf("Baslik_1 deki degeri %d\n",index);
}

baslik_2()
{
int say; /* Bu degisken sadece baslik_2 icinde gecerli */
/* ayni isimli global degiskenin yerini alir.. */

say = 53;
printf("Baslik_2 deki degeri %d\n",say);
sayac = 77;
}

baslik_3()
{
printf("Baslik_3 deki degeri ise %d\n",sayac);
}
```

Ilk tanimlanan degisken "say", butun fonksiyonlardan once tanimlandigi icin, herhangi biri tarafindan cagirilabilir, ve daima erisilebilir. Daha sonra, "sayac" isimli bir degisken tanimliyoruz. Bu da global bir degiskendir, fakat ana programdan sonra tanimlandigi icin, ana program tarafindan kullanilamaz. Global bir degisken, fonksiyonlarin disinda tanimlanan degiskenlere denir. Bu tip degiskenlere dissal degiskenler adi da verilebilir.

Ana programa geri donerek, "index" isimli degiskenin tanimina bakalim. Su an icin "register" sozcugunu goz onune almayin. Bu degisken "otomatik" bir degiskendir, yani o fonksiyon cagirildiginda olusur, ve fonksiyondan cikinca kaybolur. Ana program baska fonksiyonlari cagirdiginda bile daima calisir oldugundan, burada pek manasi yoktur. Tanimlanan diger bir degisken de, "birsey" degiskenidir. Bu degisken, sadece "for" dongusunun icinde tanimlidir, ve baska bir yerden erisilemez. Herhangi bir kume dongusunun basina, degisken tanimlamalari konulabilir. Kumeden cikinca, bu degisken tanimsiz olacaktir.

OTOMATIK DEGISKENLER HAKKINDA...

Baslik_1'e bir bakin. "index" isimli bir degisken kullaniyor. Bu degiskenin ana programdaki "index" ile arasinda, ikisinin de otomatik degisken olmasi disinda hicbir bag yoktur. Program, bu fonksiyonu islemezken, bu degisken yoktur bile. Baslik_1 cagirildiginda, bu degisken yaratilir, ve baslik_1 bitince de bu degisken silinir. Fakat bu, ana programdaki ayni isimli degiskenin degerini hic etkilemez, cunku ayri nesnelerdir.

Yani otomatik degiskenler, gerektiginde yaratilirlar, ve isleri bitince de silinirler. Hatirlamaniz gereken bir nokta da, bir fonksiyon birden fazla kere cagirildiginda, otomatik degiskenlerin eski degerleri saklanmaz, yeni bastan deger atanmalari gerekir.

STATIK DEGISKENLER ?

Bir baska degisken tipi ise, statik degiskenlerdir. Degiskeni tanimlarken basina "static" sozcugunu koyarak, o degisken yada degiskenler, fonksiyonun tekrar tekrar cagirilmasinda, eski degerlerini tutarlar.

Ayni sozcugu bir global degiskenin onune koyarak, o degiskenin sadece o kutuk icindeki fonksiyonlara tanimli olmasini saglayabiliriz. Bundanda anlayacaginiz gibi, birkac parcadan olusan kutukler arasinda global degiskenlerin tanimlanmasi mumkundur. Bunu 14. konuda daha iyi gorecegiz.

AYNI ISMI TEKRAR KULLANMAK

baslik_2 ye bir bakin. Burada "say" isimli degiskenin tekrar tanimlandigini ve 53 degerini aldigini goruyoruz. Global olarak tanimlanmasina karsin, ayni isimde bir otomatik degisken tanimlamak mumkundur. Bu degisken tumuyle yeni bir degiskendir, ve global olarak, programin basinda tanimlanan "say" ile arasinda hicbir baglanti yoktur. Bu sayede kafanizda "acaba global isimlerle karisirmi" sorusu olmadan fonksiyon yazabilirsiniz.

REGISTER DEGISKENLERI NEDIR

Sozumu tutarak, register degiskenine donelim. Bir bilgisayar bilgiyi hafizada yada registerlerde tutabilir. Register sahasina erisim, hafizaya erisimden cok daha hizlidir, fakat programcinin kullanabilecegi az sayida register vardir. Bazi degiskenlerin program tarafindan cok kullanilacagini dusunuyorsaniz, o degiskeni "register" olarak tanimlayabilirsiniz. Bilgisayar ve derleyici tipinize gore, bir yada birkac degiskeni bu sekilde tanimlayabilirsiniz. Cogu derleyicilerin hic register degiskenleri yoktur, ve "register" sozcugunu goz onune almadan derleme yaparlar.

Register degiskenleri, sadece tamsayi ve karakter tipi degiskenler ile kullanilabilir. Sectiginiz derleyiciye gore, unsigned, long yada short tipleride register olabilir.

STANDART FONKSİYON KÜTÜPHANESI

Her derleyici, icinde bircok fonksiyon olan bir kutuphane ile birlikte gelir. Bunlar genellikle giris/cikis islemleri, karakter ve katar isleme, ve matemetiksel fonksiyonlari icerir. Bunlarin cogunu sonraki konularda gorecegiz.

Bunun disinda, cogu derleyicinin, standart olmayan, ve kullandiginiz bilgisayarin ozelliklerini kullanan, ilave fonksiyonlari vardir. Ornegin, IBM-PC ve uyumlular icin, BIOS servislerini kullanan fonksiyonlar sayesinde, isletim sistemine komutlar vermeyi, yada ekrana direk yazmayi saglayan fonksiyonlar olabilir.

RECURSION NEDIR ?

```
main()
{
int index;

index = 8;
geri_say(index);
}

geri_say(rakam)
int rakam;
{
rakam--;
printf("rakam degeri %d dir.\n",rakam);
if (rakam > 0)
geri_say(rakam);
```

```
printf("Simdi rakam %d oldu..\n",rakam);
}
```

Recursion, ilk karsilasildiginda cok korkutucu gorunen bir kavramdir. Fakat RECURS.C isimli programa bakarsaniz, recursion'un butun zorlugunu yenebiliriz. Aslinda fazla basit ve dolayisi ile aptal olan bu program, bize recursion'un kullanimini gostermesi bakimindan cok yararlidir.

Recursion, kendini cagiran bir fonksiyondan baska birsey degildir. Yani, bitmek icin bir kontrol mekanizmasina ihtiyaci olan bir dongudur. Karsinizdaki programda "index" degiskeni 8 e atanir, ve "geri_say" fonksiyonunun parametresi olarak kullanilir. Bu fonksiyon da, bu degiskenin degerini teker teker azaltir, ve bize bu degeri gosterir. Sonra tekrar kendisini cagirir, degeri bir kez daha azalir, tekrar, tekrar.. Sonunda deger sifira ulasir, ve dongu artik kendini cagirmaz. Bunun yerine, daha onceki cagirmada kaldigi yere geri doner, tekrar geri doner, en sonunda ana programa geri doner, ve program sona erer.

NE OLDU ?

Fonksiyon kendisini cagirdiginda, butun degiskenlerini,ve cagirilan fonksiyonun islemesi bittiginde donmesi gereken yeri hafizaya sakladi. Bir dahaki sefere kendinin tekrar cagirdiginda, yine ayni seyi yapti, ta ki kendisini tekrar cagirmasi bitene kadar. Daha sonra tekrar bu bilgileri, ayni koyus sirasi ile geri okudu.

Hatirlamaniz gereken nokta, recursion'un bir noktada bitmesi gerektigidir, sayet sonsuz bir donguye girerseniz, bilgisayarin hafizasi bitecek ve bir hata mesaji cikacaktir.

