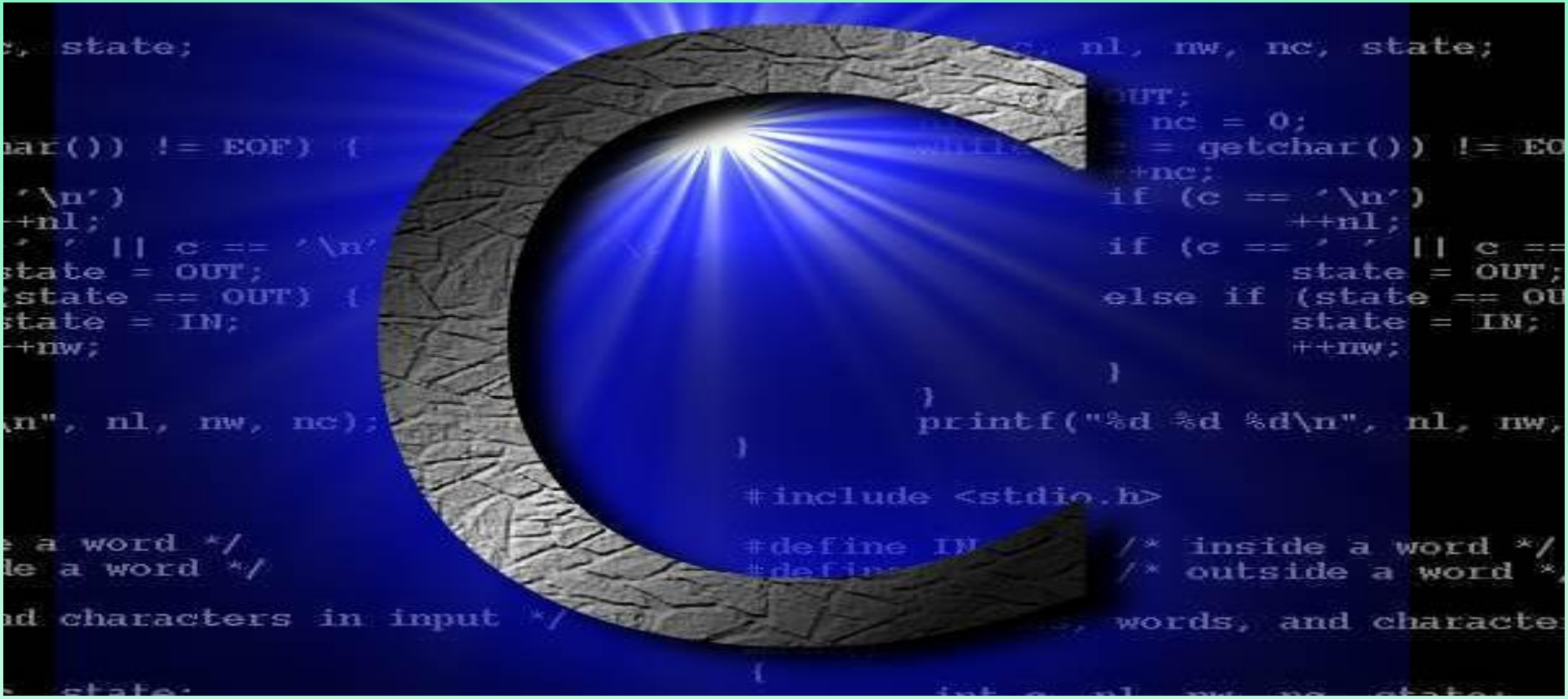


C PROGRAMLAMA
VE
UYGULAMALARI

Ders ile İlgili Bazı Bilgiler

%40 Vize

(Vize için ek 20 puana kadar ödevler)

%60 Final

(Final için ek 20 puana kadar ödevler)

Devamsızlık: 4 Hafta

BİLGİSAYAR

- **hardware** – Bilgisayarın fiziksel parçaları
  - Printer, Monitör, Klavye, anakart vs
- **software** – Bilgisayar tarafından kullanılan programlar
  - Word, Excel, Turbo Pascal vs

DONANIM

- Bir bilgisayar sisteminde 4 önemli bölüm bulunur:
  1. **Input devices**

     Bilgisayara bilgi göndermek için kullanılan araçlardır.
     - Klavye, mouse vs.
  2. **Output devices**

     Bilgisayardan kullanıcıya bilgi iletimi için kullanılır.
     - Monitör, printer vs

DONANIM

3. CPU

- CPU (Central Processing Unit – Merkezi İşletim Ünitesi veya işlemci)
  - bilgisayarın beyni olarak bilinir,
    - aritmetiksel ve mantıksal işlemleri yapan aygıtır.
      - Aritmetiksel işlemler : toplama, çıkarma, çarpma, bölme
      - mantıksal işlemler : karşılaştırma büyüklük, küçüklük, eşitlik tespiti gibi işlemlerdir. C
      - PU aynı zamanda bilgisayar birimlerin çalışması ve bu birimler arasındaki veri akışını da kontrol eder.
- CPU bir programdaki talimatları alıp program tarafından istenen işlemleri gerçekleştiren araçtır.
  - Tipik bir CPU komutu 0 ve 1 leri sayı olarak algılar
    - hafıza birimi 17deki sayıyı al, 19 daki hafıza biriminden aldığın sayıya ekle, sonucu 55teki hafıza birimine yaz" gibi düşünülebilir.

DONANIM

- CPU 3 ana parçadan oluşur: CU, ALU, ve registers
  - **CU (Kontrol Birimi)**
    - CPU da komutların gerçekleştirilme sırasını kontrol eden birimdir.
  - **ALU (Arithmetic Logic Unit - Aritmetik Lojik Birim)**
    - Bir bilgisayarda genel anlamda aritmetik ve mantıksal işlemlerin yapıldığı birime ALU adı verilir.
  - **Registers**
    - Saklayıcılar modern işlemci tasarımının merkezidir.
    - İşlemci üzerinde bulunan ve hesaplamalarda geçici depo görevi gören yüksek hızlı hafıza birimleridir.

DONANIM

4. Memory: 2 tip hafıza bulunmaktadır: Birincil hafıza ve ikincil hafıza

- **Birincil hafıza (Primary Memory)**
  - Birincil hafıza bilgisayarın rastgele erişilebilir belleğidir (**RAM -Random Access Memory**).
  - En temel fonksiyonu işlemcinin (Merkezi işlem birimi – CPU) program çalıştırırken geçici olarak verileri sakladığı ve sırası geldikçe bu verileri kullandığı alan olmasıdır.
  - Bir işlemci (CPU) çok basit toplama çıkarma seviyesinde işlemler yaparak programları çalıştırır. Komplike bir programın ise bu nevi basit işlemlere indirgenmesi mümkün olsa da bu indirgenme sonucunda çok sayıda işlemin yapılması gerekir. CPU anlık olarak bu işlemlerden sadece birisini yapabilir. Geri kalan işlemler ise sırasını beklemek ve bu sırada bir yerde saklanmak zorundadır.
  - RAM bilgisayarda çalışan her programın CPU'da çalışmak için bekletildiği ve o ana kadar çalışması sonucunda biriken verilerinin saklandığı hafıza ünitesidir.

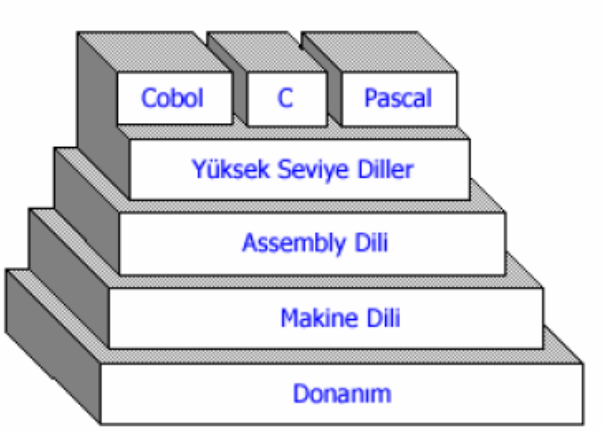
Resim 1.1: RAM bellek

Programlama

- **Her şeyden önce herkes bir programlama dilini öğrenebilir.**
  - Bilgisayar programlama yüksek bir zekâ ve matematik bilgisi gerektirmez. Sadece asla vazgeçmeme sabrı ve öğrenme isteği yeterlidir.
  - Programlama bir hünerdir. Bazı insanlar doğal olarak diğerlerinden daha iyidir, ama herkes pratik yaparak iyi olabilir.
- Başaramamaktan korkmak yerine, kendinizi bu maharete vererek, öğrenmek için uğraşın. Programlama eğlencelidir, fakat yanlış çalışma yöntemleriyle sinir bozucu da olabilir ve zamanınızın boşa geçmesine neden olabilir.

Programlama

- Programlama dili: İnsan-makina ve makina- makina arasındaki iletişimi sağlar.
- Programlama dilleri kullanım yapısına göre genel olarak üç seviyede incelenir.
  - Düşük seviyeli diller (makina ve assembly dilleri)
  - Orta seviyeli diller (C/C++ programlama dili)
  - Yüksek seviyeli diller (Basic, Fortan, Pascal. vb.)



Dillerin genel görünümleri

Problem

- **Problem Nedir?**
  - Bir işlemin, otomasyonun ya da bilimsel hesaplamanın bilgisayarla çözülmesi fikrinin ortaya çıkmasına problem denir.
  - Bu tip fikirlerde insanların bu sorunları beyinle çözmeleri ya imkansızdır ya da çok zor ve zaman alıcıdır.
  - Bu tip bir sorunu bilgisayarla çözebilme fikrinin ortaya çıkması bir bilgisayar probleminin ortaya çıkmasına neden olmuştur.
  - Bazen de bir işletme veya yönetimin otomasyonunu sağlamak amacı ile bu tip problemler tanımlanır.

Problem Çözümü

- Problemi Çözebilmek için öncelikle sorunun çok net olarak programcı tarafından anlaşılmış olması gerekir.
- Tüm ihtiyaçlar ve istekler belirlenmelidir. Gerekiyorsa bu işlem için birebir görüşmeler planlanmalı ve bu görüşmeler gerçekleştirilmelidir.
- Problemin Çözümüne ilişkin zihinsel alıştırmalar yapılır.
  - Bu alıştırmaların Bilgisayar çözümüne yakın olması hedeflenmelidir.
  - Bir sorunun tabii ki birden fazla çözümü olabilir.
    - Bu durumda bilgisayar ile en uygun çözüm seçilmelidir.
    - Çünkü bazen pratik çözümler bilgisayarlar için uygun olmayabilir.
- Oluşturulan çözüm **Algoritma** dediğimiz adımlarla ifade edilmelidir.
- Bu algoritmanın daha anlaşılabilir olması için Akış Çizgesi oluşturulmalıdır.
- Uygun bir programlama dili seçilmeli ve oluşturulan algoritma ve akış şeması bu programlama dili aracılığı ile bilgisayar ortamına aktarılmalıdır.
- Oluşturulan program bir takım verilerle test edilmelidir.
  - Oluşabilecek sorunlar ilgili kısımlar tekrar gözden geçirilerek düzeltilir. Bu adımlar defalarca gerçekleştirilmek zorunda kalınabilir.

Program ve Programlama

- **Program Nedir?**
  - Problem Çözümü kısmında anlatılan adımlar uygulandıktan sonra ortaya çıkan ve sorunumuzu bilgisayar ortamında çözen ürüne Program denir.
  - Bazı durumlarda bu ürüne yazılım denebilir.
- **Programlama Nedir?**
  - Problem Çözümünde anlatılan adımların tümüne birden programlama denilebilir.
  - Çoğunlukla ***çok iyi tanımlanmış bir sorunun*** çözümüne dair adımlar ile çözümün oluşturulup bunun bir programlama dili ile bilgisayar ortamına aktarılması Programlama diye adlandırılabilir.

