SAKARYA ÜNİVERSİTESİ

Programlamaya Giriş
HAFTA 7
İşlevler (Fonksiyonlar)

Prof. Dr. Cemil ÖZ

Doç. Dr. Cüneyt BAYILMIŞ

Arş. Gör. Dr. Gülüzar ÇİT

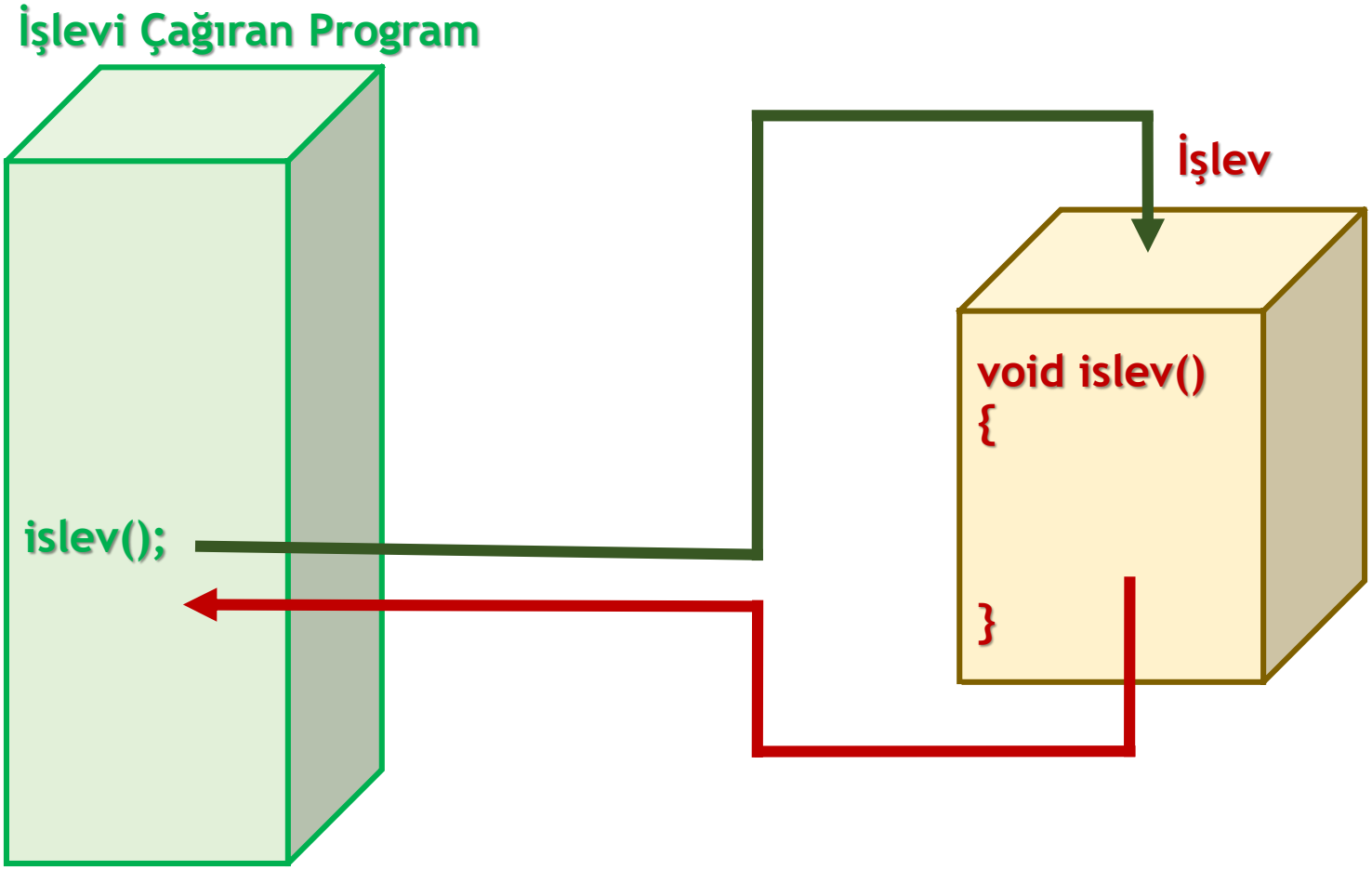
Konu & İçerik

- İşlevler (Fonksiyonlar)
  - Fonksiyon Tanımlama
  - İşlevlerin Aşırı Yüklenmesi (Function Overloading)
  - Scope (Kapsam)
  - Inline Fonksiyonlar
  - Yapılar – Fonksiyonlar
  - Özyinelemeli (Recursive) Fonksiyonlar
  - Fonksiyon Çağrıları
- Çalışma Soruları
- Kaynaklar

İşlevler (Fonksiyonlar)

- Büyük problemler, küçük parçalara bölünerek kolay çözülür.
- Bu parçalara **işlev**, **alt program**, **yöntem** ya da **procedure** denilmektedir.
- Yazılımlar benzer modüller içerirler. Dolayısıyla program içerisinde tekrar eden işlemlerin her defasında yeniden yazılması engellenerek program geliştirme kolaylaştırılır.
- Hata ayıklama küçük ölçekte daha kolaydır.
- Küçük parçalara yoğunlaşmak kolaydır.
- Parçalara ayrılan problem çok sayıda insan tarafından paralel olarak çözülebilir.
- Yapısal programlama fonksiyonlarla mümkündür.
- Daha güvenli ve verimli kod üretimi sağlanır.
- Ayrıca program boyutları da nispeten küçülür.

İşlevler (Fonksiyonlar)…



İşlevler (Fonksiyonlar)…

```
Dönüş_tipi fonksiyonAdi (parametre_listesi) {
