C Programlama Dili

Dr. Öğr. Üye. Zafer Civelek

C’de program modülleri

Tecrübeler göstermiştir ki, büyük bir programı geliştirmek ve bakımı için en iyi yol her biri esas programdan daha iyi yönetilebilen daha küçük parçalardan veya modüllerden meydana getirmektir.

C deki modüllere fonksiyon denir. C programları genellikle, programcının yazdığı yeni fonksiyonlar ile C standart kütüphanesinde önceden paketlenmiş hazır fonksiyonlar kullanılarak yazılır. C standart kütüphanesi, yaygın matematiksel hesaplamaları, karakter katarı işlemleri, karakter değiştirmeleri, giriş/çıkış ve diğer birçok kullanışlı işlemleri icra etmek için zengin bir fonksiyon kolleksiyonu sağlar. Bu programcının işini kolaylaştırır, çünkü bu fonksiyonlar ihtiyaç olan özelliklerin birçoğunu saklar. Daha önce kullanılan printf ve scanf fonksiyonları standart kütüphane fonksiyonlarıdır.

Math kütüphane fonksiyonları

Bir fonksiyon, bir program içerisinde birçok noktada kullanılabilen özel görevleri tanımlamak için yazılabilir. Fonksiyonu tanımlayan ifadeler yalnızca bir kez yazılır ve ifadeler diğer fonksiyonlardan gizlenir. Fonksiyonlar, belirlenmiş bir görevi yapmak için fonksiyonun ismini belirten ve fonksiyonun ihtiyaç duyduğu bilgileri sağlayan bir fonksiyon çağrısı tarafından başlatılır.

Standart C kütüphane fonksiyonuna örnek olarak Math kütüphane fonksiyonları

Math kütüphane fonksiyonları, belirli yaygın matematiksel hesaplamaların yapılmasını sağlar. Fonksiyonlar normalde bir programda önce fonksiyon ismi sonra parantez içerisinde fonksiyon argümanı yazılarak kullanılır.

Math kütüphane fonksiyonları

mesela 900 sayısının karekökünü hesaplamak için

sqrt(900);

fonksiyonu kullanılır. Bu ifade yürütüldüğü zaman, parantez içinde bulunan sayının (900) karekökünü hesaplamak için sqrt math kütüphane fonksiyonu çağrılır. 900 sayısı sqrt fonksiyonunun argumanıdır. sqrt fonksiyonu double tür bir argüman alır ve double tür bir sonuç gönderir. math kütüphanesinde ondalık değer gönderen tüm fonksiyonlar double tür veri gönderir. Double değerlerin float değerler gibi %f çevrim belirteci kullanılarak çıktısı alınabilir. math kütüphanesindeki fonksiyonları kullanırken, *#include<math.h>* önişlemeci talimatını kullanarak, math kütüphanesini programa dahil etmekde fayda vardır.

Math kütüphane fonksiyonları

Fonksiyon argümanları, sabit, değişken veya ifade olabilir. mesela;

```c
#include<stdio.h>
#include<math.h>

int main(void){
    int c1=13;
    int d=3;
    int f=4;
    printf("%.2f",sqrt(c1+d*f));
}
```

```
5.00
--------
```

C, math kütüphane fonksiyonlarının küçük bir örneği şu şekildedir.

Fonksiyon	Açıklama	Örnek
`sqrt( x )`	*x*'in karekökü	`sqrt( 900.0 )`, 30.00'dur `sqrt( 9.0 )`, 3.0'dır
`cbrt(x)`	*x*'in küp kökü (sadece C99 ve C11)	`cbrt( 27.0 )`, 3.0'dır `cbrt( -8.0 )` -2.0'dır
`exp( x )`	e^x üssel fonksiyonu	`exp( 1.0 )`, 2.718282 dir `exp( 2.0 )`, 7.389056'dır
`log( x )`	*x*'in doğal logaritması (e tabanı)	`log( 2.718282 )`, 1.0'dır `log( 7.389056 )`, 2.0'dır
`log10( x )`	*x*'in logaritması	`log10( 1.0 )`, 0.0'dır `log10( 10.0 )`, 1.0'dır `log10( 100.0 )`, 2.0'dır
`fabs( x )`	ondalık sayı olarak *x*'in mutlak değeri	`fabs( 13.5 )`, 13.5'dir `fabs( 0.0 )`, 0.0'dır `fabs( -13.5 )`, 13.5'tir
`ceil( x )`	*x*'ten küçük olmayan en küçük tam sayıya yuvarlar	`ceil( 9.2 )`, 10.0'dır `ceil( -9.8 )`, -9.0'dır
`floor( x )`	*x*'ten büyük olmayan en büyük tam sayıya yuvarlar	`floor( 9.2 )`, 9.0'dır `floor( -9.8 )`, -10.0'dır
`pow( x, y )`	*x*'in *y*. kuvveti (x^y)	`pow( 2, 7 )`, 128.0'dır `pow( 9, .5 )`, 3.0'dır
`fmod( x, y )`	ondalık sayı olarak *x*/*y*'den kalan	`fmod( 13.657, 2.333 )`, 1.992'dir
`sin( x )`	*x*'in sinüsü (radyan)	`sin( 0.0 )`, 0.0'dır
`cos( x )`	*x*'in kosinüsü (radyan)	`cos( 0.0 )`, 1.0'dır
`tan( x )`	*x*'in tanjantı (radyan)	`tan( 0.0 )`, 0.0'dır

Fonksiyonlar

Fonksiyonlar bir programı modüllere ayırmamıza izin verir. Fonksiyon tanımlamalarında kullanılan tüm değişkenler yerel değişkenlerdir. Yalnızca içerisinde tanımlandıkları fonksiyon içinde erişilebilirler. Çoğu fonksiyonun, fonksiyonlar arasında bilgi haberleşmesi için anlam taşıyan bir parametre listesi vardır. Bir fonksiyonun parametreleri aynı zamanda o fonksiyonun yerel değişkenleridir. Bir programı fonksiyonlaştırmanın bazı sebebleri vardır. Bir tanesi böl ve yönetdir. Program geliştirmeyi daha yönetilebilir yapar. Başka bir sebeb de yazılımın yeniden kullanılabilirliğini sağlamaktır. Yeni programlar oluştururken yapı taşları olarak var olan fonksiyonları kullanmaktır. İyi fonksiyon isimlendirme ve tanımlama ile programlar, özelleştirilmiş kod kullanarak inşa edilmek yerine, özel görevleri gerçekleştiren standartlaştırılmış fonksiyonlardan oluşturulabilirler.

Fonksiyonlar

Üçüncü sebeb programda kod tekrarından kaçınmaktır. Bir fonksiyon olarak paketlenmiş kodlar, programın başka noktalarında fonksiyon çağrılarak yürütülmesini sağlar. Her bir fonksiyon, iyi tanımlanmış tek bir görevi yapmak için sınırlandırılmalıdır. Yaptığımız her program standart kütüphane fonksiyonlarını çağıran main denilen bir fonksiyondan meydana gelir. Özel fonksiyonları ise kurallarına uyarak programcı yazar. mesela 1'den 10'a kadar sayıların karesini hesaplayan ve yazdıran bir program yazalım. Karelerini hesaplama kısmını da fonksiyon ile yapalım.

```c
//programcı tanımlı bir fonksiyon oluşturma ve kullanma
#include<stdio.h>

int square(int y);//fonksiyon bildirimi

//main yürütmeye başlar
int main(void){
    int x;//sayaç
    //10 kez dön ve her seferinde x'in karesini hesapla ve yazdır
    for (x=1;x<=10;++x){
        printf("%d ",square(x));//fonksiyon çağrısı
    }//for sonu
    puts("");
}//main sonu

//square fonksiyon tanımlaması
int square(int y)//y argümanının fonksiyona bir kopyası
{
    return y*y;//y'nin kaaresini tam sayı olarak gönderir
}//square fonksiyonu sonu
```

```
1 4 9 16 25 36 49 64 81 100
```

Fonksiyonlar

square fonksiyonu main'de

printf("%d ",square(x)); komutu içerisinde başlatılır. square fonksiyonu x'in değerinin bir kopyasını y parametresi içine alır. Sonra square, y*y'yi hesaplar. Sonuç square'nin başlatıldığı main'deki printf fonksiyonuna geri gönderilir ve printf sonucu yazar. Bu işlem for ifadesi kullanılarak 10 kere tekrar edilir. square fonksiyonunun tanımlaması, square'nin bir y tam sayı parametresi beklediğini gösterir. Fonksiyonun isminin önündeki int anahtar kelimesi, square'nin bir tam sayı sonuç gönderdiğine işaret eder. square'deki return ifadesi, y*y ifadesinin değerini, çağıran fonksiyona geri verir.