ALGORİTMA VE AKIŞ ŞEMALARI

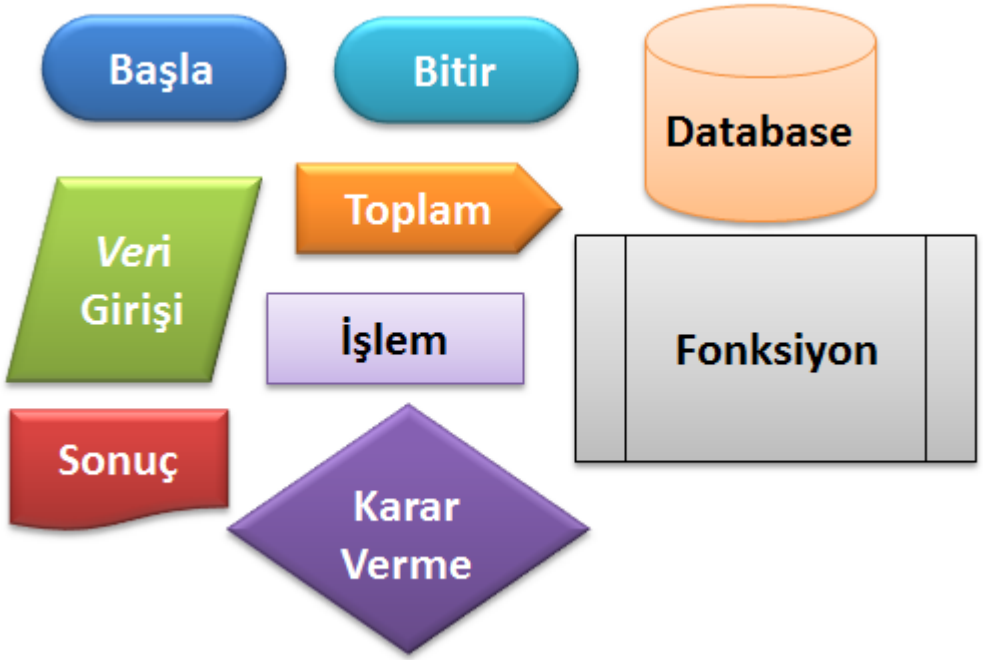
- Bilgisayarlar aptal makinelerdir.
  - Sadece ona yapmasını söylediğiniz komutları uygular
  - Bilgisayar kullanarak soru çözmek için sonuca giden yolun **tam olarak** belirlenmesi gerekir.
    - Bir sorunu çözerken, atması gereken her adımı yapması gereken her işi (görmek duymak dahil) söylemeniz gereken bir makine olarak düşünün
  - Aynı soru için değişik çözüm yolları geliştirilebilir.
  - Eğer bilgisayara verilen çözüm yanlışsa, çıkan sonuç yanlış çözüm doğru ise çıkan sonuç da doğrudur.

Algoritma

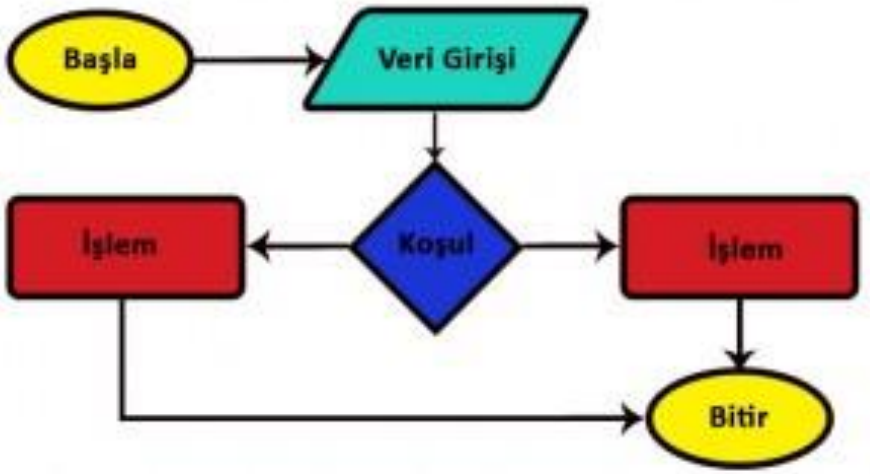
- Bir sorunu çözebilmek için gerekli olan sıralı mantıksal adımların tümüne **Algoritma** denir.
- Doğal dille yazılabileceği için fazlaca formal değildir.
- Bir algoritma için aşağıdaki ifadelerin mutlaka doğrulanması gereklidir.
  - Her adım *son derece belirleyici* olmalıdır. Hiç bir şey şansa bağlı olmamalıdır.
  - Belirli bir sayıda adım sonunda algoritma sonlanmalıdır.
  - Algoritmalar karşılaşılabilecek tüm ihtimalleri ele alabilecek kadar genel olmalıdır.

Algoritma en geniş anlamıyla, verilerden hareketle istenen sonucun nasıl alınacağını gösteren bir uygulama metodudur.

Algoritma; çözüm yolu aranan problemin herhangi bir adımında ne gibi işlemler yapılacağının tıpkı bir yemek tarifinde olduğu gibi yazıya dökülmüş şeklidir. Algoritmada her adımda giriş veya çıkış bilgileri, bölme, çarpma, aktarma, test ve benzeri işlemler yer almalıdır. Belli bir sayıda adımdan sonra algoritma mutlaka son bulmalıdır.



Algoritma anlatımda kullanılan cümlelerin kolay, basit ve benzer konular için de geçerli olmasına dikkat edilmelidir. Program geliştiricisi çözümü istenen probleme uygun algoritmayı kurabilmek için, çeşitli kaynaklardan faydalanabilir. Benzer programlar verimli bir kaynak olabilir. Gelişmiş ülkelerde, bilgisayar kitapları ve diğer süreli yayınlar, belirli problemlere ilişkin algoritma yayınlamaktadırlar. Algoritmayı paket program kütüphaneleri şeklinde bazı büyük pazarlardan da elde etmek mümkündür. Nihayet, çok fazla rafine olmuş bazı programlama dillerinde, program geliştirme aşamasında algoritmaya ihtiyaç duyulmaz. Zira, bu tür dillerde algoritma, programlama dilinin tanımı içerisinde yer almıştır.



Akış Çizgeleri

- Bir algoritmanın şekillerle görsel gösterimidir.
  - Algoritma ***doğal dille*** yazıldığı için herkes tarafından anlaşılamayabilir
  - Ancak akış çizgelerinde her bir şekil standart bir anlam taşıdığı için farklı yorumlanıp anlaşılamaması mümkün değildir.
  - Her bir ifade için ayrı bir sembol kullanılmaktadır.

İzlenecek Adımlar

- Problem iyice anlaşılır
  - Öncelikle algoritma kurulmalı
  - Problemin hangi dil kullanılarak çözüleceği belirlenir.
  - Belirlenen dilin sentaksına uygun şekilde kaynak kodları yazılır
- Kaynak kodlarını makine diline çevirecek olan derleyici kullanılır
- Çalıştırılabilir dosya oluşturulur
- Program çalıştırılır

- **Programlama Dili Nedir?**
  - Bir Problemin Algoritmik çözümünün Bilgisayara anlatılmasını sağlayan, son derece sıkı-sıkıya kuralları bulunan kurallar dizisidir.
- **Derleyici Nedir?**
  - Bir programlama dili ile bilgisayara aktarılan programın bilgisayarın anlayabileceği Makine Diline çevirmeyi sağlayan ve yazılan programda söz dizim hatalarının olup olmadığını bulan yazılımlardır.
  - *Her Programlama dili için bir derleyici olması gerekmektedir.*
  - PASCAL, C/C++, Delphi örnek olarak verilebilir.
- **Yorumlayıcı Nedir?**
  - Derleyici gibi çalışan ancak yazılmış programları o anda Makine diline çeviren yazılımlardır.
  - Bu tür bir yazılımda Programın Makine dili ile oluşturulmuş kısmı bilgisayarda tutulmaz. Programın her çalıştırılmasında her adım için Makine dili karşılıkları oluşturulur ve çalıştırılır.
  - ASP, JavaScript, MATLAB gibi programlar örnek olarak verilebilir.

ALGORİTMA VE AKIŞ ŞEMALARI

- Bilgisayar programı düzensel olarak tanımlanmış bir dizi komuttan oluşur.
- **Örnek bir Algoritma**
  - Örneğimiz bir insanın evden çıkıp işe giderken izleyeceği yolu ve işyerine girişinde ilk yapacaklarını tanımlamaktadır.
    - Evden dışarıya çık
    - Otobüs durağına yürü
    - Durakta gideceğin yöndeki otobüsü bekle
    - Otobüsün geldiğinde otobüse bin
    - Biletini bilet kumbarasına at
    - İneceğin yere yakınlaştığında arkaya yürü
    - İneceğini belirten ikaz lambasına bas
    - Otobüs durunca in
    - İşyerine doğru yürü
    - İş yeri giriş kapısından içeriye gir
    - Mesai arkadaşlarınla selamlaş
    - İş giysini giy
    - İşini yapmaya başla.

ALGORİTMA VE AKIŞ ŞEMALARI

- Başarılı bir programın algoritması için
  - Her adım *son derece belirleyici* olmalıdır. Hiç bir şey şansa bağlı olmamalıdır.
  - Belirli bir sayıda adım sonunda algoritma sonlanmalıdır. (Bir şekilde program sonlanabilmelidir)
  - Algoritmalar karşılaşılabilecek tüm ihtimalleri ele alabilecek kadar genel olmalıdır.
  - Algoritmada algoritmanın genel işleyişini etkileyebilecek hiç bir belirsizlik olmamalıdır.
  - Algoritmada bazı adımlar yer değiştirebilir . Ancak bir çok adımın kesinlikle yer değiştiremeyeceğini bilmeliyiz.
    - Yanlış sıradaki adımlar algoritmanın yanlış çalışmasına neden olacaktır.

ALGORİTMA VE AKIŞ ŞEMALARI

- Bilgisayara çözdürmek istediğiniz bir probleminiz var
  - Karşınızda bir makine olduğunu düşünün
  - İşlemi çözmek için çok basit adımlar belirleyin.
    - Hiçbir adımı atlamayın

ALGORİTMA VE AKIŞ ŞEMALARI

- Örnek: Bilgisayara verilecek iki sayıyı toplayıp sonucu ekrana yazacak bir program için algoritma geliştirmek istiyoruz

    1. BAŞLA
    2. A sayısını oku
    3. B sayısını oku
    4. TOPLAM=A + B işlemini yap
    5. TOPLAM değerini ekrana yaz
    6. SON

ALGORİTMA VE AKIŞ ŞEMALARI

- Örnek: Klavyeden girilecek iki sayıdan büyük olanından küçük olanını çıkarıp sonucu ekrana yazacak program için bir algoritma geliştiriniz.

1. BAŞLA
2. A sayısını oku
3. B sayısını oku
4. Eğer A büyüktür B SONUC=A-B
   Değilse SONUC=B-A
5. SONUC değerini ekrana yaz
6. SON

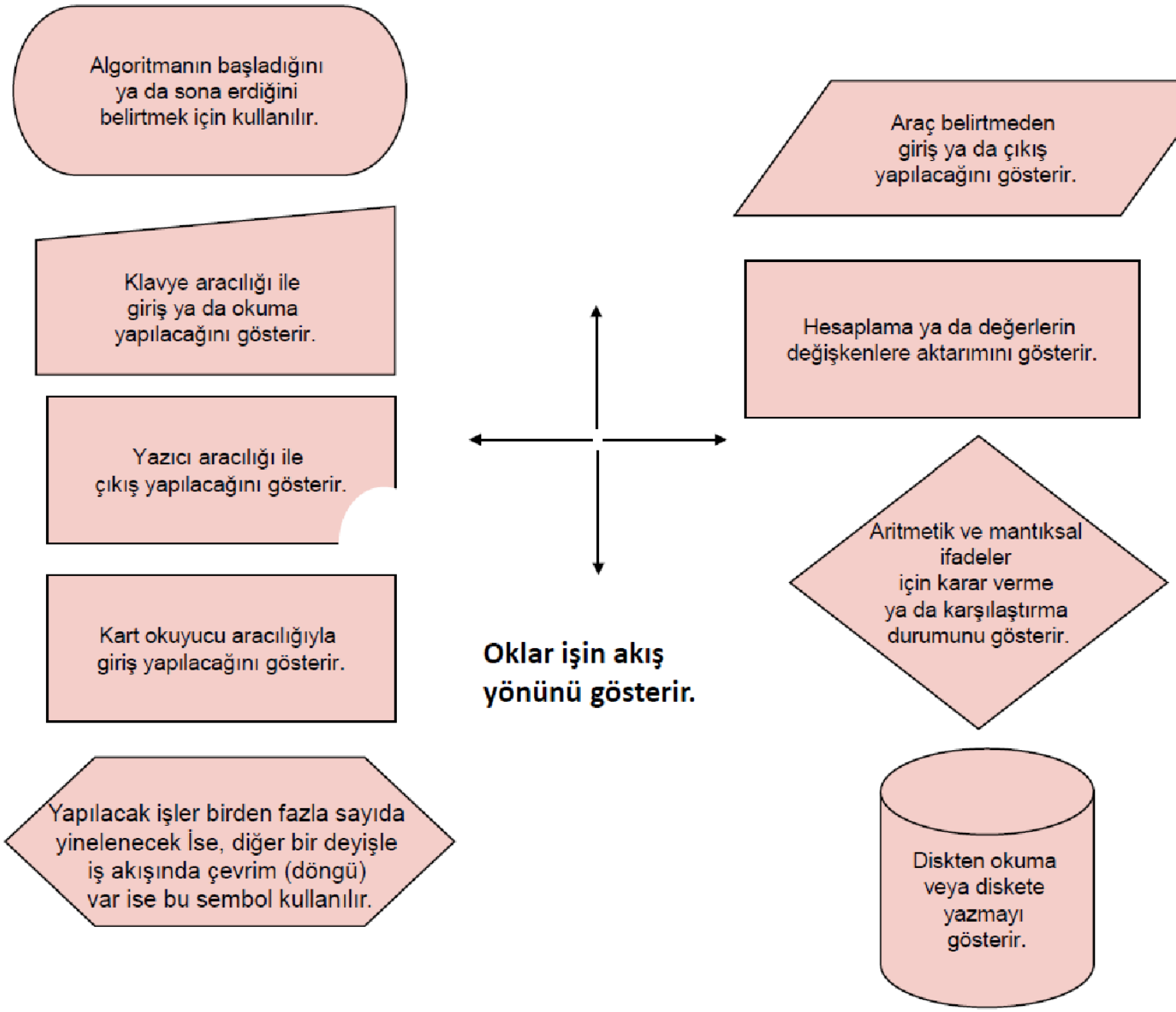
ALGORİTMA VE AKIŞ ŞEMALARI

- Örnek: 1den klavyeden girilen bir n değerine kadar sayıları toplayan ve sonucu ekrana yazan bir algoritmayı geliştirelim.

