Algoritmalar ve Programlama I – BLM 103

Hafta 2: Akış Diyagramları

Fenerbahçe Üniversitesi

2. Hafta İçeriği

- Derleyici ve Yorumlayıcı
- Programlama Süreci
  - Problemin Tanımlanması
  - Program Tasarımı
  - Kodlama Süreci
  - Test ve Hata Ayıklama
- Sözdekod
- Akış Diyagramları

Bilgisayar Programları

- Bilgisayarlar tam olarak ne kodlarsanız o işlemi gerçekleştirirler
- Bir bilgisayar programında, bir işi gerçekleştirmek için ilgili komutlar bulunur.
- Günümüzde bir onlarca bilgisayar programlama dili mevcuttur.

Bilgisayar Programı Nedir?

- Farklı türlerde yazılımlar mevcuttur. Bunlara örnek olarak:
    - İşletim Sistemleri (Win 10, Linux, Ubuntu, …).
    - Derleyici ve Yorumlayıcı (Visual Studio, Dev C++, ...).
    - Uygulamalar (Ödeme Sistemleri, Muhasebe Yazılımları…).
    - Gömülü Sistem (Televizyon, Telefon).

Programcı Kimdir?

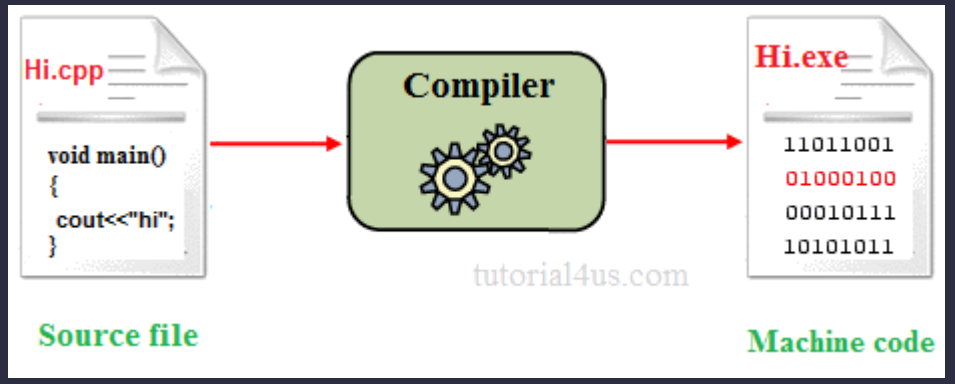
- Bir programcı, kendisine verilen genellikle doğal dilde olan bir problemi bilgisayarın anlayacağı dile dönüştürüp, yazan kişiye denir.

- Yazdığı programı derleyici ve yorumlayıcı araçları ile makinanın anlayacağı dile çevirir ve yürütür.

Derleyici (Compiler) ve Yorumlayıcı (Interpreter)

- **Derleyici, kendisine verilen insanın anlayabileceği kaynak kodu (source code), obje kodu (object code)'a dönüştüren bir yazılımdır. Obje kodu makinanın anlayabileceği yapıdadır.**

- Yorumlayıcı, kendine verilen kaynak kodu obje koduna dönüştürerek satır satır çalıştırırlar. Derleyiciden farkı ise, derleyici bir uygulamanın tamamını dönüştürürken, yorumlayıcılar ise kaynak kodun her satırını dönüştürür, çalıştırır ve bir sonraki satıra geçer.



Programın Yürütülmesi

Derleyici Kullanımı

Yorumlayıcı Kullanımı

İyi Tasarlanmış Programlar

- **<u>İyi tasarlanmış programlar:</u>**

  - Beklenen sonuçları üretmeli
  - Kolay anlaşılır olmalı
  - Bakım ve güncelleme kolay olmalı
  - Verilmli olmalı (Kaynak kullanımı)
  - Güvenli (Hata ve saldırılara karşı koruma mekanizmaları)
  - Esnek (Yeniden kullanılabilirlik)

  olmalıdır.

Programlama Süreci

1. Problemin Tanımlanması
   - Programın ne yapması gerekiyor?
   - Programın çıktıları ne olacak ve nasıl bir formatta olacak?
   - Programın girdileri ne olacak ve nasıl bir formatta olacak?

Örnek: *İki sayıdan büyük olan sayıyı bulan bir program yazın*
   - İki sayı girişi al, karşılaştır ve maksimum değeri yazdır.
   - Giriş ve çıkışlar tamsayılar
   - Girişler klavyeden olacak
   - Çıkış monitörden gösterilecek

Programlama Süreci

2. Program Tasarımı:

Algoritma tasarım sürecidir. Bilgisayarın gereken çıktıları vermesi için adım adım gereken işlemlerin tasarlanmasıdır.

