

C PROGRAMLAMA



FONKSİYONLAR



- Gerçek hayattaki problemlerin çözümü için geliştirilen programlar çok büyük boyutlardadır.
- Daha büyük programlar yazmanın en kolay yolu onları küçük parçalar halinde yazıp sonra birleştirmektir.
- Böylece çok büyük boyutlardaki program kodlarını yönetmek daha kolay olacaktır.
- Örneğin; ikinci dereceden bir bilinmeyenli bir denklemin köklerinin hesaplanması işlemini yapan bir program geliştirdiğimizi düşünelim.
 - Bu problemin çözümünde yapılacak matematiksel işlemler aşağıdaki gibidir.
 - 1- diskriminantın hesaplanması
 - 2- diskriminantın sonucuna bakarak kök olup olmadığına karar verilmesi.
 - 3- duruma göre köklerin bulunması.
- Görüldüğü gibi burada esas problemimiz kök hesabı olduğu halde, bu problemi çözebilmek için daha küçük alt problemlerin (sub problems) çözülmesi gereklidir.
- Eğer fonksiyon kullanmadan düz mantıkla bu problemi kodlarsak bütün çözüm parçacıklarını main() bloğu içine yazılmalıyız.
- Böyle yaparsak ana program bloğumuz problemin büyüklüğüne göre uzar, okunabilirliği azalır ve müdahale etmek zorlaşır.

FONKSİYONLAR

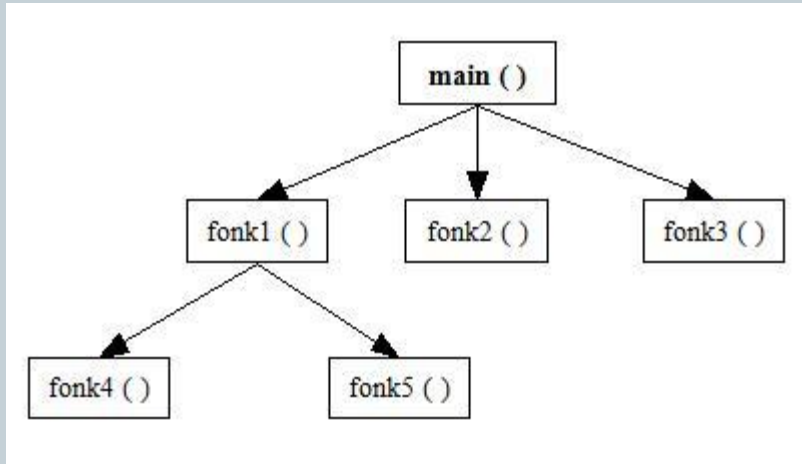


- Bu türlü büyük problemleri kendi içerisinde, her biri verilen bir işi çözmek için tasarlanmış alt program (sub program) parçacıklarından, yani modüllerden oluşturmak daha mantıklı olacaktır.
- C dilinde bu modüllere fonksiyon (function) adı verilir.
- Fonksiyonlara işlev yada alt yordam adı da verilmektedir.
- Bir fonksiyonu çalıştırma işine *çağırma* (function call) denir.
- Her fonksiyon, ismi (function name) ve kendisinden istenen işi gerçekleştirmek için gerekli olan değerler yani parametreler-argümanlar ile çağırılırlar.
- Bazı fonksiyonlar kendi içerisinde çeşitli işlemler yaptıktan sonra yaptığı işlemin sonucunu kendisini çağıran fonksiyona bildirirler. Bu değere geri dönüş değeri (return value) adı verilir.

FONKSİYONLAR



- C programlarında main() ana fonksiyon olduğu için programın çalışması main() fonksiyonundan başlar.
- Main fonksiyonu diğer fonksiyonları çağırarak çalıştırır.
- Bir fonksiyon içerisinde başka bir veya birkaç fonksiyonda çağırılabilir.



