C Programlamaya Giriş

C dili, bilgisayar programı tasarımına yapılandırılmış ve disiplinli bir yaklaşım sunar. Bu bölüm, C programlamayı tanıtmakta ve birçok temel C özelliğini gösteren birkaç örnek sunmaktadır.

Basit Bir C Programı: Bir Metin Satırını Yazdırma

Bir satır metin yazdıran basit bir C programıyla başlıyoruz. Program ve ekran çıktısı Şekil 2.1'de gösterilmiştir.

```
1  // fig02_01.c
2  // A first program in C.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main(void) {
7     printf("Welcome to C!\n");
8  } // end function main
```

1. ve 2. satırlar

// fig02_01.c

// C'deki ilk program.

// ile başlayan bu iki satır açıklama için verilmiştir. Yorum ekleme işlemi programları belgeler ve programın okunabilirliğini artırır. Programları yürüttüğünüzde bilgisayar açıklama satırlarını basitçe göz ardı eder ve dikkate almaz. Bu satırlar programı daha sonra neden ve nasıl yazdığımızı hatırlamak için eklediğimiz bize özel satırlardır.

Ayrıca, ilk satırdaki /*'dan son satırın sonundaki */'a kadar her şeyin bir yorum olduğu belirtilir. Yani /*…*/ ile çok satırlı yorumları da kullanabilirsiniz. Daha kısa yorumlar için // tercih ediyoruz, çünkü bunlar / *…*/ yorumlarıyla meydana gelen, kazara */ kapanışının atlanması gibi yaygın programlama hatalarını ortadan kaldırır.

#include Ön İşlemci Yönergesi

Satır 3

#include <stdio.h>

bir C önişlemci yönergesidir. Önişlemci, derlemeden önce # ile başlayan satırları işler. 3. satır, önişlemciye standart giriş/çıkış başlığının (<stdio.h>) içeriğini dahil etmesini söyler. Bu, derleyicinin printf (satır 7) gibi standart giriş/çıkış kütüphanesi fonksiyonlarını doğru şekilde kullandığınızdan emin olmak için kullandığı bilgileri içeren bir dosyadır. Bölüm 5, başlıkların içeriğini daha ayrıntılı olarak açıklamaktadır.

Boş Satırlar ve Beyaz Boşluk

4. satırı boş bıraktık. Programların okunmasını kolaylaştırmak için boş satırlar, boşluk karakterleri ve sekme (tab) karakterleri kullanırsınız. Bunlar beyaz boşluk olarak bilinir ve genellikle derleyici tarafından göz ardı edilir.

Ana Fonksiyon (main function)

Satır 6

int main(void) {

Her C programının bir parçasıdır. main'den sonraki parantezler, **main**'in fonksiyon adı verilen bir program yapı taşı olduğunu gösterir. C programları, biri **main** olmak zorunda olan fonksiyonlardan oluşur. Her program **main** fonksiyonda çalışmaya başlar. İyi bir uygulama olarak, her fonksiyondan önce fonksiyonun amacını belirten bir yorum (5. satırdaki gibi) yazılır. Fonksiyonlar bilgi döndürebilir. **main**'in solundaki **int** anahtar sözcüğü, main'in bir tamsayı (tam sayı) değeri "döndürdüğünü" belirtir.

Fonksiyonlar, yürütmeleri istendiğinde de bilgi alabilir. Buradaki parantez içindeki boşluk, **main**'in herhangi bir bilgi almadığı anlamına gelir.

Sol ayraç {, her fonksiyonun gövdesini başlatır (6. satırın sonu). Karşılık gelen bir sağ ayraç }, her fonksiyonun gövdesini sonlandırır (satır 8). Bir program **main**'in kapanış sağ parantezine ulaştığında, program sona erer. Parantezler ve programın aralarındaki kısmı bir blok oluşturur; sonraki bölümlerde daha ayrıntılı olarak tartışacağımız önemli bir program birimidir.