```
#define BASLA 0 /* Dongunun baslangic noktasi */
#define BITIR 9 /* Dongunun bitis noktasi */
#define MAX(A,B) ((A)>(B)?(A):(B)) /* Max makro tanimlanmasi */
```

```
#define MIN(A,B) ((A)>(B)?(B):(A)) /* Min makro tanimlanmasi */

main()
{
int index,mn,mx;
int sayac = 5;

for (index = BASLA;index <= BITIR;index++) {
mx = MAX(index,sayac);
mn = MIN(index,sayac);
printf("Max simdi %d ve min de %d ..\n",mx,mn);
}
}
```

Bu programda, ilk defa define lara ve makrolarla tanisacaksiniz. Ilk dort satirdaki "#define" sozcuklerine dikkat edin. Butun makrolar ve define'lar bu sekilde baslar. Derleme baslamadan, on-derleyici (preprocessor) bu tanimlari alir, ve programda bu sembolleri gercek degerleri ile degistirir. Ornegin, BASLA sembolunu heryerde sifir ile degistirir. Derleyicinin kendisi, bu BASLA yada BITIR sembollerini gormez bile.

Boyle ufak bir programda bu sekilde semboller tanimlamak luzumsuzdur, fakat ikibin satirlik bir programda, yirmiyedi yerde BASLA olsa idi, sayede #define'i degistirmek, programdaki rakamlari degistirmekten daha kolay olurdu.

Ayni sekilde on-derleyici, BITIS sembolu gordugu heryere 9 rakamini koyar.

C de alisilmis bir teknik de, BASLA yada BITIR gibi sembolik sabitlerin buyuk harfle, ve degisken isimlerinin de kucuk harfle yazilmasidir.

MAKRO

Makro Nedir ?

Makro, bir #define satirindan baska birsey degildir. Fakat icinde islemler yapabildigi icin, ona ozel bir isim verilmistir. Ornegin ucuncu satirda, iki rakamin hangisi buyukse onu donduren MAX isimli bir makro tanimliyoruz. Bundan sonra on-derleyici ne zaman MAX termini ve arkasindan parantezi gorurse, bu parantezlerin arasinda iki tane deger bulacagini farz eder, ve tanimda bulunan deyimi, buraya koyar. Ornegin, onikinci satira gelindiginde, "A" yerine "index" ve "B" yerine de "sayac" konur.

Ayni sekilde "MIN" isimli makro da kendisine gecirilen iki rakamin hangisi daha kucukse, o degeri dondurur.

Bu makrolarda bir suru fazlalik parantez goreceksiniz. Bunlarin nedeni, bir sonraki programda anlasilacak..

```
#define HATALI(A) A*A*A /* Kup icin hatali makro */
#define KUP(A) (A)*(A)*(A) /* Dogusu ... */
#define KARE(A) (A)*(A) /* Karesi icin dogru makro */
#define START 1
#define STOP 9

main()
{
int i,offset;
```

```
offset = 5;

for (i = START;i <= STOP;i++) {
printf("%3d in karesi %4d dir, ve kubu ise %6d dir..\n",
i+offset,KARE(i+offset),KUP(i+offset));

printf("%3d in HATALIsi ise %6d dir.\n",i+offset,HATALI(i+offset));
}
}
```

Ilk satira baktiginiza, HATALI isimli makronun bir rakamin kubunu aldigini goruyoruz. Gercektende, bu makro bazen dogru calismaktadir.

Programin kendisinde,i+offset 'in KUP unun hesaplandigi yeri inceleyelim. Sayet i 1 ise, offset de 5 olduguna gore, 1+5 = 6 olacaktir. KUP isimli makroyu kullanirken, degerler:

```
(1+5)*(1+5)*(1+5) = 6*6*6 = 216
```

olacaktir. Halbuki, HATALI yi kullanirsak, carpmanin onceligi, toplamadan fazla oldugundan, degerleri:

```
1+5*1+5*1+5 = 1+5+5+5 = 16
```

seklinde buluyoruz. Yani, parantezler, degiskenleri dogru bir sekilde birbirinden ayrimak icin gereklidir.

Programin gerisi basittir, ve sizin incelemenize birakilmistir..

KELİME KATARI (STRING)

KELIME KATARI (STRING) NEDIR?

Bir katar, genellikle harflerden olusan karakterler dizisidir. Ciktinizin guzel ve manali gorunmesi icin, icinde isimler ve adresler olabilmesi icin, programlarinizin katarlar kullanmasi sarttir. C dilinde tam tanimi, "char" tipi bilgilerin, NULL karakter (yani sifir) ile sonlandirilmasidir.

C bir katari karsilastiracagi, kopyalayacagi yada ekrana yansitacagi zaman, bunlari gerceklestiren fonksiyonlar, NULL gorunene dek bu islemi yapmak uzere programlanmistir.

ARRAY (dizi) NEDIR?

dizi, ayni tip verilerin birbiri arkasina tanimlanmasidir. Kelime katari, bir cins dizidir.

```
main()
{
char isim[7]; /* Bir karakter dizisi tanimlayalim */

isim[0] = 'T';
isim[1] = 'u';
isim[2] = 'r';
isim[3] = 'g';
isim[4] = 'u';
isim[5] = 't';
isim[6] = 0; /* Bos karakter - katarin sonu */

printf("Isim %s dur. \n",isim);
```

```
printf("Icinden bir karakter: %c\n",isim[2]);
printf("Ismin bir parcasi: %s \n",&isim[3]);
}
```

Bu programda, ilk once, "char" tipi bir tanimlama goruyoruz. Koseli parantezler icinde, kac hanelik bir dizi tanimlanacagini belirtiyoruz. C dilinde butun diziler sifirdan basladigi icin, bu tanimlama ile kullanabilecegimiz en yuksek index degeri 6 dir.

KATAR NASIL KULLANILIR

Demek ki, "isim" degiskeni, icinde 7 tane karakter tutabilir. Fakat en son karakterin sifir olmasi zorunlugu oldugu icin, kullanilabilecek olan alan 6 karakterliktir. Bu katarin icine manali birsey yuklemek icin, yedi tane komut veriyoruz - her biri, katara bir karakter atamaktadir. En sonunda da, katarin sonunu belirten sifir rakamini koyuyoruz. (Bir "#define" ile NULL karakteri, programin basinda sifir olarak tanimlayabiliriz.)

printf komutundaki %s isareti, printf'e "isim" isimli katardan, sifira rastlayincaya kadar ekrana yazmasini belirtir. Dikkat etmeniz gereken bir nokta, "isim" degiskeninin indexinin yazilmasinin gerekmedigidir.

KATARIN BIR KISMININ YAZILMASI

Ikinci printf komutu ise %c ile, katarin icinden sadece bir karakter (harf) yazilmasini gosterir. Istedigimiz karakterin index numarasini da, "isim" degiskeninin yanina, koseli parantezler arasinda gosterebiliriz.

Son printf komutunda ise, katarin 4. karakterinden itibaren yazmanin bir ornegidir. "isim" degiskeninin onundeki & (ampersand) isareti, isim[3]'un hafizada saklandigi adresin printf'e

gecirilmesini belirtir. Adresleri 8. konuda gorecegiz, fakat ufak bir ornek ile size bizleri nelerin bekledigini gostermek istedim.

BAZI KATAR FONKSIYONLARI

```
main()
{
char isim1[12],isim2[12],karisik[25];
char baslik[20];

strcpy(isim1,"Rosalinda");
strcpy(isim2,"Zeke");
strcpy(baslik,"Bu bir basliktir.");

printf(" %s\n\n",baslik);
printf("isim 1: %s \n",isim1);
printf("isim 2: %s \n",isim2);

if(strcmp(isim1,isim2)>0) /* sayet isim1 > isim2 ise, 1 dondurur */
strcpy(karisik,isim1);
else
strcpy(karisik,isim2);

printf("Alfabetik olarak en buyuk isim %s dir.\n",karisik);

strcpy(karisik,isim1);
strcat(karisik," ");
strcat(karisik,isim2);
printf("Iki isim birden %s\n",karisik);
```