fonksiyondan geri dönen değer **x** in karaköküdür.

```
a = sqrt ( x );
```

fonksiyon ismi

parametresi

Fonksiyon kullanmanın faydaları



- **Kodun gereksiz yere büyümesini engeller.**
 - Sıkça tekrarlanan işler için bir kere fonksiyon yazıldığında aynı kodlar tekrar yazılmaksızın istendiği kadar çalıştırılabilir.
- **Okunabilirliği artırarak algılamayı kolaylaştırır.**
 - Main() fonksiyonu içerisinde birbirinden ayrılmış sadece kendi işlerinin yapan fonksiyonların isimleri bulunur. Detay işlemler fonksiyonların içinde halledilir. Bu programlama tekniğine prosedürel soyutlama (procedural abstraction) adı verilir.
- **Programın test edilmesini ve hataların bulunmasını kolaylaştırır.**
 - Hata araştırılırken aranılması gereken alt program bloğuna bakılır. İstenilirse yalnız başlarına da test edilebilirler.
- **Güncelleştirilebilir olmasını ve yeniden kullanabilme kolaylığı sağlar.**
 - Modüler olarak yazıldıklarında istenilen projelerde defalarca kullanılabilirler.

C dilinde standart fonksiyonlar ve kullanıcı tanımlı fonksiyonlar olmak üzere iki tip fonksiyon bulunur.

Standart fonksiyonlar



- C dilinin geliştiricileri tarafından programcıların kullanmaları için önceden yazılmış olan hazır fonksiyonlardır.
- Programcı, kullanmak istediği hazır fonksiyon prototiplerinin bulunduğu başlık (header) dosyalarını include ön işlemci direktifi ile bildirerek kullanabilir.
- Hazır fonksiyonların kendileri .LIB uzantılı kütüphane dosyaları içindedirler.
- Standart girdi-çıkıtlı işlemleri için kullanılan **printf()** ve **scanf()** fonksiyonlar standart veya hazır fonksiyonlardır.
- **stdlib** kütüphanesinde bulunan **rand()** fonksiyonu rastgele sayı üretmek için kullanılan bir hazır fonksiyondur.

Standart fonksiyonlar



- Standart veya hazır fonksiyonlara matematiksel işlemleri gerçekleştirmek için kullanılan ve **math** kütüphane dosyasında bulunan fonksiyonlarda örnek olarak verilebilir.

Fonksiyon Bildirimi	Açıklama	Örnek	Sonuç
<code>int abs(int x);</code>	x tamsayısının mutlak değerini hesaplar	<code>abs(-4)</code>	4
<code>double fabs(double x);</code>	x gerçel sayısının mutlak değerini hesaplar	<code>fabs(-4.0)</code>	4.000000
<code>double floor(double x);</code>	x'e (x'den büyük) en yakın tamsayıyı gönderir	<code>abs(-2.7)</code>	3.000000
<code>double ceil(double x);</code>	x'e (x'den küçük) en yakın tamsayıyı gönderir	<code>abs(5.6)</code>	5.000000
<code>double sqrt(double x);</code>	pozitif x sayısının karekökünü hesaplar	<code>sqrt(4.0)</code>	2.000000
<code>double pow(double x, double y);</code>	x^y (x^y) değerini hesaplar	<code>pow(2.0, 3.0)</code>	8.000000
<code>double log(double x);</code>	pozitif x sayısının doğal logaritmasını hesaplar, $\ln(x)$	<code>log(4.0)</code>	1.386294
<code>double log10(double x);</code>	pozitif x sayısının 10 tabanındaki logaritmasını hesaplar	<code>log10(4.0)</code>	0.602060
<code>double sin(double x);</code>	radian cinsinden girilen x sayısının sinüs değerini hesaplar	<code>sin(3.14)</code>	0.001593
<code>double cos(double x);</code>	radian cinsinden girilen x sayısının kosinüs değerini hesaplar	<code>cos(3.14)</code>	-0.999999
<code>double tan(double x);</code>	radian cinsinden girilen x sayısının tanjant değerini hesaplar	<code>tan(3.14)</code>	-0.001593
<code>double asin(double x);</code>	sinüs değeri x olan açıyı gönderir. Açı $-\pi/2$ ile $\pi/2$ arasındadır.	<code>asin(0.5)</code>	0.523599
<code>double acos(double x);</code>	cosinüs değeri x olan açıyı gönderir. Açı $-\pi/2$ ile $\pi/2$ arasındadır.	<code>acos(0.5)</code>	1.047198
<code>double atan(double x);</code>	tanjant değeri x olan açıyı gönderir. Açı $-\pi/2$ ile $\pi/2$ arasındadır.	<code>atan(0.5)</code>	0.463648

Kullanıcı tanımlı fonksiyonlar



- C dilinde programcı kendi fonksiyonlarını oluşturarak kullanabilir.
- Bu tür fonksiyonlara kullanıcı tanımlı fonksiyonlar denir.
- Genel fonksiyon tanımlaması aşağıdaki gibidir.