Bir Çıktı Bildirimi

Satır 7

printf("C'ye Hoş Geldiniz!\n");

Bilgisayara bir eylem gerçekleştirmesini, yani ekranda tırnak işaretleri içine alınmış karakter dizisini görüntülemesini söyler. Bir diziye bazen karakter dizisi, mesaj veya hazır bilgi denir.

7. satırın tamamı—printf fonksiyonunun görevini yerine getirmesi için yapılan “çağrı”, printf'in parantez içindeki argümanı ve noktalı virgül (;) dahil—bir ifade olarak adlandırılır. Her deyim noktalı virgül deyimi sonlandırıcı ile bitmelidir. printf içindeki "f", "biçimlendirilmiş" anlamına gelir. 7. satır yürütüldüğünde, **C'ye Hoş Geldiniz!** ifadesi ekrana yazılır. Karakterler genellikle çift tırnak içinde göründükleri gibi yazdırılır, ancak **\n** karakterlerinin görüntülenmediğine dikkat edin.

Çıkış dizileri

Bir dizede (karakter dizisi-string’de), ters eğik çizgi (\) bir çıkış karakteridir. Printf'in sıra dışı bir şey yapması gerektiğini belirtir. Bir dizede, derleyici bir ters eğik çizgiyi bir sonraki karakterle birleştirerek bir çıkış dizisi oluşturur. Çıkış dizisi \n, yeni satır anlamına gelir. printf bir dizede yeni bir satırla karşılaştığında, çıkış imlecini bir sonraki satırın başına konumlandırır. Bazı yaygın çıkış dizileri aşağıda listelenmiştir:

*NOT: Dersin bundan sonraki bölümlerinde karakter satırlarını **dize** olarak adlandıracağız.*

Çıkış Dizisi	Açıklama
\n	İmleci bir sonraki satırın başına taşır.
\t	İmleci bir sonraki yatay sekme durağına taşır.
\a	Geçerli imleç konumunu değiştirmeden sesli veya görünür bir uyarı üretir.
\\	Bir dizede ters eğik çizgi özel bir anlama sahip olduğu için, Bir dizeye ters eğik çizgi karakteri eklemek için \\ gereklidir.
\"	Dizeler çift tırnak içine alındığından, bir dizeye çift tırnak karakteri eklemek için \" gereklidir.

Bağlayıcı ve Yürütülebilir Dosyalar

frintf ve scanf gibi standart kütüphane fonksiyonları, C programlama dilinin parçası değildir. Örneğin, derleyici printf veya scanf'te bir yazım hatası bulamaz. Bir printf deyimi derlenirken, derleyici yalnızca programda kütüphane fonksiyonuna bir "çağrı" için alan sağlar. Ancak derleyici, kütüphane fonksiyonlarının nerede olduğunu bilmez. Bağlayıcı çalıştığında, fonksiyonlarını bulur ve bu fonksiyonlara uygun çağrıları programa ekler. Fonksiyon adı yanlış yazılırsa, bağlayıcı hatayı tespit eder.

Girinti Kuralları

Her fonksiyonun tüm gövdesini, fonksiyonun gövdesini tanımlayan ayraçlar içinde bir sıra girinti (üç boşluk öneririz) ile boşluk oluşturun. Bu girinti, bir programın işlevsel yapısını vurgular ve daha kolay okunmasına yardımcı olur. Tercih ettiğiniz girinti boyutu için bir kural belirleyin ve bu kuralı eşit şekilde uygulayın. Sekme (tab) tuşu girintiler oluşturmak için kullanılabilir. Profesyonel stil kılavuzları genellikle sekme yerine boşluk kullanılmasını önerir. Bazı kod düzenleyiciler, Sekme tuşuna bastığınızda aslında boşluk ekler.

Çoklu printf kullanma

printf fonksiyonu ile C'ye Hoş Geldiniz! birkaç farklı yolla gösterilebilir. Örneğin, Şekil 2.2, Şekil 2.1 ile aynı çıktıyı üretmek için iki ifade kullanır. Bu da çalışır, çünkü her printf bir öncekinin bittiği yerden yazdırmaya devam eder. Satır 7'de Hoş Geldiniz ve ardından bir boşluk görüntülenir (ancak yeni satır yoktur). Satır 8'in printf'i, boşluğun hemen ardından aynı satıra yazdırmaya başlar.

```
1  // fig02_02.c
2  // Printing on one line with two printf statements.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main(void) {
7     printf("Welcome ");
8     printf("to C!\n");
9  } // end function main
```

```
Welcome to C!
```

Fig. 2.2 | Printing one line with two `printf` statements.