1. BAŞLA
2. N OKU
3. T=0
4. X=1
5. T=T+X
6. X=X+1
7. EĞER X<=N İSE 5. ADIMA GİT
8. T YAZ

ALGORİTMA VE AKIŞ ŞEMALARI

- **Akış Çizgeleri**
  - Bir algoritmanın şekillerle görsel ifadesidir.
    - Algoritma ***doğal dille*** yazıldığı için herkes tarafından anlaşılamayabilir
    - Ancak akış çizgelerinde her bir şekil standart bir anlam taşıdığı için farklı yorumlanıp anlaşılamaması mümkün değildir.
    - Her bir ifade için ayrı bir sembol kullanılmaktadır.

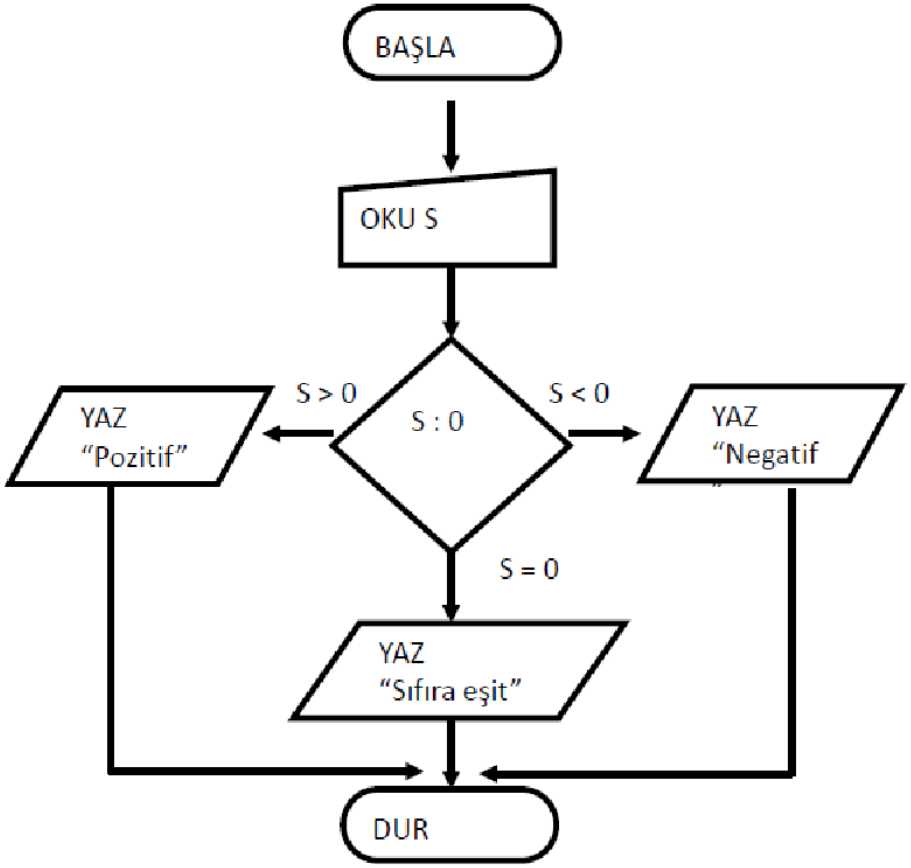

Algoritmanın başladığını ya da sona erdiğini belirtmek için kullanılır.
Araç belirtmeden giriş ya da çıkış yapılacağını gösterir.
Klavye aracılığı ile giriş ya da okuma yapılacağını gösterir.
Hesaplama ya da değerlerin değişkenlere aktarımını gösterir.
Yazıcı aracılığı ile çıkış yapılacağını gösterir.
Aritmetik ve mantıksal ifadeler için karar verme ya da karşılaştırma durumunu gösterir.
Kart okuyucu aracılığıyla giriş yapılacağını gösterir.
Oklar işin akış yönünü gösterir.
Yapılacak işler birden fazla sayıda yinelenecek İse, diğer bir deyişle iş akışında çevrim (döngü) var ise bu sembol kullanılır.
Diskten okuma veya diskete yazmayı gösterir.

ALGORİTMA VE AKIŞ ŞEMALARI

- Klavyeden girilen bir sayının pozitif, negatif veya sıfıra eşit olma durumunu hesaplayıp yazdıran algoritma ve akış şemasını hazırlayalım

ALGORİTMA

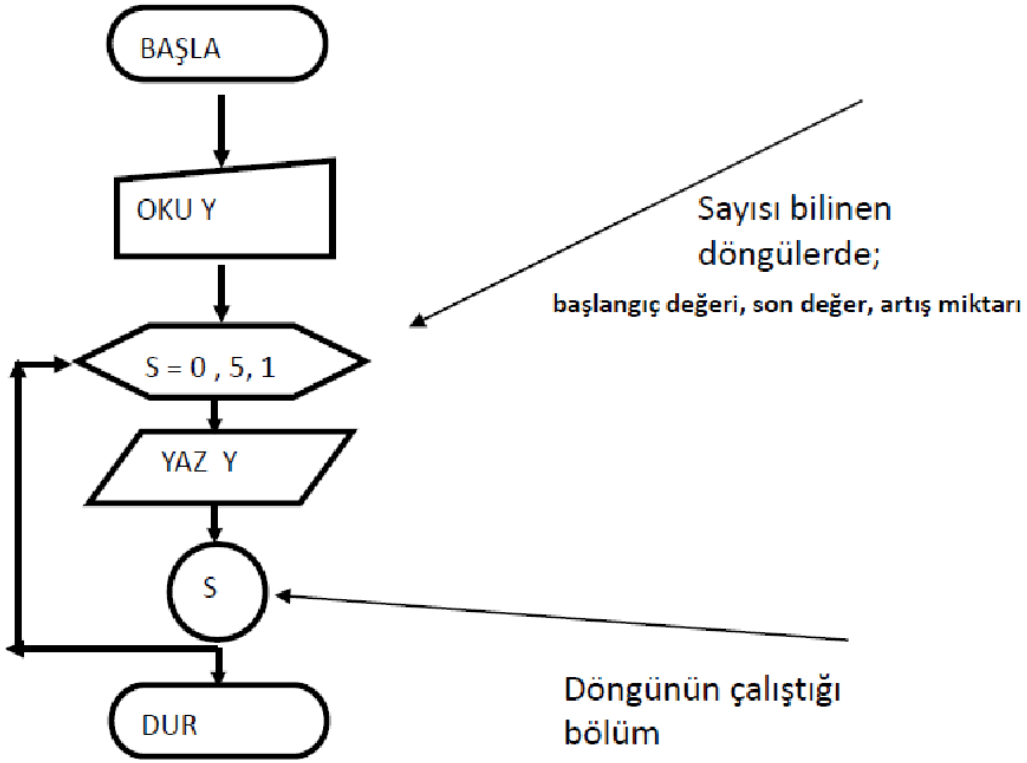
1 : Başla
2 : Oku S
3 : Eğer S > 0 ise “Pozitif” yaz,
4 : Eğer S < 0 ise “Negatif” yaz,
5 : Eğer S = 0 ise “Sıfıra eşit” yaz,
6 : Dur



ALGORİTMA VE AKIŞ ŞEMALARI

- Klavyeden girilen bir yazıyı 5 kez yazdıran algoritma ve akış şemasını oluşturunuz.

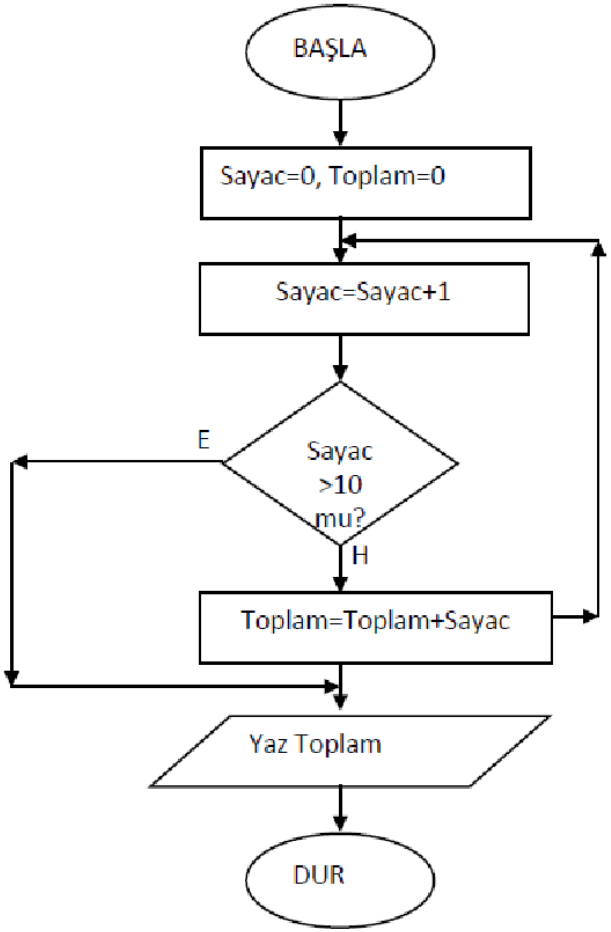
(Y : Yazı, S : Sayaç)
1 : Başla
2 : Oku Y
3 : Yaz Y
4 : S = S + 1
5 : Eğer S < 5 ise A3 e git
6 : Dur



ALGORİTMA VE AKIŞ ŞEMALARI

- 1-10 arasındaki tamsayıların toplamını bulan programın algoritma ve akış şemasını yazın?

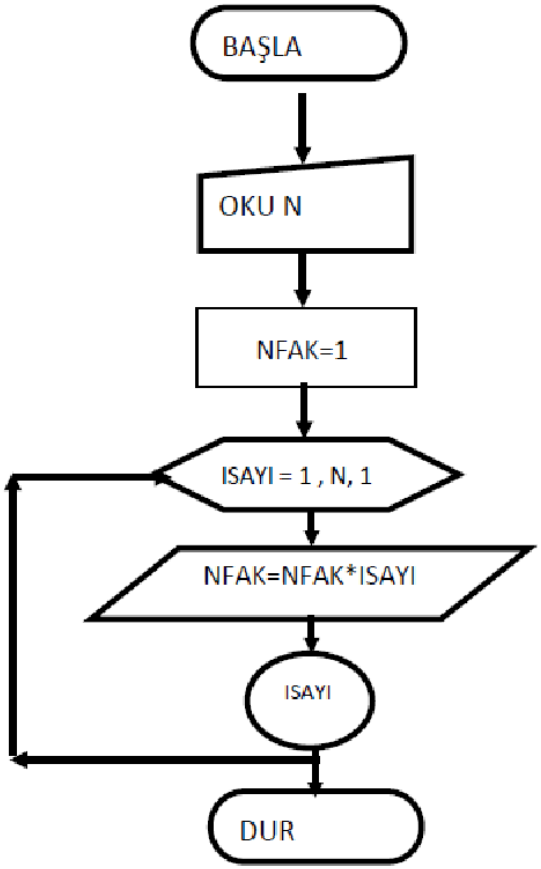
1.BAŞLA
2. Sayac=0, Toplam=0
3. Sayac=Sayac+1
4. EĞER Sayac>10 İSE GİT 7
5. Toplam=Toplam+Sayac
6. GİT 3
7. YAZ "1-10 Arası Sayıların Toplamı=",Toplam
8. DUR



ALGORİTMA VE AKIŞ ŞEMALARI

- N sayısını ekrandan okutarak faktöriyelini hesaplayan ve yazan programın algoritma ve akış şemasını yazın.

