Fonksiyonlar

Gerçek dünya sorunlarını çözen çoğu bilgisayar programı, ilk birkaç bölümde sunulanlardan çok daha büyüktür. Deneyimler, bir programı geliştirmenin ve sürdürmenin en iyi yolunun, onu her biri orijinal programdan daha yönetilebilir olan daha küçük parçalardan oluşturmak olduğunu göstermiştir. Bu tekniğe böl ve yönet denir. Büyük programların tasarlanması, uygulanması, çalıştırılması ve sürdürülmesi için bazı temel C özelliklerini açıklayalım.

C Programlarını Modülerleştirme

C'de, yazdığınız yeni fonksiyonları önceden paketlenmiş C standart kütüphane fonksiyonları ile birleştirerek programları modüler hale getirebilirsiniz. C standart kütüphanesi, ortak matematiksel hesaplamalar, dizi işlemleri, karakter işlemleri, giriş/çıkış ve diğer birçok yararlı işlemi gerçekleştirmek için zengin bir fonksiyon koleksiyonu sağlar. Önceden paketlenmiş fonksiyonlar, ihtiyacınız olan yeteneklerin çoğunu sağladıkları için işinizi kolaylaştırır. C standardı, C dilini ve onun standart kütüphanesini içerir—standart C derleyicileri her ikisini de uygular. Önceki bölümlerde kullandığımız printf, scanf ve pow işlevleri standart kütüphane fonksiyonlarıdır.

Bir programda birçok noktada kullanılabilecek belirli görevleri gerçekleştirmek için fonksiyonlar tanımlayabilirsiniz. Fonksiyonu tanımlayan ifadeler bir kez yazılır ve diğer fonksiyonlardan gizlenir. Göreceğimiz gibi, bu tür bir gizleme, iyi bir yazılım mühendisliği için çok önemlidir. Fonksiyonlar, adını ve fonksiyonun belirlenen görevi gerçekleştirmek için ihtiyaç duyduğu bilgileri (argümanlar olarak) sağlayan bir fonksiyon çağrısı tarafından çağrılır.

Matematik Kütüphanesi Fonksiyonları

C'nin matematik kütüphanesi fonksiyonları (math.h başlığı), yaygın matematiksel hesaplamalar yapmanızı sağlar. Bu bölümde bu fonksiyonların çoğunu kullanıyoruz. 900.0'ın karekökünü hesaplamak ve yazdırmak için;

```
printf("%.2f", sqrt(900.0));
```

Bu ifade yürütüldüğünde, 900.0'ın karekökünü hesaplamak için sqrt <math.h> kütüphane fonksiyonunu çağırır ve ardından sonucu 30.00 olarak yazdırır. **sqrt** fonksiyonu, double türünde bir argüman alır ve double türünde bir sonuç döndürür. Aslında<math.h> kütüphanesinde kayan nokta değerleri döndüren tüm fonksiyonlar, double veri türünü döndürür. Float değerler gibi çift değerlerin %f dönüştürme belirtimi kullanılarak çıktı alınabileceğini unutmayın. Bir fonksiyon çağrısının sonucunu aşağıdaki gibi daha sonra kullanmak üzere bir değişkende saklayabilirsiniz;

```
double result = sqrt(900.0);
```

Fonksiyon bağımsız değişkenleri sabitler, değişkenler veya ifadeler olabilir. c = 13.0, d = 3.0 ve f = 4.0, ise şu ifade;

```
printf("%.2f", sqrt(c + d * f));
```

13.0 + 3.0 * 4.0 = 25.0'in karekökünü hesaplar ve 5.00 olarak yazdırır. Aşağıdaki tablo birkaç C <math.h> kütüphane fonksiyonunu özetlemektedir. Tabloda x ve y değişkenleri double tipindedir. C11 standardında, complex.h başlığı aracılığıyla komplex sayı işlemleri de eklenmiştir.

Function	Description	Example
sqrt(x)	x'in karekökü	sqrt(900.0); değeri=30.0 sqrt(9.0); değeri=3.0
cbrt(x)	x'in küp kökü (yalnızca C99 ve C11)	cbrt(27.0); değeri=3.0 cbrt(-8.0) ; değeri= -2.0
exp(x)	üstel fonksiyon e^x	exp(1.0) ; değeri=2.718282 exp(2.0) ; değeri= 7.389056
log(x)	x'in doğal logaritması (e tabanı)	log(2.718282) ; değeri= 1.0 log(7.389056) ; değeri= 2.0
log10(x)	x'in logaritması (10 tabanında)	log10(1.0) ; değeri= 0.0 log10(10.0) ; değeri= 1.0 log10(100.0) ; değeri= 2.0
fabs(x)	kayan noktalı sayı olarak x'in mutlak değeri	fabs(13.5) ; değeri= 13.5 fabs(0.0) ; değeri= 0.0 fabs(-13.5) ; değeri=13.5
ceil(x)	X değerini x'ten küçük olmayan en küçük tam sayıya yuvarlar	ceil(9.2) ; değeri= 10.0 ceil(-9.8) ; değeri= -9.0
floor(x)	X değerini x'ten büyük olmayan en büyük tamsayıya yuvarlar	floor(9.2) ; değeri= 9.0 floor(-9.8) ; değeri= -10.0
pow(x, y)	x'in üssü y (x^y)	pow(2, 7) ; değeri= 128.0 pow(9, .5) ; değeri= 3.0
fmod(x, y)	kayan noktalı sayı olarak x/y'nin kalanı	fmod(13.657, 2.333) ; değeri= 1.992
sin(x)	x'in trigonometrik sinüsü (x radyan cinsinden)	sin(0.0) ; değeri= 0.0
cos(x)	x'in trigonometrik kosinüsü (x radyan cinsinden)	cos(0.0) ; değeri= 1.0
tan(x)	x'in trigonometrik tanjantı (x radyan cinsinden)	tan(0.0) ; değeri= 0.0