```
}
```

Ilk once 4 tane katar tanimliyoruz. Daha sonra, "strcpy" isimli cok pratik bir fonksiyona geliyoruz. Yaptigi is, bir katari, bir digerine, ta ki sifir bulunana kadar kopyalamak. Hangi katarin hangisine kopyalancagini hatirlamak icin, bir atama komutunu dusunun ("x=23" gibi). Veri, sagdakinden, soldakine kopyalanir. Bu komutun yapilmasindan sonra, isim1 in icinde, "Rosalinda" olacaktir - den-densiz olarak. Den-denler, derleyicinin sizin bir katar tanimladiginizi anlamasi icin gereklidir.

KATARLARIN ALFABETIK OLARAK SIRAYA KONMASI

Ilginizi cekebilecek diger bir fonksiyonda, "strcmp" dur. Sayet kendisine gecirilen birinci katar ikinciden daha buyukse, 1 dondurur, ayni ise 0, ve ikinci daha buyukse -1 dondurur. "Zeke" katarinin kazanmasi, sizi herhalde sasirtmaz. Burada katarin boyu onemli degildir, sadece icindeki karakterler. Ayrica harflerin buyuk yada kucuk harf olmasi da fark ettirir. C de bir katarin butun harflerini kucuk yada buyuge ceviren fonksiyonlar da vardir. Bunlari daha ileri kullanacagiz.

KATARLARI BIRBIRINE EKLEMEK

En son satirda, "strcat" isimli yeni bir fonksiyon goreceksiniz. Gorevi, bir katarin sonuna diger katari eklemektir. Bunu yaparken NULL karakterin de yerli yerinde olmasini saglar. Burada, "isim1", "karisik" 'a kopyalanir, daha sonra "karisik" a iki bosluk ve "isim2" eklenir.

Katarlar zor degildir, ve son derece faydalidirlar. Onlari kullanmayi iyice ogrenmenizde fayda

vardir.

DİZİLER

Bir Tamsayı Dizisi

```
main()
{
int degerler[12];
int index;

for (index = 0;index < 12;index++)
degerler[index] = 2 * (index + 4);

for (index = 0;index < 12;index++)
printf("Index = %2d deki degeri %3d dir..\n",index,degerler[index]);

}
```

Bu programda, bir tamsayi dizisi tanimliyoruz. Gordugunuz gibi, ayni katar tanimlama gibi.. Bu sayede, index degiskeni haric oniki tane degiskenimiz oluyor. Bu degiskenlerin isimleri "degerler[0]" , "degerler[1]" ,vs. dir. Ilk "for" dongusunde, bunlara deger atiyoruz, ikincisi ise, index degiskeni ve "degerler" dizisinin icindekileri ekrana yaziyor.

BİR KAYAR NOKTA DİZİNİ

```
char isim1[] = "Birinci Program basligi";

main()
```

```
{
int index;
int ivir[12];
float tuhaf[12];
static char isim2[] = "Ikinci Program Basligi";

for (index = 0;index < 12;index++) {
ivir[index] = index + 10;
tuhaf[index] = 12.0 * (index + 7);
}

printf("%s\n",isim1);
printf("%s\n\n",isim2);
for (index = 0;index < 12;index++)
printf("%5d %5d %10.3f\n",index,ivir[index],tuhaf[index]);
}
```

Burada, "float" olarak tanimli bir kayar nokta dizisi goruyorsunuz. Ayrica bu program, katarlara nasil baslangic degeri atanabilecegini gosteriyor. Koseli parantezlerin icini bos birakarak, derleyicinin o veriyi saklamak icin yeteri kadar yer ayarlamasini sagladik. Programin icinde, bir katar daha ilk degerini veriyoruz. Burada onune "static" koymak zorunlugumuz var. Baska yeni birsey yok bu programda. Degiskenler rastgele degerlere atanir, ve sonra da bu degerler ekrana yazdirilir.

BİR FONKSIYONDAN DEĞER DÖNDÜRME

```
main()
{
int index;
```

```
int matrix[20];

for (index = 0;index < 20;index++) /* veriyi uretelim */
matrix[index] = index + 1;

for (index = 0;index < 5;index++) /* orjinal veriyi, ekrana. */
printf("Baslangic matrix[%d] = %d\n",index,matrix[index]);

yapbirsey(matrix); /* fonksiyona gidip, deger degistirme */

for (index = 0;index < 5;index++) /* degismis matrix i yazalim */
printf("Geri donen matrix[%d] = %d\n",index,matrix[index]);
}

yapbirsey(list) /* Veri donusunu gosterir */
int list[];
{
int i;

for (i = 0;i < 5;i++) /* print original matrix */
printf("Onceki matrix[%d] = %d\n",i,list[i]);

for (i = 0;i < 20;i++) /* add 10 to all values */
list[i] += 10;

for (i = 0;i < 5;i++) /* print modified matrix */
printf("Sonraki matrix[%d] = %d\n",i,list[i]);
}
```

Bir fonksiyondan deger dondurmenin bir yolu da, diziler kullanmaktir. Buradam 20 hanelik bir dizi tanimladiktan sonra, icine degerler atiyoruz, bu degerlerin ilk besini ekrana yazdiktan sonra, "yapbirsey" isimli fonksiyona atliyoruz. Burada goreceginiz gibi, bu fonksiyon "matrix" isimli diziye "list" demeyi tercih ediyor. Fonksiyona, ne cins bir dizi gececegini bildirmek icin, "int" olarak "list"i tanimliyoruz. Fonksiyona kac elemanlik bir dizi gecegini soylememize luzum yok, fakat istenirse belirtilebilir. Bu nedenle bos koseli parantezler kullaniyoruz.

Bu fonksiyon da, kendisine gecen degerleri gosterdikten sonra, bu degerlere 10 ekliyor, ve yeni degerleri gosterip, ana programa geri donuyor. Ana programda goruyoruz ki, fonksiyonun yaptigi degisiklikler, "matrix" degerlerini de degistirmis.

Dizilerin, normal degiskenlerin aksine, fonksiyondaki degerleri degisince, cagiran programdaki dizinin degerlerinin degismesini garipsiyebilirsiniz. Pointerlar konusuna gelince butun bunlar daha manali olacaktir.

BİRDEN FAZLA BOYUTLU DİZİLER

```
main()
{
int i,j;
int buyuk[8][8],dev[25][12];

for (i = 0;i < 8;i++)
for (j = 0;j < 8;j++)
buyuk[i][j] = i * j; /* Bu bir carpim tablosudur */

for (i = 0;i < 25;i++)
for (j = 0;j < 12;j++)
```

```
dev[i][j] = i + j; /* Bu da bir toplama tablosudur */

buyuk[2][6] = dev[24][10]*22;
buyuk[2][2] = 5;
buyuk[buyuk[2][2]][buyuk[2][2]] = 177; /* bu, buyuk[5][5] = 177; demek */

for (i = 0;i < 8;i++) {
for (j = 0;j < 8;j++)
printf("%5d ",buyuk[i][j]);
printf("\n"); /* Her i nin degeri artinca, bir RETURN */
}
}
```

Burada iki tane iki boyutlu dizi kullaniyoruz. "buyuk" adli 8 e 8 lik dizinin elemanlari [0][0] dan [7][7] ye kadar, toplam 64 tanedir. Diger tanimli "dev" dizi ise, kare degildir, fakat dizinin kare olmasinin sart olmadigini gosteren bir ornektir.

Iki dizi de biri carpim tablosu, digeri de toplama tablosu ile doldurulur.

Dizi elemanlarinin tek tek degistirilebilecegini gostermek icin, once "buyuk" un elemanlarinda birine, "dev" in bir elemani ile, 22 ile carpildiktan sonra atanir. Ikinci atamada ise, "buyuk[2][2]" elemani 5 degerine atanir. Herhangi bir islemin index olarak kullanilabilecegini gosteren ucuncu atama ise, aslinda "big[5][5] = 177;" dir.

POINTER

POINTER NEDIR?

Basitce, pointer, bir adrestir. Bir degisken olmak yerine, bir degiskenin hafizadaki adresini tasiyan bir 'ok isareti'dir.