    gerekli değişken tanımlamaları…;
    fonksiyon gövdesi…;
    return geriDonusDegeri / Degiskeni;
}
```

- **Fonksiyon adı**
  - Yöntemin adını belirler. Değişken isimlendirme kuralları aynen geçerlidir.
- **Parametre listesi**
  - Yöntem içerisinde kullanılacak giriş parametreleri tanımlanır.
- **Geri Dönüş Değeri / Değişkeni**
  - Yöntemden geriye dönen değer / değişken belirtilir.
- **Dönüş Tipi**
  - Yöntemden geri dönen değerin tipi yazılırGeriye değer dönmeyecek ise **void** yazılmalıdır. void kullanımı durumunda fonksiyon return komut satırı içermez. Fonksiyonda geri dönüş değeri/değişkeni return komut satırı ile belirtilir.

İşlevler (Fonksiyonlar)…

```
float aylikMaas(float mesaiSaati, float saatUcreti)
```

```
float bol(int x, float f)
```

```
int kareAl(int x)
{

    return x*x;
}
```

```
void mesaj(int mesajNo)
{
    switch (mesajNo)
    {
    case 0 : cout << "Mesaj 0"; break;
    case 1 : cout << "Mesaj 1"; break;
    default: cout << "Tanımsız..."; break;
    }
}
```

İşlevler (Fonksiyonlar)…

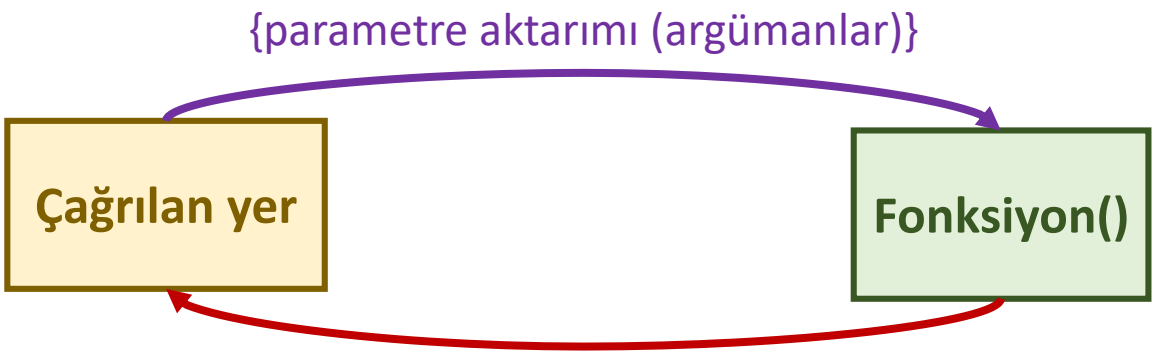
- Fonksiyon tanımı çağıran fonksiyondan önce yapılmalıdır.
- Eğer sonra yapılacaksa, tanıtım (prototip) mutlaka çağıran fonksiyondan önce yapılmalıdır.