Fonksiyonlar

Fonksiyonlar, bir programı modüler hale getirmenize izin verir. Çok sayıda fonksiyon içeren programlarda, **main** genellikle programın çalışmasının büyük bölümünü gerçekleştiren fonksiyonlara yapılan bir çağrı grubu olarak uygulanır.

Bir programı "işlevselleştirmek" için çeşitli motivasyonlar vardır.

- Böl ve yönet yaklaşımı, program geliştirmeyi daha yönetilebilir hale getirir.
- Başka bir motivasyon, mevcut fonksiyonları kullanarak yeni programlar oluşturmaktır. Bu tür bir yazılımın yeniden kullanılabilirliği, C++, Java, C# ("C sharp" olarak telaffuz edilir), Objective-C ve Swift gibi C'den türetilen nesne yönelimli programlama dillerinde önemli bir kavramdır. İyi bir fonksiyon adlandırma ve tanımlamayla, özel kod yerine belirli görevleri yerine getiren standartlaştırılmış fonksiyonlardan programlar oluşturabilirsiniz. Bu soyutlama olarak bilinir. Printf, scanf ve pow gibi standart kütüphane fonksiyonlarını her kullandığımızda soyutlamayı kullanırız.
- Üçüncü bir motivasyon, bir programda kodu tekrarlamaktan kaçınmaktır. Bir fonksiyon olarak paketleme kodu, o fonksiyonu çağırarak diğer program konumlarından yürütülmesine izin verir.

Her fonksiyon, iyi tanımlanmış tek bir görevi gerçekleştirmekle sınırlandırılmalı ve fonksiyon adı bu görevi ifade etmelidir. Bu, soyutlamayı kolaylaştırır ve yazılımın yeniden kullanımını teşvik eder. Fonksiyonun ne yaptığını açıklamak için kısa ve öz bir ad seçemiyorsanız, çok fazla farklı görev gerçekleştiriyor olabilir. Böyle bir fonksiyonu daha küçük işlevlere bölmek genellikle en iyisidir; Bu, ayrıştırma adı verilen bir işlemdir.

Fonksiyon Tanımları

Sunduğumuz her program, görevlerini yerine getirmek için standart kütüphane fonksiyonları olarak adlandırılan main adı verilen bir fonksiyondan oluşuyordu. Şimdi özel fonksiyonların nasıl yazılacağına bakalım.

1- kare Fonksiyonu

1'den 10'a kadar olan tam sayıların karelerini hesaplamak ve yazdırmak için bir kare fonksiyonu kullanan bir program düşünün (Şekil 5.1).

NOT: Bu programda int tanımını aşağıdaki gibi for döngüsünün dışında tanımlayın.

....
int x;
.....

```
1  // fig05_01.c
2  // Creating and using a function.
3  #include <stdio.h>
4
5  int square(int number); // function prototype
6
7  int main(void) {
8     // loop 10 times and calculate and output square of x each time
9     for (int x = 1; x <= 10; ++x) {
10       printf("%d  ", square(x)); // function call
11    }
12
13    puts("");
14 }
15
16 // square function definition returns the square of its parameter
17 int square(int number) { // number is a copy of the function's argument
18    return number * number; // returns square of number as an int
19 }
```

“printf” ifadesinde (satır 10) kare alma fonksiyonu (square) çağrılır. square fonksiyonu (satır 17), bağımsız değişken x'in değerinin bir kopyasını alır. Daha sonra square, number*number değerini hesaplar ve sonucu main'de square'in çağrıldığı 10. satıra geri iletir. 10. satır, sonucu ekranda görüntüleyen printf fonksiyonuna iletir. Bu işlem, for ifadesinin her yinelemesinde bir kez olmak üzere 10 kez tekrarlanır. square fonksiyonunun tanımı (satır 17–19), bir int parametre değeri beklediğini gösterir. square fonksiyonundaki return ifadesi, number*number sonucunu çağıran fonksiyona geri iletir.