```
main() /* Pointer kullanimi ornegi */
{
int index,*pt1,*pt2;

index = 39; /* herhangi bir deger */
pt1 = &index; /* 'index' in adresi */
pt2 = pt1;

printf("Deger simdi %d %d %d dir.\n",index,*pt1,*pt2);

*pt1 = 13; /* 'index' in degerine degisiklik yapalim */

printf("Degistikten sonra ise %d %d %d\n",index,*pt1,*pt2);
}
```

Su an icin, programin index degiskenini ve iki tane astrisk ile baslayan terimlerin tanimlandigi yere bakmayin. Aslinda astrisk denilen bu isarete, biz simdilik 'yildiz' diyelim.

Programda ilk once, index degiskenine 39 degerini atiyoruz. Bunun altindaki satirda ise, pt1'e tuhaf bir deger atanmasini goruyoruz - index degiskeni, ve onunde bir & ampersand isareti ile. Bu ornekte, pt1 ve pt2 pointer dir, ve index de basit bir degiskendir. Simdi bir problemle karsi

karsiyayiz. Bu programda pointer kullaniliyor, fakat nasil kullanilacagini ogrenmedik.

Bu gorecekleriniz biraz aklinizi karistiracak, fakat bunlari anlamadan gecmeyin.

İKİ ÖNEMLİ KURAL

- Onune ampersand isareti konmus bir degisken, o degiskenin adresini belirtir. Yani altinci satir, soyle okunabilir: "pt1, index isimli degiskenin adresini alir."
- Onune yildiz konmus bir pointer, kendisinin tuttugu adreste bulunan degeri gosterir. Programin dokuzuncu satiri, soyle okunabilir: "pt1 pointer'inin gosterdigi yere, 13 degeri atandi."

HAFIZA YARDIMCISI

- & 'i bir adres olarak dusunun
- * 'i adresteki deger olarak dusunun.

pt1 ve pt2 pointer olarak, kendileri bir deger tasimazlar, fakat bellekteki bir adresi gosterirler. Bu programda, 'index' degiskenini gosteren pointer'lar oldugu icin, degiskenin degerini hem index ile, hemde onun adresini tasiyan pointer'lar ile degistirebiliriz.

Dokuzuncu satirda, index degiskeninin degeri, pt1 pointer'i ile degistiriliyor. Program icinde 'index' i kullandigimiz herhangi biryerde, (pt1 baska birseye atanincaya kadar), '*pt1' i de kullanmamiz mumkundur, cunku pt1, index'in adresini tasimaktadir.

BIR BASKA POINTER

Programa degisklik katmak icin, birbaska pointer daha tanimladim. "pt2" isimli bu pointer, yedinci satirda "pt1"'in tasidigi adresi almaktadir. Bu atamadan once, ayni henuz deger atanmamis degiskenler gibi icinde rastgele bilgiler vardir. Bundan sonra, "pt2" de "index" degiskeninin adresini tasimaktadir. Ornegin, dokuzuncu satirda "*pt1" i "*pt2" ile degistirsek de, sonuc ayni olacaktir - cunku iki pointer da ayni adresi tasimaktadir.

SADECE BİR DEĞİŞKEN

Bu programda uc tane degisken var gibi gorunse de, aslinda bir tane degisken tanimlidir. Iki pointer ise, bu degiskenin adresini tutmaktadir. Bu durum, "printf" komutunun hep 13 degerini yazmasindan da anlasilabilir.

Bu gercekten anlamasi zor bir kavramdir, fakat en kucuk C programlari disinda hepsi tarafindan kullanildigi icin, ogrenmeniz gereklidir.

POINTER NASIL TANIMLANIR

Programin ucuncu satirinda, ilk once "index" isimli degisken tanimlanir, daha sonra da iki tane pointer tanimlamasi goreceksiniz. Ikinci tanim, su sekilde okunabilir: "pt1'in gosterecegi adres, bir tamsayi degiskenine ait olacak." Yani, "pt1", tamsayi bir degiskeninin pointer'i olur. Ayni sekilde, "pt2" de, yine bir tamsayi degiskeninin pointer'i olur.

Bir pointer, bir degiskenin adresini tasimak icin tanimlanir. Tanimlandigindan baska bir degisken tipi icin kullanimi "uyumsuz veri tipi" hatasinin olusmasina sebep olur. Ornegin, "float" tipi bir pointer, "int" tipli bir degiskenin adresini alamaz.

POINTER'LI İKİNCİ PROGRAMIMIZ

```
main()
{
char katar[40],*orada,bir,iki;
int *pt,list[100],index;

strcpy(katar,"Bu bir karakter kataridir.");

bir = katar[0]; /* bir ve iki ayni degeri tasirlar */
iki = *katar;
printf("Ilk cikti %c %c\n",bir,iki);

bir = katar[8]; /* bir ve iki ayni degeri tasirlar */
iki = *(katar+8);
printf("Ikinci cikti %c %c\n",bir,iki);

orada = katar+10; /* katar+10 ve katar[10] aynidir. */
printf("Ucuncu cikti %c\n",katar[10]);
printf("Dorduncu cikti %c\n",*orada);

for (index = 0;index < 100;index++)
list[index] = index + 100;
pt = list + 27;
printf("Besinci cikti %d\n",list[27]);
printf("Altinci cikti %d\n",*pt);
}
```

Bu programda, iki tane pointer, iki tane dizi ve uc tane degisken tanimliyoruz. "orada" isimli

pointer, karakter tipi, ve "pt" ise, tamsayi tipindedir.

BİR KATAR DEĞİŞKENI ASLINDA BİR POINTER DIR

C programlama dilinde, bir katar degiskeni, o katarin baslangicini gosteren bir pointer olarak tanimlanmistir. Programda bir bakin: once "katar" isimli diziye sabit bir katar atiyoruz. Daha sonra, "bir" isimli degiskene, "katar" in ilk harfini atiyoruz. Sonra, "iki" isimli degiskene, ayni degeri atiyoruz. Ikinci satirda "*katar[0]" yazmak yalnis olurdu, cunku yildiz isareti, koseli parantezlerin yerini almaktadir.

"katar" i neredeyse tam bir pointer gibi kullanabilirsiniz, yegane farki, tuttugu adres degistirilemez, ve daima o katarin baslangic adresini gosterir.

Onkinci satira gelince, katarin dokuzuncu karakterinin (sifirdan basladigimiz icin), iki ayri sekilde "bir" ve "iki" isimli degiskenlere atandigini goruyoruz.

C programlama dili, pointer'in tipine gore, index ayarlamasini otomatik olarak yapar. Bu durumda, "katar" bir "char" olarak tanimlandigi icin, baslangic adresine 8 eklenir. Sayet "katar" "int" (tamsayi) olarak tanimlanmis olsa idi, index iki ile carpilip, "katar" in baslangic adresine eklenirdi.

"orada" bir pointer oldugu icin, 16. satirda "katar" in 11. elemaninin adresini tasiyabilir. "orada" gercek bir pointer oldugu icin, herhangi bir karakter degiskeninin adresini gosterebilir.

POINTER VE ARİTMETİK

Her cesit islemler, pointer'lar ile mumkun degildir. Pointer bir adres oldugundan, ona bir sabit rakam ekleyip, daha ilerideki bir adrese erismek mumkundur. Ayni sekilde, pointer'in adresinde

bir rakam cikartip, daha onceki hafiza bolgelerine erismek mumkundur. Iki pointer'i toplamak pek mantikli degildir, cunku bilgisayardaki adresler sabit degildir. Cikacak rakamin tuhaf olacagi icin pointer ile carpma da yapilamaz. Ne yaptiginizi dusunurseniz, yapabilecekleriniz ve yapamayacaklariniz kendini belli edecektir.

TAMSAYI POINTER'I

"list" isimli tamsayi dizisine, 100 den 199 a kadar degerler verilir. Daha sonra, 28. elemanin adresini, "pt" isimli pointer'a atiyoruz. Daha sonra ekrana yazdigimizda, gercektende, o degeri aldigini goruyoruz.

Daha onceki konularda, bir fonksiyondan veri degerlerini dondurmek icin iki metod oldugunu soylemistim. Ilki, bir dizi kullanarakti. Ikincisini herhalde tahmin edersiniz. Sayet tahmininiz "pointer sayesinde" idiyse,tebrikler.