Dönüş_tipi *fonksiyonAdi* (parametre_listesi) ;

```
int kareAl(int);
```

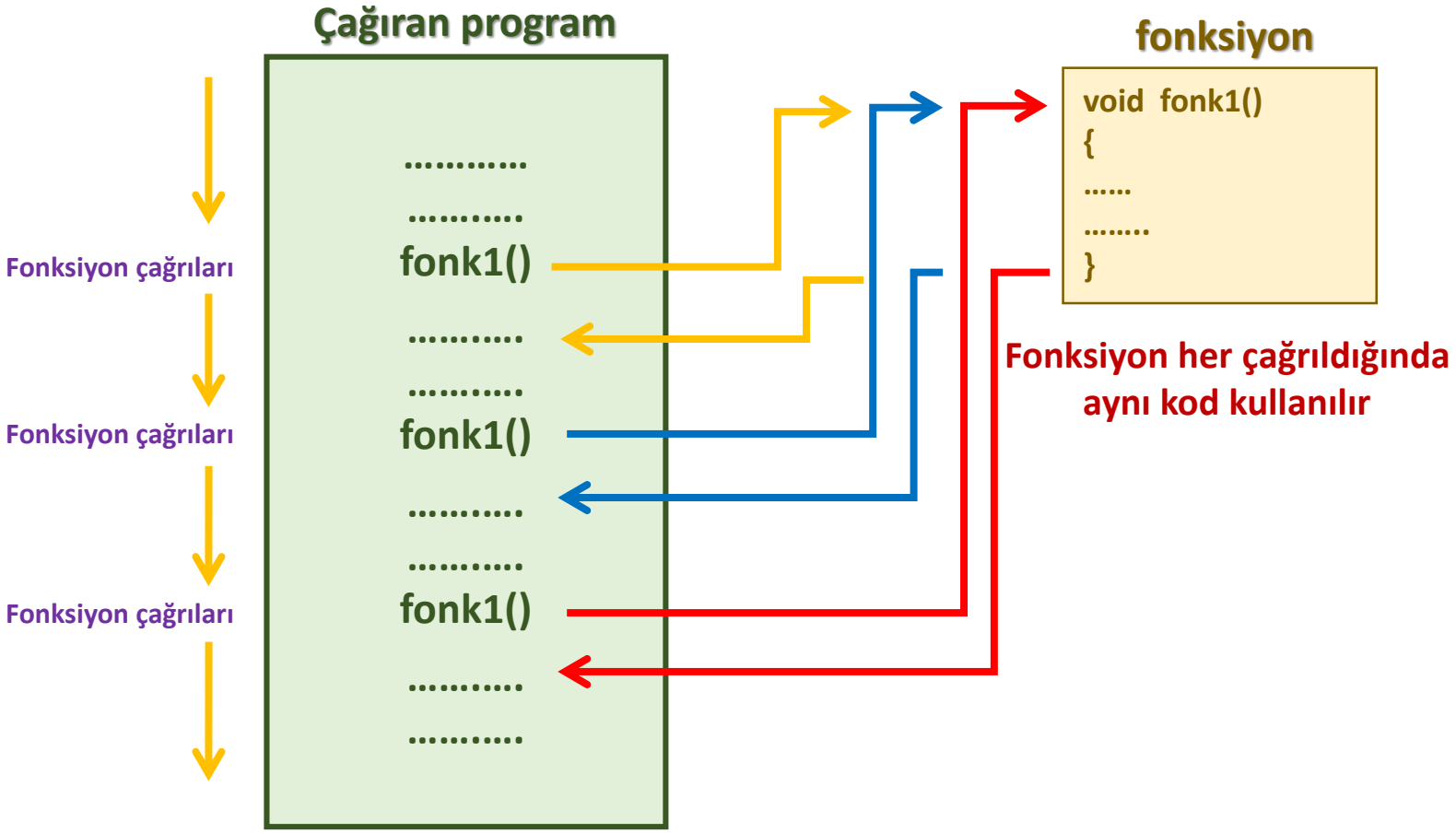
- Diğer bir değişle; bir C/C++ programında fonksiyonlar ana fonksiyon olan **main()** fonksiyonundan önce veya sonra tanımlanabilir. Kullanılacak fonksiyon main() fonksiyonundan önce tanımlanır ise bu durumda fonksiyonun ön bildirimine ihtiyaç yoktur.
- Fakat özellikle büyük boyutlu programlar oluştururken programın okunabilirliği açısından main() fonksiyonundan önce ön bildirim yapıp fonksiyonu main() fonksiyonundan sonra tanımlamak daha uygundur.
- Parametreler/argümanlar yerel değişkenlere benzerler ve sadece fonksiyon içerisinde tanınırlar.