Tek bir printf ile Birden Fazla Satır Görüntüleme

Bir printf, Şekil 2.3'teki gibi birkaç satır görüntüleyebilir. Her \n çıktı imlecini bir sonraki satırın başına taşır.

```c
// fig02_03.c
// Printing multiple lines with a single printf.
#include <stdio.h>

// function main begins program execution
int main(void) {
   printf("Welcome\nto\nC!\n");
} // end function main
```

```
Welcome
to
C!
```

Fig. 2.3 | Printing multiple lines with a single `printf`.

Bir sonraki programımız, bir kullanıcı tarafından klavyede yazılan iki tam sayıyı girmek için scanf standart kütüphane fonksiyonunu kullanır, ardından bunların toplamını hesaplar ve sonucu printf kullanarak görüntüler. Program ve örnek çıktı Şekil 2.4'te gösterilmiştir.

```c
// fig02_04.c
// Addition program.
#include <stdio.h>

// function main begins program execution
int main(void) {
   int integer1 = 0; // will hold first number user enters
   int integer2 = 0; // will hold second number user enters

   printf("Enter first integer: "); // prompt
   scanf("%d", &integer1); // read an integer

   printf("Enter second integer: "); // prompt
   scanf("%d", &integer2); // read an integer

   int sum = 0; // variable in which sum will be stored
   sum = integer1 + integer2; // assign total to sum

   printf("Sum is %d\n", sum); // print sum
} // end function main
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

2. satırdaki yorum, programın amacını belirtir. Yine, program yürütmeye main ile başlar (6-20 satırları)—6 ve 20 satırlarındaki parantezler sırasıyla main gövdesinin başlangıcını ve sonunu işaretler.

7. ve 8. satırlar tanımlardır.

int integer1 = 0; // kullanıcının girdiği ilk sayıyı tutacaktır

int integer 2 = 0; // kullanıcının girdiği ikinci sayıyı tutacak

integer 1 ve integer2 adları, programın daha sonra kullanmak üzere değerleri depolayabileceği bellekteki konumlar olan değişkenlerdir. Bu tanımlar, integer1 ve integer2'nin **int** tipine sahip olduğunu belirtir. Bu, herhangi bir tamsayı değerlerini tutacakları anlamına gelir. 7 ve 8 numaralı satırlar, her değişkeni 0 olarak başlatır. Her değişkeni açıkça başlatmak gerekli olmasa da, bunu yapmak birçok yaygın sorunun önlenmesine yardımcı olacaktır.

Tüm değişkenler, bir programda kullanılmadan önce bir ad ve tür ile tanımlanmalıdır. Her değişken tanımını, o değişken kodda ilk kez kullanılmadan önce main içinde herhangi bir yere yerleştirebilirsiniz. Genel olarak değişkenleri ilk kullanımlarına yakın tanımlamanız gerekir.

Bir değişken adı herhangi bir geçerli tanımlayıcı olabilir. Her tanımlayıcı harflerden, rakamlardan ve alt çizgilerden (_) oluşabilir, ancak bir rakamla başlayamaz. C büyük/küçük harfe duyarlıdır, dolayısıyla a1 ve A1 farklı tanımlayıcılardır. Bir değişken adı küçük harfle başlamalıdır. Metnin ilerleyen kısımlarında, büyük harfle başlayan tanımlayıcılara ve tamamı büyük harf kullanan tanımlayıcılara özel bir anlam vereceğiz. Anlamlı değişken adları seçmek, bir programın kendi kendini belgelemesine yardımcı olur, bu nedenle daha az yorum gerekir. Derleyici tarafından oluşturulan tanımlayıcılar ve standart kütüphane tanımlayıcıları ile çakışmaları önlemek için tanımlayıcıları alt çizgi (_) ile başlatmaktan kaçının.