```
main()
{
int cevizler,elmalar;


cevizler = 100;
elmalar = 101;
printf("Baslangic degerleri %d %d\n",cevizler,elmalar);


/* "degistir" i cagirinca, */
degistir(cevizler,&elmalar); /* cevizlerin DEGERI ve, */
/* elmalarin adresini geciriyoruz */


printf("Bitis degerleri ise, %d %d dir..\n",cevizler,elmalar);
```

```
}

degistir(kuru_yemis,meyvalar) /* kuru_yemis tamsayidir */
int kuru_yemis,*meyvalar; /* meyvalar bir tamsayi pointer'idir */
{
printf("Degerler %d %d\n",kuru_yemis,*meyvalar);
kuru_yemis = 135;
*meyvalar = 172;
printf("Sonraki degerler %d %d\n",kuru_yemis,*meyvalar);
}
```

Burada, iki tane tamsayi degiskeni (pointer degil) tanimliyoruz: "cevizler" ve "elmalar". Once bunlara birer deger atiyoruz, ve "degistir" isimli fonksiyonu cagiriyoruz. Cagirirken, "cevizler" in degeri (100), ve "elmalar" degiskeninin adresini geciriyoruz. Fakat, fonksiyona da, bir deger ve bir adres gelecegini haber vermemiz gereklidir. Bunun icin, fonksiyonun parametreleri tanimlanirken, bir adres tasiyacak olan sembolun basina bir yildiz koymamiz yeterlidir.

Fonksiyonun icinde, bu iki degeri degistirip, eski ve yeni degerleri ekrana yaziyoruz. Bu program calistiginda, ana programdaki "cevizler" in degerinin ayni kaldigini fakat "elmalar" in yeni degerlerini aldigini goreceksiniz.

"cevizler" in degerinin ayni kalmasinin nedeni, fonksiyona bir deger gecirildiginde, C dilinin o degerin bir kopyasini fonksiyona gecirmesi yuzundendir. Programa geri dondugunuzde, degerin bir kopyasini kullandigimiz icin asil degerin degismedigini goreceksiniz.

"elmalar" in degerinin degismesi ise, yine fonksiyona "elmalar" degiskeninin adresinin bir kopyasi gecirildigi halde, bu adres ana programdaki "elmalar" a karsilik geldigi icin, fonksiyonda bu adresteki degeri degistirir degistirmez, "elmalar" in da degeri degismis olur.

Standart IO

Standart Input/Output

```
#include <stdio.h> /* input/output icin standard header */

main()
{
char c;

printf("Herhangi bir tusa basin. X = Programi durdurur. \n");

do {
c = getchar(); /* klavyeden bir tus okuyalim */
putchar(c); /* ekranda gosterelim. */
} while (c != 'X'); /* ta ki okunan bir X oluncaya dek... */

printf("\nProgramin sonu.\n");
}
```

Standart I/O deyimi, verinin girildigi ve ciktigi en normal yerleri, klavyeyi ve ekrani kast eder. Bu kutuge ilk baktiginizda, "#include <stdio.h>" komutunu goreceksiniz. Bu komut on-derleyiciye, kucuktur ve buyuktur isaretleri arasinda yer alan kutuk isminin programa eklenmesini soyler. Bazen, < > isaretleri yerine den-den " " isaretleri de gorebilirsiniz. Aralarindaki fark, <> isaretlerinin on-derleyiciye, su anda calistiginiz diskte / dizinde degil de, bu tip kutuklerin konuldugu yerde aramasini bildirir. Halbuki den-den isaretleri ile belirlenmis bir kutuk ismi, sizin su anda bulundugunuz disk / dizinde aranir. Genellikle, "bu tip kutuklerin

konuldugu yer", derleyiciye daha onceden belirtilir. Ornegin, Quick C derleyicisinde, derleyiciye girmeden once:

```
SET INCLUDE=C:\INCLUDE
```

yazmak, derleyicinin bundan sonra butun 'include' edilecek, yani eklenecek kutuklerin C: diskinin \INCLUDE dizininde aranmasini belirtir.

Sonu .h ile biten kutuklerin, ozel bir fonksiyonu vardir. Bunlara header yada baslik kutukleri denir. Genellikle iclerinde, bazi fonksiyonlari kullanmak icin gereken tanimlamalar yer alir. Bu kullandigimiz "stdio.h" kutugu ise, bir suru "#define" komutundan olusur.

C DE INPUT/OUTPUT İŞLEMLERI

C dilinde lisanin bir parcasi olarak tanimlanmis input/output komutlari yoktur, bu nedenle bu fonksiyonlarin kullanici tarafindan yazilmasi gereklidir. Her C kullanan kisi, kendi input/output komutlarini yazmak istemediginden, derleyici yazarlari bu konuda calisma yapmislar, ve bize bir suru input/output fonksiyonlari saglamislardir. Bu fonksiyonlar standart hale gelmislerdir, ve hemen her C derleyicisinde ayni input/output komutlarini bulabilirsiniz. C nin lisan tanimi, Kernigan ve Richie tarafindan yazilmis bir kitaptir, ve onlar bu gorecegimiz input/output fonksiyonlari bu kitaba katmislardir.

Bu "stdio.h" isimli kutugu incelemenizde fayda vardir. Icinde bircok anlamadiginiz nokta olacaktir, fakat bazi kisimlar tanidik olacaktir.

DİĞER INCLUDE KUTUKLERI

C de buyuk programlar yazmaya basladiginizda, programlari ufak parcalara ayirip ayri ayri

derlemek isteyebilirsiniz. Bu degisik parcalarin ortak kisimlarini tek bir kutukte toplayip, bir degisiklik gerektiginde sadece o ortak kutukten yapmayi isteyebilirsiniz (ornegin global degisken tanimlari.) Bu gibi durumlarda "#include" kutukleri cok faydali olacaktir.

"BASITIO" YA GERI DONELIM

"c" isimli degisken tanimlanir, ve ekrana mesaj yazilir. Daha sonra, kendimizi "c", buyuk harf X e esit olmadigi surece devam eden bir dongunun icinde buluyoruz. Bu programdaki iki yeni fonksiyon, su an icin ilgi noktamiz. Bunlar klavyeden bir tus okumak, ve ekrana bir karakter yazmayi saglarlar.

"getchar()" isimli fonksiyon, klavyeden okudugu tusu dondurur, bu deger "c" ye atanir. "putchar()" fonksiyonu ise, bu degeri ekrana yansitir.

Bu programi derleyip calistirdiginizda, bir surpriz ile karsilasacaksiniz. Klavyeden yazdiginizda, ekrana herseyin iyi bir sekilde yansitildigini goreceksiniz. RETURN tusuna bastiginizda ise, butun satirin tekrar ekrana yazildigini goreceksiniz. Her karakteri teker teker ekrana getirmesini soyledigimiz halde, programimiz sanki butun satiri sakliyor gibi.

DOS BİZE YARDIMCI OLUYOR (YADA IŞE KARIŞIYOR)

Bu durumu anlayabilmek icin, DOS un nasil calistigini anlamamiz gereklidir. Klavyeden tuslar DOS kontrolu ile okundugu zaman, RETURN tusu basilana dek, basilan tuslar bir sahada saklanir. RETURN basilinca da, butun satir programa dondurulur. Tuslara basilirken, karakterler ekrana da yansitilir. Bu duruma da "eko" ismi verilir.

Simdi anlatilanlari goz onunde bulundurarak, programimiz calisirken ekrana eko edilenlerin, DOS tarafindan yapildigini anlayabilirsiniz. Siz RETURN e basinca da, bu saklanan tuslar, programa gonderilir. Bunu daha iyi anlamak icin, icinde buyuk harf X olan bir satir yazin. DOS, buyuk X in ozel bir tus oldugundan habersiz, siz RETURN e basana kadar tuslari kabul etmeye devam eder. RETURN e basinca ise, bu katar programa gecirilir, ve program X e rastlayincaya kadar ekrana karakterleri birer birer yazar.

Isletim sisteminin bu tuhafliklari karsisinda yilmayin. Bazi programlarinizda, bu ozellik isinize yarayabilir. Fakat simdi biz, az once yazdigimiz programin, dusundugumuz gibi calismasini saglayalim.