Fonksiyonlar

int square(int y); komutu bir fonksiyon bildirimidir. Parantez içindeki int, square'in çağrıcıdan bir tam sayı değer almayı beklediğini derleyiciye bildirir. Fonksiyon ismi square'nin solundaki int, square'nin çağrıcıya bir tam sayı sonuç gönderdiğini derleyiciye bildirir. Derleyici, square'nin herhangi bir çağrısının doğru dönüş türü, doğru sayıda argüman ve doğru argüman türleri ve argümanların doğru sırada olduklarını kontrol etmek için fonksiyon bildirimine başvurur.

Bir fonksiyon tanımlamasının biçimi:

Fonksiyonlar

dönüş_değeri_türü fonksiyon_ismi (parametre listesi)
{
tanımlamalar
ifadeler
}

fonksiyon ismi herhangi bir geçerli kimliktir. dönüş_değeri_türü, çağırıcıya gönderilen sonucun veri türüdür. void, dönüş_değeri_türü, fonksiyonun bir değer göndermediğini gösterir. parametre listesi, fonksiyon çağrıldığı zaman, fonksiyon tarafından alınan parametreleri belirten virgülle ayrılmış bir listedir. Eğer fonksiyon herhangi bir değer almıyorsa, parametre listesi void olur. Her bir parametre için tür açıkca listelenmelidir.

Fonksiyonlar

Aynı türden fonksiyon parametrelerini, double x, double y yerine double x,y olarak belirtmek derleme hatası ile sonuçlanır. Bir fonksiyon tanımlamasında parametre listesini kapatan sağ parantezden sonra noktalı virgül koymak bir söz dizim hatasıdır. Bir parametreyi fonksiyon içinde yerel değişken olarak tekrar tanımlamak bir derleme hatasıdır. Süslü küme parantezi içindeki tanımlamalar ve ifadeler aynı zamanda blok olarak adlandırılan fonksiyon gövdesini oluşturur. Değişkenler herhangi bir blokta bildirilebilir ve bloklar kümelenebilir.

Fonksiyonlar

Başka bir fonksiyon içinde bir fonksiyon tanımlama bir söz dizim hatasıdır.

Anlamlı fonksiyon ve parametre isimleri seçmek programları daha okunabilir yapar ve aşırı açıklama kullanımını önlemeye yardımcı olur.

Küçük fonksiyonlar yazılımın tekrar kullanılabilirliğini kolaylaştırır.

Programlar küçük fonksiyonların birleşimi olarak yazılmalıdır. Bu durum, programları yazmasını, hata ayıklamasını, bakımını ve geliştirmesini kolaylaştırır.

Fonksiyonlar

Fonksiyon bildirimi ile fonksiyon çağrıları, fonksiyon başlığı, argüman sayısı, türü, sırası ve dönüş değeri türünde eşleşmelidir.

Çağrılan bir fonksiyondan, fonksiyonun başlatıldığı noktaya üç şekilde döner;

fonksiyon bir sonuç göndermiyorsa

a- fonksiyon sonu sağ küme parantezine ulaşıldığı zaman

b- return; ifadesini yürüterek

fonksiyon bir sonuç gönderiyorsa;

c-return ifade;

ile ifade değerini çağırıcıya geri gönderir.

Fonksiyonlar

main'in geri dönüş değeri programın doğru bir şekilde yürütülüp yürütülmediğini göstermek için kullanılır. geri dönüş değerinin sıfır olması programın başarılı bir şekilde yürütüldüğünü gösterir.

Örnek: üç tam sayıdan en büyüğünü belirlemek ve geri göndermek için programcı tanımlı maksimum fonksiyonunu kullanır. Tam sayılar scanf ile girilir. Sonra bunlar en büyük tam sayıyı belirleyen maksimum'a aktarılır. Bu değer maksimumdaki return ifadesi tarafından main'e gönderilir. Gönderilen değer daha sonra printf ifadesiyle yazılır.

```c
//üç tam sayıdan en büyüğünü  bulma
#include<stdio.h>

int maksimum(int x,int y,int z);//fonksiyon bildirimi

//main fonksiyonu yürütülmeye başlar
int main(void){

    int sa1;
    int sa2;
    int sa3;//kullanıcı tarafından girilen üç tam sayı

    printf("%s","uc tam sayi girin: ");
    scanf("%d%d%d",&sa1,&sa2,&sa3);

    //sa1,sa2,sa3 maksimum fonksiyonunun argümanlarıdır
    printf("maksimum: %d\n",maksimum(sa1,sa2,sa3));
}//main sonu

//maksimum fonksiyonu tanımlamasıdır
//x,y,z parametrelerdir
int maksimum(int x,int y,int z)
{
    int max=x;//x'i en büyük kabul et

    if(y>max){//y max dan büyükse
    max=y;//y'yi max'a aktar
    }//if sonu

    if(z>max){//z max dan büyükse
    max=z;//z'yi max'a aktar
    }//if sonu

    return max;//max en büyük değer
}//maksimum fonksiyonu sonu
```

```
uc tam sayi girin: 22 85 17
maksimum: 85
```

```
uc tam sayi girin: 47 32 14
maksimum: 47
```

```
uc tam sayi girin: 35 8 79
maksimum: 79
```

Fonksiyonlar

Fonksiyon Bildirimleri

C'nin önemli bir özelliği fonksiyon bildirimidir. Bu özellik C++'dan ödünç alınmıştır. Derleyici fonksiyon çağrılarını doğrulamak için fonksiyon bildirimlerini kullanır. C'nin önceki sürümleri bu tür kontrolleri yapmazdı, bu nedenle derleyici hata saptamaksızın, fonksiyonları uygun olmayan bir şekilde çağırmak mümkündü. Böyle çağrılar sonlandırıcı hatalar veya ince, bulunması zor problemlere neden olan sonlandırıcı olmayan hatalar ile sonuçlanır. Fonksiyon bildirimleri bu eksikliği düzeltmiştir. C'nin tür kontrol etme özelliğinden faydalanmak için tüm fonksiyonların fonksiyon bildirimleri programa eklenmelidir.

Fonksiyonlar

Programdaki

int maksimum(int x, int y, int z);

komut bir fonksiyon bildirimidir. maksimum'un int türünden üç argüman aldığını ve int türünden bir sonuç gönderdiğini ifade eder. Fonksiyon bildirimi, maksimum fonksiyonunun ilk satırıyla aynıdır.

Fonksiyon bildirimi ile eşleşmeyen bir fonksiyon çağrısı derleme hatasıdır. Eğer fonksiyon bildirimi ve fonksiyon tanımlaması birbirine uymuyorsa bir hata üretilir. Mesela programdaki fonksiyon bildirimi

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

şeklinde yazılsaydı fonksiyon bildirimindeki void dönüş türü fonksiyon başlığındaki int dönüş türünden farklılık gösterdiği için derleyici bir hata üretecekti.

Fonksiyonlar

Arguman uyumu ve genel aritmetik çevrim kuralları

Fonksiyon bildirimlerinin bir başka önemli özelliği argüman uyumudur yani argümanları uygun türe zorlamaktır. Mesela sqrt math kütüphane fonksiyonu <math.h> deki fonksiyon bildirimi bir double parametre belirtse bile bir tam sayı argümanla çağrılabilir ve fonksiyon doğru bir şekilde çalışmaya devam eder.

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

ifadesi sqrt(4) ü doğru bir şekilde yapar ve 2.000 değerini yazar. Fonksiyon bildirimi derleyicinin tam sayı değeri 4'ün bir kopyesini, kopye, sqrt'ye geçirilmeden önce double değer 4.0'a dönüştürmesine neden olur.

Fonksiyonlar

Genelde, fonksiyon bildirimindeki parametre türlerine tam olarak karşılık gelmeyen argüman değerleri fonksiyon çağrılmadan önce uygun türe dönüştürülür. C'nin genel aritmetik çevrim kuralları takip edilmezse, bu çevrimler doğru olmayan sonuçlara yönlendirebilirler. Bu kurallar veri kaybı olmaksızın değerlerin diğer türlere nasıl dönüştürülebileceğini belirler. sqrt misalinde, bir int değeri değişmeksizin bir double'a dönüştürülür. Bununla birlikte, bir int e dönüştürülen double, double değerin esas değerini değiştirerek ondalık kısmını kırpar. Büyük tam sayı türleri küçük tam sayı türlere dönüştürmek (mesela long'dan short'a) de aynı zamanda değişmiş değerlerle sonuçlanır.