Çok sözcüklü değişken adları, programları daha okunaklı hale getirebilir. Bu tür isimler için:

• kelimeleri **toplam_komisyon** 'da olduğu gibi alt çizgi ile ayırın veya

• kelimeleri birlikte sıralayın ve sonraki her kelimeye **toplamKomisyon** 'ta olduğu gibi büyük harfle başlayın.

İkinci stile deve kılıfı denir, çünkü büyük ve küçük harflerin deseni bir deve silüetini andırır.

Satır 10

printf("Enter First Integer: ");

"İlk tamsayıyı girin: " yazısını görüntüler. Bu mesaj, kullanıcıya belirli bir eylemde bulunmasını söylediği için bilgi istemi olarak adlandırılır.

Satır 11

scanf("%d", &integer1); // bir tamsayı oku

Kullanıcıdan bir değer elde etmek için scanf kullanır. Fonksiyon, genellikle klavye olan standart girişten okur.

scanf'deki "f", "biçimlendirilmiş" anlamına gelir. Bu scanf'in iki bağımsız değişkeni vardır —"%d" ve & integer1. "%d" biçim kontrol dizesidir. Kullanıcının girmesi gereken veri türünü belirtir.

%d dönüştürme özelliği, verilerin bir tamsayı olması gerektiğini belirtir; d, "ondalık tamsayı" anlamına gelir. Her dönüştürme belirtiminin başında bir % karakteri vardır.

scanf'in ikinci bağımsız değişkeni bir ve (ampersant) işaretiyle (&) başlar ve ardından değişken adı gelir. &, adres operatörüdür ve değişken adıyla birleştirildiğinde, scanf'e integer1 değişkeninin belleğindeki konumunu (veya adresini) söyler. scanf daha sonra kullanıcının girdiği değeri o hafıza konumunda saklar. Ve işaretini (&) kullanmak, genellikle yeni başlayan programcılar ve bu gösterimi gerektirmeyen diğer dillerde programlama yapan kişiler için kafa karıştırıcıdır. Şimdilik, her scanf çağrısında her değişkenden önce & işareti koymayı unutmayın. Bu kuralın bazı istisnaları 6. ve 7. Bölümlerde ele alınmıştır. &'nin kullanımı, 7. Bölümde işaretçileri (pointers) inceledikten sonra netleşecektir.

Bir scanf deyimindeki bir değişkenden önce ve işaretini (&) unutmak, genellikle yürütme zamanı hatasına neden olur. Birçok sistemde bu, bir "segmentasyon hatasına" veya "erişim ihlaline" neden olur. Böyle bir hata, bir kullanıcının programı, bilgisayar belleğinin erişim ayrıcalıklarına sahip olmadığı bir bölümüne erişmeye çalıştığında ortaya çıkar. Bu hatanın kesin nedeni Bölüm 7'de açıklanacaktır.

11. satır yürütüldüğünde, bilgisayar kullanıcının integer1 için bir değer girmesini bekler. Kullanıcı bir tamsayı yazar ve ardından sayıyı bilgisayara göndermek için Enter tuşuna (veya Return tuşuna) basar. Bilgisayar daha sonra sayıyı (veya değeri) integer1'e yerleştirir. Programda integer1'e sonraki tüm başvurularda aynı değeri kullanır. printf ve scanf fonksiyonları, kullanıcı ve bilgisayar arasındaki etkileşimi kolaylaştırır. Bu etkileşim bir diyaloğa benzer ve genellikle etkileşimli bilgi işlem olarak adlandırılır.

Satır 13

printf("Enter second integer: ");

kullanıcıdan ikinci tamsayıyı girmesini ister

scanf("%d", &integer2);