```
#include <stdio.h>

main()
{
char c;

printf("Herhangi bir tusa basin. X = Programi durdurur. \n");

do {
c = getch(); /* bir tus oku */
putchar(c); /* basilan tusu goster */
} while (c != 'X'); /* ta ki c == 'X' olana dek */

printf("\nProgramin sonu.\n");
}
```

Bu programdaki yegane degisiklik olan yeni fonksiyon "getch()", yine klavyeden tek bir karakter okur. Farki, "getchar" gibi DOS'a takilmamasidir. Bir karakter okur, ve ekrana yansitmadan bu

tusu programa dondurur.

Bu programi calistirdiginizda, bir oncekindeki gibi tekrarlanan satirlar olmadigini goreceksiniz. Ayrica program artik 'X' e basar basmaz durmaktadir. Burada baska bir problemimiz var. RETURN'e basinca cursor, ekranin soluna gitmektedir, ama bir alt satira inmemektedir.

SATIR ATLAMAMIZ LAZIM

Cogu uygulama programi siz RETURN e basinca, program o RETURN e ek olarak bir de "Line Feed" yani satir atlama karakteri ilave eder. Satir atlama otomatik olarak yapilmaz. Bundan sonraki programda, bu sorunu da halletmis olacagiz.

```
#include "stdio.h"
#define CR 13 /* CR sembolunu 13 olarak tanimlar */
#define LF 10 /* LF sembolunu 10 olarak tanimlar */

main()
{
char c;

printf("Tuslara basin. Durmak icin X e basin.\n");

do {
c = getch(); /* Bir karakter oku */
putchar(c); /* basilan tusu ekrana yaz */
if (c == CR) putchar(LF); /* sayet basilan RETURN tusu ise,
bir SATIR ATLAMA karakteri yolla */
} while (c != 'X');
```

```
printf("\nProgramin sonu.\n");
}
```

Programin ilk basinda CR 'nin artik 13 e esit oldugunu ve LF nin de 10 oldugunu belirtiyoruz. Sayet ASCII tablosundan bakarsaniz, RETURN tusuna karsilik gelen kodun 13 oldugunu gorursunuz. Ayni tabloda, satir atlama kodu da 10 dur.

Ekrana basilan tusu yazdiktan sonra, sayet bu tus RETURN tusu ise, bir satir atlayabilmemiz icin, satir atlama kodunu ekrana yaziyoruz.

Programin basindaki "#define" lar yerine "if (c == 13) putchar(10);" diyebilirdik, fakat ne yapmak istedigimiz pek belirgin olmazdi.

HANGI METOD DAHA IYI?

Burada ekrandan bir harf okumanin iki yolunu inceledik. Her ikisinin de avantajlari ve dezavantajlari var. Bunlara bir bakalim.

Ilk metodda, butun isi DOS ustlenmektedir. Programimiz baska islerle ugrasirken, DOS bizim icin satiri hazirlayabilir, ve RETURN'e basilinca bu satiri programa dondurebilir. Fakat, bu metodda karakterleri basildiklari anda fark etmemiz imkansizdir.

Ikinci metodda, tuslari teker teker fark etmemiz mumkundur. Fakat, program bu okuma sirasinda butun zamanini okumaya harcar ve baska bir is yapamaz, ve bilgisayarin tum zamanini bu isle almis oluruz.

Hangi metodun uzerinde calistiginiz program icin daha uygun oldugunu programci olarak siz karar vereceksiniz.

Burada, "getch()" fonksiyonun tersi olan "ungetch()" isimli bir fonksiyon daha oldugunu da belirtmeliyim. Sayet bir karakteri "getch()" le okuduktan sonra fazla okudugunuzu fark ederseniz, bu fonksiyon ile okunan tusu geri koyabilirsiniz. Bu bazi programlarin yazilimini kolaylastirmaktadir cunku bir tusu istemediginizi onu okuyuncaya kadar bilemezsiniz. Sadece bir tek tusu "ungetch" edebilirsiniz, fakat genellikle bu yeterlidir.

BIRAZ TAMSAYI OKUYALIM

```
#include <stdio.h>

main()
{
int deger;

printf("0 ila 32767 arasinda bir rakam yazin, durmak icin 100 girin.\n");

do {
scanf("%d",&deger); /* bir tamsayi oku (adresi ile) */
printf("Okunan deger %d idi. \n",deger);
} while (deger != 100);

printf("Programin sonu\n");
}
```

Alistigimiz tip bir program olan TAMOKU'da, "scanf" isimli yeni bir fonksiyon goruyoruz. Cok kullandigimiz "printf" fonksiyonuna cok benzeyen bu fonksiyonun gorevi, istenilen tip verileri okuyup, degiskenlere atamak.

"printf" den en buyuk farki, "scanf" in degisken degerleri yerine, adreslerini kullanmasidir.

Hatirlayacaginiz gibi, bir fonksiyonun parametrelerinin degerlerini degistirebilmesi icin, degiskenin adresine ihtiyaci vardir. "scanf" fonksiyonuna adres yerine deger gecirmek, C dilinde en SIK rastlanan hatalardan biridir.

"scanf" fonksiyonu, girilen satiri, satirdaki bosluklara bakmadan, ve bu sekilde kullanildiginda, rakam olmayan bir karakter bulana kadar bir tamsayi okur.

Sayet 32766 den buyuk bir rakam girerseniz, programin hata yaptigini gorursunuz. Ornegin 65536 girerseniz, programin 0 degerini dondurdugunu gorursunuz. Buna sebep, tamsayilarin hafizada saklanisinda onlara 16 bitlik bir saha ayrilmasindandir. Programinizda daha buyuk rakamlar kullanacaksaniz, 'long' yada 'float' tiplerini secebilirsiniz.

KARAKTER KATARI GİRİŞİ

```
#include <stdio.h>

main()
{
char big[25];

printf("Karakter katari girin, en fazla 25 karakter.\n");
printf("Birinci kolonda X yazarak programi bitirin.\n");

do {
scanf("%s",big);
printf("Yazdiginiz katar -> %s\n",big);
} while (big[0] != 'X');
```

```
printf("Programin sonu.\n");
}
```

Bu program bir oncekine cok benzer, fakat bu sefer bir kelime katari giriyoruz. 25 elemanli bir dizi tanimlanmistir, fakat en son deger bir '0' olmasi gerektiginden, kullanilabilen kisimi 24 dur. "scanf" deki degiskenin onune & ampersand isareti gerekmez cunku, koseli parantezleri olmayan bir dizi degiskeni, C dilinde o dizinin baslangicini gosteren bir adrestir.

Calistiginizda, sizi bir supriz bekliyor. Yazdiginiz cumleyi, program ayri satirlarda gosterir. Bunun sebebi, "scanf" bir katar okurken, satirin sonuna yada bir bosluga rastlayincaya kadar okumasina devam eder. Bir dongu icinde oldugumuzdan, program tekrar tekrar "scanf" i cagirarak, DOS'un giris sahasinda kalan butun karakterleri okur. Cumleleri kelimelere boldugunden, X ile baslayan herhangi bir kelimeye rastlayinca, bu program durur.

24 karakterden daha fazlasini girmeye calisin. Ne olduguna bakin. Size bir hata mesaji verebilir, yada programiniz aleti kilitleyebilir. Gercek bir programda, boyle seylerin sorumlulugu sizlerin omuzlarinizdadir. C dilinde yazdiginiza size cok sey duser, fakat ayni zamanda bircok kolaylik da saglar.

C DE INPUT/OUTPUT PROGRAMLAMA

C dili cok miktarda input/output yapan programlar icin degil de, bir bircok icsel islemler yapan sistem programlari icin yazilmistir. Klavye'den bilgi alma rutinleri cok kullanislidir, fakat C size az yardimci olur. Yani, yapmaniz gereken I/O islemlerinde sorun cikmasini onlemek icin detaylarla sizin ugrasmaniz lazimdir. Fakat genellikle herhangi bir program icin bu tip fonksiyonlari bir defa tanimlamaniz yeterlidir.