İşlevler (Fonksiyonlar)…

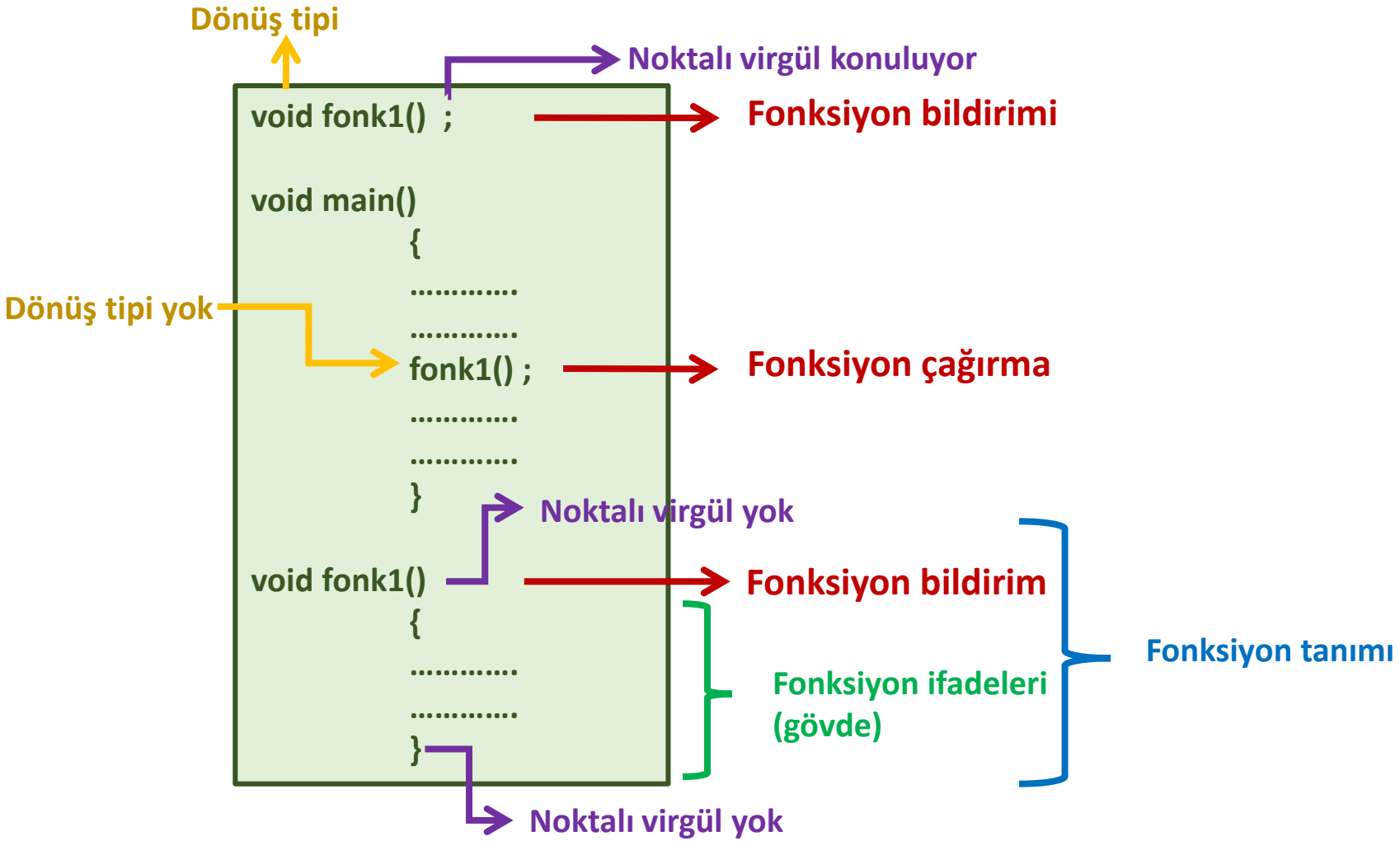


İşlevler (Fonksiyonlar)…

- Bir fonksiyonun kullanımı ve kontrol akışı



İşlevler (Fonksiyonlar)…



İşlevler (Fonksiyonlar)…

➢ **ÖRNEK:** ⇒ [1]_void1.cpp

```
#include <iostream>

using namespace std;

void ciz();

int main()
{

    ciz();                          //fonksiyon çağırma
    cout << "Veri Tipi       Aralik" << endl;
    ciz();                          //fonksiyon çağırma
    cout << "char         -128 to 127" << endl
         << "short       -32,768 to 32,767" << endl
         << "int         System dependent" << endl
         << "long        -2,147,483,648 to 2,147,483,647" << endl;
    ciz();

    system("pause");

    return 0;
}

void ciz()
{
   for(int j=0; j<45; j++)
      cout << '-';
   cout << endl;
}
```

Prototip tanımlamamak için **void ciz ()** fonksiyon bloğu main fonksiyonundan önce yazılmalıdır.

İşlevler (Fonksiyonlar)…

- C/C++’ da bir fonksiyonuna parametre aktarımı (argüman kullanımı) iki şekilde yapılabilir:
  - **Değer İle Çağırma (Calling By Value)**
    - Verinin değeri doğrudan aktarılabilir yani değer doğrudan parametre olarak kullanılabilir