Fonksiyonlar

Genel aritmetik çevrim kuralları iki tür veri türünden (karışık tür) değerler içeren ifadelere otomatik olarak uygulanır ve derleyici tarafından ele alınır. Karışık tür bir ifadede, derleyici dönüştürülecek değerin geçici bir kopyasını yapar, sonra kopyayı ifadedeki en yüksek türe dönüştürür, esas değer dönüşmeden kalır. En az bir ondalık değer içeren karışık tür bir ifade için genel aritmetik çevrim kuralları şöyledir:

Değerlerden biri long double ise, diğeri long double'a dönüştürülür. Değerlerden biri double ise, diğeri double'a dönüştürülür. Değerlerden biri float ise, diğeri float'a dönüştürülür.

Fonksiyonlar

Karışık tür ifadeler yalnızca tam sayı türler içeriyorsa, genel aritmetik çevrimler bir dizi tam sayı kurallar belirtir. Çoğu durumda aşağıdaki tablodaki daha düşük tam sayı türleri daha yüksek tam sayı türlere dönüştürülür. Ondalık ve tam sayı veri türlerinin her türü printf ve scanf çevrim belirteçleri ile gösterilir.

Data type	printf conversion specification	scanf conversion specification
Floating-point types		
`long double`	`%Lf`	`%Lf`
`double`	`%f`	`%lf`
`float`	`%f`	`%f`
Integer types		
`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`

Fonksiyonlar

Değerleri yukarıdaki tabloda olan daha düşük türlere dönüştürmek doru olmayan değerlerle sonuçlanabilir, bu nedenle derleyici böyle durumlar için uyarı verir. Bir değer yalnızca kasıtlı olarak daha düşük türden değere atanarak veya tür dönüşüm işlemi kullanılarak daha düşük bir türe dönüştürülebilir. Bir fonksiyon çağrısında argümanlar o türden değişkenlere direk olarak atanıyormuş gibi fonksiyon bildiriminde belirtilen parametre türlerine dönüştürülür. int parametresi kullanan square fonksiyonu bir ondalık argüman ile çağrılırsa argüman int'e dönüştürülür ve square bu durumda doğru olmayan bir değer gönderir. Mesela square(4.5), 20.25 değil 16 değerini döndürür.

Fonksiyonlar

Bir fonksiyon için fonksiyon bildirimi yoksa derleyici fonksiyonun ilk defa ortaya çıktığı halini ya fonksiyon tanımlamasını yada bir fonksiyon çağrısını kullanarak kendi fonksiyon bildirimini oluşturur. Bu durum derleyiciye bağlı olarak uyarı yada hataya yönlendirir.

Fonksiyonlar

Örnek: klavyeden girilen bir tam sayının tek mi çift mi olduğunu ekrana yazdıran bir programı fonksiyon kullanarak yapın. Tek ve çift kontrolünü fonksiyon yerine getirsin.

```c
/* Sayının tek veya çift olmasını
   kontrol eder. */
#include<stdio.h>

void tekCiftMi(int);//fonksiyon bildirimi

int main( void )
{
    int girilenSayi;
    printf( "Lutfen bir sayi giriniz> " );
    scanf( "%d",&girilenSayi );
    tekCiftMi( girilenSayi );

    return 0;
}
void tekCiftMi( int sayi )
{
    if( sayi%2 == 0 )
        printf( "%d cift bir sayidir.\n", sayi );
    else
        printf( "%d tek bir sayidir.\n", sayi );
}
```

```
Lutfen bir sayi giriniz> 67
67 tek bir sayidir.
```

```
Lutfen bir sayi giriniz> 568
568 cift bir sayidir.
```

Fonksiyonlar

Örnek: klavyeden girilen bir tam sayının kare ve küpünü ekrana yazdıran bir program yapınız. kare ve küpünü fonksiyonlar ile yazdırınız.

```c
#include<stdio.h>
// Verilen sayinin karesini ve küpünü hesaplar

void kareHesapla( int sayi );
void kupHesapla( int sayi );

int main( void )
{
    // main( ) fonksiyonunda
    // a degiskeni tanimliyoruz.
    int a;
    printf( "Bir sayi giriniz> ");
    scanf( "%d",&a );
    printf( "Girdiginiz sayi\t: %d\n", a );
    kareHesapla( a );
    // Eger a degiskeni lokal olmasaydi,
    // kare_hesapla fonksiyonundan sonra,
    // a'nin degeri bozulur ve kup yanlis
    // hesaplanirdi.
    kupHesapla( a );
    return 0;
}

```

Fonksiyonlar

```
24  void kareHesapla( int sayi )
25  {
26      // kare_hesapla fonksiyonunda
27      // a degiskeni tanimliyoruz.
28      int a;
29      a = sayi * sayi;
30      printf( "Sayinin karesi\t: %d\n", a );
31  }
32
33  // Verilen sayinin kupunu hesaplar
34  void kupHesapla( int sayi )
35  {
36      // kup_hesapla fonksiyonunda
37      // a degiskeni tanimliyoruz.
38      int a;
39      a = sayi * sayi * sayi;
40      printf( "Sayinin kupu\t: %d\n", a );
41  }
42
```

```
Bir sayi giriniz> 5
Girdiginiz sayi : 5
Sayinin karesi  : 25
Sayinin kupu    : 125
```

Fonksiyonlar

Örnek: Bir kürenin yarıçapını klavyeden giriş olarak alıp, kürenin yüzey alanını ve hacmini hesaplatın. (yüzey alanı=4*pi*r^2, hacmi=4*pi*r^3/3)

```c
//kürenin alanı ve hacmi
#include<stdio.h>
#include<math.h>

double yariCap;
double pi=3.1415;

double kureAlan(double); //fonksiyon bildirimleri
double kureHacim(double);

int main(void){

    printf("kurenin yari capini giriniz:");
    scanf("%lf",&yariCap);
    printf("yari capi %.2f olan kurenin yuzey alani\t: %.2f\n",yariCap,kureAlan(yariCap));
    printf("yari capi %.2f olan kurenin hacmi\t: %.2f\n",yariCap,kureHacim(yariCap));
}

//kürenin alanını hesaplayan fonksiyon
double kureAlan(double yariCap){
    return (4*pi*pow(yariCap,2));
}

//kürenin hacmini hesaplayan fonksiyon
double kureHacim(double yariCap){
return (4*pi*pow(yariCap,3)/3);
}
```

```
kurenin yaricapini giriniz:6
yari capi 6.00 olan kurenin yuzey alani : 452.38
yari capi 6.00 olan kurenin hacmi       : 904.75
```

Fonksiyonlar

Fonksiyon Çağrı Yığını ve Yığın Yapısı

C'nin fonksiyonları nasıl çağırdığını anlamak için yığın olarak bilinen bir veri yapısını anlamak gerekir. Bir yığın üst üste konulmuş tabaklara benzetilebilir. Yeni bir tabak koyarken en üste koyarsınız. Bir tabak alırken en üstten alırsınız. Yığınlar son giren ilk çıkar (LIFO) veri yapısı olarak bilinir. Yığından bir eleman çekerken en son yığına itilen elemanı çekersiniz.

Fonksiyonlar, fonksiyon çağrı yığını mekanizması ile çalışır. Bu veri yapısı arka planda çalışan fonksiyon çağrı/dönüş mekanizmasını destekler. Aynı zamanda her çağrılan fonksiyonun otomatik değişkenlerini oluşturmayı, bakımını ve yok edilmesini destekler.

Fonksiyonlar

Her fonksiyon çağrıldığı zaman, başka fonksiyonları çağırabilen başka fonksiyonlar çağırabilir. Her fonksiyon kontrolü onu çağıran fonksiyona en sonunda vermek zorundadır. Bu nedenle onu çağıran fonksiyona kontrolü vermek için her fonksiyonun ihtiyacı olan dönüş adreslerinin takibi yapılmak zorundadır. Bir fonksiyon başka bir fonksiyonu çağırdığı zaman yığına bir giriş itilir. Bu giriş, yığın yapısında çağıran fonksiyona dönmek için çağrılan fonksiyonun ihtiyacı olan dönüş adresini içerir. Bazı ek bilgileri de içerebilir. Geri dönmeden önce başka fonksiyon çağırmak yerine çağrılan fonksiyon dönerse, fonksiyon çağrısının yığın yapısı çıkarılır ve kontrol çıkarılmış yığın yapısındaki dönüş adresine aktarılır.

Fonksiyonlar

Her çağrılan fonksiyon, çağrıldığı yere geri dönmek için ihtiyaç duyduğu bilgiyi çağrı yığınının en üstünde bulur. Fonksiyon başka bir fonksiyona çağrı yaparsa, yeni fonksiyon çağrısının yığın yapısı, çağrı yığını üzerine girilir. Bu durumda çağrıldığı yere dönmek için yeni çağrılan fonksiyon tarafından gerekli olan dönüş adresi yığının en üstünde bulunur.