Yukarıdan Aşağıya (Top-Down) Tasarım Yöntemi

- Ana problem alt küçük problemlere bölünür.
- Her bir bölünmüş daha basit problemler çözülerek bir araya getirilir.

Programlama Süreci

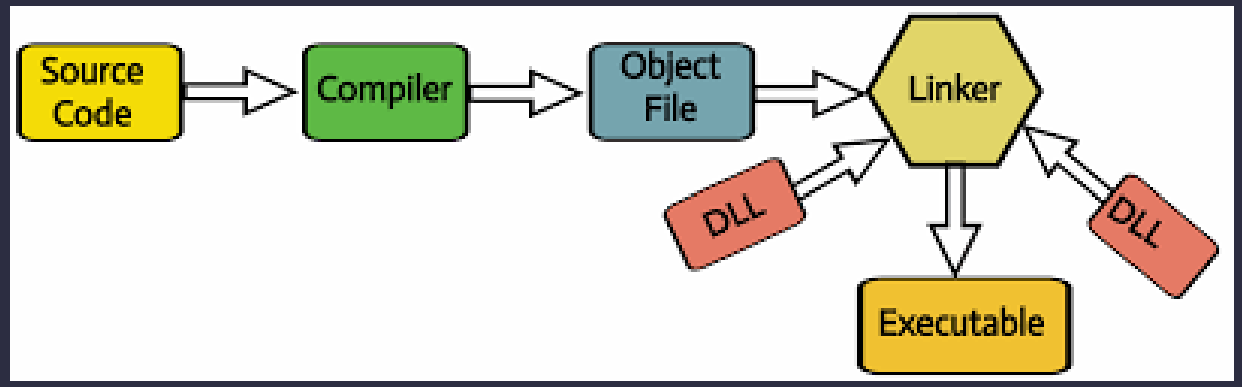
3. Kodlama Süreci:

Geliştirilen bir algoritmanın bir programlama dilinde ifade edilmesidir.

```
#include <stdio.h>

int main()
{
      int number1, number2;
      int maximum;

       printf("Please, enter two numbers: ");
       scanf("%d %d", &number1, &number2);

        if (number1 >= number2)
                                maximum = number1;
                else
                                maximum = number2;

         printf(“%d is maximum\n“, maximum);

                return 0;
}
```

Programlama Süreci

- *Derleme (Compilation): Bir kaynak kodunun obje koduna dönüştürülme işlemidir.*
- Bağlama (Linking) üretilmiş olan ve bir uygulama için gerekli olan tüm obje kodlarını alıp, çalıştırılabilir bir uygulama çıkarma işlemidir.
- Derleme ve Bağlama işleminin 2 adımlı olmasının avantajları:
  - Kaynak kod çok uzun olabilir, kod birden çok parçaya bölünebilir.
  - Bölünmüş kodlar birbirinden bağımsız çalıştırılabilirler.
  - İstendiğinde tek bir çalıştırılabilir dosyada (Windows için EXE) saklanabilirler.



Programlama Süreci

4. Program Testi ve Hata Ayıklama

- Bir program yazıldığında ilk denemede, genellikle hatalar barındırır.
- Bu hataları gidermek için deneme girişleri ile uygulamanın test edilmesi gerekmektedir.
- Hata ayıklama (Debugging), hataların tespiti ve giderilmesi sürecidir.

Hata Türleri

- **Sözdizimi (Syntax) Hataları**: Programlama dilinin kurallarına uyulmayarak yazılmış bir ifadeden kaynaklı hatalardır.
  - Bu hata gerçekleştiğinde, derleyici yazılımı, hata verip derleme işlemini gerçekleştirmez.
- **Anlamsal (Semantic) Hatalar**: Kaynak kodda anlamsal yani mantık hatasının gerçekleşmesidir.
  - Derleme işlemi gerçekleşir ancak programın üretmesi gereken sonuçlardan farklı sonuçlar ortaya çıkar.
- **Çalışma Zamanı (Run-time) Hataları**: Program çalışırken meydana gelmektedir. Beklenenden farklı bir formatta veri girişi veya bellekte programın erişemeyeceği bir yere erişmeye çalışması gibi sorunlar nedeniyle yaşanır.
  - Gerçekleştiğinde program patlar (Crash) ve kapanır.

Programlama

Programlama iki aşamaya ayrılabilir.