  - **Adres İle Çağırma (Calling By Reference)**
    - Verinin adresi (değişken) aktarılabilir yani değişken parametre olarak kullanılabilir
  - Bunun ile birlikte fonksiyon argümanların bir kısmı **değer** bir kısmı da **adres** şeklinde de olabilir.

İşlevler (Fonksiyonlar)…

➢ **ÖRNEK:** ⇒ [2]_ciz2.cpp

```
void ciz(char, int);

int main()
{

    ciz('*', 60);                          //fonksiyon çağırma
    cout << "Veri Tipi       Aralik" << endl;
    ciz('-',50);                          //fonksiyon çağırma
    cout << "char         -128 to 127" << endl
        << "short       -32,768 to 32,767" << endl
        << "int         Sisteme bağlı" << endl
        << "long        -2,147,483,648 to 2,147,483,647" << endl;
    ciz('-', 50);

    system("pause");

    return 0;
}
```

```
void ciz(char ch, int n)
{
   for(int j=0; j<n; j++)
      cout << ch;
   cout << endl;
}
```

İşlevler (Fonksiyonlar)…

İşlevlerin Aşırı Yüklenmesi

- Aynı isme sahip fonksiyonun aldığı parametrelere göre farklı şekilde çalışmasıdır.

```
float aylikMaas(float mesaiSaati, float saatUcreti)
```