Satır 5’teki
int square(int number); // fonksiyon prototipi

ifadesi bir fonksiyon prototipidir. Parantez içindeki int, derleyiciye square'in arayandan bir tamsayı değeri almayı beklediğini bildirir. square fonksiyon adının solundaki int, derleyiciye square'in çağırana bir tamsayı sonucu döndürdüğünü bildirir. Bir fonksiyon prototipinin sonunda noktalı virgülü unutmak bir sözdizimi (syntax) hatasıdır.

Anlamlı fonksiyon adları ve anlamlı parametre adları seçmek, programları daha okunaklı hale getirir ve aşırı yorumlardan kaçınmaya yardımcı olur. Programlar küçük fonksiyonların koleksiyonları olarak yazılmalıdır. Bu, programların yazılmasını, hata ayıklamasını, bakımını, değiştirilmesini ve yeniden kullanılmasını kolaylaştırır. Çok sayıda parametre gerektiren bir fonksiyon çok fazla görev gerçekleştiriyor olabilir. Fonksiyonu ayrı görevleri gerçekleştiren daha küçük fonksiyonlara bölmeyi alışkanlık haline getirmek gerekir. Fonksiyonun dönüş tipi, adı ve parametre listesi mümkünse tek satıra

sığmalıdır. Fonksiyon tanımlarında tanımlanan tüm değişkenler yerel değişkenlerdir; bunlara yalnızca tanımlandıkları fonksiyonda erişilebilir. Çoğu fonksiyonun, fonksiyon çağrılarındaki argümanlar aracılığıyla fonksiyonlar arasında iletişim kurmayı sağlayan parametreleri vardır. Bir fonksiyonun parametreleri aynı zamanda o fonksiyonun yerel değişkenleridir.

Derleyici, aşağıdakileri sağlamak için square'in çağrısını (10. satır) prototipiyle karşılaştırır:
• argüman sayısı doğru mu?,
• bağımsız değişkenler doğru türde mi?,
• bağımsız değişken türleri doğru sırada mı?,
• dönüş türü, fonksiyonun çağrıldığı bağlamla tutarlı mı?.

Fonksiyon prototipi, fonksiyon tanımının ilk satırı ve fonksiyon çağrılarının tümü, bağımsız değişkenlerin ve parametrelerin sayısı, türü ve sırası konusunda uyumlu olmalıdır. Fonksiyon prototipi ve fonksiyon başlığı, fonksiyonun çağrılabileceği yeri etkileyen aynı dönüş türüne sahip olmalıdır.

Fonksiyon Tanımının Formatı

Bir fonksiyon tanımının formatı şöyledir;
dönüş değerinin türü - fonksiyon adı (parametre listesi) {
ifadeler
}

Fonksiyon adı, herhangi bir geçerli tanımlayıcıdır. Dönüş değeri türü, çağırana döndürülen sonucun türüdür. Dönüş değeri tipi **void** ise, bir fonksiyonun bir değer döndürmediğini gösterir. Dönüş değeri türü, fonksiyon adı ve parametre listesi birlikte bazen fonksiyon başlığı olarak ta anılır.
Parametre listesi, çağrıldığında fonksiyon tarafından alınan parametreleri virgülle ayrılmış bir listedir. Bir fonksiyon herhangi bir parametre almazsa, parametre listesi **void** anahtar kelimeyi içermelidir. Her parametre kendi türünü içermelidir; aksi halde bir derleme hatası oluşur.

Bir fonksiyon tanımında parametre listesinin sağ parantezinden sonra noktalı virgül koymak, bir parametreyi bir fonksiyonda yerel değişken olarak yeniden tanımlamak gibi bir hatadır. Bunu yapmak yanlış olmasa da, bir fonksiyonun bağımsız değişkenleri ve fonksiyon tanımındaki ilgili parametreler için aynı adları kullanmayın; bu, belirsizliği önlemeye yardımcı olur.
Parantez içindeki ifadeler, aynı zamanda bir blok olan fonksiyon gövdesini oluşturur. Yerel değişkenler herhangi bir blokta bildirilebilir ve bloklar iç içe olabilir. Fonksiyonlar iç içe olamaz—bir fonksiyonu başka bir fonksiyonun içinde tanımlamak bir sözdizimi hatasıdır (syntax error).

Fonksiyon bir sonuç döndürmezse, kontrol, yalnızca fonksiyonu sonlandıran sağ ayraçlara ulaşıldığında veya aşağıdaki komutu çalıştırarak döndürülür;
return;
Fonksiyon bir sonuç döndürürse, aşağıdaki komut ifadenin değerini çağırana döndürür.
return *expression;*

main fonksiyonun int dönüş değeri, programın doğru yürütülüp yürütülmediğini gösterir. C'nin önceki sürümlerinde, aşağıdakileri açıkça main'in sonuna yerleştirilirdi;

return 0 ; *// 0, programın başarıyla sonlandırıldığını gösterir*

C standardı, main'in dolaylı olarak 0 döndürdüğünü belirtir.