Çoğu fonksiyonun, parametreler ve yerel değişkenler gibi otomatik değişkenleri vardır. Fonksiyon yürütülürken otomatik değişkenlerin var olması gerekir. Fonksiyon diğer fonksiyonlara çağrı yaparsa aktif olmaları gerekir. Ancak çağrılan fonksiyon çağırana döndüğü zaman, çağrılan fonksiyonun otomatik değişkenleri yok edilmelidir.

Fonksiyonlar

Çağrılan fonksiyonun yığın yapısı otomatik değişkenleri hafızada saklar. Bu yığın yapısı fonksiyon aktif olduğu sürece var olur. Fonksiyon geri döndüğü ve yerel otomatik değişkenlere ihtiyaç olmadığı zaman yığın yapısından çıkarılır. Bu yerel otomatik değişkenler artık program tarafından tanınmaz.

Bir bilgisayardaki hafıza miktarı sınırlıdır. Bu nedenle fonksiyon çağrı yığınında yığın yapılarını saklamak için belirli bir hafıza miktarı kullanılır. Fonksiyon çağrı yığınında saklanabilen yığın yapılarından daha fazla fonksiyon çağrısı olursa, yığın taşması olarak bilinen sonlandırıcı bir hata olur.

Fonksiyonlar

Eylemdeki Fonksiyon Çağrı Yığını

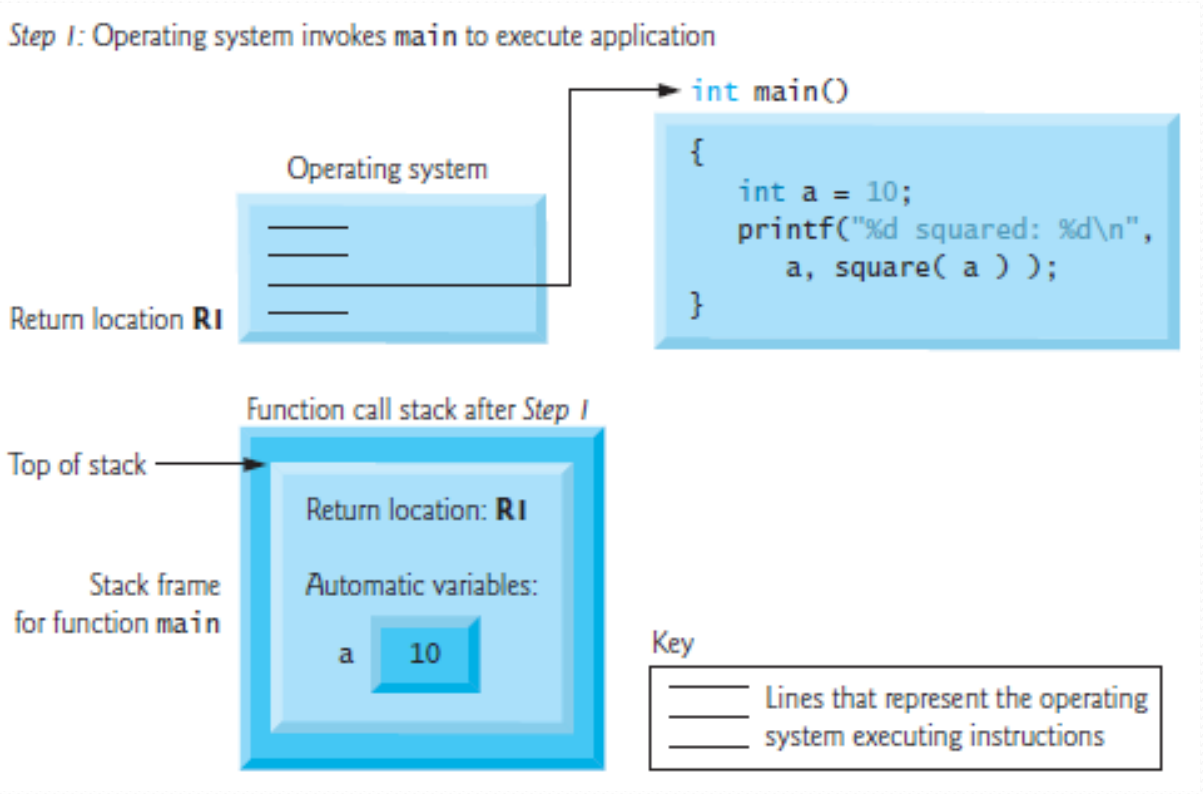
Aşağıdaki programdaki çağrı yığınının main tarafından çağrılan square fonksiyonunun işleyişini nasıl desteklediğini ele alalım.

```
1   /* Fig. 5.3: fig05_03.c
2      Creating and using a programmer-defined function */
3   #include <stdio.h>
4
5   int square( int ); /* function prototype */
6
7   /* function main begins program execution */
8   int main( void )
9   {
10     int a=10;
11
12        printf( "%d un karesi %d\n ", a,square( a ) ); /* function call */
13
14  } /* end main */
15
16  /* square function definition returns square of parameter */
17  int square( int x ) /* y is a copy of argument to function */
18  {
19     return x * x; /* returns square of y as an int */
20  } /* end function square */
```

```
10 un karesi 100
```

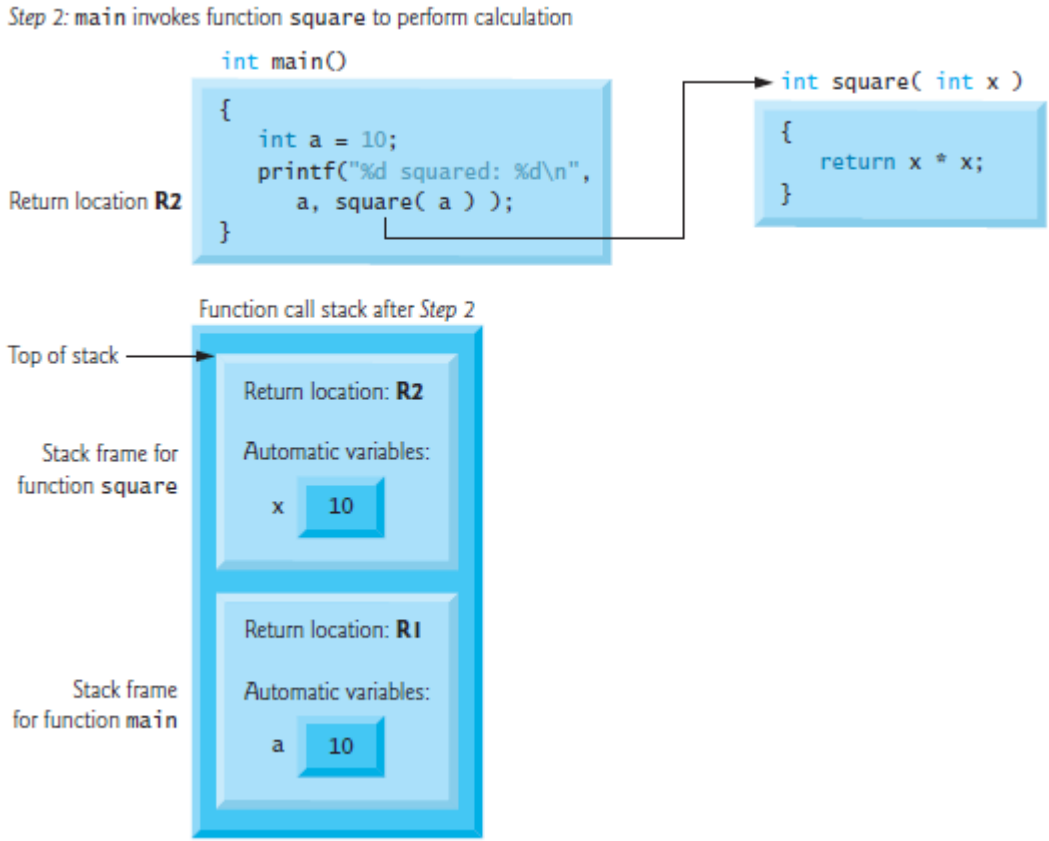
Fonksiyonlar

Önce işletim sistemi main'i çağırır, bu yığın yapısı yığına girer. Aşağıdaki şekilde gösterilmiştir. Yığın yapısı main'e işletim sistemine nasıl döneceğini söyler (R1 dönüş adresi) ve main'in otomatik değişkenleri için yer içerir(a=10 için).