İşlevin imzası

```
float aylikMaas(float mesai, float saatUcreti);
float aylikMaas(float maas);
float aylikMaas();
```

İşlevler (Fonksiyonlar)…

➢ **ÖRNEK:** ⇒ [3]_ciz3.cpp

```
void ciz(char, int);
void ciz();

int main()
{

    ciz();                                  //fonksiyon çağırma
    cout << "Veri Tipi       Aralik" << endl;
    ciz('-', 50);                           //fonksiyon çağırma
    cout << "char         -128 to 127" << endl
        << "short       -32,768 to 32,767" << endl
        << "int         Sisteme bağlı" << endl
        << "long        -2,147,483,648 to 2,147,483,647" << endl;
    ciz('-', 50);

    system("pause");

    return 0;
}
```

```
void ciz(char ch, int n)
{
    for (int j = 0; j < n; j++)
        cout << ch;
    cout << endl;
}

void ciz()
{
    for (int j = 0; j < 45; j++)
        cout << '*';
    cout << endl;
}
```

İşlevler (Fonksiyonlar)…

- **İşlevlere Parametre Olarak Yapıların Aktarımı**
  - ÖRNEK: ⇒ [4]_parametre_yapi.cpp

```cpp
#include <iostream>

using namespace std;

struct olcu {
    int mt;
    int cm;
};

void olcuGoster(olcu);

int main()
{
    olcu d1, d2;

    cout << "uzunluk(metre) giriniz: ";
    cin >> d1.mt;
    cout << "uzunluk(cmetre) giriniz:: ";
    cin >> d1.cm;

    olcuGoster(d1);

    system("pause");

    return 0;
}
```

```cpp
void olcuGoster(olcu a)
{
    cout << a.mt << " m, " << a.cm << " cm \n";
}
```

İşlevler (Fonksiyonlar)…

İşlevlerden Değer Döndürmek

- Bir fonksiyon çalışmasını tamamladıktan sonra kendisini çağıran programa tek bir değer döndürebilir. Genellikle döndürülen bu değer, fonksiyonun çözdüğü problemin cevabını içerir.
- ÖRNEK: ⇒ [5]_değer_dondurme1.cpp

```
#include <iostream>

using namespace std;

float cevir(float);

int main()
{
    float libre, kilo;

    cout << "Kilonuzu Giriniz (kg) ";
    cin >> kilo;

    libre = cevir(kilo);
    cout << "Kilonuz (lb) " << libre << endl;

    system("pause");

    return 0;
}
```

```
float cevir(float kg)
{
    float lb = kg / 0.453592f;
    return lb;
}
```

İşlevler (Fonksiyonlar)…

- **İşlevlerden Return Kullanmadan Değer Döndürmek**
  - ÖRNEK: ⇒ [6]_değer_dondurme2.cpp

```
#include <iostream>

using namespace std;

void degistir(int, int);
int x = 10, y = 5;
int main()
{
    cout << "x = " << x << " y = " << y << endl;
    degistir(x, y);
    cout << "x = " << x << " y = " << y << endl;

    system("pause");

    return 0;
}

void degistir(int ilk, int ikinci)
{
    x = ikinci;
    y = ilk;
}
```

```
x = 10 y = 5
x = 5 y = 10
Press any key to continue . . .
```

İşlevler (Fonksiyonlar)…

- **Değişken Kapsamları**
  - ÖRNEK: ⇒ [7]_kapsam.cpp

```
int x = 4, y = 5;               // Global Değişkenler
float c;
char ch;

int f1(int a, int b)
{
    int x = 3;                  // f1'in yerel x değişkeni
    int y;
}

void f2()
{
    int x = 4;                  // f2'nin yerel x değişkeni
    static int y = 2;           // f2'nin statik değişkeni
}

int main()
{

    int x = 5;                  // main fonksiyonunun yerel x değişkeni
    y = 10;

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

İşlevler (Fonksiyonlar)…

- **Değişken Kapsamları…**
  - ÖRNEK: ⇒ [8]_static.cpp
    - **static** değişken kullanımı