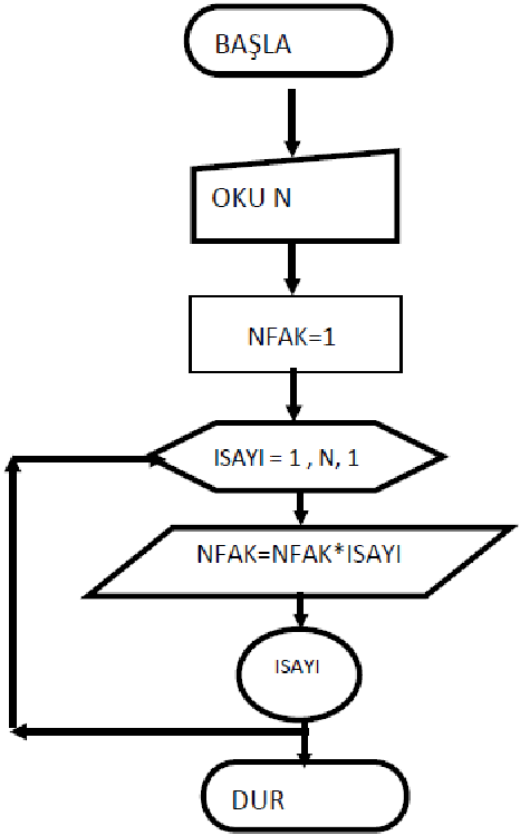
Adım 1-Başla
Adım 2-N'i klavyeden oku
Adım 3-NFAK=1
Adım 4-ISAYI=1
Adım 5-ISAYI=ISAYI+1
Adım 6-NFAK=NFAK*ISAYI
Adım 7-Eğer ISAYI<N ise Adım 5e git
Adım 8-NFAK yaz
Adım 9-Dur



ALGORİTMA VE AKIŞ ŞEMALARI

- N sayısını ekrandan okutarak faktöriyelini hesaplayan ve yazan programın algoritma ve akış şemasını yazın.

Adım 1-Başla
Adım 2-N'i klavyeden oku
Adım 3-NFAK=1
Adım 4-ISAYI=1
Adım 5-ISAYI=ISAYI+1
Adım 6-NFAK=NFAK*ISAYI
Adım 7-Eğer ISAYI<N ise Adım 5e git
Adım 8-NFAK yaz
Adım 9-Dur



Basit C Programları: Bir Metni Yazdırmak

```
/* C ile ilk program */
#include <stdio.h>
/* program çalışmaya main fonksiyonundan başlar */
int main()
{
printf("C programlama diline hosgeldiniz...\n");
return 0; /* programın başarı ile sonlandığını gösterir */

} /* main fonksiyonunun bitişi */
```

- Yorumlar
  - Derleyici `/* ve */` çevrili metinlere işlem yapmaz
  - Programın okunurluluğunu arttırmak için kullanılır.
- `#include <stdio.h>`
  - Önişlemci talimatı
    - Belirli bir dosyanın içeriğini bilgisayara yüklemesini söyler.
  - `<stdio.h>` standart giriş/çıkış işlemlerine izin verir.

Basit C Programları: Bir Metni Yazdırmak

- `int main()`
  - C programları bir yada daha fazla fonksiyon içerebilirler. Ama bunların içinden mutlaka bir tanesi `main` olmak zorundadır.
  - Parantezler fonksiyon olduğunu gösterir.
  - `int` tamsayı değerinde `main` fonksiyonun bir değer döndüreceği anlamına gelir.
  - Küme parantezi (`{` ve `}`) bir blok olduğunu gösterir.
    - Her fonksiyonun gövde kodları küme parantezleri içinde yazılır.

- `printf("C diline hosgeldiniz...\n" );`
  - Komutu bilgisayara bir iş yaptırır.
    - Yaptırdığı iş: tırnak içindeki (" ") karakterleri ekrana yazdırmaktır.
    - Bu satıra *ifade* (statement) denir.
    - Her ifade noktalı virgül (`;`) ile bitmelidir.
  - Ters bölü (\)
    - printf olağan dışı bir şeyler yapıyor.
    - `\n` yeni satır karakteri

Çıkış Sırası	Tanım
\n	Yeni satır. İmleci yeni satırın başına geçirir.
\t	Yatay sekme. İmleci bir sonraki sekme başlangıcına taşır.
\a	Alarm. Sistem zili sesi.
\\	Ters bölü. **printf** içinde ters bölü karakterini yazdırır.
\"	Çift tırnak. **printf** içinde tırnak karakterini yazdırır.

- `return 0;`
  - Fonksiyondan çıkış için bir yöntem
  - `return 0`, bu durumda, program normal olarak sonlandırıldı anlamını taşır.
- Sağ küme parantezi `}`
  - `main` fonksiyonun bitişini gösterir.
- Bağlayıcı
  - Fonksiyon çağrıldığı zaman, bağlayıcı fonksiyonu kütüphanede arar.
  - Uygun kütüphane fonksiyonlarının kodlarını programa yerleştirir.
  - Böylece makine diline çevrilmiş program tamamlanır.
  - Eğer fonksiyon ismi yanlış yazılmış ise, bağlayıcı kütüphanede o isimde fonksiyon bulamadığından hata üretir.

```
#include <stdio.h>
/* Program main çalıştırılarak başlar. */
int main()
{
printf( "C dili " );
printf( «programlama dersine" );
printf( "hos geldiniz…n" );
return 0;  /* programın başarı ile sonlandığını gösterir */
} /* main fonksiyonunun bitişi */
```

Basit C Programları : İki Tam Sayıyı Toplamak

- Önceki programlar gibi
  - Yorumlar, `#include <stdio.h>` ve `main`
- `int tamsayi1, tamsayi2, toplam;`
  - Değişken tanımı
    - Programın kullanabileceği bir değerin saklandığı hafıza konumlarıdır.
  - `int` değişkenlerin sadece tamsayı değerlerini saklamasını sağlar(-1, 3, 0, 47)
  - Değişken isimleri
    - `Tamsayi1,tamsayi2`, toplam
    - Tanıtıcılar: harf,rakam (rakam ile başlanılmaz) ve alt çizgi(_)
      - Büyük küçük harf duyarlılığı vardır.
  - Bildirimler çalıştırılabilir ifadelerden önce yazılmalıdır.
    - Örn: 12 satırdan sonra bildirimleri yapsaydık , programda yazım hatası olacaktı. Bu hataya derleyici hatası da denir.

Basit C Programları:İki Tam Sayıyı Toplamak

- `scanf( "%d", &tamsayi1 );`
  - Kullanıcıdan bir değer ister
    - `scanf` standart giriş (çoğunlukla klavye) kullanır.
  - `scanf` ifadesinin iki argümanı (bağımsız değişkeni) vardır.
    - %d – verinin tamsayı olması gerektiğini belirtir.
    - `&integer1` – değişken değerinin saklanacağı hafıza yerini belirtir.
    - & operatörü başlangıçta karışık gelebilir – şimdilik, `scanf` ifadelerinde değişkenle beraber kullanılması gerektiğini hatırlamanız yeterlidir.
  - Program çalışmaya başladığı zaman kullanıcı `scanf` ifadesine rakam yazarak cevap verir, ardından *enter* (geridönüş) tuşuna basarak sayıyı bilgisayara gönderir.

Basit C Programları : İki Tam Sayıyı Toplamak

- = (atama operatörü)
  - Bir degeri bir değişkene atama
  - İki operatör kullanılmış

```
toplam = degisken1 + degisken2;
Toplam degisken1 + degisken2 elde edilmiş;
```

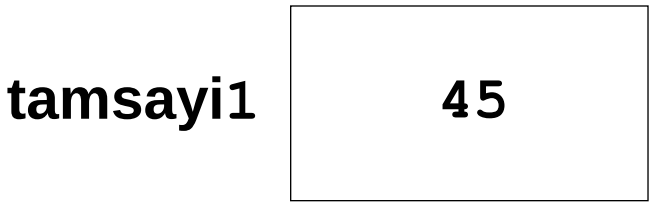
  - Değişken değerini eşitliğin sol tarafında alır.
- `printf( “Toplam %d\n", toplam );`
  - `scanf` benzer
    - %d bir tamsayının yazdırılacak anlamında kullanılır.
    - `toplam` hangi değerin yazılacağını belirler.
  - Hesaplamalar `printf` ifadesinin içindede oluşturulabilir.

```
printf( “Toplam %d\n", tamsayi1 + tamsayi2 );
```

```
/* Toplama programı */
#include <stdio.h>
/* Program main çalıştırılarak başlar. */
int main()
{
int tamsayi1; /* bildirim  */
int tamsayi2; /* bildirim */
int toplam;  /*  bildirim */
printf( "İlk tamsayıyı giriniz\n" );   /* mesaj yazdırma */
scanf( "%d", &tamsayi1 );              /* ilk tamsayının okunması */
printf( "İkinci tamsayıyı giriniz\n" ); /* mesaj yazdırma */
scanf( "%d", &tamsayi2 );              /* ikinci tamsayının okunması */
toplam = tamsayi1 + tamsayi2;          /* toplamın atanması */
printf( "Toplam %d dir\n", toplam );   /* toplamın yazdırılması */
return 0;
 } /* main fonksiyonun bitişi */
```

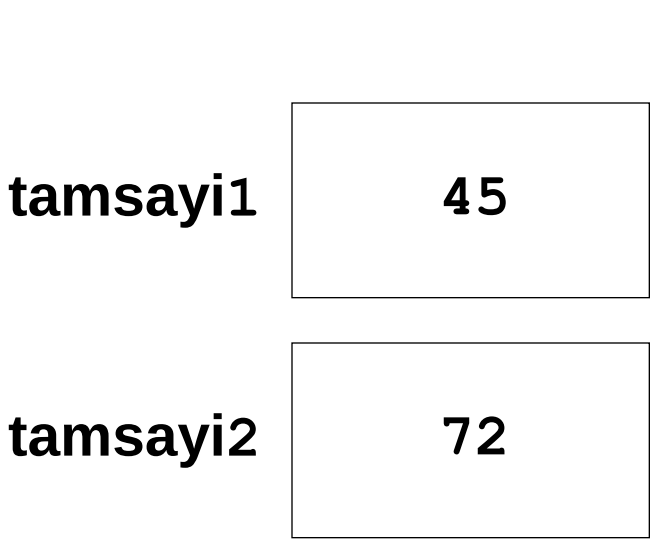
Hafıza Konuları

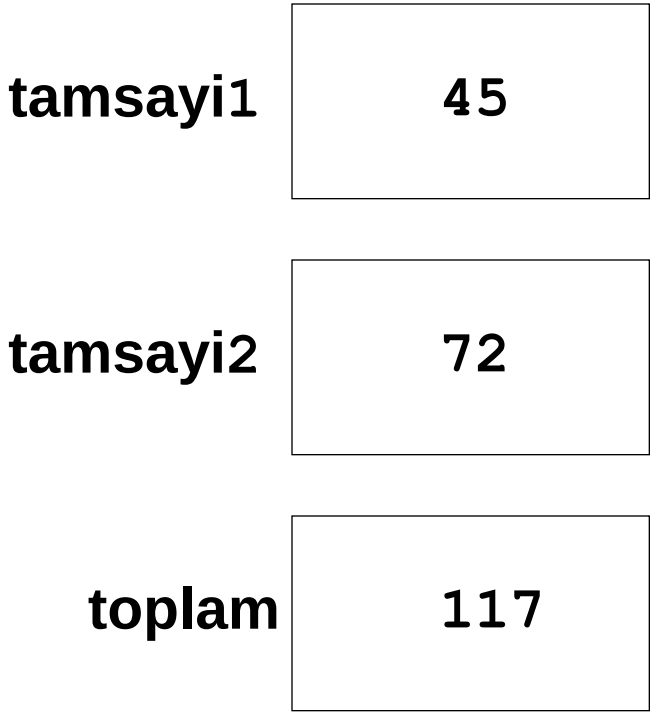
- Değişkenler
  - Değişkenler bilgisayar hafızasındaki yerlere karşılık gelen konumları belirtir.
  - Her değişkenin bir ismi, tipi, boyutu ve bir değeri vardır.
  - Bir değişkene yeni bir değer atandığı zaman (`scanf`, mesala), öncki değer silinir, yeni değer onun yerine yerleşir.
  - Hafızadan değişkenleri okumak ile değerleri değişmez.
- Şekilsel gösterimi



Hafıza Konuları

- Şekilsel gösterim (devam)





Aritmetik İşlemler

- Aritmetik hesaplamalar
  - * çarpma işlemi ve / bölme işlemi için kullanılır.
  - Tamsayı bölümü kalanı iptal ederek sonucu verir
    - `7 / 5` sonucunu `1` olarak verir
  - Mod operatörü (%) bölüm işleminde kalanı geri dönderir.
    - `7 % 5` sonucunu `2` olarak verir
- Operatör önceliği
  - Bazı aritmetik operatörler bazılarından önce işleme girerler. (i.e., çarpım toplamdan önce gelir)
    - İhtiyaç oldukça parantezler kullanılmalı
  - Örnek: `a`, `b` ve `c` değişkenlerinin ortalamasının bulunması
  - Bu şekilde kullanmayın: `a + b + c / 3`
    - Doğrusu: `(a + b + c ) / 3`

Aritmetik

- Aritmetik Operatörler:

C işlemi	Arittmetik operatör	Matematiksel deyim	C deyimi
toplama	+	$f + 7$	`f + 7`
çıkarma	-	$p - c$	`p - c`
çarpma	*	bm	`b * m`
bölme	/	x / y	`x / y`
Mod alma	%	$r \bmod s$	`r % s`

- Operatörlerin öncelik kuralları:

Operatör(ler)	İşlem(ler)	Öncelik sırası
`()`	Parentez	İlk önce hesaplanır. Eğer parantezler içice yazılmış ise, en içteki parantez ilk önce hesaplanır. Eğer bir satırda birden fazla parantez varsa (iç içe değillerse) bunlar soldan sağa doğru hesaplanır.
`*`, `/`, or `%`	Çarpım, bölüm, mod alma	İkinci olarak hesaplanır. Eğer birden fazla varsa soldan sağa doğru hesaplanır.
`+` or `-`	Toplama, çıkartma	En son hesaplanırlar. Eğer birden fazla varsa, soldan sağa doğru hesaplanırlar.