```
main()
```

```
{
int rakam[5], sonuc[5], index;
char satir[80];

rakam[0] = 5;
rakam[1] = 10;
rakam[2] = 15;
rakam[3] = 20;
rakam[4] = 25;

sprintf(satir,"%d %d %d %d %d\n",rakam[0],rakam[1],
rakam[2],rakam[3],rakam[4]);

printf("%s",satir);

sscanf(satir,"%d %d %d %d %d",&sonuc[4],&sonuc[3],
(sonuc+2),(sonuc+1),sonuc);


for (index = 0;index < 5;index++)
printf("Sonuc %d dir. \n",sonuc[index]);

}
```

Bu programda, birkac tane degisken tanimliyoruz, ve "rakamlar" isimli diziye de, "sprintf" fonksiyonunu incelemek icin rastgele sayilar atiyoruz. Bu fonksiyon, "printf" e cok benzer. Yegane farki, ciktisini ekrana yazmak yerine, bir karakter dizisine yazmasidir. Bunu da, ilk parametresi olarak veriyoruz. Yani program bu fonksiyondan dondukten sonra, "satir" dizisinin icinde, bes tane rakam olacaktir. Ikinci ile ucuncu rakamlar arasindaki bosluk, "sscanf"

fonksiyonunun bunlarin uzerinden atlamasini gormek icindir.

Bunun altinda "printf" i kullanarak bu hazirladigimiz satiri yaziyoruz. Daha sonra gordugunuz, "sscanf" fonksiyonu ise, "scanf" gibi ekrandan okumak yerine, bizim "satir" dizimizden degerleri okur. Gordugunuz gibi, "sscanf" e rakamlarin konacagi dizinin adreslerini cok degisik sekillerde verebiliyoruz. Ilk ikisi, sadece dizideki 5. ve 4. elemanlarin adreslerini index vererek tanimliyorlar, sonraki ikisi ise, dizinin baslangic adresine bir offset (bir rakam) ekleyerek buluyorlar. Sonuncusu ise, koseli parantezi olmayan bir dizinin, o dizinin baslangic elemaninin adresini gostereceginden, hicbir sey gerektirmiyor.

Bazen, bir programin ciktilarini, standart ciktidan (ekrandan), bir baska kutuge yoneltmek istenir. Fakat, hata mesajlarini gibi bazi mesajlari hala ekrana yollamak isteyebilirsiniz:

```
#include <stdio.h>

main()
{
int index;

for (index = 0;index < 6;index++) {
printf("Bu satir, standart ciktiya gidiyor.\n");
fprintf(stderr,"Bu satir ise standart hataya gidiyor.\n");
}

exit(4); /* Bu komut, DOS 'un ERRORLEVEL komutu ile bir batch file'da
(yigit kutugunde) kontrol edilebilir. Bu programin
d"nd£rd£g£ deger, soyle kontrol edilebilir:
```

```
A> COPY CON: DENE.BAT <RETURN>

OZEL
IF ERRORLEVEL 4 GOTO DORT
(Dortten kucukse, buraya devam eder..)
.
.
GOTO BITTI
:DORT
(dort yada buyukse, buraya devam eder)
.
.
:BITTI

<F6> <RETURN>

*/
}
```

Bu program, bir dongu, ve icinde iki satirdan olusur. Bu satirlardan bir tanesi standart ciktiya, bir tanesi de standart hataya gider. Burada gordugunuz "fprintf" komutu, "printf" e cok benzer, fakat ciktinin nereye gidecegini de belirtmenizi saglar. Bu alanda bir sonraki konuda daha uzun duracagiz.

Program calisinca, ekranda on iki tane satir goreceksiniz. Sayet bu programi:

```
A> OZEL > CIKTI
```

seklinde calistirirsaniz, ekranda sadece alti tane standart hataya giden mesajlari goreceksiniz. Geri kalan (standart ciktiya giden) alti tanesi ise, "cikti" isimli kutukte yer alacaktir.

YA exit(4) KOMUTU ?

Bu programdaki en son satir olan "exit(4)" komutu, programi sona erdirir, ve dort degerini DOS a dondurur. Parantezlerin arasinda 0 ila 9 degerleri kullanilabilir. Sayet bir "batch" (yigit) kutugu icinde bu programi calistiriyorsaniz, bu degeri ERRORLEVEL komutu ile kontrol edebilirsiniz.

BIR KUTUGE YAZMAK

```
#include <stdio.h>
main()
{
FILE *fp;
char ivir[25];
int index;

fp = fopen("ONSATIR.TXT","w"); /* yazmak icin acalim */
strcpy(ivir,"Bu bir ornek satirdir.");

for (index = 1;index <= 10;index++)
fprintf(fp,"%s Satir no: %d\n",ivir,index);

fclose(fp); /* Kutugu kapayalim */
}
```

Bir kutuge yazan ilk programimiz. Herzamanki gibi, "stdio.h" i programa ekliyoruz, ve daha sonra cok tuhaf bir degisken tanimliyoruz.

"FILE" tipi, bir kutuk degiskenidir, ve "stdio.h" in icinde tanimlanmistir. Kullanacagimiz kutuge erismek icin bir 'kutuk pointeri' tanimlamaktadir.

KUTUGUN ACILMASI

Bir kutuge yazmadan once, onu acmamiz gereklidir. Acmak demek, sisteme o kutugun ismini bildirmek, ve yazmak istedigimizi belirtmektir. Bunu, "fopen" fonksiyonu ile yapiyoruz. "fp" isimli kutuk pointer'i, bu acilan kutuge ait bazi bilgileri tutar. "fopen" ise, iki parametre gerektirir. Birincisi, kutugun ismidir. Buyuk harf, kucuk harf, yada karisik fark etmez.

OKUMAK "r"

"fopen" in ikinci parametresi ise, acilacak kutuk ile ne yapilacagini belirtir. Buraya, "r" "w" yada "a" yazabiliriz. "r" kullanildiginda, kutugun okuma icin acilacagini belirtir. "w", kutuge yazilacagini, ve "a" ise zaten var olan bir kutuge bilgi ekleyeceginizi belirtir. Bir kutugu okumak icin acmak icin, o kutugun diskte var olmasini geretirir. Sayet kutuk yok ise, "fopen", geriye NULL degerini dondurur.

YAZMAK "w"

Bir kutuk yazmak icin acilinca, sayet diskte yoksa yaratilir, sayet varsa, cindeki bilgiler silinir.

EKLEMEK "a"

Bir kutuk eklemek modunda acildiginda, sayet yoksa yaratilir, varsa, veri giris pointer'i bu kutugun sonuna ayarlanir. Bu sayede yeni bilgi yazilinca, kutugun sonuna yazilmis olur.

KUTUGE YAZMAK

Bir kutuge yazmak, ekrana yazmak ile neredeyse aynidir. En onemli farklar, yeni fonksiyon

isimleri, ve kutuk pointer'inin bu fonksiyonlara parametre olarak eklenmesidir. Ornek programda, "fprintf" komutu "printf" komutunun yerini alir.

KUTUGU KAPATMAK

Bir kutugu kapatmak icin, sadece "fclose" komutunu kullanmak yeterlidir. Parametre olarak da kutugun pointer'ini gecirmek yeterlidir. DOS, program sona erince kullandigi kutukleri kapattigindan, "fclose" u kullanmak sart degildir, fakat bir aliskanlik yapmasi icin, kullandiginiz kutukleri kapatmanizi tavsiye ederim.

Bu programi calistirdiginizda, ekranda hicbir sey cikarmaz. Program bittikten onra, "ONSATIR.TXT" isimli kutugu inceleyin. Icinde programin yazdigi on satirlik ciktiyi goreceksiniz.

KARAKTERLERI TEKER TEKER YAZMAK

```
#include <stdio.h>
main()
{
FILE *kutukpoint;
char digerleri[35];
int index,say;

strcpy(digerleri,"Ek satirlar.");
kutukpoint = fopen("onsatir.txt","a"); /* eklemek icin acmak */

for (say = 1;say <= 10;say++) {
for (index = 0;digerleri[index];index++)
```

```
putc(digerleri[index],kutukpoint); /* bir karakter yaz */
putc('\n',kutukpoint); /* bir de <RETURN> */
}
fclose(point);
}
```

Normal "include" kutugumuzden sonra, "kutukpoint" isimli bir kutuk pointeri tanimliyoruz. Yazacagimiz bilgileri tutmasi icin, "digerleri" isminde bir karakter dizisi tanimliyoruz. Daha sonra bu actigimiz sahaya, "strcpy" fonksiyonu ile "Ek satirlar." sozcugunu yaziyoruz. Bundan sonra, yine ayni kutugu "append" yani eklemek icin aciyoruz.

Bu program iki tane ic ice donguden olusuyor. Distaki dongu, sadece birden ona kadar sayiyor.. Icindeki dongu ise, yazilan karakter sifir olmadigi surece, "putc" fonksiyonunu cagirir.

"putc" FONKSİYONU

Bu programin ilgimizi ceken yonu, "putc" fonksiyonudur. Belirtilen kutuge bir karakter yazan bu fonksiyon, ilk parametre olarak yazilacak karakteri, ikinci olarak da kutuk pointer'ini veriyoruz. "Digerleri" isimli dizi bitince satirin sonuna bir <RETURN> karakteri koymak icin "putc" yi tekrar cagiriyoruz.

Dis dongu on kere tekrarlandiktan sonra, program kutugu kapatip sona eriyor. Bu program calistiktan sonra kutugu incelerseniz, gercektende sonuna 10 satir eklendigini gorursunuz.

BIR KUTUGU OKUMAK