```
unsigned int sayi;
float ort;

cout << "Sayı.....:";
cin >> sayi;

while (sayi != 0)
{
    cout << "Ortalama" << ortaAl(sayi) << endl;
    cout << "Sayı.....:";
    cin >> sayi;
}
```

```
float ortaAl(int sayi)
{
    static float toplam = 0;
    static int say = 0;

    say++;
    toplam += sayi;

    return toplam / say;
}
```

Aynı örneği, **static** ifadelerini kaldırarak çalıştırınız.

İşlevler (Fonksiyonlar)…

- **Inline Fonksiyonlar**
  - Program hızlanır, fakat boyutu artar.
  - Normal fonksiyon tanımının başına inline ifadesi yazılması yeterli.
  - Özellikle sık kullanılan ve küçük fonksiyonlar için tercih edilir.

```
inline void ciz(char ch, int n) // inline fonksiyon tanımlayıcısı
{                               // fonksiyon gövdesi
    for (int i = 0; i < n; i++)
        cout << ch;
    cout << endl;
}
```

İşlevler (Fonksiyonlar)…

- **Özyinelemeli (Recursive) Fonksiyonlar**
  - Fonksiyonun kendi kendini çağırması ile döngüsel yapı kurulur
  - Seri hesaplamalar, döngüsel hesaplama zorluklarında kullanılır.
  - Özyinelemeli yöntemler problemi daha gerçekçi ifade ettiği için tercih edilir.
  - Aynı problemler döngü kullanılarak (iteratif) da çözülebilir.
  - Özyinelemeli çağrılar zaman alır ve ek bellek tüketimine neden olur. Bu nedenle performans durumunda özyinelemeli fonksiyon kullanımından kaçınılmalıdır.

```
int fonksiyon(int k)
{
    //...
    int a;
    if (k<0)
        return a;        // dönüş noktası
    else
        fonksiyon(k-1); // öz yineleme
}
```

İşlevler (Fonksiyonlar)…

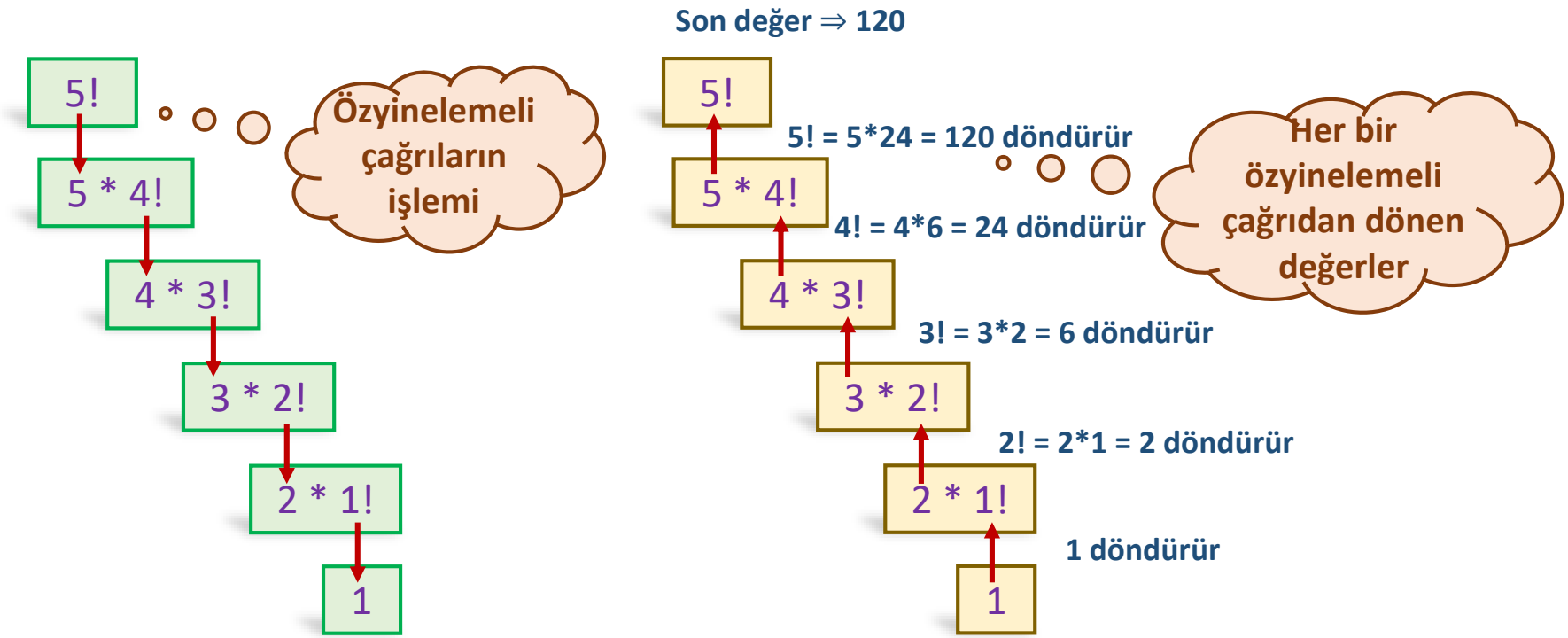
- **Özyinelemeli (Recursive) Fonksiyonlar…**
  - ÖRNEK: ⇒ [9]_foktoriyel_rekursif.cpp
    - Faktöriyel hesabı ⟹ **n!=n*(n-1)!**