1. ***Problem Çözme Aşaması***
   - Problemin çözümü için adım adım yapılması gerekenlerin tarif edilmesidir.
   - Adım adım çözüm yöntemine algoritma denir.

2. ***Gerçekleme Aşaması***
   - Algoritmanın bir programlama dilinde gerçeklenmesidir.

Problem Çözme Aşamaları

- Algoritmanın genel akışının çıkartılması (Sözde Kod/ *pseudocode kullanılabilir)*
- Algoritmayı programlama diline yakın olacak bir biçimde adım adım detaylandırılır.
- *Sözde Kod (Pseudocode), doğal dilde ifade edilip, algoritmanın bir yazılım diline geçmesi sürecinde ara aşamadır.*

Sözde Kod ve Algoritmalar

- Örnek 1: Öğrencilerin dersten geçme durumlarını hesaplayan bir uygulama yazın. Geçme durumu 4 sınavın ortalamasına göre yapılacaktır. Geçme notu 50'dir.

Sözde Kod ve Algoritmalar

Sözde Kod:

- *4 notu giriş olarak al*
- *Bu notları topla ve 4'e böl*
- *Eğer 50'nin altında ise*

  Yazdır "Kaldı"

  Değilse

  Yazdır "Geçti"

Sözde Kod ve Algoritmalar

- Detaylı Algoritma
- Adım 1: Giriş M1,M2,M3,M4
  Adım 2: Not ← (M1+M2+M3+M4)/4
  Adım 3: Eğer (NOT < 50)
  Bastır “Kaldı”
  Değilse
  Bastır “Geçti”

Akış Diyagramları (Flowcharts)

- Bir bilgisayar programının veya bir fabrikadaki üretim sürecindeki akışın, şematik olarak gösterim biçimidir.

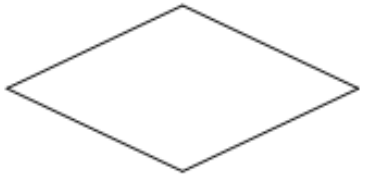
- Algoritmalar, akış diyagramları şeklinde ifade edilebilirler. Algoritmadaki komutların akışının görselleştirilmesinde faydalıdırlar.

Akış Diyagramları (Flowcharts)

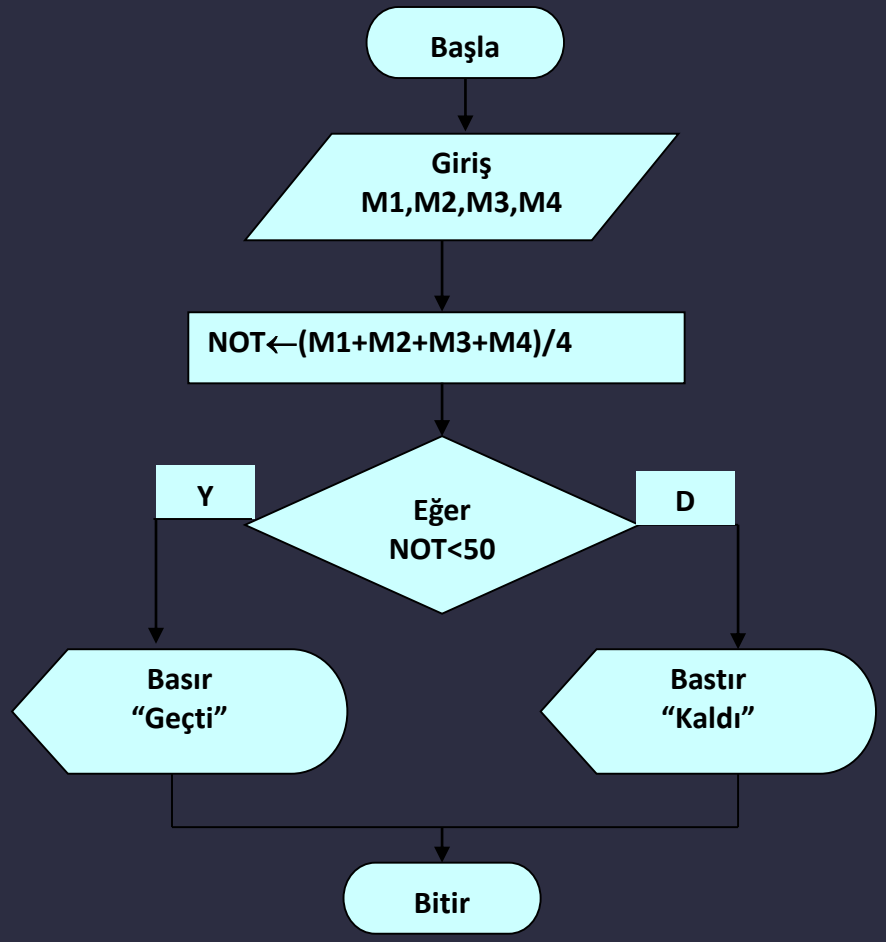
Bir akış diyagramı

- Algoritmanın akışını gösterir
- Adımları ve bağlantıları vurgular (kontrol yapıları, bir sonraki aşamanın ne olacağı gibi …)