2- maksimum Fonksiyonu

Üç tam sayıdan en büyüğünü döndüren özel bir maksimum fonksiyonu ele alalım (Şekil 5.2). Bu değer, maksimum dönüş ifadesiyle (satır 32) main'e döndürülür. 17. satırdaki printf ifadesi daha sonra maksimum tarafından döndürülen değeri yazdırır.

```
1  // fig05_02.c
2  // Finding the maximum of three integers.
3  #include <stdio.h>
4
5  int maximum(int x, int y, int z); // function prototype
6
7  int main(void) {
8     int number1 = 0; // first integer entered by the user
9     int number2 = 0; // second integer entered by the user
10    int number3 = 0; // third integer entered by the user
11
12    printf("%s", "Enter three integers: ");
13    scanf("%d%d%d", &number1, &number2, &number3);
14
15    // number1, number2 and number3 are arguments
16    // to the maximum function call
17    printf("Maximum is: %d\n", maximum(number1, number2, number3));
18 }
19
20 // Function maximum definition
21 int maximum(int x, int y, int z) {
22    int max = x; // assume x is largest
23
24    if (y > max) { // if y is larger than max,
25       max = y; // assign y to max
26    }
27
28    if (z > max) { // if z is larger than max,
29       max = z; // assign z to max
30    }
31
32    return max; // max is largest value
33 }
```

Fonksiyon başlangıçta ilk bağımsız değişkeninin (x parametresinde depolanan) en büyük olduğunu varsayar ve onu maksimuma atar (satır 22). Ardından, 24-26. satırlardaki if ifadesi, y'nin max'tan büyük olup olmadığını belirler ve öyleyse, y'yi max'a atar. Ardından, 28–30 satırlarındaki if ifadesi, z'nin max'tan büyük olup olmadığını belirler ve öyleyse, z'yi max'a atar. Son olarak, 32. satır çağırana max değerini döndürür.

Fonksiyon Prototipleri: Daha Detaylı Bir Bakış

Önemli bir C özelliği, fonksiyon prototipidir. Derleyici, fonksiyon çağrılarını doğrulamak için fonksiyon prototiplerini kullanır.
Şekil 5.2'deki (satır 5) maximum için fonksiyon prototipi şöyle idi;

int maximum(int x, int y, int z); *// fonksiyon prototipi*

Maximum'un int türünde üç argüman aldığını ve bir int sonucu döndürdüğünü belirtir. Fonksiyon prototipinin (noktalı virgül hariç) maximum tanımın ilk satırıyla aynı olduğuna dikkat edin. Dokümantasyon amacıyla fonksiyon prototiplerine parametre adlarını dahil ediyoruz. Derleyici bu adları yok sayar, dolayısıyla aşağıdaki prototip de geçerlidir:

int maximum(int, int, int);

Fonksiyon prototipiyle eşleşmeyen bir fonksiyon çağrısı, bir derleme hatasıdır. Fonksiyon prototipi ve fonksiyon tanımı aynı değilse de bu bir hatadır. Örneğin Şekil 5.2'de fonksiyon prototipi şöyle yazılmış olsaydı;

void maximum(int x, int y, int z);

Bu durumda derleyici bir hata üretirdi, çünkü fonksiyon prototipinin **void** return türü, fonksiyon başlığındaki **int** return türünden farklı olacaktır. Fonksiyon prototiplerinin bir diğer önemli özelliği argüman yönlendirmedir, yani argümanları dolaylı olarak uygun türe dönüştürmektir. Örneğin, bir tamsayı bağımsız değişkeniyle **sqrt** fonksiyonu için <math.h> kütüphane fonksiyonunun çağrılması, <math.h> içindeki fonksiyon prototipi bir double parametresi belirtse bile yine de çalışır. Aşağıdaki ifade sqrt(4)'ü doğru bir şekilde değerlendirir ve 2.000'i yazdırır:

printf("%.3f\n", **sqrt**(4));

Fonksiyon prototipi, derleyicinin **int** değeri 4'ün bir kopyasını sqrt'ye geçirmeden önce double değer 4.0'a dönüştürmesine neden olur. Genel olarak, fonksiyon prototipinin parametre türlerine tam olarak karşılık gelmeyen bağımsız değişken değerleri, fonksiyon çağrılmadan önce uygun türe dönüştürülür. C'nin olağan aritmetik dönüştürme kurallarına uyulmazsa, bu tür dönüştürmeler yanlış sonuçlara yol açabilir. Bu kurallar, verileri kaybetmeden değerlerin diğer türlere nasıl dönüştürülebileceğini belirtir. **sqrt** örneğimizde, bir **int** değeri değiştirilmeden otomatik olarak double'a dönüştürülür; double int'den çok daha geniş bir değer aralığını temsil edebilir. Bununla birlikte, bir **int**'e dönüştürülen bir **double**, **double**'ın kesirli kısmını keser ve böylece orijinal değeri değiştirir. Büyük tamsayı türlerinin küçük tamsayı türlerine dönüştürülmesi (örneğin, uzundan kısaya) değerleri de değiştirebilir.