```
unsigned int rec_faktoriyel(unsigned int n)
{
    if (n == 1)
        return 1;
    else
        return rec_faktoriyel(n - 1)*n;
}
```

```
unsigned int rec_faktoriyel(unsigned int);

int main()
{
    // Sonuç ekranında Türkçe karakterleri kullanabilmek için
    setlocale(LC_ALL, "Turkish");

    unsigned int sayi;

    cout << "Sayı........:";
    cin >> sayi;

    cout << "[REKÜRSİF]...." << sayi << "! = " << rec_faktoriyel(sayi) << endl;

    system("pause");

    return 0;
}
```

İşlevler (Fonksiyonlar)…

- **Özyinelemeli (Recursive) Fonksiyonlar…**
  - ÖRNEK: ⇒ [6]_foktoriyel_rekursif.cpp
    - Faktöriyel hesabı ⟹ **n!=n*(n-1)!**



Çalışma Soruları

- Parametre olarak aldığı tam sayının ikinin kuvveti olup olmadığını bulan ve geriye boolean olarak sonucu döndüren ***bool tam(unsigned int)*** fonksiyonunu yazınız.
- Giriş parametresi olarak aldığı gercel ve sanal değerlerini kullanarak karmaşık sayının kutupsal koordinatlarını bulan ve ekrana yazdıran fonksiyonu yazınız.
- Kendisine gönderilen sayının asal olup olmadığını kontrol ederek geriye bool bir değer döndüren fonksiyonu tanımlayınız.
- Aldığı ortalama not bilgisini harfe dönüştürerek geriye döndüren ***harfeDonustur*** fonksiyonunu tanımlayınız.
- İkinci dereceden bir denklemin a, b ve c katsayılarını giriş parametresi olarak alan ve bu denklemin köklerini ekrana yazan fonksiyonu tanımlayınız.
- Giriş parametresi olarak aldığı sayının mutlak değerini döndüren fonksiyonu yazınız.

KAYNAKLAR

- Deitel, C++ How To Program, Prentice Hall
- Horstmann, C., Budd,T., Big C++, Jhon Wiley&Sons, Inc.
- Robert Lafore, Object Oriented Programming in C++, Macmillan Computer Publishing
- Prof. Dr. Celal ÇEKEN, Programlamaya Giriş Ders Notları
- Prof. Dr. Cemil ÖZ, Programlamaya Giriş Ders Notları