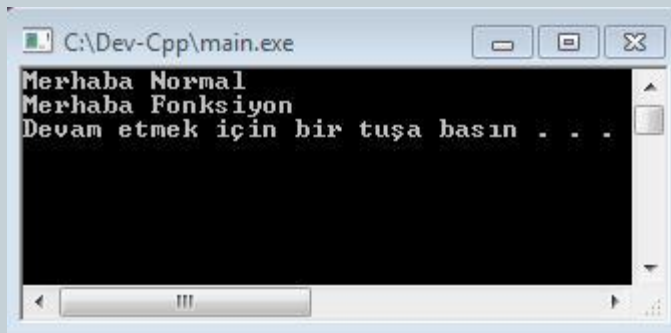
```
[FonksiyonTürü] <FonksiyonAdı> ([Parametreler]) {  
    .....;  
    .....; /* Fonksiyon içerisinde Yapılacak işlemler.. */  
    .....;  
}
```

- Fonksiyonların, parametre alıp almamaları ve geriye değer döndürüp döndürmemelerine göre farklı kullanımları vardır.

Kullanıcı tanımlı fonksiyonlar



- Geriye değer döndürmeyen ve parametre almayan fonksiyonlar:
 - Bu tür fonksiyonlar, çağıran fonksiyondan ne bir değer alırlar nede geriye değer döndürürler.
 - Geriye dönüş değeri olmadığı için fonksiyonun türü kısmına **void** ifadesi yazılır.
 - Fonksiyonun türü aynı zamanda geriye dönüş değerinin türüdür.
 - Fonksiyon türü yazılmadığında derleyici **int** olarak varsayar.



```
/* geriye değer döndürmeyen ve
 * parametre almayan fonksiyona örnek
 */
#include <stdio.h>
#include <stdlib.h>

void fonksiyon () {
    printf("Merhaba Fonksiyon\n");
}

int main()
{
    printf("Merhaba Normal\n");

    fonksiyon();

    system("PAUSE");
    return 0;
}
```

Kullanıcı tanımlı fonksiyonlar



- **Parametre alıp geriye değer döndürmeyen fonksiyonlar:**
 - Bu tür fonksiyonlar çağıran fonksiyondan bir yada birden fazla parametre alır ve geriye değer döndürmezler.
 - Bu tür fonksiyonlar çağrılırken fonksiyona gönderilecek değerler yada değerleri tutan değişkenler fonksiyon parantezinin içine yazılırlar.
 - Bu değerleri karşılayan değişkenlerde fonksiyon tanımlama kısmında parantezler içerisinde belirtilir.
 - Gönderilen değerler karşılayan değişkenler içerisine kopyalanır.
 - Gönderilen değerler ile karşılayan değişkenlerin sayısı eşit ve aynı sırada olmalıdır.

Kullanıcı tanımlı fonksiyonlar



- Parametre alıp geriye değer döndürmeyen fonksiyonların kullanımına bir örnek.

```
/* Bu fonksiyon kendisine gönderilen değere göre
   ekrana tek yada çift mesajını yazar
*/
#include <stdio.h>

void tek_cift (int gelenSayi) {
    if(gelenSayi%2==0)
        printf("Girilen sayi ( %d ) CIFTTIR. \n", gelenSayi);
    else
        printf("Girilen sayi ( %d ) TEKTIR. \n", gelenSayi);
}

int main()
{
    int sayi;
    printf("Sayiyi giriniz = ");
    scanf("%d",&sayi);

    tek_cift(sayi);

    system("PAUSE");
    return 0;
}
```



Kullanıcı tanımlı fonksiyonlar



- Parametre alıp geriye değer döndüren fonksiyonlar:
 - Bu tür fonksiyonlar çağıran fonksiyondan bir yada daha fazla parametre alır ve bunlar üzerinde çeşitli işlemler gerçekleştirerek geriye bir değer döndürürler.
 - Bir fonksiyondan geriye değer döndürmek için **return** komutu kullanılır.
 - Bu fonksiyonlarda **return** komutu çalıştığı anda fonksiyonun işi biter ve fonksiyon sonlanır.
 - Bu fonksiyon çağrıldığı fonksiyon içinde bir değişkene değer atama işleminin sol tarafında çağrılmalıdır.
 - ✧ Örnek;

```
sonuc = topla (sayi1, sayi2);
```

Kullanıcı tanımlı fonksiyonlar



- Parametre alan ve geriye değer döndüren fonksiyon kullanımına bir örnek.