Olağan aritmetik dönüştürme kuralları derleyici tarafından işlenir. Karma türde ifadeler, yani birden çok veri türünün değerlerini içeren ifadeler için geçerlidirler. Bu tür ifadelerde derleyici, dönüştürülmesi gereken değerlerin geçici kopyalarını oluşturur, ardından kopyaları ifadedeki "en yüksek" türe dönüştürür; bu, yükseltme olarak bilinir. En az bir kayan nokta değeri içeren karışık türde ifadeler için:

- Değerlerden biri long double ise, diğer değerler long double'a dönüştürülür.
- Değerlerden biri double ise, diğer değerler double'a dönüştürülür.
- Değerlerden biri kayan nokta (floating point) ise, diğer değerler kayan nokta (floating point) değere dönüştürülür.

Karışık türde ifade yalnızca tamsayı türleri içeriyorsa, olağan aritmetik dönüştürmeler bir dizi tamsayı yükseltme kuralı belirtir. Aşağıdaki tablo, her türün printf ve scanf dönüştürme belirtimleriyle birlikte kayan noktalı ve tamsayı veri türlerini listeler. Çoğu durumda, aşağıdaki tabloda daha düşük olan tamsayı türleri daha yüksek türlere dönüştürülür:

Veri tipi	**printf** dönüştürme belirtimi	**scanf** dönüştürme belirtimi
Floating-point tipi		
long double	%lf	%lf
double	%f	%lf
float	%f	%f
Integer tipi		
unsigned long long int	%llu	%llu
long long int	%lld	%lld
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
unsigned short	%hu	%hu
short	%hd	%hd
char	%c	%c

Bir değer, yalnızca değeri açıkça daha düşük türdeki bir değişkene atayarak veya bir atama operatörü kullanılarak daha düşük bir türe dönüştürülebilir. Bağımsız değişkenler, bir fonksiyon prototipinde belirtilen parametre türlerine, sanki bağımsız değişkenler bu türlerin değişkenlerine atanıyormuş gibi dönüştürülür.

Dolayısıyla, Şekil 5.1'deki square fonksiyonumuza bir double iletirsek, double değer int'e (daha düşük bir tür) dönüştürülür ve square genellikle yanlış bir değer döndürür. Örneğin, square(4.5), 20.25 değil, 16 döndürür. Dönüştürme hiyerarşisinde daha yüksek bir veri türünden daha düşük bir veri türüne dönüştürme, veri değerini değiştirebilir. Birçok derleyici bu gibi durumlarda uyarı verir.

Fonksiyon Prototip Notları

Bir fonksiyon için fonksiyon prototipi yoksa, derleyici, fonksiyonun ilk oluşumundan itibaren bir tane oluşturur. Bu, derleyiciye bağlı olarak genellikle uyarılara veya hatalara yol açabilir. Derleme hatalarını ve uyarıları önlemeye yardımcı olmak için, programınızda tanımladığınız veya kullandığınız fonksiyonlar için her zaman fonksiyon prototiplerini dahil etmek gerekir.

Herhangi bir fonksiyon tanımının dışına yerleştirilen bir fonksiyon prototipi, fonksiyon prototipinden sonra görünen fonksiyona yapılan tüm çağrılar için geçerlidir. Bir fonksiyon gövdesine yerleştirilen bir fonksiyon prototipi, yalnızca o fonksiyonda o prototipten sonra yapılan çağrılar için geçerlidir.

Fonksiyon Çağrısı ve Yığın Bellek Yapısı

C'nin Fonksiyon çağrılarını nasıl gerçekleştirdiğini anlamak için, önce yığın olarak bilinen bir veri yapısını (yani, ilgili veri öğelerinin koleksiyonunu) dikkate almamız gerekir. Bir yığını, üst üste konulmuş tabak yığınına benzer şekilde düşünün. Genellikle en üste bir tabak yerleştirirsiniz - bu, tabağı yığının üzerine koymak **Push** olarak adlandırılır. Benzer şekilde, tipik olarak en üstteki tabağı geri alırsınız - bu, tabağı yığından çıkarmak **Pop** olarak adlandırılır. Yığınlar, Son Giren İlk Çıkar (LIFO- Last In Firs Out) veri yapıları olarak bilinir; yığına eklenen son öğe, yığından çıkarılan ilk öğedir.

Bilgisayar öğrencilerinin anlaması gereken önemli bir mekanizma, fonksiyon çağrısı yığınıdır. "Perde arkasında" çalışan bu veri yapısı, fonksiyon çağırma/döndürme mekanizmasını işletir. Bu bölümde göreceğiniz gibi, fonksiyon çağrısı yığını, çağrılan her fonksiyonun yerel değişkenlerini oluşturmayı, sürdürmeyi ve yok etmeyi de sağlar.