Fonksiyonlar

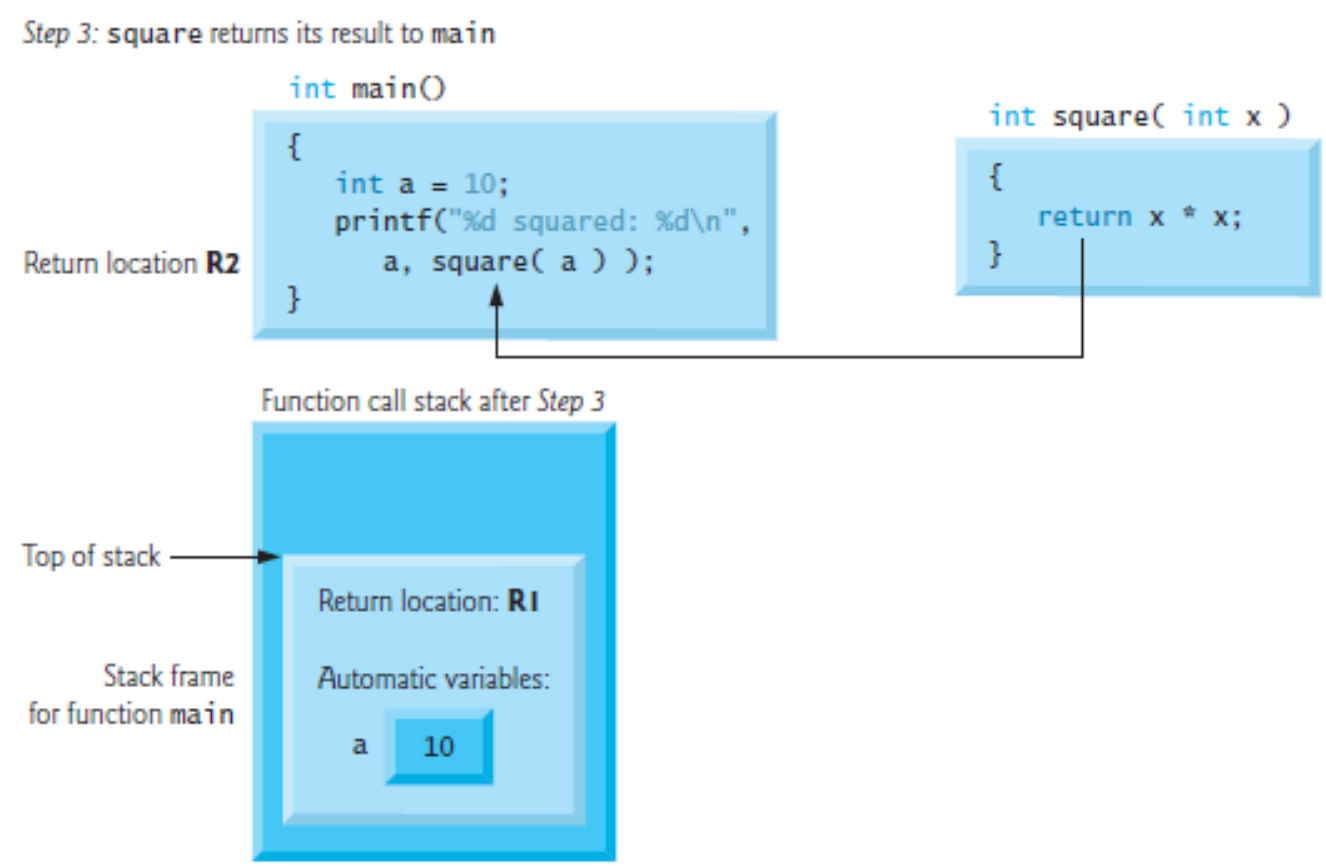
main fonksiyonu işletim sistemine dönmeden önce square fonksiyonunu çağırır. Bu durum square için bir yığın yapısının fonksiyon çağrı yığınına girmesine neden olur. Bu yığın yapısı square'in main'e dönmek için ihtiyaç duyduğu dönüş adresini (R2) ve square'in otomatik değişkeni (yani x) için hafıza içerir. Bu durum aşağıdaki şekilde gösterilmiştir.



Fonksiyonlar

square argümanın karesini hesapladıktan sonra, main'e geri dönmesi gerekir ve x otomatik değişkeni için artık hafızaya ihtiyacı yoktur. Bu nedenle square fonksiyonunun main'e dönüş yeri (yani R2) ve sqaure'nin otomatik değişkeni (yani x) yığından çekilir. Program R2 adresine yani main'de kaldığı yere dönerken, square'nin otomatik değişkeni x ortadan kaldırılır. square'nin yığın yapısı çıkarıldıktan sonra fonksiyon çağrı yığını aşağıda şekilde gösterildiği gibi olur.

Fonksiyonlar



Fonksiyonlar

Değer ve Adres ile argüman aktarma

Bir çok programlama dilinde argümanları aktarmak için iki yol vardır; değer ile aktarım ve adres ile aktarım. Argümanlar değer ile aktarıldığı zaman, argüman değerinin bir kopyası yapılır ve çağrılan fonksiyona aktarılır. Kopyada yapılan değişiklikler çağırandaki esas değişken değerini etkilemez. Argüman, adres ile aktarılırsa, çağırıcı çağrılan fonksiyonun esas değişken değerini değiştirmesine izin verir. Ne zaman çağrılan fonksiyonun çağıranın esas değişken değerini değiştirmesine gerek olmazsa değer ile aktarım kullanılır. Adres ile aktarım yalnızca esas değişkeni değiştirmesi gereken güvenilir fonksiyonlarla kullanılmalıdır.

Fonksiyonlar

Bellek Sınıfları

Değişken özellikleri; isim, tür, boyut ve değer içerir. Programdaki her kimlik, bellek sınıfı, bellek süresi, kapsam ve bağlantı dahil olmak üzere başka özelliklere sahiptir.

C, auto, register, extern ve static bellek sınıf belirteçleri sağlar. Bir kimliğin bellek sınıfı; saklama süresini, kapsamını ve bağlantısını belirler. Kimliğin süresi, kimliğin hafızada var olduğu zaman boyunca geçen süredir. Bazıları kısa bir süre var olur, bazıları tekrar tekrar oluşturulur ve yok edilir ve diğerleri programın başından sonuna kadar tüm yürütmesinde var olurlar.

Fonksiyonlar

Kimliğin kapsamı, programda kimliğe referans verilen yerdir. Bazılarına program boyunca referans verilebilir, diğerlerine yalnızca program parçalarından verilebilir. Bir kimlik bağlantısı, çoklu kaynak dosyalı programlar için kimliğin sadece o kaynak dosyasında mı yoksa uygun açıklamalarla herhangi bir kaynak dosyasında mı bilindiğini belirler.

Bellek sınıf belirteçleri; otomatik saklama süresi ve durağan saklama süresi olarak ikiye ayrılabilir. Otomatik saklama süresi değişkenlerini tanımlamak için auto anahtar kelimesi kullanılır. Otomatik saklama süreli değişkenler, tanımlandıkları yapıya girildiği zaman oluşturulur, yapı aktif olduğu sürece var olurlar ve yapıdan çıkıldığında yok edilirler.

Fonksiyonlar

Sadece değişkenlerin otomatik saklama süresi vardır. Bir fonksiyonun yerel değişkenleri (parametre listesinde veya fonksiyon gövdesinde tanımlanmış olanlar) normalde otomatik saklama süresine sahiptir. Auto anahtar kelimesi otomatik saklama süresi değişkenlerini tanımlar. Yerel değişkenler, varsayılan olarak otomatik saklama süresine sahiptirler, bu nedenle auto anahtar kelimesi nadiren kullanılır. Otomatik değişkenler yalnızca ihtiyaç duyulduklarında var oldukları için otomatik saklama bir hafıza koruma aracıdır. Bir fonksiyona girildiğinde oluşturulurlar ve fonksiyondan çıkıldığında yok edilirler.

Fonksiyonlar

Durağan Saklama Sınıfı

extern ve static anahtar kelimeleri durağan saklama süresinden değişkenlerin ve fonksiyonların kimlik tanımlamalarında kullanılır. Durağan saklama süresi kimlikleri, program yürütmeye başlamasından sonlanana kadar var olurlar. static değişkenler için, program yürütmeye başlamadan önce yalnızca bir kez hafızadan yer ayrılır ve başlangıç değeri atanır. Bununla birlikte, program yürütmesi başlangıcından itibaren değişkenler ve fonksiyonlar var olsa bile, bu kimliklerin program boyunca erişilebilir olduğu anlamına gelmez. Saklama süresi ve kapsam ayrı konulardır.