kullanıcıdan integer2 değişkeni için bir değer alır.

Satır 16

int sum = 0; // toplamın saklanacağı değişken

int değişkeni sum'ı tanımlar ve biz 17. satırda sum'ı kullanmadan önce onu 0 olarak başlatır.

17. satırdaki atama bildirimi

sum = integer1 + integer2;

integer1 ve integer2 değişkenlerinin toplamını hesaplar, ardından atama operatörünü (=) kullanarak sonucu toplam değişkenine atar.

19. satırdaki "sum %d\n" biçim kontrol dizesi

printf("sum: %d\n", sum); // toplamı yazdır

görüntülenecek bazı hazır bilgi karakterleri ("sum ") ve bir tamsayı için yer tutucu olan %d dönüştürme belirtimini içerir. Toplam, %d yerine eklenecek değerdir. Bir tamsayı (%d) için dönüştürme belirtimi hem printf hem de scanf'te aynıdır—bu, tümü için olmasa da çoğu C veri türü için geçerlidir.

Bir değişkeni tanımında başlatabilirsiniz. Örneğin, 16. ve 17. satırlar integer1 ve integer2 değişkenlerini ekleyebilir, ardından sum değişkenini şu sonuçla başlatabilir:

int sum = integer1 + integer2; // toplamı sum değişkenine atamak

Aslında sum değişkenine ihtiyacımız yok çünkü printf ifadesinde hesaplamayı yapabiliyoruz. Böylece, 16–19 satırları şu şekilde değiştirilebilir:

printf("Sum: %d\n", integer1 + integer2);

Her değişkenin bilgisayarın belleğinde bir adı, türü, değeri ve konumu vardır. Şekil 2.4'ün toplama programında, 11. satır

scanf("%d", &tamsayı1);

kullanıcının girdisini integer1'in bellek konumuna yerleştirir. Kullanıcının integer1 değeri olarak 45 girdiğini varsayalım.

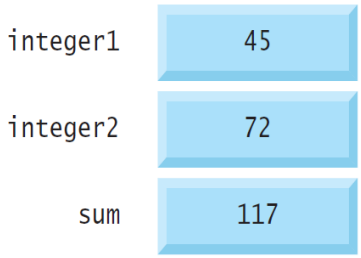
Bir hafıza konumuna bir değer yerleştirildiğinde, konumun kaybolan önceki değerinin yerine geçer. 14. satırda tekrar toplama programımıza dönelim.

scanf("%d", &integer2);

Burada da kullanıcının 72 girdiğini varsayalım. Bu konumların bellekte mutlaka bitişik olması gerekmez. integer1 ve integer2 için değerlerimiz olduğunda, satır 18 bu değerleri toplar ve toplamı önceki değerini değiştirerek sum değişkenine yerleştirir.

sum = integer1 + integer2;

Kavramsal olarak, bellek artık şu şekilde görünür:



integer1 ve integer2 değerleri toplama sonucunda değişmez. Yani, bir bellek konumundan bir değer okumak, o değere zarar vermez.

Çoğu C programı, aşağıdaki aritmetik operatörlerini kullanarak hesaplamalar gerçekleştirir:

C işlemi	**Aritmetik operatör**	**Cebirsel ifade**	**C ifadesi**
Toplama	+	f + 7	f + 7
Çıkarma	–	p – c	p - c
Çarpma	*	bm	b * m
Bölüm	/	x / y	x / y
Kalan	%	r mod s	r % s

Cebirde kullanılmayan çeşitli özel sembollerin kullanımına dikkat edin. Yıldız işareti (*) çarpmayı gösterir ve yüzde işareti (%) kalan operatörünü gösterir. Cebirde a ile b'yi çarpmak için bu tek harfli değişken adlarını ab'deki gibi yan yana koyarız. C'de ab, tek, iki harfli bir ad (veya tanımlayıcı) olarak yorumlanır. Çoğu programlama dili, a * b'de olduğu gibi * operatörünü kullanarak çarpmayı belirtir.

Tamsayı bölme (yani, bir tamsayıyı diğerine bölme) bir tamsayı sonucu verir, bu nedenle 7/4, 1 olarak değerlendirilir ve 17/5, 3 olarak değerlendirilir. 7%4, 3 verir ve 17%5, 2 verir.

