Tip Dönüşümleri (Conversions)

Farklı tiplerdeki değişkenleri birbirlerine eşitlerken gözetmemiz gereken bazı kurallar vardır. En önemli kural, ondalıklı bir sayıyı(double/float) daha düşük duyarlıklı(örn:int) bir değişkene atamamamız gerektiğidir. Bu yapılırsa, sayıların ondalıklı kısımlarını kaybetmiş, sadece tam kısımlarını eşitlemiş oluruz. Diğer bir durum da bir long değişkeni bir integer değişkene atamaya kalktığımız zaman olabilir. Eğer long çok fazla basamaklı sayı içeriyorsa integer haline dönüşürken bozulmalara uğrayabilir.

```
#include <stdio.h>

int main(){
	double d;
	int i;

	i=56;
	d=i;

	printf("d=%lf\n",d);
	/* d=56.000000 basacaktir. */


	d=45.392;

	i=d;

	printf("i=%d\n",i);
	/* i=45 basacaktir. Kusurat yok olur. Cunku tamsayi bir degiskene ondalikli sayi atamaya kalkarsan kaldiramaz, sadece tam kismini alir. */

	return 0;
}
```

Dönüşüm Belirteçleri (Casts)

Türler arasında dönüşüm yapmak için bazı yöntemlere başvururuz:

```
double d;
int i;
float ortalama;

d=(double)i; // i'nin değerini double olarak d'ye yazar.
d=i/3; //i'nin 3'e bölümünün tam kısmını d'ye eşitler. d=16/3=5 gibi.
d=(double)i/3; /* i'nin double halinin üçe bölünmüşünü d'ye yazar.
16/3=5.333333  gibi */

ortalama=(float)(x+y+z)/3; /*toplamın float'a dönüştürülmüş halinin üçe bölümünü float türündeki ortalama değişkenine yazıyoruz.*/
ortalama=(x+y+z)/3.00;
```

Veri Belirteç Tablosu

Veri Türü	printf belirteçi	scanf belirteçi
long double	%Lf	%Lf
double	**%f**	**%lf**
float	%f	%f
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
short	%hd	%hd
char	%c	%c

ETKİNLİK ALANI (SCOPE)

Her değişkenin etkin olduğu bir alan vardır. Bu alan, değişkenin tanımlandığı yere göre belirlenir. Eğer değişkeni bir fonksiyonun içerisinde belirlediysek, o değişken o fonksiyon çalıştığı sürece varlığını sürdürür ve sadece o fonksiyonun içinde etkin olur. Fonksiyonun çalışması sona erdiğinde değişken de yok olur.

```
#include <stdio.h>

void digerFonksiyon(); /*prototip*/

int main(){
	int x=4;

	printf(“x=%d”,x);
	/* Duzgun bir bicimde x=4 diye basacaktir. */

	digerFonksiyon();

	printf(“y=%d”,y);
/* Hata verecektir. Cunku y digerFonksiyon icinde tanimli. main'in icinde degil!*/

	return 0;
}

void digerFonksiyon(){
	int y=9;
	printf(“x=%d”,x);
/* Hata verecektir. Cunku x degiskeni main icinde tanimli. digerFonksiyon icinde
degil */

}
```

```
#include <stdio.h>

int a=5;
void digerFonksiyon1(); // prototip
void digerFonksiyon2(); // prototip

int main(){

	printf(“a=%d”,a);
	/*Duzgun bir bicimde a=5 basacaktir. Cunku a degiskeni global olup tum fonksiyonlardan erisilebilir. */

	return 0;
}



void digerFonksiyon1(){
	printf(“a=%d”,a);
	/*Duzgun bir bicimde a=5 basacaktir. Cunku a degiskeni global olup tum fonksiyonlardan erisilebilir. */
}

void digerFonksiyon2(){
	int a=93;
	printf(“a=%d”,a);
/* a=93 basacaktir. Normalde global degisken olan a(5)'ya ulasmasina ragmen fonksiyon icinde degisken yeniden tanimlandigi için yeni deger gecerli oldu. */

}
```

Süslü Parantezlerle Etki Alanı Oluşturmak

Herhangi bir fonksiyon içerisinde özerk bir etki alanı oluşturmak da mümkün. Ne kadar işlevsel olduğu sorgulanır ama sınavlarda sorulan çıktı sorularında sorulur genelde.

```
#include <stdio.h>

int main(){
	int a=8;

	printf(“a=%d\n”,a);
	/* a=8 basar. */

	{
		int a=4;
		int c=2;
		printf(“a=%d ,  c=%d\n”,a,c);
	/* a=4 , c=2 basar.  cunku icerideki a, disaridakine baskin durumda.*/
	}
```

```
	printf(“a=%d\n”,a);
	/* a=8 basar. Eski degerine geri dondu. Cunku iceride tanimlanan etki alani bitti. main'in etki alani yonetimi devraldi. */

	/* printf(“c=%d”,c); dersek hata verir. Cunku c degiskeni sadece { } arasindaki etki alaninda tanimli. */

	return 0;
}
```