Her fonksiyon çağrıldığında, diğer fonksiyonları çağırabilir ve bu da diğer fonksiyonları çağırabilir. Her fonksiyon sonunda program kontrolünü çağırana geri vermelidir. Bu nedenle, her fonksiyonun kontrolü onu çağıran fonksiyona döndürmek için ihtiyaç duyduğu dönüş adreslerini takip etmelidir. Fonksiyon çağrısı yığını, bu bilgileri tutmak için mükemmel bir veri yapısıdır. Bir fonksiyon başka bir fonksiyonu her çağırdığında, yığına bir giriş gönderilir. Yığın çerçevesi (stack frame) adı verilen bu giriş, çağrılan fonksiyonun çağıran fonksiyona geri dönmesi için ihtiyaç duyduğu dönüş adresini içerir. Ayrıca bazı ek bilgiler de içerir. Çağrılan bir fonksiyon geri döndüğünde, fonksiyon çağrısı için yığın açılır ve programın kontrolü açılan yığın çerçevesinde belirtilen dönüş adresine aktarılır.

Çağrılan her fonksiyon, çağırana geri döndürmek için ihtiyaç duyduğu bilgileri her zaman çağrı yığınının en üstünde bulur. Çağrılan bir fonksiyon başka bir fonksiyonu çağırırsa, yeni fonksiyon çağrısı için bir yığın çerçevesi çağrı yığınına itilir. Böylece, yeni çağrılan fonksiyonun çağırana geri dönmesi için gereken dönüş adresi artık yığının en üstünde yer alır.

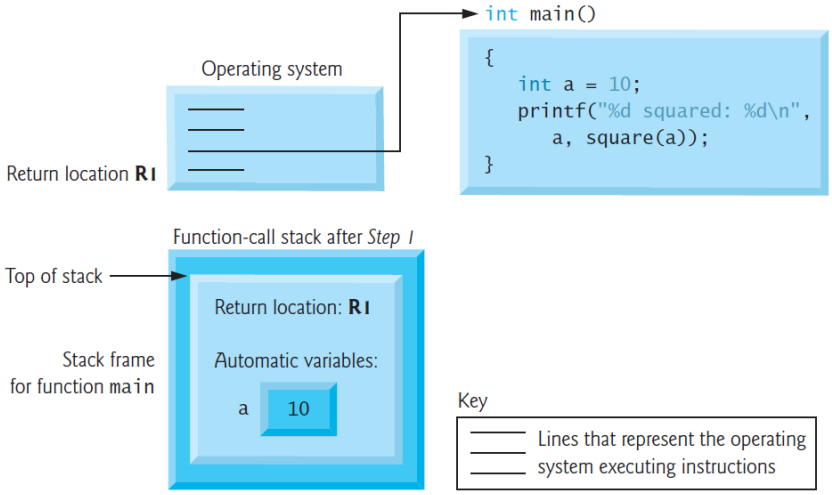
Yığın çerçevesinin bir başka önemli sorumluluğu daha vardır. Çoğu fonksiyonun, yerel değişkenleri vardır. Fonksiyon diğer fonksiyonları çağırırsa, etkin kalmaları gerekir. Ancak çağrılan bir fonksiyon çağırana geri döndüğünde, çağrılan fonksiyonun yerel değişkenlerinin "gitmesi" gerekir. Çağrılan fonksiyonun yığın çerçevesi, belleği yerel değişkenler için ayırmak için mükemmel bir yerdir. Bu yığın yapısı, yalnızca çağrılan fonksiyon etkin olduğu sürece var olur. Bu fonksiyon geri döndüğünde ve artık yerel değişkenlerine ihtiyaç duymadığında, yığın çerçevesi (stack frame) yığından çıkarılır. Bu yerel değişkenler artık program tarafından bilinmez.

Elbette, bir bilgisayardaki bellek miktarı sınırlıdır, bu nedenle fonksiyon çağrısı yığınında yığın çerçevelerini depolamak için yalnızca sınırlı bellek kullanılabilir. Yığın çerçevelerinin fonksiyon çağrısı yığınında depolanabileceğinden daha fazla fonksiyon çağrısı gerçekleşirse, yığın taşması (stack overflow) olarak bilinen bir hata (fatal error) oluşur.