Fonksiyonlar

Durağan saklama süreli kimliklerin birkaç türü vardır; dış kimlikler (genel değişkenler ve fonksiyon isimleri gibi) ve static saklama sınıfı belirteçli tanımlanmış yerel değişkenler. Genel değişkenler ve fonksiyon isimleri varsayılan olarak extern saklama sınıfındandır. Genel değişkenler, değişken tanımlamasını herhangi bir fonksiyonun dışında yaparak oluşturulur ve değerlerini program yürütmesi boyunca korurlar. Genel değişkenler ve fonksiyonlara, dosyada bildirimlerinden ve tanımlanmalarından sonra gelen herhangi bir fonksiyon tarafından referans verilebilir. Bu durum fonksiyon bildirimleri kullanmak için bir sebebtir.

Fonksiyonlar

Mesela printf çağıran bir programda stdio.h dahil edildiğinde , printf isminin dosyanın geri kalanında bilinmesi için dosyamızın başına fonksiyon bildirimi yerleştirilir.

Bir değişkeni yerel (local) yerine genel (global) olarak tanımlamak, ihtiyacı olmayan bir fonksiyonun değişkeni değiştirmek için yanlışlıkla veya kasden eriştiği zaman istenmeyen yan etkilerin olmasına izin verir. Genelde özgün performans gerekliliği olan belirli durumlar dışında genel değişken kullanılmasından kaçınmak lazımdır. Yalnızca belirli bir fonksiyonda kullanılan değişkenler, dış değişken yerine o fonksiyonda yerel değişken olarak tanımlanmalıdır.

Fonksiyonlar

static anahtar kelimesi ile tanımlanmış yerel değişkenler de yalnızca tanımlandıkları fonksiyon içinde tanınırlar, ancak otomatik değişkenlerden farklı olarak, static yerel değişkenler fonksiyondan çıkıldığında değerlerini korurlar. Sonra aynı fonksiyon çağrıldığında, static yerel değişkenler fonksiyondan son çıkıldığında sahip olduğu değeri içerir. Durağan saklama süresinden tüm sayısal değişkenler açıkça başlangıç değeri verilmediyse varsayılan olarak sıfıra eşitlenir.

Fonksiyonlar

Kapsam Kuralları

Kimlik kapsamı kimliğin referans verildiği programın parçasıdır. Mesela bir yapıda yerel değişken tanımladığımız zaman, o yapıdaki veya o yapı içinde kümelenmiş yapılarda tanımlanmasından sonra referans verilebilir. Dört kimlik kapsamı vardır, bunlar; fonksiyon kapsamı, dosya kapsamı, yapı kapsamı ve fonksiyon bildirimi kapsamıdır. Etiketler yalnızca fonksiyon kapsamlı kimliklerdir. Etiketler görüldüğü fonksiyonda herhangi bir yerde kullanılabilir, ancak fonksiyon gövdesi dışında referans verilemez. Etiketler switch ve goto ifadelerinde kullanılabilir. Etiketler fonksiyonları bir diğerinden saklayan uygulama ayrıntısıdır. Bu saklama (en düşük seviyede erişim hakkı ile) iyi yazılım mühendisliğinin temel ilkesini gerçekleştirir.

Fonksiyonlar

Bir uygulamada ilke; koda belirlenmiş görevini yapmak için yalnızca gereken, fazlası değil, öncelik ve erişim miktarının verilmesi gerektiğini ifade eder.

Herhangi bir fonksiyonun dışında tanımlanmış bir kimlik dosya kapsamıdır. Böyle bir kimlik, kimliğin tanımlandığı noktadan dosya sonuna kadar tüm fonksiyonlarda tanınır (yani erişilebilir). Bir fonksiyonun dışında bulunan genel değişkenler, fonksiyon tanımlamaları ve fonksiyon bildirimlerinin tümü dosya kapsamına sahiptir.

Bir yapı içerisinde tanımlanmış kimlikler yapı kapsamına sahiptir. Yapı kapsamı yapının sonlandırıcı sağ küme parantezinde biter. Fonksiyonun başında tanımlanmış yerel değişkenler fonksiyon tarafından yerel değişken olarak nitelenen fonksiyon parametreleri gibi yapı kapsamına sahiptir.

Fonksiyonlar

Yapılar kümelendiğinde ve dış yapıdaki bir kimlik iç yapıdakilerden biriyle aynı isme sahip olduğu zaman, iç yapı sonlanana kadar dış yapıdaki kimlik saklanır. Bu durum, içyapı yürütülürken, dış yapıdaki aynı isimdeki kimliği değil iç yapıdaki kendi yerel kimliğin değerini görmesi anlamına gelir. static olarak ifade edilmiş yerel değişkenler de program başlamadan önce var olsalar bile yapı kapsamına sahiptir. Bu nedenle, saklama süresi kimliğin kapsamını etkilemez.

Fonksiyon bildirimi kapsamlı tek kimlik fonksiyon bildiriminin parametre listesinde kullanılanlardır. Fonksiyon bildirimlerinin parametre listesindeki isimlere ihtiyacı yoktur, sadece türler gereklidir. Fonksiyon bildirimindeki parametre listesinde bir isim kullanılırsa, derleyici ismi göz ardı eder. Fonksiyon bildiriminde kullanılan kimlikler programın başka bir yerinde belirsizliğe yol açmadan kullanılabilir.

Fonksiyonlar

Aşağıdaki program, genel değişkenler, otomatik yerel değişkenler ve static yerel değişkenler ile olan kapsama konularını gösterir. Programda x genel değişkeni tanımlanır ve başlangıç değeri 1'e eşitlenir (9. satır). Bu genel değişken x isimli bir değişkenin tanımlandığı herhangi bir yapıda veya fonksiyonda gizlenir. main'de x yerel değişkeni tanımlanır ve başlangıç değeri 5'e eşitlenir (14. satır). Bu değişken genel x'in main'de gizlendiğini göstermek için yazılır. Sonra main'de başlangıç değeri 7'ye eşitlenen başka bir yerel x değişkeni ile yeni bir yapı tanımlanır (19. satır). Bu değişken main'in dış yapısındaki x'i gizlediğini göstermek için yazdırılır. 7 değerine sahip x değişkeni yapıdan çıkıldığı zaman otomatik olarak yok edilir ve artık gizli olmadığını göstermek için main'in dış yapısındaki x yerel değişkeni tekrar yazdırılır.

Fonksiyonlar

```c
/* Fig. 5.12: fig05_12.c
   A scoping example */
#include <stdio.h>

void useLocal( void ); /* function prototype */
void useStaticLocal( void ); /* function prototype */
void useGlobal( void ); /* function prototype */

int x = 1; /* global variable */

/* function main begins program execution */
int main( void )
{
   int x = 5; /* local variable to main */

   printf("local x in outer scope of main is %d\n", x );

   { /* start new scope */
      int x = 7; /* local variable to new scope */

      printf( "local x in inner scope of main is %d\n", x );
   } /* end new scope */

   printf( "local x in outer scope of main is %d\n", x );

```

Fonksiyonlar

Program, hiç argüman almayan ve hiçbir şey göndermeyen üç fonksiyon tanımlar. useLocal fonksiyonu x otomatik değişkenini tanımlar ve başlangıç değeri olarak 25'e eşitler (39. satır). useLocal çağrıldığında , değişkeni yazdırılır, 1 arttırılır ve fonksiyondan çıkmadan önce tekrar yazdırılır. Bu fonksiyon her çağrıldığında x otomatik değişkeni 25'e tekrar eşitlenir. useStaticLocal fonksiyonu bir static x değişkeni tanımlar ve bunu 52. satırda 50'ye eşitler (static değişkenler için program yürütmeye başlamadan önce hafızada yalnızca bir kez yer ayrılır ve değer ataması yapılır). static olarak ifade edilmiş yerel değişkenler kapsam dışında olsalar bile değerlerini korur. useStaticLocal çağrıldığı zaman x yazdırılır, 1 arttırılır ve fonksiyondan çıkmadan önce tekrar yazdırılır. Bu fonksiyonun sonraki çağrısında x static yerel değişkeni 51 değerini alacaktır.