Akış Diyagramları (Flowcharts) Sembolleri

Name	Symbol	Use in Flowchart
Oval		Denotes the beginning or end of the program
Parallelogram		Denotes an input operation
Rectangle		Denotes a process to be carried out e.g. addition, subtraction, division etc.
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes. (e.g. IF/THEN/ELSE)
Hybrid		Denotes an output operation
Flow line		Denotes the direction of logic flow in the program

Örnek 1



- Detaylı Algoritma
- Adım 1: Giriş M1,M2,M3,M4
  Adım 2: Not ← (M1+M2+M3+M4)/4
  Adım 3: Eğer (NOT < 50)
  Bastır “Kaldı”
  Değilse
  Bastır “Geçti”

Örnek 2

- Girilen bir feet birimindeki girişi cm'e dönüştüren bir uygulama geliştirin.

Sözde Kod:

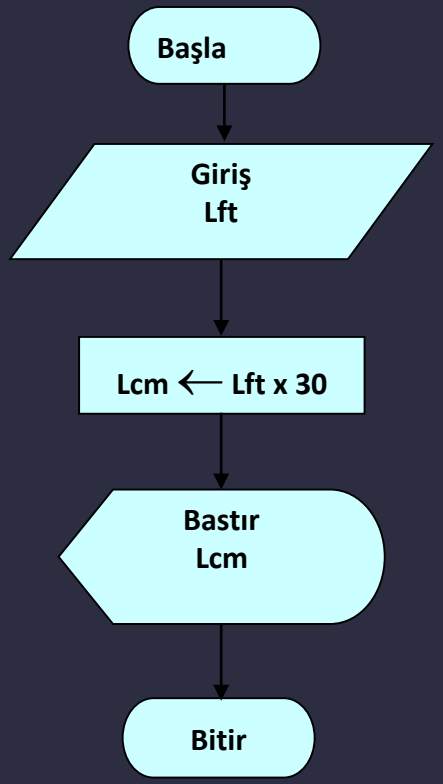
- *Feet (LFT) girişini al*
- *CM (LCM) olarak hesaplamak için LFT'i 30 ile çarp*
- *Ekrana LCM'i bastır.*

Örnek 2

Algoritma

- Adım 1: Giriş LFT
- Adım 2: LCM ← LFT x 30
- Adım 3: Bastır LCM

Akış Diyagramı



Örnek 3

Bir dikdörtgen iki kenarını giriş alıp, alanını hesaplayan bir uygulama yazınız.