Karar Verme: Eşitlik ve Karşılaştırma Operatörleri

Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;` (Leftmost multiplication)

`2 * 5 is 10`

Step 2. `y = 10 * 5 + 3 * 5 + 7;` (Leftmost multiplication)

`10 * 5 is 50`

Step 3. `y = 50 + 3 * 5 + 7;` (Multiplication before addition)

`3 * 5 is 15`

Step 4. `y = 50 + 15 + 7;` (Leftmost addition)

`50 + 15 is 65`

Step 5. `y = 65 + 7;` (Last addition)

`65 + 7 is 72`

Step 6. `y = 72;` (Last operation—place 72 in y)

Karar Verme: Eşitlik ve Karşılaştırma Operatörleri

- Çalıştırabilir ifadeler
  - İşlem gerçekleştirir (hesaplamalar, verinin giriş/çıkış işlemleri)
  - Karar verilir
    - Sınav notuna göre “geçti” yada “kaldı” yazdırmak isteyebiliriz.
- `if` kontrol ifadesi
  - Bu bölümde basit versiyonu anlatılacak, detaylı anlatım ilerki bölümlerde ele alınacak.
  - Eğer koşul `doğru` ise, if yapısının gövde kısmı çalışır.
    - `0` `yanlış(false)`, 0 olmayan değerler ise (`doğru`) `true`
  - `If` yapısından sonraki ifade ile programın çalışması devam eder.
- Anahtar Kelimeler
  - C için ayrılmış özel kelimler
  - Tanımlayıcı yada değişken adları olarak kullanılmazlar.

Karar Verme: Eşitlik ve Karşılaştırma Operatörleri

Operatörler	C'deki karşılığı	C 'de örneği	C'de anlamı
Eşitlik Operatörleri			
=	==	x == y	x eşittir y
≠	!=	x != y	x eşit değildir y
Karşılaştırma Operatörleri			
>	>	x > y	x büyüktür y
<	<	x < y	x küçüktür y
>=	>=	x >= y	x büyüktür yada eşittir y
<=	<=	x <= y	x küçüktür yada eşittir y

```c
/* Şekil. 2.13: fig02_13.c
   if yapılarını,karşılaştırma ve eşitlik operatörlerini
   kullanmak */
#include <stdio.h>

/* Program main çalıştırılarak başlar. */
int main()
{
   int sayi1;
    int sayi2;

    printf( "İki tamsayı girin\n" );
    printf( "Bu iki sayının karşılaştırması yapılacaktır:" );

    scanf( "%d%d", &sayi1, &sayi2 ); /* iki tamsayiyi okuma */

    if ( sayi1 == sayi2 ) {
       printf( "%d eşittir %d\n", sayi1, sayi2 );
    } /*if bitti */

    if ( sayi1 != sayi2 ) {
       printf( "%d eşit değil %d\n", sayi1, sayi2 );
    } /*if bitti */

```

```c
   if ( sayi1 < sayi2 ) {
      printf( "%d küçük %d\n", sayi1, sayi2 );
   } /*if bitti */

   if (sayi1 > sayi2 ) {
      printf( "%d büyük %d\n", sayi1, sayi2 );
   } /*if bitti */

   if (sayi1 <= sayi2 ) {
      printf( "%d küçük yada eşit %d\n", sayi1, sayi2 );
   } /*if bitti */

   if (sayi1 >= sayi2 ) {
      printf( "%d büyük yada eşit %d\n", sayi1, sayi2 );
   } /*if bitti */

   return 0;   /* program başarı ile sonlandırıldı */

} /* main fonksiyonun bitişi */
```

```
İki tamsayı girin
Bu iki sayının karşılaştırması yapılacaktır :  3 7
3 eşit değildir 7
3 küçüktür 7
3 küçük yada eşit 7
```

```
İki tamsayı girin
Bu iki sayının karşılaştırması yapılacaktır 22 12
22 eşit değil 12
22 büyük 12
22 büyük yada eşit 12
```

```
İki tamsayı girin
Bu iki sayının karşılaştırması yapılacaktır 7  7
7 eşit 7
7 küçük yada eşit 7
7 büyük yada eşit 7
```

Karar Verme: Eşitlik ve Karşılaştırma Operatörleri

Operatörler				İşleyişleri
*	/	%		Soldan sağa
+	-			Soldan sağa
<	<=	>	>=	Soldan sağa
==	!=			Soldan sağa
=				Sağdan sola

Şu ana kadar anlatılan operatörlerin öncelikleri ve işleyişleri

Karar Verme: Eşitlik ve Karşılaştırma Operatörleri

Anahtar Kelimeler			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

C’nin anahtar kelimeleri.

Amaçlar

- Bu bölümde öğreneceksiniz:
  - `for` ve `do…while` döngü yapılarını kullanmak.
  - `switch` seçim yapısını kullanarak çoklu seçimler yapmak.
  - Program kontrolünde `break` ve `continue` kullanmak.
  - Mantık operatörlerini kullanmak.

Giriş

- Bu bölümde öğrenececkleriniz
  - Yeni döngü kontrol yapıları
    - `for`
    - `do...while`
  - `switch` çoklu seçim yapısı
  - `break` ifadesi
    - Belli kontrol yapılarından derhal ve hızlı bir biçimde çıkmak için kullanılır
  - `continue` ifadesi
    - Bir döngü gövdesinin geri kalan kısmını atlayarak döngünün diğer kısımlarınının çalışmasını sağlar

Döngülerin Temelleri

- Döngü
  - Döngü devam koşulları doğru (`true`) kaldığı sürece bilgisayar bir grup emri defalarca çalıştırır
- Sayıcı Kontrollü Döngü
  - Belirli döngü: döngünün kaç defa çalıştırılacağı önceden bilinir
  - Tekrarların sayısını saymak için kontrol değişkeni kullanılır
- Nöbetçi Kontrollü Döngü
  - Belirsiz döngü
  - Tekrar sayısının belli olmadığı durumlarda kullanılır
  - Nöbetçi değer veri girişinin sonlandığını belirtir

Artırma ve azaltma operatörleri

Operatör	Örnek deyim	Açıklama
++	++a	a' yı bir artır ve deyimi hesaplarken a'nın yeni değerini kullan.
++	a++	a'nın değerini deyimi hesaplarken kullan ve sonra a'nın değerini 1 artır.
--	--b	b' yı bir azalt ve deyimi hesaplarken b'nin yeni değerini kullan.
--	b--	b'nin değerini deyimi hesaplarken kullan ve sonra b'nin değerini 1 azalt.
Artırma ve Azaltma operatörleri		

Artırma ve azaltma operatörleri

- Artırma operatörü (++)
  - `c+=1` 'in yerine kullanılabilir
- Decrement operator (`--`)
  - `c-=1` 'in yerine kullanılabilir
- Ön artırma
  - Operatör değişkenden önce kullanılır (`++c` or `--c`)
  - Değişkenin değeri deyim hesaplanmadan önce değişir.
- Son artırma
  - Operatör değişkenden sonra kullanılır(`c++` or `c--`)
  - Değişkenin değeri deyim hesaplandıktan sonra değişir.

Artırma ve azaltma operatörleri

- Eğer `c` 'in değeri `5` ise,

  `printf( "%d", ++c );` ekrana `6` **yazar**

  `printf( "%d", c++ );` **ekrana** `5` **yazar**

  – Her iki durumda da, `c` nin son değeri `6` dır.

- Eğer değişken bir deyimin içinde değilse

  – Ön artırma ve son artırma aynı sonucu verir.

  ```
  ++c;
  printf( “%d”, c );
  ```

  Yukarıdaki ile aşağıdaki program parçası aynı etkiyi yapar.

  ```
  c++;
  printf( “%d”, c );
  ```

```c
/* Fig. 3.13: fig03_13.c
   Ön artırma ve son artırma*/
#include <stdio.h>


int main()
{
   int c;


   c = 5;
   printf( "%d\n", c );
   printf( "%d\n", c++ ); /* önartırma */
   printf( "%d\n\n", c );


   c = 5;
   printf( "%d\n", c );
   printf( "%d\n", ++c ); /* son artırma*/
   printf( "%d\n", c );

   return 0;

}
```

5
5
6

5
6
6

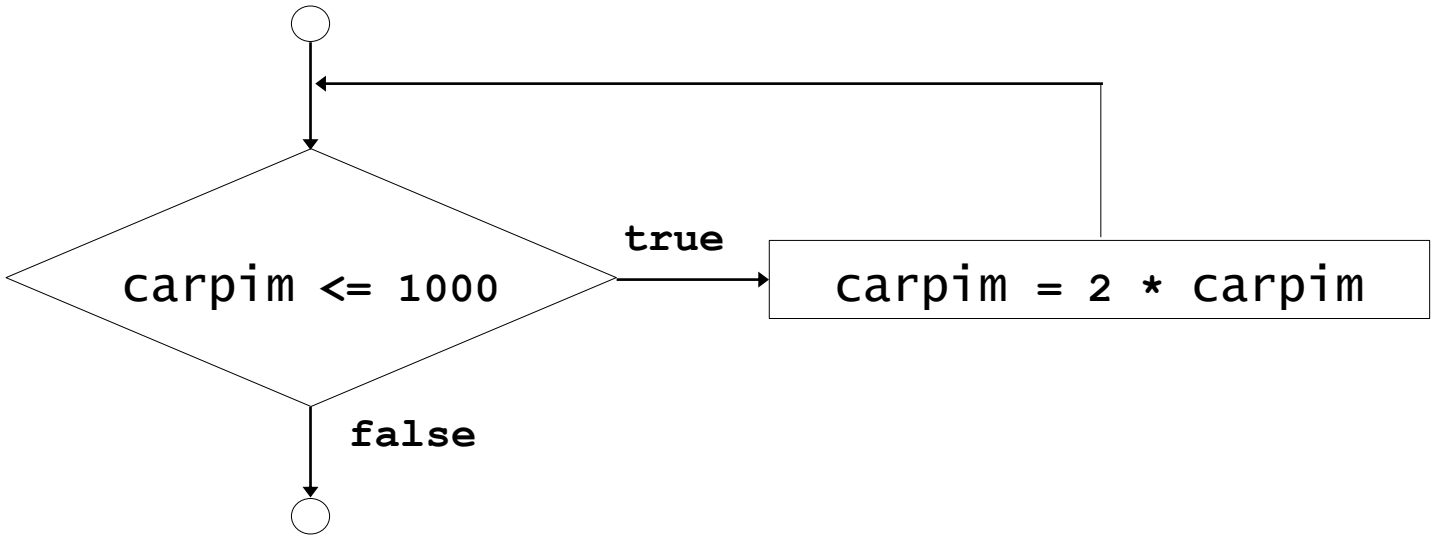
`while` tekrar deyimi

- Tekrar yapısı
  - While daki koşul doğru olduğu sürece bir grup işlemi tekrarlayan yapıdır.
  - `while` daki koşul yanlış olana kadar işlemler tekrar eder.

while tekrar deyimi

- Örnek:

```
int carpim = 2;
while ( carpim <= 1000 )
       carpim = 2 * carpim;
```

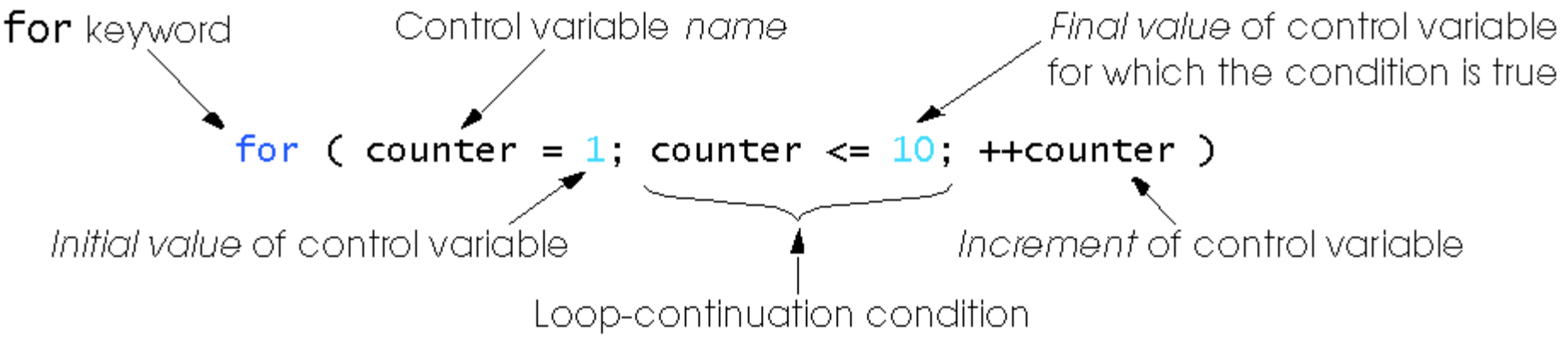


```c
/* Fig. 3.6: fig03_06.c
   /* Sayac kontrollü döngü ile sınıf ortalamısının bulunması */
#include <stdio.h>


int main()
{
   int sayac;
   int not;
    int toplam;
    int ortalama;

    /* ilk değerlerin verilmesi */
    toplam = 0;
    sayac = 1;

    /* işlem bölümü */
    while ( sayac <= 10 ) {
       printf( "Notu girin: " );
       scanf( "%d", &not );
       toplam = toplam + not;
       sayac = sayac + 1;
    }