Fonksiyonlar

```
26    useLocal(); /* useLocal has automatic local x */
27    useStaticLocal(); /* useStaticLocal has static local x */
28    useGlobal(); /* useGlobal uses global x */
29    useLocal(); /* useLocal reinitializes automatic local x */
30    useStaticLocal(); /* static local x retains its prior value */
31    useGlobal(); /* global x also retains its value */
32
33    printf( "\nlocal x in main is %d\n", x );
34    return 0; /* indicates successful termination */
35 } /* end main */
36
37 /* useLocal reinitializes local variable x during each call */
38 void useLocal( void )
39 {
40    int x = 25; /* initialized each time useLocal is called */
41
42    printf( "\nlocal x in useLocal is %d after entering useLocal\n", x );
43    x++;
44    printf( "local x in useLocal is %d before exiting useLocal\n", x );
45 } /* end function useLocal */
46
```

Fonksiyonlar

useGlobal fonksiyonu herhangi bir değişken tanımlamaz. Bundan dolayı, x değişkenine atıf yaptığında, genel x (9. satır) kullanılır. useGlobal çağrıldığında, genel değişken yazdırılır, 10'la çarpılır ve fonksiyondan çıkmadan önce tekrar yazdırılır. useGlobal fonksiyonu tekrar çağrıldığında, genel değişken 10 değerine sahip olacaktır. Sonunda, tüm fonksiyonlar farklı kapsamlardaki değişkenlere atıfda bulunduğundan fonksiyon çağrılarından hiçbirinin main'deki x'in değerini değiştirmediğini göstermek için program main'deki x yerel değişkenini tekrar yazdırır (33. satır).

Fonksiyonlar

```
47  /* useStaticLocal initializes static local variable x only the first time
48     the function is called; value of x is saved between calls to this
49     function */
50  void useStaticLocal( void )
51  {
52     /* initialized only first time useStaticLocal is called */
53     static int x = 50;
54
55     printf( "\nlocal static x is %d on entering useStaticLocal\n", x );
56     x++;
57     printf( "local static x is %d on exiting useStaticLocal\n", x );
58  } /* end function useStaticLocal */
59
60  /* function useGlobal modifies global variable x during each call */
61  void useGlobal( void )
62  {
63     printf( "\nglobal x is %d on entering useGlobal\n", x );
64     x *= 10;
65     printf( "global x is %d on exiting useGlobal\n", x );
66  } /* end function useGlobal */
```

Fonksiyonlar

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal

local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal

global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal

local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal

local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal

global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal

local x in main is 5
```

Fonksiyonlar

Her bir standart kütüphaneye karşılık gelen o kütüphanedeki tüm fonksiyonların bildirimlerini ve bu fonksiyonlar tarafından ihtiyaç duyulan çeşitli veri türlerinin ve sabitlerin tanımlamalarını içeren bir kütüphane başlığı vardır. Programcı kendisi de bir kütüphane oluşturup ona bir başlık vererek kullanabilir. Bazı standart C kütüphane başlıkları şu şekildedir.

Kütüphane Başlığı	Açıklama
`<assert.h>`	Programda hata ayıklamaya yardımcı olan teşhisleri ilave etmek için bilgi içerir.
`<ctype.h>`	Belirli özellikler için karakterleri test eden fonksiyonlar için fonksiyon bildirimleri ve küçük harfleri büyük harflere çevirmek ve vb. için kullanılabilen fonksiyonlar için fonksiyon bildirimleri içerir.
`<errno.h>`	Hata koşullarını raporlamak için kullanışlı makroları tanımlar.
`<float.h>`	Sistemin ondalık boyut sınırlarını içerir.
`<limits.h>`	Sistemin tümleşik boyut sınırlarını içerir.
`<locale.h>`	Bir programın o anda çalıştığı mahalde değiştirilmesine musade eden fonksiyon bildirimlerini ve diğer bilgileri içerir. locale kavramı bilgisayar sistemlerinin dünya genelindeki tarihler, zaman, para miktarları ve büyük sayılar gibi veriyi ifade etmek için farklı kuralları ele almasını sağlar.
`<math.h>`	Math kütüphane fonksiyonlarının fonksiyon bildirimlerini içerir.
`<setjmp.h>`	Genel fonksiyon çağırma ve dönüş sırasını atlamaya izin veren fonksiyonların bildirimlerini içerir. .
`<signal.h>`	Program yürütmesi sırasında ortaya çıkabilen çeşitli koşulların üstesinden gelen fonksiyon bildirimleri ve makrolar içerir.
`<stdarg.h>`	Sayıları ve türleri bilinmeyen bir fonksiyonun arguman listesi ile ilgilenen makroları tanımlar.
`<stdio.h>`	Standart giriş/çıkış kütüphane fonksiyonları için fonksiyon bildirimlerini ve onlar tarafından kullanılan bilgileri içerir.
`<stdlib.h>`	Sayıları yazıya ve yazıyı sayılara çevirme, hafıza ayırma, rastgele sayılar ve diğer işe yarar fonksiyonlar için fonksiyon bildirimleri içerir.
`<string.h>`	Karakter katarı işleme fonksiyonlarının bildirimlerini içerir.
`<time.h>`	Zaman ve tarih işlemleri yapmak için fonksiyon bildirimleri içerir.

Fonksiyonlar

Bazı problemler için kendi kendini çağıran fonksiyonlar kullanmak faydalı olur. Özyineli bir fonksiyon ya direk olarak yada başka bir fonksiyon üzerinden dolaylı olarak kendini çağıran bir fonksiyondur.

Özyineli problem-çözme yaklaşımı yaygın olarak birkaç maddeden oluşur. Fonksiyon aslında en basit durumun nasıl çözüleceğini bilir. Fonksiyon eğer temel bir durumla çağrılırsa, sonucu geri gönderir. Fonksiyon daha karmaşık bir problemle çağrılırsa, fonksiyon problemi iki kavramsal kısma ayırır: fonksiyonun nasıl yapılacağını bildiği ve bilmediği kısımlar.

Fonksiyonlar

Özyinelemenin mümkün olması için, ikinci kısım esas probleme benzemek, ancak biraz daha basit ve küçük hali olmak zorundadır. Bu yeni problem esas problem gibi gözüktüğü için, fonksiyon daha küçük problem üzerinde çalışmak için kendisinin yeni bir kopyasını başlatır-bu özyineli çağrı olarak adlandırılır. Özyineli çağrı esas çağrıcıya geri gönderilecek bir sonuç oluşturmak için kendi sonucuyla fonksiyonun problemin nasıl çözüleceğini bildiği kısmını birleştirmek için return anahtar kelimesini de içerir.

Fonksiyonlar

Özyinelemeyi sonlandırmak için, her seferinde fonksiyon kendini esas problemin biraz daha küçük bir haliyle çağırır. Küçük problemlerin bu sırası en sonunda temel duruma yakınsamak zorundadır. Fonksiyon temel durumu algıladığında, fonksiyonun önceki kopyasına bir sonuç gönderir ve sonunda fonksiyona yapılan esas çağrı main'e esas istenen sonucu geri gönderene kadar ardışık bir geri dönüş sırası meydana gelir.

Örnek: özyineli olarak faktöriyel hesaplama

n! şeklinde yazılan, 1!=1 ve 0!=1 olarak tanımlandığı negatif olmayan n tam sayının faktöriyeli

n.(n-1).(n-2). 1

Çarpımıdır.

Fonksiyonlar

5!, 120'ye eşit olan 5*4*3*2*1 çarpımına eşittir. 0'dan büyük yada eşit number isimli bir tam sayının faktöriyeli aşağıdaki gibi bir for fonksiyonu kullanılarak tekrarlamalı (özyinelemesiz) olarak hesaplanabilir.

Fonksiyonlar

Sıfırdan büyük veya eşit bir tam sayının faktöriyeli bir for ifadesi kullanılarak özyinelemesiz olarak şu şekilde hesaplanabilir.

```
1   //for döngüsü ile faktöriyel hesabı
2   #include<stdio.h>
3
4   int main(void){//main başlangıcı
5
6   unsigned long long int factorial=1;//faktöriyel için değişken tayin eder
7   int counter;//sayacı tanımlar
8   int number;//faktöriyeli hesaplanacak sayıyı alır
9   printf("faktoriyeli hesaplanacak sayiyi girin: ");//girişi ister
10  scanf("%d",&number);//faktöriyeli hesaplanacak sayıyı alır
11
12  for(counter=number;counter>=1;--counter)
13  factorial*=counter;
14
15  printf("sonuc: %llu\n",factorial);//sonucu yazdırır
16
17  }//main sonu
```