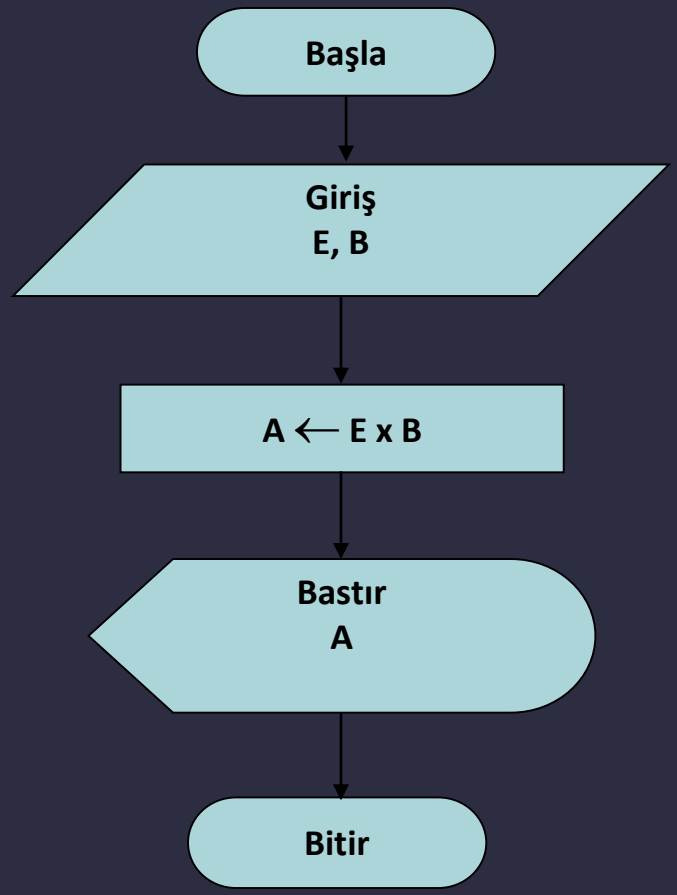
Sözde kod

- *Eni (E) ve boyunu (B) al.*
- *Alanı (A) E ve B'i çarparak hesapla*
- *A'ı bastır*

Örnek 3

Algoritma

- Adım 1: Giriş E,B
- Adım 2: A ← E x B
- Adım 3: Bastır A



Örnek 4

- Aşağıdaki denklemin köklerini bulan bir uygulama geliştirin.

$$ax^2 + bx + c = 0$$

- İpucu: **d** = karekök ($b^2 - 4ac$) denklemi ile kökler bulunabilir:

***x*1** = (–*b* + *d*)/2*a*

***x*2** = (–*b* – *d*)/2*a*

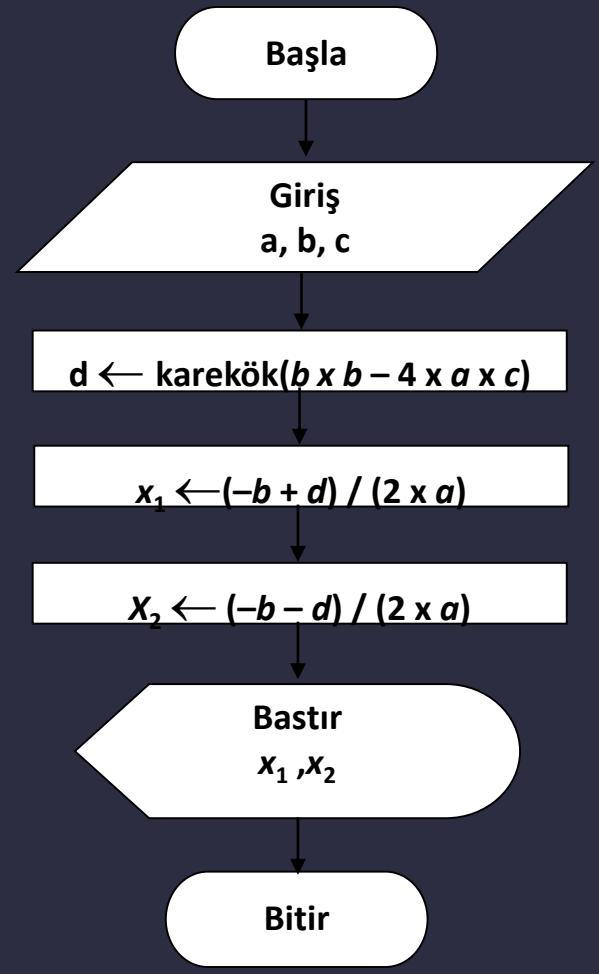
Örnek 4

Sözde Kod:

- *Denklemin a, b,c katsayılarını al*
- **d'i hesapla**
- ***x*1'i hesapla**
- **x2'i hesapla**
- x*1 ve* x2'i bastır

Örnek 4

- **Algoritma**:
  - Adım 1: Giriş a, b, c
  - Adım 2: $d \leftarrow$ karekök ($b \times b - 4 \times a \times c$)
  - Adım 3: $x1 \leftarrow (-b + d) / (2 \times a)$
  - Adım 4: $x2 \leftarrow (-b - d) / (2 \times a)$
  - Adım 5: Bastır $x1$, $x2$