```

```
25    /* Bitiş bölümü */
26    ortalama = toplam / 10;
27
28
29    printf( "Sınıf ortalaması %d dir.\n", ortalama );
30
31    return 0;
32
33 } /* program başarılı bir şekilde bitti */
```

```
Notu girin: 98
Notu girin : 76
Notu girin : 71
Notu girin : 87
Notu girin : 83
Notu girin : 90
Notu girin : 57
Notu girin : 79
Notu girin : 82
Notu girin : 94
Sınıf ortalaması 81 dir
```

for Döngü Yapısı



for Döngü Yapısı

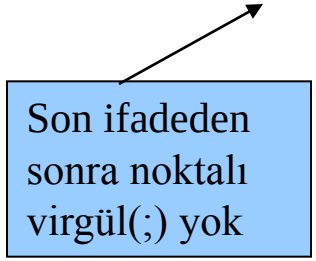
- `for` döngüsünün biçimi

  `for` (*kontol değişkenine ilk değeri atama*; *döngü devam koşulu*; *artırım*)
  ifade

- Örnek:

```
for(sayici = 1; sayici <= 10; sayici++ )
{
   printf( "%d\n", sayici );,
}
```

  – 1 den 10 kadar olan tamsayıları ekrana basar



```c
/* Fig. 4.5: fig04_05.c
   for ile toplama */
#include <stdio.h>

int main()
{
   int toplam = 0, sayi;

   for ( sayi = 2; sayi <= 100; sayi += 2 ) {
      toplam += sayi;       }

   printf( "Toplam %d\n", toplam );

   return 0;
}
```

```
Sum is 2550
```

Data Tipleri

Data Type	Range	Bytes	Format
signed char	-128 to +127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf

```
#include <stdio.h>
int main()
{
   int n, count;
   unsigned long long int factorial=1;
   printf("Enter an integer: ");
   scanf("%d",&n);
   if ( n< 0)
       printf("Error!!! Factorial of negative number doesn't exist.");
   else
   {
     for(count=1;count<=n;++count)    /* for loop terminates if count>n */
     {
       factorial*=count;              /* factorial=factorial*count */
     }
   printf("Factorial = %lu",factorial);
   }
   return 0;
}
```

Output 1

```
Enter an integer: -5
Error!!! Factorial of negative number doesn't exist.
```

Output 2

```
Enter an integer: 10
Factorial = 3628800
```

switch Çoklu Seçim Yapısı

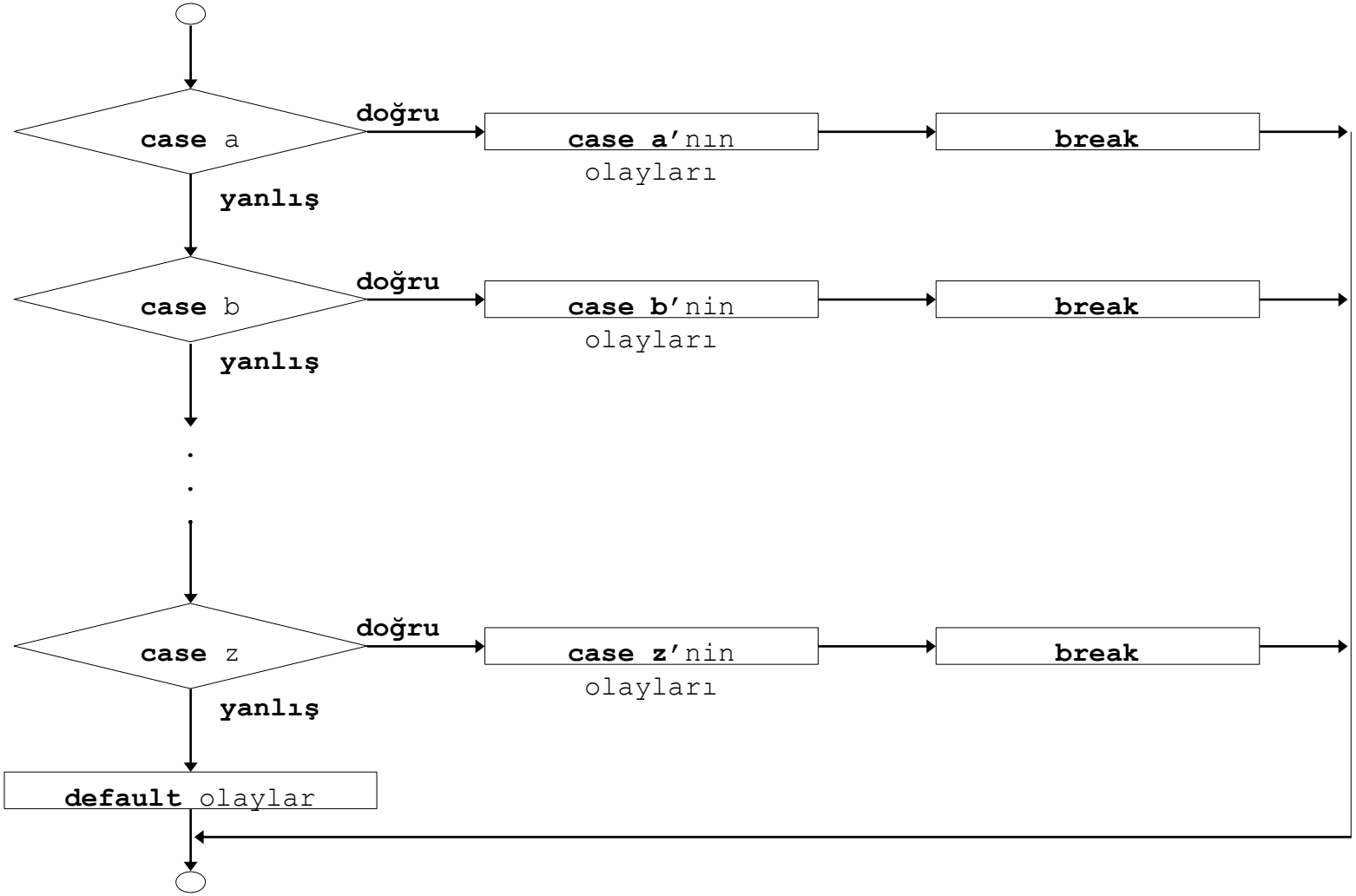
- `switch`
  - Bir değişken yada ifadenin ayrı ayrı sabitlerle karşılaştırılması ve buna bağlı olarak farklı işlemlerin yapılması gerektiren durumlarda kullanılır
- Biçim
  - Bir seri `case` yapısı ve isteğe bağlı `default` yapısı kısımlarından oluşur

```
switch ( değer ){
    case '1':
        işlemler
    case '2':
        işlemler
    default:
        işlemler
    }
```

  - `break;` ifadeden çıkar

switch Çoklu Seçim Yapısı

- `Switch` yapısının akış grafiği



```
/* C program to demonstrate the working of switch...case statement */
/* C Program to create a simple calculator for addition, subtraction,
   multiplication and division */

# include <stdio.h>
int main() {
    char o;
    float num1,num2;
    printf("Select an operator either + or - or * or / \n");
    scanf("%c",&o);
    printf("Enter two operands: ");
    scanf("%f%f",&num1,&num2);
    switch(o) {
        case '+':
            printf("%.1f + %.1f = %.1f",num1, num2, num1+num2);
            break;
        case '-':
            printf("%.1f - %.1f = %.1f",num1, num2, num1-num2);
            break;
        case '*':
            printf("%.1f * %.1f = %.1f",num1, num2, num1*num2);
            break;
        case '/':
            printf("%.1f / %.1f = %.1f",num1, num2, num1/num2);
            break;
        default:
            /* If operator is other than +, -, * or /, error message is shown */
            printf("Error! operator is not correct");
            break;
    }
    return 0;
}
```

```
#include <stdio.h>
int main()
{
  int n,i;
  printf("Enter a positive integer: ");
  scanf("%d",&n);
  printf("Factors of %d are: ", n);
  for(i=1;i<=n;++i)
  {
      if(n%i==0)
         printf("%d ",i);
  }
  return 0;
}
```

```
Enter a positive integer: 60
Factors of 60 are: 1 2 3 4 5 6 12 15 20 30 60
```

```
/*C program to demonstrate the working of while loop*/
#include <stdio.h>
    int main(){
    int number,factorial;
    printf("Enter a number.\n");
    scanf("%d",&number);
    factorial=1;
    while (number>0){      /* while loop continues util test condition
number>0 is true */
          factorial=factorial*number;
          --number;
}
printf("Factorial=%d",factorial);
return 0;
}

Enter a number.
5
Factorial=120
```

```
/* Üç basamaklı, basamaklarının küpleri toplamı kendisine eşit olan tam
   sayılara  Armstrong sayı denir. Örneğin: 371 = 3^3 + 7^3 + 1^3.
   Bu program İç-içe geçmiş 3 döngü ile bütün Aramstrong sayıları bulur. */

#include <stdio.h>

int main()
{
   int a,b,c, kup, sayi, k=1;

   for(a=1; a<=9; a++)
   for(b=0; b<=9; b++)
   for(c=0; c<=9; c++)
   {
      sayi = 100*a + 10*b + c;       /* sayi = abc (üç basamaklı) */
      kup  = a*a*a + b*b*b + c*c*c;  /* kup  = a^3+b^3+c^3        */

      if( sayi==kup )
            printf("%d. %d\n",k++,sayi);
   }
return 0;
}
```

```
#include <stdio.h>

main()
{
  int n, c;

  printf("Enter a number\n");
  scanf("%d", &n);

  if ( n == 2 )
    printf("Prime number.\n");
  else
  {
    for ( c = 2 ; c <= n - 1 ; c++ )
    {
      if ( n % c == 0 )
        break;
    }
    if ( c != n )
      printf("Not prime.\n");
    else
      printf("Prime number.\n");
  }
  return 0;
}
```

```
#include <stdio.h>

main()
{
   int n, c;

   printf("Enter a number\n");
   scanf("%d", &n);

   if ( n == 2 )
      printf("Prime number.\n");
   else
   {
      for ( c = 2 ; c <= n - 1 ; c++ )
      {
         if ( n % c == 0 )
            break;
      }
      if ( c != n )
         printf("Not prime.\n");
      else
         printf("Prime number.\n");
   }
   return 0;
}
```

```
/* Girilen 3 tamsayinin bir ucgenin kenari olup olmadigi kontrol edilir */

#include<stdio.h>
int main( void )
{
 int a, b, c;
 int temp;

 printf("Birinci kenar uzunlugu> ");
 scanf("%d", &a);
 printf("Ikinci kenar uzunlugu> ");
 scanf("%d", &b);
 printf("Ucuncu kenar uzunlugu> ");
 scanf("%d", &c);

 /* a ile b den buyuk olan a ya kucuk olan b ye atanir */
 if(a < b) {
 temp = a;
 a = b;
 b = temp;
 }

 if( ((a + b) < c) || ((a - b) > c) )
 printf("Bu kenar uzunluklarina sahip bir ucgen olamaz.\n");
 else
 printf("Bu kenar uzunluklarina sahip bir ucgen cizilebilir.\n");

 return 0;
}
```

Ahmet, bir romanın her gün bir önceki gün okuduğu sayfadan 5 sayfa fazlasını okumaktadır. İlk gün 10 sayfa okuyarak başlayan Ahmet'in 1.000 sayfalık bir romanı kaç günde bitireceğini bulan programı C dilinde kodlayınız.

```
#include <stdio.h>
main()
{
 int sayfa=10, okunansayfa=0, gun = 0;
 while(okunansayfa < 1000) {
 okunansayfa += sayfa;
 sayfa += 5;
 gun ++;
 }
 printf("%d. gun icinde kitap bitmis olur", gun);
return 0;
}
```

```
#include <stdio.h>

int main()
{
 int X;
 printf ("sayi gir");
 scanf ("%d",&X);
 if (X%2==0)
 printf ("çift");
 else
 printf("tek");
return 0;
}
```

```
#include "stdio.h"
#include "conio.h"

int main()
{
int SS=0,EBS=0,i,N=5,S;
for(i=1;i< =N;i++) // 5 Tane sayı girilmesi için döngü
başlatıldı. { printf("%d. Sayiyi giriniz.",i); scanf("%d",&S);
if(S>EBS) //EBS Değeri ile S(Sayiyi karşılaştırıyor.)
{
EBS=S; // Eğer S büyükse EBS den yeni EBS Sayi oluyor.
SS=i; // Sayinin konumu SS atanıyor.
}
}
printf("%d sayisi %d. Sirada girilmistir.",EBS,SS);
}
```

```c
#include "stdio.h"

int main()
{
int N=5,T=0,i,S;
for(i=1;i< =N;i++) // 5 Tane sayı girilmesi için döngü
başlatıldı.
{
printf("%d. sayiyi giriniz.",i);
scanf("%d",&S);
if(S%2==1) // Sayinin tek olup olmadığı denetlendi.
{
T=T+1; // sayi tek ise T(Sayaç) 1 artırıldı.
}
}
printf("%d tane Tek Sayi vardir.",T);
}
```

```
#include <stdio.h>
int main()
{
  int n, reverse=0, rem;
  printf("Enter an integer: ");
  scanf("%d", &n);
  while(n!=0)
  {
     rem=n%10;
     reverse=reverse*10+rem;
     n/=10;
  }
  printf("Reversed Number = %d",reverse);
  return 0;
}
```