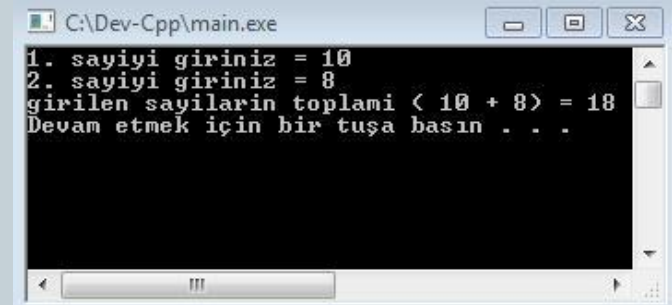
```
/* kendisine gönderilen iki sayının
   toplamını geri döndüren bir fonksiyon örneği
*/
#include <stdio.h>

int topla (int x, int y) {
    int toplam;
    toplam = x + y;
    return toplam;
}

int main()
{
    int sayi1, sayi2, sonuc;
    printf("1. sayiyi giriniz = ");
    scanf("%d",&sayi1);
    printf("2. sayiyi giriniz = ");
    scanf("%d",&sayi2);

    sonuc = topla (sayi1, sayi2);

    printf("girilen sayilarin toplami ( %d + %d) = %d \n",sayi1,sayi2,sonuc);
    system("PAUSE");
    return 0;
}
```



Fonksiyonlarda parametre akış türleri



- Fonksiyonlara değerler iki farklı yöntemle gönderilirler.
- 1 – Değerle çağırma (call by value):
 - Bu yöntemle çağırılan fonksiyondan gönderilen değerler, çağrılan fonksiyondaki parametrelerin içerisine kopyalanırlar.
 - Kopyaların değişmesi orijinal değerleri etkilemez.
 - Şu ana kadar fonksiyon çağırımlarında hep bu yöntem kullanıldı.
- 2 – Referansla çağırma (call by reference):
 - Bu yöntemle kopyalanan değerlerin fonksiyon içerisinde değiştirilmesi orijinallerini de etkiler.
 - Bu yöntem ileriki bölümlerde ayrıntılı olarak ele alınacaktır.

Fonksiyon prototipi tanımlama



- Bu bölüme kadar olan örneklerde fonksiyonlar, kendisini çağıran fonksiyonların üzerinde olacak şekilde yazıldı (main() fonksiyonunun üstünde).
- Bunun sebebi, fonksiyonların geriye döndürecekleri değerlerin tiplerinin fonksiyonun çağrıldığı komut satırına gelmeden önce derleyiciye bildirilmesi gerektiğidir.
- Derleme işlemi yukarıdan aşağıya doğru yapıldığı için fonksiyonlar üstte tanımlandığında derleyici fonksiyonun geri dönüş değerinin tipini bilir.
- Fakat fonksiyonların aşağıda da tanımlandığı durumlar olabilir.
- Bu durumda en üstte fonksiyonların prototipini tanımlamak bu problemi çözecektir.

Fonksiyon prototipi örnek



```
/* kendisine gönderilen iki sayının
   toplamını geri döndüren fonksiyon protitipi
*/
#include <stdio.h>

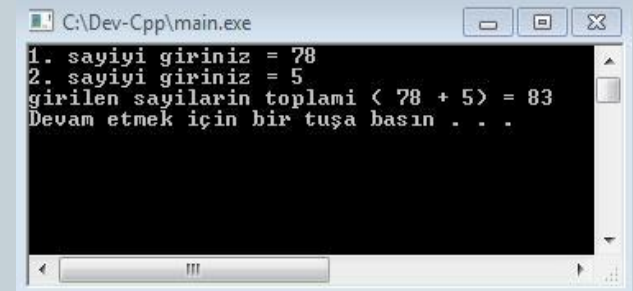
int topla (int, int); // fonksiyon protitipi

int main()
{
    int sayi1, sayi2, sonuc;
    printf("1. sayiyi giriniz = ");
    scanf("%d",&sayi1);
    printf("2. sayiyi giriniz = ");
    scanf("%d",&sayi2);

    sonuc = topla (sayi1, sayi2);

    printf("girilen sayilarin toplami ( %d + %d) = %d \n",sayi1,sayi2,sonuc);
    system("PAUSE");
    return 0;
}

int topla (int x, int y) {
    return (x+y);
}
```



```
C:\Dev-Cpp\main.exe
1. sayiyi giriniz = 78
2. sayiyi giriniz = 5
girilen sayilarin toplami < 78 + 5> = 83
Devam etmek için bir tuşa basın . . .
```