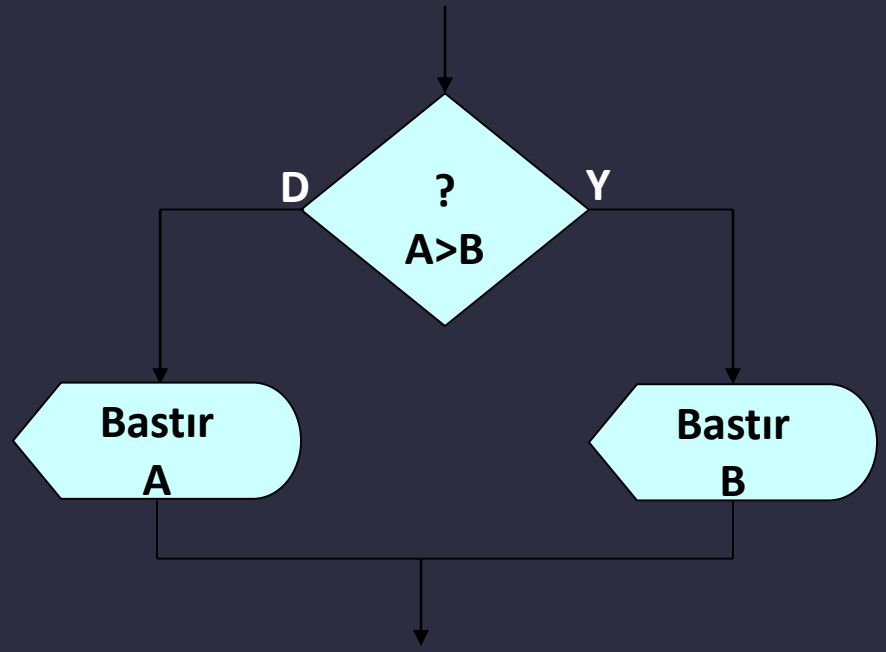
Kontrol Yapıları

- A>B ifadesi, bir mantıksal bir ifadedir, sonucu doğru veya yanlıştır
- Test edilmek istenen bir durumu ifade eder.

Örnek:

- ***Eğer A>B İfadesi doğru ise (Yani A, B'den daha büyük ise)***
  - *A'nın değerini Bastır*
- ***Eğer A>B ifadesi yanlış ise (Yani A, B'den daha büyük değil ise)***
  - B'nin değerini bastır

Kontrol Yapıları



Eğer, Değilse (if, else) Yapıları

- Yapı şu şekildedir:

eğer durum

doğru olduğu durumdaki aksiyon

değilse

yanlış olduğu durumdaki aksiyon

Eğer, Değilse (if, else) Yapıları

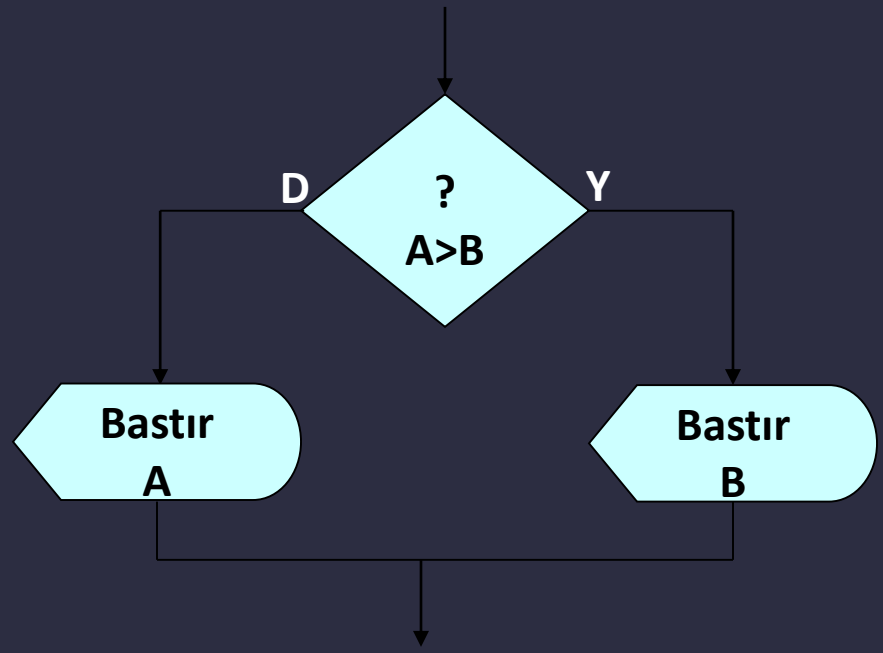
- Algoritmanın akış diyagramı şu şekildedir:

Eğer A>B

bastır A

Değilse

bastır B



Kontrol Yapıları

Kontrol Operatörleri

Operatör	Açıklama
>	Büyüktür
<	Küçüktür
=	Eşittir
≥	Büyüktür veya Eşittir
≤	Küçüktür veya Eşittir
≠	Eşit Değildir

Örnek 5

- İki sayı girişi alıp, en büyük sayının ne olduğunu ekrana gösteren bir uygulama geliştirin.