Statik Değişkenler

Bir fonksiyon içerisinde statik olarak tanımlanan bir değişken, fonksiyon sonlandığında değerini bir sonraki fonksiyon çağrısında kullanılmak üzere saklar.

```
#include <stdio.h>

void fonksiyon1(int x); //prototip

int main(){

/*Basilacak yazilari ozet olarak yanlarina yaziyorum*/

	fonksiyon1(5); //Once=2 ; Sonra=7
	fonksiyon1(9); //Once=7 ; Sonra=16
	fonksiyon1(3); //Once=16 ; Sonra=19
	fonksiyon1(49); //Once=19 ; Sonra=68

	return 0;

}

void fonksiyon1(int x){
	static int a=2;

/* fonksiyon ilk cagrildiginda a'nin gecerli olan degeri 2'dir. Sonraki cagrilarda bu ifade onemsenmeyip saklanan ifade dikkate alinacaktir.*/

	printf(“Islem yapilmadan once: a=%d\n”,a);

	a=a+x;

	printf(“Islem yapildiktan sonra: a=%d\n”,a);


}
```

ÖZ YİNELEMELİ (RECURSIVE) FONKSİYONLAR

Bazen yazdığımız kodun daha anlaşılır olmasını sağlamak için öz yinelemeli fonksiyonlara başvururuz. Bu fonksiyonlar kendi içlerinde tekrar kendilerini çağırırlar ve bu işlem belirtilen sınıra ulaştığı zaman durur. Hemen bir örnek üzerinde konuyu inceleyelim.

```
#include <stdio.h>

long faktoriyel(int x);

int main(){

	printf("%d!=%ld\n",5,faktoriyel(5));
	return 0;

}

long faktoriyel(int x){

	if (x==0)
		return 1;

	return x*faktoriyel(x-1);
}
```

Burada faktoriyel(5) ile fonksiyona 5 değerini gönderiyoruz. x=5.
x=5=0 olmadığı için 5*faktoriyel(4) değerini döndürüyoruz.

Burada kullandığımız faktoriyel(4) çağrısı ise kendi içinde 4*faktoriyel(3) değerini döndürüyor. O da faktoriyel(2), o da faktoriyel(1), o da faktoriyel(1), o da faktoriyel(0), o da 1.

Böylece durum şu hale geliyor.
faktoriyel(5)
5*faktoriyel(4)
5*4*faktoriyel(3)
5*4*3*faktoriyel(2)
5*4*3*2*faktoriyel(1)
5*4*3*2*1*faktoriyel(0)
5*4*3*2*1*1

Dikkat ederseniz her bir çağrı, sonlanmak için kendi içinde çağırdığı çağrının sonlanmasını bekliyor. Dolayısıyla ilk çağırdığımız faktoriyel(5) çağrısının sona ermesi için faktoriyel(0) çağrısının sona ermesi, döndürdüğü değerlerin kendisini çağıran fonksiyona(örn: faktoriyel(1)) cevabını döndürmesi gerekmektedir. Bunu bir insan kuyruğuna benzetebiliriz.
A-B-C-D-E-F her biri birer insan olsun.
A, B'ye 4 diyor.
B, C'ye 3 (A'dan duyduğunun bir eksiğini) diyor.
C, D'ye 2 diyor.
D, E'ye 1 diyor.

E, F'ye 0 diyor.

F de bakıyor ki 0'a gelmişim, ben de E'ye 1 cevabını vermeliyim diyor.
1 Cevabını alan E, 1(kendisine ilk söylenen)*1(kendisine verilen cevap)=1 cevabını döndürüyor D'ye.
1 Cevabını alan D, 2*1=2 cevabını döndürüyor C'ye.
2 Cevabını alan C, 3*2=6 cevabını döndürüyor B'ye.
6 Cevabını alan B, 4*6=24 cevabını döndürüyor A'ya.
24 Cevabını alan A, 5*24 = 120 değerini ilk çağrıldığı yere(örn:main) döndürüyor.

Böylece imece tarzı bir hesaplama yapılmış oluyor. Bu hesaplama tarzının eksik yönü ise çok fazla bellek yemesi ve yavaş olması. Çünkü herkes birbirini beklemekle kalmıyor, her bir insanın(çağrılmış fonksiyonun) kapladığı yer hesaplama boyunca kullanılıyor.

Ama kodun anlaşılır olması bazı karmaşık durumlarda işleri kolaylaştırabiliyor.