Bilgisayar sistemlerinde sıfıra bölme girişimi genellikle tanımsızdır. Genel olarak, programın işini başarıyla gerçekleştirmeden hemen sonlandırılmasına neden olan ölümcül bir hatayla (fatal error) sonuçlanır. Ölümcül olmayan hatalar ise programların genellikle yanlış sonuçlara yol açarak tamamlanmasına sebep olur.

C ifadelerinde parantezler cebirsel ifadelerde olduğu gibi kullanılır. Örneğin a ile b + c ifadelerini çarpmak için a * (b + c) yazarız.

C, aritmetik ifadelerdeki operatörleri, genellikle cebirdekilerle aynı olan operatör önceliği kuralları tarafından belirlenen bir sırayla uygular:

Aşağıdaki ifade bir doğrunun denklemidir. Burada paranteze ihtiyaç yoktur :

Cebir: y = mx + b

C: y = m * x + b;

Cebir: z = pr mod q + w/x – y

C: z = p * r % q + w / x - y;

Operatör öncelik kurallarını daha iyi anlamak için, C'nin ikinci dereceden bir polinomu nasıl değerlendirdiğine bir göz atalım.

y = a * x * x + b * x + c;

Cebirde olduğu gibi, bir ifadeyi daha net hale getirmek için gereksiz parantezler kullanmak kabul edilebilir. Dolayısıyla, önceki ifade aşağıdaki gibi parantez içine alınabilir:

y = (a* x * x) + (b * x) + c;

Koşul İfadeleri

Yürütülebilir ifadeler ya hesaplamalar, gerçekleştirir ya da kararlar verir. Örneğin, bir program bir kişinin bir sınavdaki notunun 60'a eşit mi olduğunu belirleyebilir, böylece “Tebrikler! Geçtin." Koşul, doğru (yani koşul karşılanmıştır) veya yanlış (yani koşul karşılanmamıştır) olabilen bir ifadedir.

Bu bölüm, bir programın bir koşulun değerine dayalı olarak karar vermesini sağlayan **if** ifadesini tanıtır. Koşul doğruysa, **if** ifadesinin gövdesindeki ifade yürütülür; Aksi takdirde, diğer işleme geçer.

Aşağıdaki eşitlik ve ilişki operatörleri kullanılarak koşullar oluşturulur:

Cebirsel eşitlik veya ilişkisel operatör	C eşitliği veya ilişkisel operatör	Örnek C durumu	C koşulunun anlamı
ilişkisel operatörler			
>	>	x > y	x, y'den büyüktür
<	<	x < y	x, y'den küçüktür
≥	>=	x >= y	x y'den büyük veya eşittir
≤	<=	x <= y	x, y'den küçük veya eşittir
Eşitlik operatörleri			
=	==	x == y	x eşittir y
≠	!=	x != y	x y'ye eşit değil

<, <=, > ve >= ilişki operatörleri aynı önceliğe ve soldan sağa aynı gruba sahiptir. Eşitlik operatörleri == ve != aynı önceliğe sahiptir ve bu öncelik, ilişkisel operatörlerden ve soldan sağa gruplama operatörlerinden daha düşüktür. C'de bir koşul aslında sıfır (false) veya sıfır olmayan (true) bir değer üreten herhangi bir ifade olabilir.

(==) ifadesini atama operatörüyle (=) karıştırmak yaygın bir programlama hatasıdır. Göreceğiniz gibi, bu operatörlerin karıştırılması, derleme hatalarından ziyade bulunması zor mantık hatalarına neden olabilir.

(==) öğesini atama operatörüyle (=) karıştırmak yaygın bir programlama hatasıdır. Bu operatörlerin karıştırılması, derleme hatalarından ziyade bulunması zor mantık hatalarına neden olabilir.

Şekil 2.5, kullanıcı tarafından girilen iki sayıyı karşılaştırmak için altı tane “**if**” ifadesi kullanmıştır. Doğru koşulu olan her “if” ifadesi için karşılık gelen printf yürütülür. Program ve üç örnek yürütme çıktısı şekilde gösterilmiştir.