Algoritma

Adım 1: *Girişler* girdi1, girdi2

Adım 2: *Eğer (girdi*1 > girdi2)

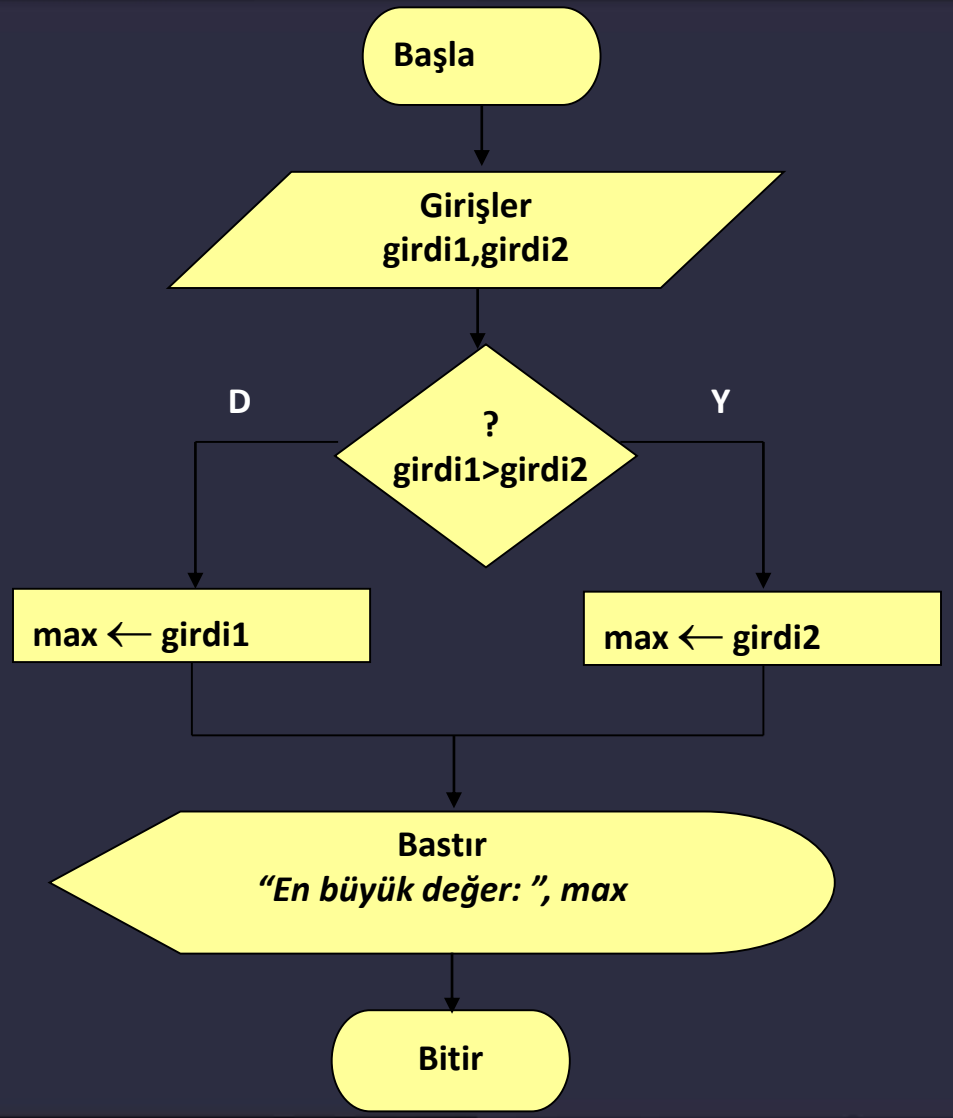
max ← girdi1

Değilse

max ← girdi2

Adım 3: *Bastır "En büyük değer: ", max*

Örnek 5


Başla
Girişler
girdi1,girdi2
?
girdi1>girdi2
D
Y
max ← girdi1
max ← girdi2
Bastır
"En büyük değer: ", max
Bitir

İç İçe Kontrol Mekanizmaları

- Bir eğer, değil (if, else) mekanizması birden çok eğer, değil mekanizması içerebilir.

Örnek:

- Kullanıcıdan 3 sayı alıp, en büyüğünü bulan bir uygulama geliştirin.