#define Önişlemci Komutu

#define komutu sayesinde, PI'nin

aslında 3.14 olduğu derleyici (compiler) tarafından kabul edilmiştir.

printf yerine, ekrana_yazdir; scanf yerine de, deger_al isimlerini kullandık.

```
/* Yarıçapa göre daire alanı hesaplar */
#include<stdio.h>
#define PI 3.14
#define ekrana_yazdir printf
#define deger_al scanf
int main( )
{
int yaricap;
float alan;
ekrana_yazdir( "Çemberin yarı çapını giriniz>
" );
deger_al( "%d", &yaricap );
alan = PI * yaricap * yaricap;
ekrana_yazdir( "Çember alanı: %.2f\n", alan );
return 0;
}
```

#undef #ifdef ve #ifndef Önişlemci Komutları

```
#include<stdio.h>
#define PI 3.14
int main( )
{
// Tanımlı PI değeri, tanımsız hâle getiriliyor.
#undef PI
int yaricap;
float alan;
printf( "Çemberin yarı çapını giriniz> " );
scanf( "%d", &yaricap );
// PI değerinin tanımlı olup olmadığı kontrol
ediliyor.
#ifdef PI
//PI tanımlıysa, daire alanı hesaplanıyor.
alan = PI * yaricap * yaricap;
printf( "Çember alanı: %.2f\n", alan );
#else
//PI değeri tanımsızsa, HATA mesajı veriliyor.
printf("HATA: Alan değeri tanımlı değildir.\n");
#endif
return 0;
}
```

#undef:

#define komutuyla tanımladığımız şeyleri, iptal etmek isteriz

#define ile tanıtılmışsa, şöyle yapılsın; yok tanıtılmadıysa, böyle olsun gibi farklı durumlarda ne

olacağını belirten yapılar:

#ifdef (*if defined - şayet tanımlandıysa*)

#ifndef (*if not defined - şayet tanımlanmadıysa*) operatörleri kullanılır.

#if, #else, #endif, #elif Önişlemci Komutları

```
/* Daire alanını hesaplar */
#include<stdio.h>
#define HASSASLIK_DERECESI 2
int main( ){
int yaricap;
float alan;
printf( "Çemberin yarı çapını giriniz> " );
scanf( "%d", &yaricap );
#if ( HASSASLIK_DERECESI == 0 )
alan = 3 * yaricap * yaricap;
#elif ( HASSASLIK_DERECESI == 1 )
alan = 3.1 * yaricap * yaricap;
#elif ( HASSASLIK_DERECESI == 2 )
alan = 3.14 * yaricap * yaricap;
#else
alan = -1;
#endif
printf( "Çember alanı: %.2f\n", alan );
return 0;}
```

Bazen bir değerin tanımlanıp, tanımlanmadığını bilmek yetmez. Yani eğer doğruysa, böyle yapılması lâzım, aksi hâlde böyle olacak gibi...

Hassaslık derecesi, pi sayısının virgülden kaç basamak sonrasının hesaba katılacağını belirtir. Eğer hassaslık derecesi bunlara uymuyorsa, alan değeri -1 yapılır.

Fonksiyonlar

- C gibi prosedürel dillerin önemli konularından birisi fonksiyonlardır. Java veya C# gibi dillerde metot (method) ismini alırlar. Adı n'olursa olsun, görevi aynıdır. Bir işlemi birden çok yaptığınızı düşünün. Her seferinde aynı işlemi yapan kodu yazmak oldukça zahmetli olurdu. Fonksiyonlar, bu soruna yönelik yaratılmıştır. Sadece bir kereye mahsus yapılacak işlem tanımlanır. Ardından dilediğiniz kadar, bu fonksiyonu çağırırsınız. Üstelik fonksiyonların yararı bununla da sınırlı değildir.

- Fonksiyonlar, modülerlik sağlar. Sayının asallığını test eden bir fonksiyon yazıp, bunun yanlış olduğunu farkederseniz, bütün programı değiştirmeniz gerekmez. Yanlış fonksiyonu düzeltirsiniz ve artık programınız doğru çalışacaktır. Üstelik yazdığınız fonksiyonlara ait kodu, başka programlara taşımanız oldukça basittir.

Fonksiyonlar

- Fonksiyonlar, çalışmayı kolaylaştırır. Diskten veri okuyup, işleyen; ardından kullanıcıya gösterilmek üzere sonuçları grafik hâline dönüştüren; ve işlem sonucunu diske yazan bir programı baştan aşağı yazarsanız, okuması çok güç olur. Yorum koyarak kodun anlaşılabilirliğini, artırabilirsiniz. Ancak yine de yeterli değildir. İzlenecek en iyi yöntem, programı fonksiyon parçalarına bölmektir. Örneğin, diskten okuma işlemini *disten_oku( )* isimli bir fonksiyon yaparken; grafik çizdirme işini *grafik_ciz( )* fonksiyonu ve diske yazdırma görevini de *diske_yaz( )* fonksiyonu yapabilir. Yarın öbür gün, yazdığınız kodu birileri incelediğinde, sadece ilgilendiği yapıya göz atarak, aradığını çok daha rahat bulabilir. Binlerce satır içinde çalışmaktansa, parçalara ayrılmış bir yapı herkesin işine gelecektir.

Fonksiyonlar

main( ) Fonksiyonu

- Şimdiye kadar yazdığımız bütün kodlarda, main( ) şeklinde bir notasyon kullandık. Bu kullandığımız ifade, aslında main( ) fonksiyonudur. C programlama dilinde, bir kodun çalışması main( ) fonksiyonun içeresinde olup olmamasına bağlıdır. Bir nevi başlangıç noktası olarak düşünebiliriz. Her programda sadece bir tane main( ) fonksiyonu bulunur. Başka fonksiyonların, kütüphanelerin, kod parçalarının çalıştırılması main( ) içersinde direkt veya dolaylı refere edilmesiyle alakalıdır.

Fonksiyonlar

Fonksiyonlarda ait genel yapı:

donus_tipi **fonksiyon_adi(** *alacagi_arguman[lar]*)
{
.
.
FONKSİYON İÇERİĞİ
(YAPILACAK İŞLEMLER)
.
.
[***return*** *deger*]
}

Fonksiyon Oluşturma Örneği

```
/* Ev sekli cizen program */
#include<stdio.h>
int catiyi_ciz(  ){
printf( "    /\\ \n" );
printf( "   /  \\ \n" );
printf( "  /    \\ \n" );
printf( " /      \\\n" );
printf( "----------\n" );
}
int kat_ciz(   ){
printf( "|  |\n" );
printf( "|  |\n" );
printf( "|  |\n" );
printf( "----------\n" );
}
int main(   ){
catiyi_ciz( );
kat_ciz( );
return 0; }
```

// Evin catisini cizen fonksiyon.

// Evin katini cizen fonksiyon.

// Programin calismasini saglayan
// ana fonksiyon.

Fonksiyon Oluşturma Örneği

```
#include<stdio.h>
int sonuc = 0;
int kare_hesapla(int sayi)
{   sonuc = sayi * sayi;  }
int kup_hesapla(int sayi)
{ sonuc = sayi * sayi * sayi;    }
int main(  ){
int a;
printf( "Sayi giriniz> ");
scanf( "%d",&a );
printf( "Girdiginiz sayi\t: %d\n", a );
kare_hesapla( a );
printf("Sayinin karesi\t: %d\n", sonuc );
kup_hesapla( a );
printf("Sayinin kupu\t: %d\n", sonuc );
return 0; }
```

Fonksiyon Oluşturma Örneği

```
/* Sayının tek veya çift olmasını kontrol eder. */
#include<stdio.h>
int tek_mi_cift_mi( int sayi )
{
if( sayi%2 == 0 )
printf( "%d, cift bir sayidir.\n", sayi );
else
printf( "%d, tek bir sayıdir.\n", sayi );
}
int main(  ) {
int girilen_sayi;
printf( "Lütfen bir sayi giriniz> " );
scanf( "%d",&girilen_sayi );
tek_mi_cift_mi( girilen_sayi );
return 0;  }
```

Fonksiyon Oluşturma Örneği

```
#include <stdio.h>
int mult ( int x, int y );
int main()
{
  int x;
  int y;
    printf( "Please input two numbers to be multiplied: " );
  scanf( "%d", &x );
  scanf( "%d", &y );
  printf( "The product of your two numbers is %d\n", mult( x, y ) );
}
int mult (int x, int y){
  return x * y;
}
```

Fonksiyon Oluşturma Örneği

```
#include<stdio.h>
int sum(int,int);
int main() {
int a,b,c;
printf("\nEnter the two numbers : ");
scanf("%d%d",&a,&b);
c = sum(a,b);
printf("\nAddition of two number is : %d",c);
}
int sum (int num1,int num2)
{
int num3;
num3 = num1 + num2 ;
return(num3); }
```

Fonksiyon Oluşturma Örneği

```
#include<stdio.h>
int bankamatik(int para)
{
  int a,yirmilik,onluk,beslik;
  if(para%5==0)
  {
      yirmilik = para/20;
      para -= yirmilik*20;
      onluk = para/10;
      para -= onluk*10;
      beslik = para/5;
      para -= beslik*5;
      printf("\nYirmilik = %d",yirmilik);
      printf("\nOnluk    = %d",onluk);
      printf("\nBeslik   = %d\n",beslik);
    }
    else
      printf("Girilen miktar 5 TL ve katlari olmali!\n");   }
int main()
{
   int miktar;
   printf("Cekilecek para miktari (TL) = ");
   scanf("%d",&miktar);
   bankamatik(miktar);  /* fonksiyon bir değişkene atanmamış ! */
  return 0;    }
```

```c
/* Fig. 5.4: fig05_04.c
   Finding the maximum of three integers */
#include <stdio.h>

int maximum( int x, int y, int z ); /* function prototype */

/* function main begins program execution */
int main()
{
   int number1; /* first integer */
   int number2; /* second integer */
   int number3; /* third integer */

   printf( "Enter three integers: " );
   scanf( "%d%d%d", &number1, &number2, &number3 );

   /* number1, number2 and number3 are arguments
      to the maximum function call */
   printf( "Maximum is: %d\n", maximum( number1, number2, number3 ) );

   return 0; /* indicates successful termination */

} /* end main */

```

```
25 /* Function maximum definition */
26 /* x, y and z are parameters */
27 int maximum( int x, int y, int z )
28 {
29    int max = x;      /* assume x is largest */
30
31    if ( y > max ) { /* if y is larger than max, assign y to max */
32       max = y;
33    } /* end if */
34
35    if ( z > max ) { /* if z is larger than max, assign z to max */
36       max = z;
37    } /* end if */
38
39    return max;       /* max is largest value */
40
41 } /* end function maximum */
```

```
Enter three integers: 22 85 17
Maximum is: 85
Enter three integers: 85 22 17
Maximum is: 85
Enter three integers: 22 17 85
Maximum is: 85
```

Öncü (Header) Dosyalar

- Öncü Dosyalar
  - Kütüphane fonksiyonları için fonksiyon prototipleri içerir
  - `<stdlib.h> , <math.h>`, vs...
  - Yükleme... `#include <filename>`

    `#include <math.h>`
- Öncü dosya düzenleme
  - Fonksiyonlarla dosya oluşturma
  - `filename.h olarak kayıt etme.`
  - `#include "filename.h“` ile başka dosyalara yükleme
  - Fonksiyonları tekrar kullanma.