```c
// fig02_05.c
// Using if statements, relational
// operators, and equality operators.
#include <stdio.h>

// function main begins program execution
int main(void) {
   printf("Enter two integers, and I will tell you\n");
   printf("the relationships they satisfy: ");

   int number1 = 0; // first number to be read from user
   int number2 = 0; // second number to be read from user

   scanf("%d %d", &number1, &number2); // read two integers

   if (number1 == number2) {
      printf("%d is equal to %d\n", number1, number2);
   } // end if

   if (number1 != number2) {
      printf("%d is not equal to %d\n", number1, number2);
   } // end if

   if (number1 < number2) {
      printf("%d is less than %d\n", number1, number2);
   } // end if

   if (number1 > number2) {
      printf("%d is greater than %d\n", number1, number2);
   } // end if

   if (number1 <= number2) {
      printf("%d is less than or equal to %d\n", number1, number2);
   } // end if

   if (number1 >= number2) {
      printf("%d is greater than or equal to %d\n", number1, number2);
   } // end if
} // end function main
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
 the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

Program, number1 ve number2 **int** değişkenlerine iki tamsayı okumak için scanf'i (14. satır) kullanır. İlk %d, number1 değişkeninde depolanacak bir değeri dönüştürür. İkincisi, number2 değişkeninde depolanacak bir değeri dönüştürür.

16–18 satırlarındaki “if” ifadesi, eşitlik için number1 ve number2 değişkenlerinin değerlerini karşılaştırır.

```
if (number1 == number2) {

printf("%d eşittir %d\n", number1, number2);

}
```

Değerler eşitse, 17. satırda sayıların eşit olduğunu belirten bir metin satırı görüntülenir. 20, 24, 28, 32 ve 36. satırlarda başlayan “if” ifadelerindeki her doğru koşul için, ilgili ifade bir metin satırı görüntüler. Her bir “if” ifadesinin gövdesini girintilemek ve her bir “if” ifadesinin üstüne ve altına boş satırlar yerleştirmek, programın okunabilirliğini artırır.

Sol ayraç {, her “if” ifadesinin gövdesini başlatır (örneğin, satır 16). Karşılık gelen bir sağ ayraç }, her bir “if” ifadesinin gövdesini sonlandırır (örneğin, satır 18). Bir “if” ifadesinin gövdesine herhangi bir sayıda ifade yerleştirilebilir.

Aşağıdaki tablo, operatörlerin önceliğini en yüksekten en düşüğe listeler:

Operatör	Gruplama
()	soldan sağa
* / %	soldan sağa
+ -	soldan sağa
< <= > >=	soldan sağa
== !=	soldan sağa
=	soldan sağa

Atama operatörü (=) sağdan sola gruplar. Çok sayıda operatör (işleç) içeren ifadeler yazarken operatör (işleç) öncelik tablosuna bakın. İfadedeki operatörlerin doğru sırada uygulandığından emin olun. Karmaşık bir ifadede değerlendirme sırasından emin değilseniz, ifadeleri gruplandırmak için parantez kullanın veya ifadeyi birkaç basit ifadeye bölün.

Bu bölümdeki örneklerde kullandığımız “int”, “if” ve “void” gibi bazı sözcükler, dilin anahtar sözcükleridir veya ayrılmış sözcükleridir. Derleyici için özel anlamlara sahiptir. Aşağıdaki tablo C’deki anahtar kelimelerini vermektedir. Bunları tanımlayıcı olarak ya da değişken olarak kullanmayın.

anahtar kelimeler
auto do goto signed unsigned break double if sizeof void case else int static volatile char enum long struct while const extern register switch continue float return typedef default for short union
C99 standardına eklenen anahtar kelimeler
_Bool _Complex _Imaginary inline restrict
C11 standardına eklenen anahtar kelimeler
_Alignas _Alignof _Atomic _Generic _Noreturn _Static _assert _Thread _local

Bu bölüm, ekranda veri görüntüleme, kullanıcıdan veri girme, hesaplama yapma ve karar verme gibi birçok önemli C özelliğini tanıttı. Bir sonraki bölümde, yapılandırılmış programlamayı tanıtırken bu teknikleri geliştireceğiz. İfadelerin yürütülme sırasını nasıl belirleyeceğimizi inceleyeceğiz.