```
#include <stdio.h>

main()
```

```
{
int deger;

printf("0 ila 32767 arasinda bir rakam yazin, durmak icin 100 girin.\n");

do {
scanf("%d",&deger); /* bir tamsayi oku (adresi ile) */
printf("Okunan deger %d idi. \n",deger);
} while (deger != 100);

printf("Programin sonu\n");
}
```

Bir kutuk okuyan ilk programimiz! "stdio.h" ve iki degisken tanimindan sonra, "fopen" fonksiyonunda okumak icin "r" parametresini veriyoruz. Daha sonra, kutuk acmanin basarili olip olmadigini kontrol ediyoruz. Sayet basarili degilse, geriye NULL degeri donecektir.

Program, bir "do while" dongusunun icinde tek bir karakter okuyup, ekrana yaziyor. Bu dongu, ta ki, "getc" fonksiyonu kutugun sonunu belirten EOF dondurene kadar surer. EOF donunce de, kutuk kapatilir, ve program sona erer.

DIKKAT DIKKAT DIKKAT

Bu noktada, C nin en sasirtici ve en cok yapilan hatasina rastliyoruz. "getc" fonksiyonundan geri donen degisken bir karakterdir, dolayisi ile bunu "char" tipi bir degiskene atayabiliriz. Hatirlayalim ki, bir "char" degiskeni 0 ila 255 arasindaki degerleri alabilir.

Fakat, cogu C derleyicilerinde EOF karakteri, -1 olarak tanimlanmistir - yani, "char" degiskeninin disinda - Bu nedenle sayet char kullanirsak, program kutugun sonunun geldigini bulamaz, ve

sonsuz bir dongude takilir. Bunun onune gecmesi kolaydir: EOF karakteri donmesini beklediginiz durumlarda, daima "int" tipi bir degisken kullanin.

Sayet sizin derleyiciniz icin EOF karakterinin ne oldugunu ogrenmek isterseniz, "stdio.h" isimli header'i okuyabilirsiniz.

KELIME KELIME OKUMAK

```
#include "stdio.h"

main()
{
FILE *fp1;
char birkelime[100];
int c;

fp1 = fopen("ONSATIR.TXT","r");

do {
c = fscanf(fp1,"%s",birkelime); /* kutukten bir kelime okuyalim */
printf("%s\n",birkelime); /* ekrana yazalim */
} while (c != EOF); /* ta ki EOF olana kadar */

fclose(fp1);
}
```

Bu program, nerdeyse bir oncekinin aynisidir. Burada, kelime kelime okumak icin "fscanf" fonksiyonunu kullaniyoruz, cunku "fscanf" fonksiyonu, bir bosluga gelince, okumayi birakir.

FAKAT BIR PROBLEM VAR

Programi inceleyince, verinin kutukten okundugunu, ekrana yazildigini ve daha sonra EOF olup olmadiginin kontrol edildigini goruyoruz. Bu nedenle, istemedigimiz birsey ekrana yazilmis oluyor. Buyuk ihtimalle, programin sonunda, en son kelimeyi bir daha yaziyoruz - cunku zaten "birkelime" nin icinde idi o deger.

Buna mani olmak icin, bir baska program gorelim. Ismi IYIOKU.C olsun:

```
#include "stdio.h"

main()
{
FILE *fp1;
char birkelime[100];
int c;

fp1 = fopen("onsatir.txt","r");

do {
c = fscanf(fp1,"%s",birkelime); /* kutukten bir kelime oku... */
if (c != EOF)
printf("%s\n",birkelime); /* ekrana yaz... */
} while (c != EOF); /* ta ki EOF olana dek.. */

fclose(fp1); /* kutugu kapa */
}
```

Gordugunuz gibi, bir "if" komutu ile, sayet kutugun sonuna gelip gelmedigimize bakiyoruz.

Aslinda bu problem KAROKU.C da da vardi, fakat orada pek gorunmuyordu.

SONUNDA, BUTUN BIR SATIR OKUYORUZ

```
#include "stdio.h"

main()
{
FILE *fp1;
char birkelime[100];
char *c;

fp1 = fopen("ONSATIR.TXT","r");

do {
c = fgets(birkelime,100,fp1); /* bir satir okuyalim */
if (c != NULL)
printf("%s",birkelime); /* ekrana yazalim */
} while (c != NULL); /* ta ki NULL olana kadar.. */

fclose(fp1);
}
```

Bu program, simdiye de gorduklerimize benziyor, fakat NULL isimli yeni bir nesne de katildi.

"fgets" fonksiyonu ile, bir butun satiri, ve sonundaki yeni satir karakterini (\n), bir diziye okur. Ilk parametre olarak, donen karakterleri koyacagimiz yerin adresi tanimlanir, ikinci parametrede en fazla kac karakter okunmasina izin verecegimizi belirtiyoruz, ve son olarak da kutuk degiskeninin ismini veriyoruz.

o Yani bu fonksiyon, ya bir yeni satir karakterine rastlayana kadar, yada izin verilen karakter sayisi eksi bir kadar okur. Eksi birin sebebi ise, katarin sonunu belirten (\0) sifir degerine yer birakmasidir.

Tabi sonunda, kutugu kapatiyoruz..

DEĞİŞKEN BİR KUTUK İSMİ

```
#include "stdio.h"

main()
{
FILE *fp1;
char birkelime[100],kutukismi[25];
char *c;

printf("Kutuk ismini girin -> ");
scanf("%s",kutukismi); /* istenilen kutuk ismini alalim */

fp1 = fopen(kutukismi,"r");

do {
c = fgets(birkelime,100,fp1); /* kutukten bir satir okuyalim */
if (c != NULL)
printf("%s",birkelime); /* ekrana yazalim */
} while (c != NULL); /* ta ki NULL olana kadar */

fclose(fp1);
}
```

Burada, ilk once kullanicidan "scanf" ile kutuk ismini kullanicidan aliyoruz, daha sonra kutugu acip, satir satir ekrana yaziyoruz.

YAZICIYA NASIL BİRŞEY YOLLAYABİLİRİZ

```
#include "stdio.h"

main()
{
FILE *guzel,*printer;
int c;

guzel = fopen("onsatir.txt","r"); /* kutugu acalim */
printer = fopen("PRN","w"); /* printeri acalim */

do {
c = getc(guzel); /* kutukten bir karakter okuyoruz */
if (c != EOF) {
putchar(c); /* ekranda goruntuleyelim */
putc(c,printer); /* ve yaziciya yollayalim */
}
} while (c != EOF); /* ta ki (End Of File) kutuk bitene kadar */

fclose(guzel);
fclose(printer);
}
```

Okumak icin, "onsatir.txt" yi actiktan sonra, yazmak icin "PRN" isimli kutugu aciyoruz. Printere bir bilgi yollamak, ayni bir kutuge yazmak gibidir, fakat standart bir kutuk ismi kullanmak

zorundayiz. Bu konuda kesin standartlar yoktur, fakat genellikle bu isimler "PRN" , "LPT", "LPT1" yada "LPT2" dir.

Bazi yeni derleyicilerin, "stdprn" diye, onceden tanimli bir kutuk tanimliyicilari vardir. Bu sayede, siz printer'i bir kutuk gibi acmadan, ona veri yollayabilirsiniz.

Program, birer birer butun kutugu okuyup, ekranda gosterir, ve printer'e yollar. EOF, kutuk sonu bulundugunda, kutukler kapanir, ve program biter.