Depolama sınıfları



- Daha önceki konularda değişkenler anlatılırken, değişkenin türü, ismi ve değeri olmak üzere üç özellik üzerinde duruldu.
- Bu bölümde bu özelliklere ek olarak değişkenlerin depolama sınıfı (storage classes), ömrü (duration) ve faaliyet alanı (scope) kavramları üzerinde durulacaktır.
- C programlama dilinde *auto*, *register*, *static*, *extern*, belirleyicileri ile tanımlanabilecek 4 tür depolama sınıfı vardır.
- Bu belirleyiciler ile tanımlanan değişkenler ömürleri itibariyle ikiye ayrılırlar.
- Bunlar otomatik depolama ömürlüleri (automatic storage classes) ve durağan depolama ömürlüleri (static storage classes) dir.

Otomatik depolama ömürlüleri (Automatic storage classes)



- Bu tür değişkenler auto veya register anahtar kelimeleri kullanılarak tanımlanan değişkenlerdir.
- Bir değişkenin otomatik ömürlü olması; tanımlandıkları bloğun çalışması ile hafızaya yerleştirilip, bloğun çalışması bittiği anda bellekten yok edilmesi anlamına gelir.
- Sadece değişkenler otomatik ömürlüdürler.
- Bir fonksiyonun yerel (local) değişkenleri (bunlar fonksiyonun parametre listesinde yada gövdesi içinde tanımlanmış olanlar olabilir) normal olarak bir belirleyici yazılmadığında varsayılan olarak otomatik ömürlüdürler.

Durağan depolama ömürlüler (Static Storage Classes)



- Bu tür değişkenler yada fonksiyonlar *extern* veya *static* anahtar kelimeleri kullanılarak tanımlanırlar.
- Bir tanımın durağan depolama ömürlü olması; programın çalışmaya başlaması ile hafızaya yerleştirileceği ve program sonlanana kadar hafızadan silinmeyeceği anlamına gelir.
- Global değişkenler, global fonksiyonlar ve *static* anahtar kelimesi ile tanımlanmış olan yerel değişkenler bu sınıf içerisine girerler.
- Global değişkenler ve fonksiyonlar varsayılan olarak *extern* tanımlıdırlar.
- Global değişkenler fonksiyonların dışında tanımlanan değişkenlerdir.
- Bir değişkenin global olması programın çalıştığı andan itibaren bellekte tutulması anlamına gelir.

Durağan depolama ömürlüleri (Static Storage Classes)



- Global değişkenler ve fonksiyonlar tanılandıkları noktadan programın sonuna kadar her yerde kullanılabilirler.
- *Static* anahtar kelimesi ile tanımlanmış olan yerel değişkenler yalnızca tanımlandıkları fonksiyon içerisinde kullanılabilirler ancak otomatik değişkenlere benzemezler.
- Bu tür değişkenler fonksiyon çalıştığı anda hafızaya yerleşirler ve program bitene kadar bellekte kendileri ve değerleri saklanır.
- Aynı fonksiyon bir kere daha çalıştırıldığında *static* olarak tanımlanmış olan değişken, bir önceki çalışmadan kalan değerini kullanmaya devam eder.
- Bütün durağan ömürlü tanımlanan sayısal değişkenlerin ilk değerleri 0 olarak belirlenir.

Faaliyet alanı kuralları (scope rules)



- Değişkenin geçerli olduğu yani kullanılabileceği program parçasına o değişkenin faaliyet alanı denir.
- Örneğin bir blok ({ ve } işaretleri arasındaki program parçası) içerisinde tanımlanan yerel değişkenler yalnızca o blok ve o bloğun içerisindeki bloklarda geçerlidir.
- C programlarında, dosya faaliyet alanı (File scope), blok faaliyet alanı (Block scope) ve fonksiyon faaliyet alanı (Function scope) olmak üzere üç tür faaliyet alanı vardır.