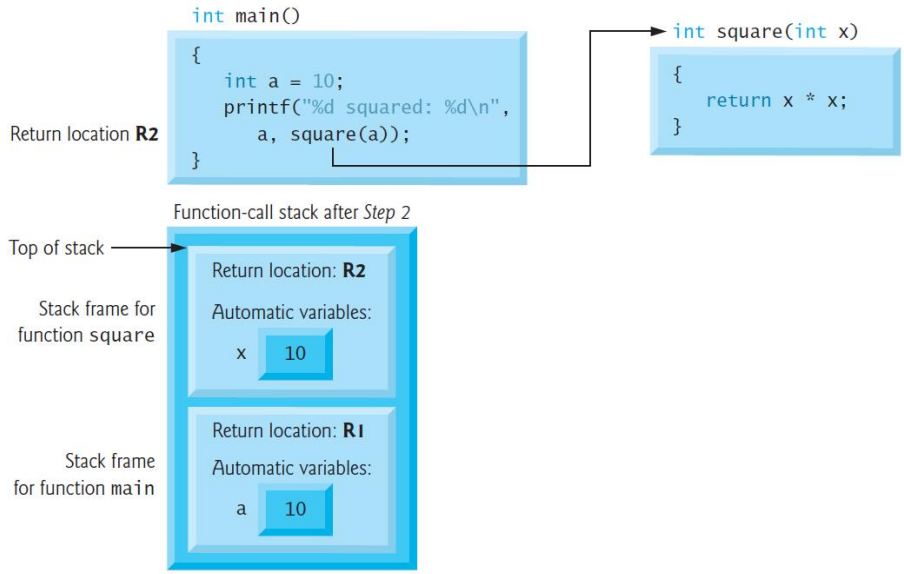
Şimdi çağrı yığınının main tarafından çağrılan bir kare fonksiyonun çalışmasını nasıl desteklediğine bakalım (Şekil 5.3'ün 8–12 satırları).

```
1  // fig05_03.c
2  // Demonstrating the function-call stack
3  // and stack frames using a function square.
4  #include <stdio.h>
5
6  int square(int x); // prototype for function square
7
8  int main() {
9     int a = 10; // value to square (local variable in main)
10
11    printf("%d squared: %d\n", a, square(a)); // display a squared
12 }
13
14 // returns the square of an integer
15 int square(int x) { // x is a local variable
16    return x * x; // calculate square and return result
17 }
```

Adım 1: İşletim Sistemi Uygulamayı Yürütmek için main'i Çağırır
İlk olarak, işletim sistemi main'i çağırır; bu, fonksiyonun yığın çerçevesini yığına iter (aşağıdaki diyagramda gösterildiği gibi). Yığın çerçevesi, main'e işletim sistemine nasıl geri dönüleceğini (yani, R1 dönüş adresine transfer edileceğini) söyler ve main'in 10 olarak başlatılan yerel değişkeni a için alanı içerir.

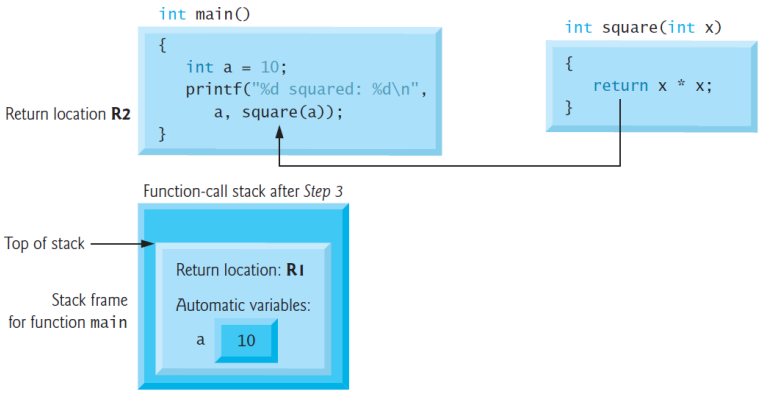


Adım 2: main, Hesaplamayı Gerçekleştirmek için square Fonksiyonunu Çağırır
main fonksiyonu—işletim sistemine dönmeden önce—Şekil 5.3'ün 11. satırındaki square fonksiyonunu çağırır. Bu, aşağıdaki şemada gösterildiği gibi, square için bir yığın çerçevesinin (15-17. satırlar) fonksiyon çağrısı yığınına itilmesine neden olur:
Bu yığın çerçevesi (stack frame), square'in main'e dönmesi için ihtiyaç duyduğu dönüş adresini (yani, R2) ve square'in yerel değişkeni (yani, x) için belleği içerir.



Adım 3: square Sonucunu main'e Döndürür

square, bağımsız değişkeninin karesini hesapladıktan sonra, main'e geri dönmesi gerekir ve artık **x** yerel değişkeni için belleğe ihtiyacı yoktur. Böylece yığın açılır - square'e main'deki dönüş konumunu verir (yani, R2) ve square'in yerel değişkenini yok eder. Aşağıdaki diyagram, square'in yığın çerçevesi açıldıktan sonraki fonksiyon çağrısı yığınını gösterir:



main fonksiyonu şimdi square çağrısının sonucunu görüntüler (Şekil 5.3'teki satır 11). Main'in kapanış sağ parantezine ulaşmak, yığın çerçevesini yığından çıkarır. Bu, main'e işletim sistemine döndürmek için ihtiyaç duyduğu adresi verir (yani, önceki diyagramda R1). Bu noktada, ana yerel değişkeni (yani a) için ayrılan bellek kullanılamaz. Önceki tartışma ve şemalarda aslında verilmeyen bir durum var. main'in square fonksiyonunu çağırdığını ve square'in main'e döndüğünü gösterdik, ama tabii ki printf de bir fonksiyon. Şekil 5.3'teki kodu incelerken, main'in printf'i, ardından printf'in square'i çağırdığını söyleyebiliriz. Ancak, printf'in argüman değerlerinin, printf'in çağrılabilmesi için tam olarak bilinmesi gerekir. Yani yürütme şu şekilde ilerler:

1. İşletim sistemi main'i çağırır, böylece main'in yığın çerçevesi yığının üzerine itilir.
2. main, square'ı çağırır, böylece square'ın yığın çerçevesi yığının üzerine itilir.
3. square, printf'in bağımsız değişken listesinde kullanılmak üzere bir değer hesaplar ve main'e döndürür, böylece square'in yığın çerçevesi yığından çıkarılır.
4. main, printf'i çağırır, böylece printf'in yığın çerçevesi yığının üzerine itilir.
5. printf, bağımsız değişkenlerini görüntüler, ardından ana ekrana döner, böylece printf'in yığın çerçevesi yığından çıkarılır.
6. main sonlandırılır, böylece main'in yığın çerçevesi yığından çıkarılır.

Bu örnekte, yığın veri yapısının, program yürütmeyi destekleyen bir anahtar mekanizmayı uygulamada ne kadar önemli olduğunu gördük. Veri yapılarının bilgisayar biliminde birçok önemli uygulaması vardır. Yığınları, Kuyrukları, Bağlantılı listeleri ve Ağaçları Bölüm 12'de daha detaylı inceleyeceğiz.

Başlıklar (Headers)- (Fonksiyon Kütüphaneleri)

Her standart kütüphanenin, o kütüphanedeki tüm fonksiyonlar için fonksiyon prototiplerini ve bu fonksiyonların ihtiyaç duyduğu çeşitli veri türlerinin ve sabitlerin tanımlarını içeren bir başlığı vardır. Aşağıdaki tablo, programlara dahil edilebilecek çeşitli standart kütüphane başlıklarını alfabetik olarak listeler. Ayrıca C standardı ek başlıklar da içerir. Bu tabloda birkaç kez kullanılan "makrolar" terimi, Bölüm 14'te ayrıntılı olarak tartışılmaktadır.

Başlık	**Açıklama**
<assert.h>	Program hata ayıklamasına yardımcı olan tanılama eklemek için bilgiler içerir.
<ctype.h>	Belirli özellikler için karakterleri test eden fonksiyonlar için fonksiyon prototiplerini ve küçük harfleri büyük harflere veya tersini yapmak için kullanılabilecek fonksiyonlar için fonksiyon prototiplerini içerir.
<float.h>	Sistemin kayan nokta boyutunun sınırlarını içerir.
<limits.h>	Sistemin bütünleyici boyut sınırlarını içerir.
<math.h>	Matematik kütüphane fonksiyonları için fonksiyon prototiplerini içerir.
<signal.h>	Program yürütme sırasında ortaya çıkabilecek çeşitli koşulları işlemek için fonksiyon prototiplerini ve makroları içerir.
<stdarg.h>	Sayısı ve türü bilinmeyen bir fonksiyona ilişkin bir argüman listesini kullanmak için makroları tanımlar.
<stdio.h>	Standart giriş/çıkış kütüphane fonksiyonları ve bunlar tarafından kullanılan bilgiler için fonksiyon prototiplerini içerir.
<stdlib.h>	Sayıların metne ve metnin sayılara dönüştürülmesi, bellek tahsisi, rasgele sayılar ve diğer yardımcı fonksiyonlar için fonksiyon prototiplerini içerir.
<string.h>	Dize (string) işleme fonksiyonları için fonksiyon prototiplerini içerir.
<time.h>	Saat ve tarihi değiştirmek için foksiyon prototiplerini ve türlerini içerir.
Diğer başlıklar:	
<errno.h>	Hata durumlarını raporlamak için yararlı olan makroları tanımlar.
<locale.h>	Bir programın çalışmakta olduğu geçerli yerel ayar için değiştirilmesini sağlayan fonksiyon prototiplerini ve diğer bilgileri içerir. Yerel kavramı, bilgisayar sisteminin dünya çapında tarihler, saatler, para birimi miktarları ve büyük sayılar gibi verileri ifade etmek için farklı kuralları işlemesini sağlar.
<setjmp.h>	Olağan fonksiyon çağrısı ve dönüş sırasının atlanmasına izin veren fonksiyonlar için fonksiyon prototiplerini içerir.
<stddef.h>	C tarafından kullanılan yaygın tür tanımlarını içerir.

Özel başlıklar oluşturabilirsiniz. #include önişlemci yönergesi kullanılarak programcı tanımlı bir başlık eklenebilir. Örneğin, kare fonksiyonumuzun prototipi square.h başlığında bulunuyorsa, programın en üstünde aşağıdaki yönergeyi kullanarak bu başlığı programımıza dahil edebiliriz:

```
#include "square.h"
```