```
faktoriyeli hesaplanacak sayiyi girin: 5
sonuc: 120
```

Fonksiyonlar

Faktöriyel fonksiyonunun özyinelemeli tanımı aşağıdaki ilişkinin tanımlanmasıyla ortaya çıkar:

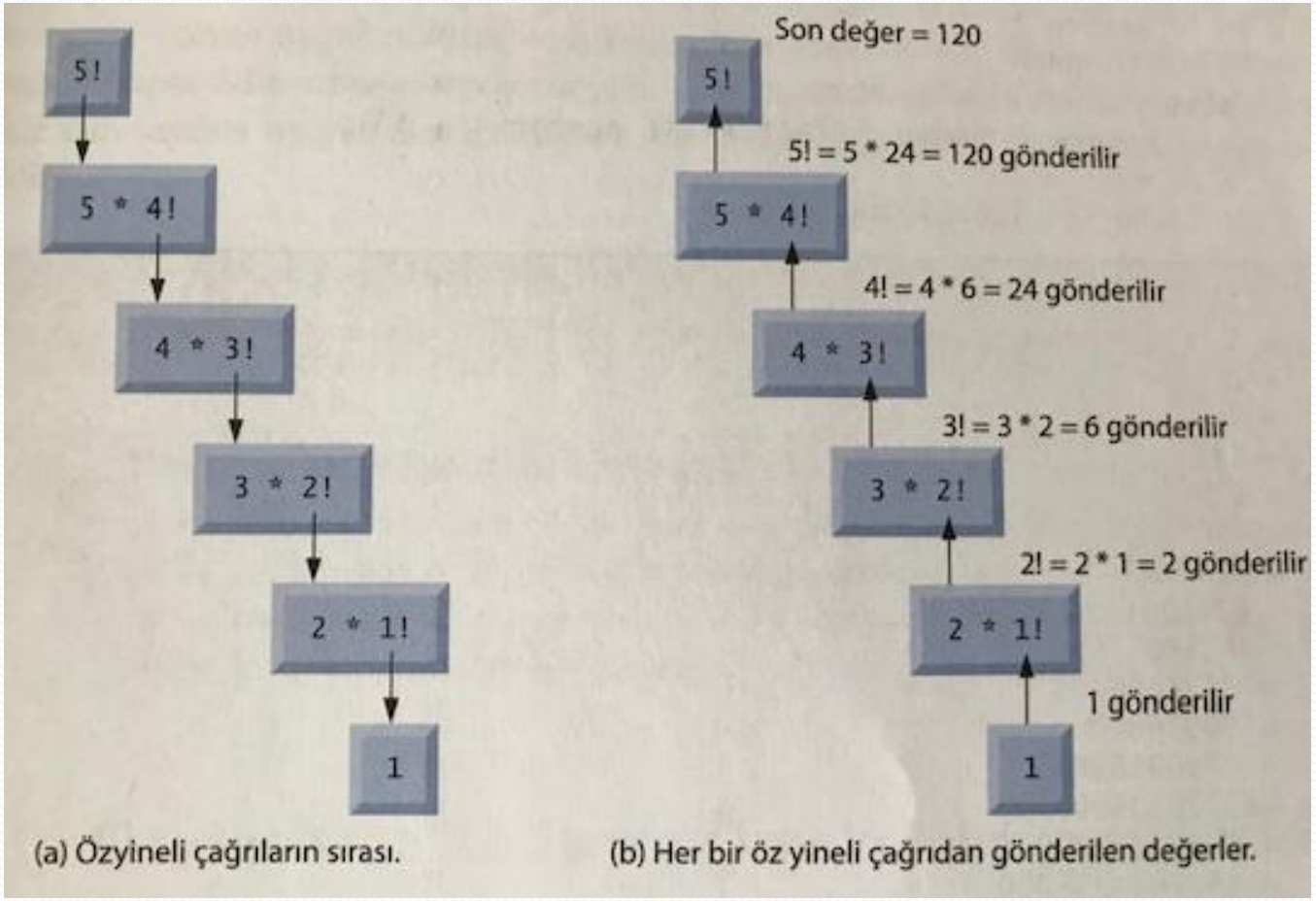
n!=n.(n-1)!

mesela

5!=5.4!

5!'in hesaplanması aşağıdaki şekildeki gibi olur. (a) şekli özyineli çağrıların yinelemeyi sonlandıran 1!'e (yani temel durum) kadar ardışık olarak nasıl ilerlediğini gösterir. (b) ise son değer hesaplanıp gönderilene kadar her bir özyineli çağrıdan çağrılan yere gönderilen değerleri gösterir.

Fonksiyonlar



Fonksiyonlar

1'den 21'e kadar olan sayıların faktöriyelini for döngüsü ve özyinelemeli olarak hesaplayıp yazdıralım.

```c
//özyinelemeli faktöriyel fonksiyonu
#include<stdio.h>

unsigned long long int factorial(unsigned int number);//fonksiyon bildirimi

//main fonksiyonu program yürütmesine başlar
int main(void){
    unsigned int i;//sayaç
    //her bir hesaplamada factorial(i)'yi hesapla
    //ve sonucu göster
    for(i=0;i<=21;++i){
        printf("%u! = %llu\n",i,factorial(i));
    }//for sonu
}//main sonu

//factorial fonksiyonunun özyineli tanımı
unsigned long long int factorial(unsigned int number)
{
    //temel durum
    if(number<=1){
        return 1;
    }//if sonu
    else{//özyineli adım
    return(number*factorial(number-1));
    }//else sonu
}//factorial fonksiyonu sonu
```

```
faktoriyeli hesaplanacak sayiyi girin: 21
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 14197454024290336768

--------------------------------
Process exited after 2.89 seconds with return value 21
Press any key to continue . . .
```

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 14197454024290336768

--------------------------------
Process exited after 0.02952 seconds with return value 27
Press any key to continue . . .
```

Fonksiyonlar

Özyineli factorial fonksiyonu ilk önce sonlandırma koşulunun doğru olup olmadığını, yani number'ın 1'den küçük veya eşit olup olmadığını kontrol eder. number, 1'den küçük veya eşitse, factorial geri dönüş değeri olarak 1'i gönderir, daha fazla özyinelemeye ihtiyaç yoktur. number, 1'den büyükse,

*return number*factorial(number-1);*

komut satırına program yönlenir. Bu komut satırı number-1'in faktöriyeli ile number sayısını çarpar. factorial fonksiyonu unsigned int alır ve unsigned long long int türünden bir sonuç gönderir. C standartı, unsigned long long int türünden bir değişkenin en fazla 18.446.744.073.709.551.615 kadar büyük bir değer tutabilir.

Fonksiyonlar

Fibonacci serisi 0 ve 1 ile başlar ve sonra gelen Fibonacci sayısı önceki iki Fibonacci sayısının toplamına eşittir.

0,1,1,2,3,5,8,13,21,.....

Ardışık Fibonacci sayılarının oranı 1,618 sabit değerine yakınsar. Buna altın oran denir. Fibonacci serisi özyineli olarak şu şekilde tanımlanabilir;

fibonacci(0)=0

fibonacci(1)=1

fibonacci(n)=fibonacci(n-1)+fibonacci(n-2)

Fibonacci sayısını fibonacci fonksiyonu kullanarak özyineli olarak hesaplayan program şu şekildedir;

```c
//özyineli fibonacci fonksiyonu
#include<stdio.h>

unsigned long long int fibonacci(unsigned int);//fonksiyon bildirimi

//main fonksiyonu program yürütmesine başlar
int main(void)
{
    unsigned long long int result;//fibonacci değeri
    unsigned int number;//kullanıcı tarafından girilen sayı

    //kullanıcıdan tam sayı al
    printf("bir tam sayi girin: ");
    scanf("%u",&number);

    //kullanıcı tarafından girilen sayı için fibonacci değeri hesapla
    result=fibonacci(number);

    //sonucu göster
    printf("Fibonacci(%u)=%llu\n",number,result);
}//main sonu

//fibonacci fonksiyonunun yinelemeli tanımı
unsigned long long int fibonacci(unsigned int n)
{
    //temel durum
    if(0==n || 1==n){
        return n;
    }//if sonu
    else{//yineleme adımı
    return fibonacci(n-1)+fibonacci(n-2);
    }//else sonu
}//fibonacci fonksiyonu sonu
```

```
bir tam sayi girin: 0
Fibonacci(0)=0
```

```
bir tam sayi girin: 1
Fibonacci(1)=1
```

```
bir tam sayi girin: 2
Fibonacci(2)=1
```

```
bir tam sayi girin: 3
Fibonacci(3)=2
```

```
bir tam sayi girin: 10
Fibonacci(10)=55
```

```
bir tam sayi girin: 20
Fibonacci(20)=6765
```

```
bir tam sayi girin: 30
Fibonacci(30)=832040
```

```
bir tam sayi girin: 40
Fibonacci(40)=102334155
```

Fonksiyonlar

main'den fibonacci'ye yapılan çağrı özyineli bir çağrı değildir, ancak sonra yapılan tüm çağrılar özyinelidir. fibonacci her başlatıldığında, ilk önce temel durumu kontrol eder (n, 0'a yada 1'e eşit mi). Eğer bu doğru ise n geri gönderilir. n, 1'den büyükse, yineleme adımı her bir fibonacci'ye yapılan esas çağrıdan biraz daha basit olan iki özyineli çağrı üretir. Aşağıdaki şekil fibonacci fonksiyonunun fibonacci(3)'ü nasıl hesapladığını gösterir.

Fonksiyonlar