Öncü Dosyalar

ANSI C kütüphanesi öncü dosyalar [değiştir | kaynağı değiştir]

<assert.h>	Mantıksal hataları bulmada yardımcı olmak için assert makrolarını içerir.
<complex.h>	Karmaşık sayılarla işlem yapmak için fonksyionlar içerir.
<ctype.h>	Contains functions used to classify characters by their types or to convert between upper and lower case in a way that is independent of the used character set (typically ASCII or one of its extensions, although implementations utilizing EBCDIC are also known).
<errno.h>	Hata kodlarını denemek için fonksiyonlar içerir.
<fenv.h>	Ondalıklı sayıların kontrolünü sağlayan fonksiyonlar tanımlar.
<float.h>	Contains defined constants specifying the implementation-specific properties of the floating-point library, such as the minimum difference between two different floating-point numbers (_EPSILON), the maximum number of digits of accuracy (_DIG) and the range of numbers which can be represented (_MIN, _MAX).
<inttypes.h>	For precise conversion between integer types. (New with C99)
<iso646.h>	For programming in ISO 646 variant character sets. (New with **NA1**)
<limits.h>	Contains defined constants specifying the implementation-specific properties of the integer types, such as the range of numbers which can be represented (_MIN, _MAX).
<locale.h>	For setlocale() and related constants. This is used to choose an appropriate locale.
<math.h>	Yaygın matematik fonksiyonlarını içerir.
<setjmp.h>	Declares the macros setjmp and longjmp, which are used for non-local exits
<signal.h>	Çeşitli istisnai durumların konrolünü için gerekir.
<stdarg.h>	For accessing a varying number of arguments passed to functions.
<stdbool.h>	For a boolean data type. (New with C99)
<stdint.h>	For defining various integer types. (New with C99)
<stddef.h>	Yeni bazı tipler ve makrolar yaratmak içindir.
<stdio.h>	Çekirdek girdi ve çıktı modülünü içerir. This file includes the venerable `printf` function.
<stdlib.h>	For performing a variety of operations, including conversion, pseudo-random numbers, memory allocation, process control, environment, signalling, searching, and sorting.
<string.h>	Dizelerle işlemler yapmak içindir.
<tgmath.h>	For type-generic mathematical functions. (New with C99)
<time.h>	Zaman ve tarih biçimlerini dönüştürmek içindir.
<wchar.h>	For manipulating wide streams and several kinds of strings using wide characters - key to supporting a range of languages. (New with **NA1**)
<wctype.h>	For classifying wide characters. (New with **NA1**)

Matematik Kütüphanesindeki Fonksiyonlar

- Math kütüphanesindeki fonksiyonlar
  - Yaygın olarak kullanılan matematiksel hesaplamalar
  - `#include <math.h>`
- Çağrılan Fonksiyonları biçimlendirme
  - `FonksiyonAdı( argüman );`
    - Eğer çoklu argüman varsa listeleri virgül kullanarak ayır
  - `printf( "%.2f", sqrt( 900.0 ) );`
    - Sqrt fonksiyonunu çağırır,girilen değerin karekök'ünü geriye döndürür
    - Bütün Matematik fonk. verileri "`double`"olarak geri döndürür.
  - Argümanlar sabit, değişken veya ifade olabilir

Math. Kütüphanesinin Fonksiyonları

Fonksiyonlar	Tanım	Örnek
`sqrt( x )`	x in kare kökü	`sqrt( 900.0 ) is 30.0` `sqrt( 9.0 ) is 3.0`
`exp( x )`	e^x ifadeli fonksiyonlar	`exp( 1.0 ) is 2.718282` `exp( 2.0 ) is 7.389056`
`log( x )`	x in doğal logaritması (*e tabanında*)	`log( 2.718282 ) is 1.0` `log( 7.389056 ) is 2.0`

`log10( x )`	*x* in logaritması (10 tabanında)	`log10( 1.0 ) is 0.0`
		`log10( 10.0 ) is 1.0`
		`log10( 100.0 ) is 2.0`

`fabs( x )`	absolute value of *x*	`fabs( 5.0 ) is 5.0`
		`fabs( 0.0 ) is 0.0`
		`fabs( -5.0 ) is 5.0`

`ceil( x )`	*x* i kendinden büyük en küçük tamsayıya yuvarlar	`ceil( 9.2 ) is 10.0` `ceil( -9.8 ) is -9.0`
`floor( x )`	*x* den küçük n büyük tam sayıya yuvarlar	`floor( 9.2 ) is 9.0` `floor( -9.8 ) is -10.0`
`pow( x, y )`	*x* in *y ninci* kuvvetini alır (x^y)	`pow( 2, 7 ) is 128.0` `pow( 9, .5 ) is 3.0`
`fmod( x, y )`	remainder of *x*/*y* as a floating point number	`fmod( 13.657, 2.333 ) is 1.992`
`sin( x )`	x in trigonometrik sinüsü (*x* radyan olarak)	`sin( 0.0 ) is 0.0`
`cos( x )`	x in trigonometrik kosinüsü (*x* radyan olarak)	`cos( 0.0 ) is 1.0`
`tan( x )`	x in trigonometric tanjantı (*x* in radians)	`tan( 0.0 ) is 0.0`

Fig. 5.2 Yaygın kullanılan matematik kütüphanesindeki fonk.

Math. Kütüphanesinin Fonksiyonları

The **pow** functions compute *x* raised to the power of *y*, i.e.

```
#include <math.h>
#include <stdio.h>
int main(void) {
  for(int i = 1; i < 5; i++)
  printf("pow(3.2, %d) = %lf\n", i, pow(3.2, i));
  return 0; }
```

Output:

```
pow(3.2, 1) = 3.200000
pow(3.2, 2) = 10.240000
pow(3.2, 3) = 32.768000
pow(3.2, 4) = 104.857600
```

Rastgele Sayılar Üretme

- `rand` fonksiyonu
  - yükle `<stdlib.h>`
  - 0 ve `RAND_MAX` (en az `32767`) arasında sayı "random" geri döndürme.

```
i = rand();
```

    - "random" sayıların önceden ayarlanmış dizisi
    - Her fonksiyon çağrılmasında aynı dizi tekrar eder.
- Derecelendirme (Scaling)
  - 1 ile n arasında random sayı alma

```
1 + ( rand() % n )
```

    - `rand() % n` 0 ile `n-1 arasında bir sayı döndürür`
    - `1` ile `n` arasında random sayıya 1 ekleme yapar.

```
1 + ( rand() % 6)
```

      - `1` and `6` arasında sayı

Rastgele Sayılar Üretme

- `srand` fonksiyonu (rassallaştırma)
  - `<stdlib.h>`
  - Farklı bir dizide rasgele sayılar oluşturulmasını sağlar
  - Unsigned(işaretsiz) tipte bir tamsayıyı argüman olarak kullanır

    `srand(` *seed* **`);`**
  - `srand( time( NULL ) );/*yükle <time.h> */`
    - `time( NULL )`
      - Program çalıştırıldığı esnadaki saniye cinsinde oluşturur
      - “(Randomizes) rasgele seçmeler" çekirdek

```c
/* Fig. 5.7: fig05_07.c
   Shifted, scaled integers produced by 1 + rand() % 6 */
#include <stdio.h>
#include <stdlib.h>

/* function main begins program execution */
int main()
{
   int i; /* counter */

   /* loop 20 times */
   for ( i = 1; i <= 20; i++ ) {

      /* pick random number from 1 to 6 and output it */
      printf( "%10d", 1 + ( rand() % 6 ) );

      /* if counter is divisible by 5, begin new line of output */
      if ( i % 5 == 0 ) {
         printf( "\n" );
      } /* end if */

   } /* end for */

   return 0; /* indicates successful termination */

} /* end main */
```

6	6	5	5	6
5	1	1	5	3
6	6	2	4	2
6	2	3	4	1

enum Deyimi (Enumeration Constants)

Bu tip, değişkenin alabileceği değerlerin belli (sabit) olduğu durumlarda programı daha okunabilir hale getirmek için kullanılır. Genel yazım biçimi:

enum tip_adı{değer_1, değer_2, ..., değer_n} değişken_adı;

tip_adı programcı tarafından verilen tip ismidir. değişken_adı ise program içinde kullanılacak olan değişkenin adıdır. Eğer kullanılmazsa program içinde daha sonra enum ile birlikte kullanılır. Örneğin:

enum bolumler{programcilik, donanim, muhasebe, motor};

tanımı ile derleyici programcilik için 0, donanim için 1, muhasebe için 2 ve motor için 3 değerini kabul ederek atamaları buna göre yapar. Değişken adı bildirilirse daha sonra enum kullanmaya gerek kalmaz. Örneğin:

enum renkler {kirmizi, mavi, sari} renk;

enum gunler {pazartesi, sali, carsamba, persembe, cuma, cumartesi, pazar};

gibi yapılan sabit tanımlamaları program içinde kullanılabilir:

```
enum bolumler bolum;
enum gunler gun;
bolum = muhasebe; /* bolum = 2 anlamında */
gun   = cuma;     /* gun   = 4 anlamında */
renk  = kirmizi;  /* renk  = 0 anlamında */
```

enum Deyimi (Enumeration Constants)

```
/* 15prg01.c: Klavyeden girilen bir sayının tek olup olmadığını sınar */

#include <stdio.h>

enum BOOLEAN{ FALSE, TRUE };  /* 0, 1 */

int tek(int n){ return (n % 2);  }

int main() {
    enum BOOLEAN sonuc;
    int x;

    printf("Bir sayi girin: ");
    scanf("%d",&x);

    sonuc = tek(x);   /* tek mi? */

    if( sonuc == TRUE )
        puts("Girilen sayi tek ");
    else
        puts("Girilen sayi cift");

  return 0; }
```

enum Deyimi (Enumeration Constants)

```
#include<stdio.h>
int main( void )
{
// Degisken tipinin nasil olacagini tanimliyoruz
enum ana_renkler {
Kirmizi,
Mavi,
Sari
};
// Degiskeni tanimliyoruz.
enum ana_renkler piksel;
// Degisken degerini Mavi olarak belirliyoruz.
// Dilersek Sari ve Kirmizi da girebiliriz.
piksel = Mavi;
// Degisken degeri karsilastiriliyor.
if( piksel == Kirmizi )
printf( "Kırmızı piksel\n" );
else if( piksel == Mavi )
printf( "Mavi piksel\n" );
else
printf( "Sarı piksel\n" );
return 0;
}
```

Örnek: Şans Oyunu

- Barbut(craps) similasyonu
- Kurallar
  - İki zar atılır
    - İlk atışta 7 veya 11 (toplam), oyuncu kazanır.
    - İlk atışta 2, 3, veya 12 (toplam), oyuncu kaybeder.
    - İlk atışta 4, 5, 6, 8, 9, 10 – oyuncunun puanı
  - Kazanmak için kendi puanını 7 gelmeden önce tekrar atması gerekir.

```c
/* Fig. 5.10: fig05_10.c
   Craps */
#include <stdio.h>
#include <stdlib.h>
#include <time.h> /* contains prototype for function time */

/* enumeration constants represent game status */
enum Status { CONTINUE, WON, LOST };

int rollDice( void ); /* function prototype */

/* function main begins program execution */
int main()
{
   int sum;         /* sum of rolled dice */
   int myPoint;     /* point earned */

   enum Status gameStatus; /* can contain CONTINUE, WON, or LOST */

   /* randomize random number generator using current time */
   srand( time( NULL ) );

   sum = rollDice(  ); /* first roll of the dice */

```

```
25    /* determine game status based on sum of dice */
26    switch( sum ) {
27
28       /* win on first roll */
29       case 7:
30       case 11:
31          gameStatus = WON;
32          break;
33
34       /* lose on first roll */
35       case 2:
36       case 3:
37       case 12:
38          gameStatus = LOST;
39          break;
40
41       /* remember point */
42       default:
43          gameStatus = CONTINUE;
44          myPoint = sum;
45          printf( "Point is %d\n", myPoint );
46          break; /* optional */
47    } /* end switch */
48
```

```
49    /* while game not complete */
50    while ( gameStatus == CONTINUE ) {
51       sum = rollDice(  ); /* roll dice again */
52
53       /* determine game status */
54       if ( sum == myPoint ) { /* win by making point */
55          gameStatus = WON;
56       } /* end if */
57       else {
58
59          if ( sum == 7 ) { /* lose by rolling 7 */
60             gameStatus = LOST;
61          } /* end if */
62
63       } /* end else */
64
65    } /* end while */
66
67    /* display won or lost message */
68    if ( gameStatus == WON ) {
69       printf( "Player wins\n" );
70    } /* end if */
71    else {
72       printf( "Player loses\n" );
73    } /* end else */
74
```

```
75    return 0; /* indicates successful termination */
76
77 } /* end main */
78
79 /* roll dice, calculate sum and display results */
80 int rollDice( void )
81 {
82    int die1;    /* first die */
83    int die2;    /* second die */
84    int workSum; /* sum of dice */
85
86    die1 = 1 + ( rand() % 6 ); /* pick random die1 value */
87    die2 = 1 + ( rand() % 6 ); /* pick random die2 value */
88    workSum = die1 + die2;     /* sum die1 and die2 */
89
90    /* display results of this roll */
91    printf( "Player rolled %d + %d = %d\n", die1, die2, workSum );
92
93    return workSum; /* return sum of dice */
94
95 } /* end function rollRice */
```

```
Player rolled 5 + 6 = 11
Player wins

Player rolled 4 + 1 = 5
Point is 5
Player rolled 6 + 2 = 8
Player rolled 2 + 1 = 3
Player rolled 3 + 2 = 5
Player wins

Player rolled 1 + 1 = 2
Player loses

Player rolled 1 + 4 = 5
Point is 5
Player rolled 3 + 4 = 7
Player loses
```