- **Dosya faaliyet alanı:**

- Bir tanımlama fonksiyonların dışında yapıldığı zaman tanımlanan değişken dosya faaliyet alanına sahiptir.
- Bunlar tanımlamanın yapıldığı noktadan dosyanın sonuna kadar bütün fonksiyonlar tarafından tanınır ve kullanılırlar.
- Global değişkenle, fonksiyon tanımlamaları ve fonksiyon prototipleri dosya faaliyet alanına girer.

- **Blok faaliyet alanı:**

- Bir blok içerisindeki bütün bildiriler blok faaliyet alanına sahiptir.
- Bunların faaliyet alanı blok sonunu gösteren } işaretine kadardır.
- Bloklar iç içe yazıldığında dıştaki blok içerisindeki değişkenler içteki blok içerisinde de geçerlidir.
- Dıştaki ve içteki bloklar aynı isimde değişkenlere sahip olabilirler, ancak iç bloktaki değişken dış bloktakini maskeler.

- **Fonksiyon faaliyet alanı:**

- Fonksiyon faaliyet alanı da blok faaliyet alanı gibi düşünülebilir.
- Fonksiyon parametre listesinde ve gövdesinde tanımlanan değişkenler yalnızca o fonksiyon içerisinde geçerlidirler.
- *Static* anahtar kelimesi ile tanımlanmış olan yerel değişkenler programın çalışmasından sonuna kadar bellekte tutulurlar ancak blok faaliyet alanına sahiptirler.
- Faaliyet alanları değişkenlerin ömürlerinin etkilemez.

faaliyet alanlarına göre değişkenlerin değerlerinin nasıl değiştiğini gösteren bir program



```
/* faaliyet alanlarına göre değişkenlerin değerlerinin
   nasıl değiştiğini gösteren bir program
*/
#include <stdio.h>

void a( void ); // a global fonksiyon prototipi.
void b( void ); // b global fonksiyon prototipi.
void c( void ); // c global fonksiyon prototipi.
int x = 1;      // global x değişkeni.

int main()
{
    int x = 5; // yerel (local) x değişkeni.
    printf("(main deki) x = %d\n", x);
    {
        // yeni bir blok yeni bir faaliyet alanı başlıyor.
        int x = 7;
        printf("(main de blokdaki) x = %d\n", x);
    } // faaliyet alanı sona erdi.
    printf("(main deki) x = %d\n", x);
    a(); // otomatik ömürlü yerel x değişkeni var.
    b(); // durağan ömürlü yerel x değişkeni var.
    c(); // global x değişkenini kullanıyor.
    printf("\n**Fonksiyonlar 2. defa cagrililiyor**\n");
    a(); // yerel x'i yeniden oluşturun.
    b(); // yerel x'i eski değerinden devam ediyor.
    c(); // global x'in eski değerini kullanıyor
    printf("\n(main deyiz son) x = %d\n", x);

    system("PAUSE");
    return 0;
}
```

```
C:\Dev-Cpp\Project1.exe
<main deki> x = 5
<main de blokdaki> x = 7
<main deki> x = 5

<a deyiz> x = 25
<a da ++x den sonra> x = 26

<b deyiz> x = 50
<b de ++x den sonra> x = 51

<c deyiz> x = 1
<c deyiz x*=10 den sonra> x = 10

**Fonksiyonlar 2. defa cagrililiyor**

<a deyiz> x = 25
<a da ++x den sonra> x = 26

<b deyiz> x = 51
<b de ++x den sonra> x = 52

<c deyiz> x = 10
<c deyiz x*=10 den sonra> x = 100

<main deyiz son> x = 5
Devam etmek için bir tuşa basın . . .
```

faaliyet alanlarına göre değişkenlerin değerlerinin nasıl değiştiğini gösteren bir program



```
void a(){
    int x = 25;
    /* a fonksiyonu her çalıştığında
    x değişkeni yeniden oluşturulup ilk değeri atanır */
    printf("\n (a dayız) x = %d\n",x);
    ++x;
    printf("(a da ++x den sonra) x = %d\n",x);
}

void b(){
    static int x = 50;
    /* durağan tanımlı yerel x değişkeni
    b fonksiyonu ışk çağrıldığında belleğe yerleşir
    ilk değer atama işlemide yalnızca ilk anda geçerlidir */
    printf("\n (b deyiz) x = %d\n",x);
    ++x;
    printf("(b de ++x den sonra) x = %d\n",x);
}

void c(){
    printf("\n (c deyiz) x = %d\n",x);
    x*=10;
    printf("(c deyiz x*=10 den sonra) x = %d\n",x);
}
```