Örnek 6

Adım 1: ***Giriş*** **N1, N2, N3**
Adım 2: ***Eğer (*****N1>N2){**
Eğer (**N1>N3) *****{**
MAX ← N1
}
Değilse{
MAX ← N3
}
}
Değilse{
Eğer (**N2>N3){**
MAX ← N2
}
Değilse{
MAX ← N3
}
}
Adım 3: ***Bastır "En büyük sayı: ", MAX***

Örnek 6

- **Ödev 1 – Soru 1: Yandaki algoritmanın akış diyagramını çiziniz**

```
Adım 1: Giriş          N1, N2, N3
Adım 2: Eğer (N1>N2){
            Eğer (N1>N3) {
                    MAX ← N1
            }
            Değilse{
                    MAX ← N3
            }
        }
        Değilse{
            Eğer (N2>N3){
                    MAX ← N2
            }
            Değilse{
                    MAX ← N3
            }
        }
Adım 3: Bastır "En büyük sayı: ", MAX
```

Örnek 7

Bir çalışanın ismini (isim), kaç saat fazla çalıştığı (fSaat) ve kaç saat şirkete gelmediğini isteyip (gsaat), ek mesai ücretinin ne kadar olacağını bulan bir program yazınız.

Bonus Tablosu	
fSaat – (2/3)*gSaat	Ödenecek Bonus
>40 Saat	$50
>30 ve ≤ 40 hours	$40
>20 ve ≤ 30 hours	$30
>10 ve ≤ 20 hours	$20
≤ 10 hours	$10

Örnek 7

- **Ödev 1 – Soru 2: Yandaki algoritmanın akış diyagramını çiziniz**

Adım 1: *Girişler* isim, fSaat, gSaat
Adım 2: *Eğer* (fSaat–(2/3)*gSaat > 40)
ödeme ← 50
*Değilse Eğer (*fsaat–(2/3)*gsaat > 30)
ödeme ← 40
*Değilse Eğer (*fsaat–(2/3)* gsaat > 20)
ödeme ← 30
*Değilse Eğer (*fsaat–(2/3)* gsaat > 10)
ödeme ←20
Değilse
ödeme ← 10

Adım 3: *Bastır* isim, “ kişisi için $”, ödeme," ödeme yapılacak."

Döngüler

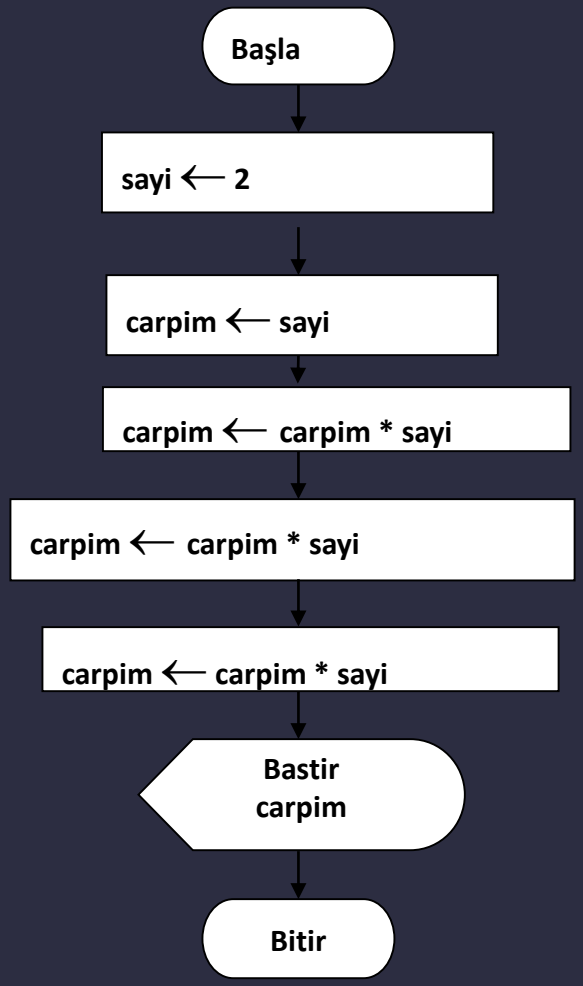
- Bir uygulamada, aynı işlem birçok kez tekrar ediyorsa, bu işlem için döngü (loop) yapıları kullanılır.

- Döngü yapılarının kullanılması, uygulamanın karmaşıklığı ve boyutunu azaltmaktadır.

Örnek 8

- **Algoritma**:
- Adım 1: sayi ← 2
- Adım 2: carpim ← sayi
- Adım 3: carpim ← carpim * sayi
- Adım 4: carpim ← carpim * sayi
- Adım 5: carpim ← carpim * sayi
- Adım 6: *Bastır* carpim

Örnek 8


Başla
sayi ← 2
carpim ← sayi
carpim ← carpim * sayi
carpim ← carpim * sayi
carpim ← carpim * sayi
Bastir
carpim
Bitir

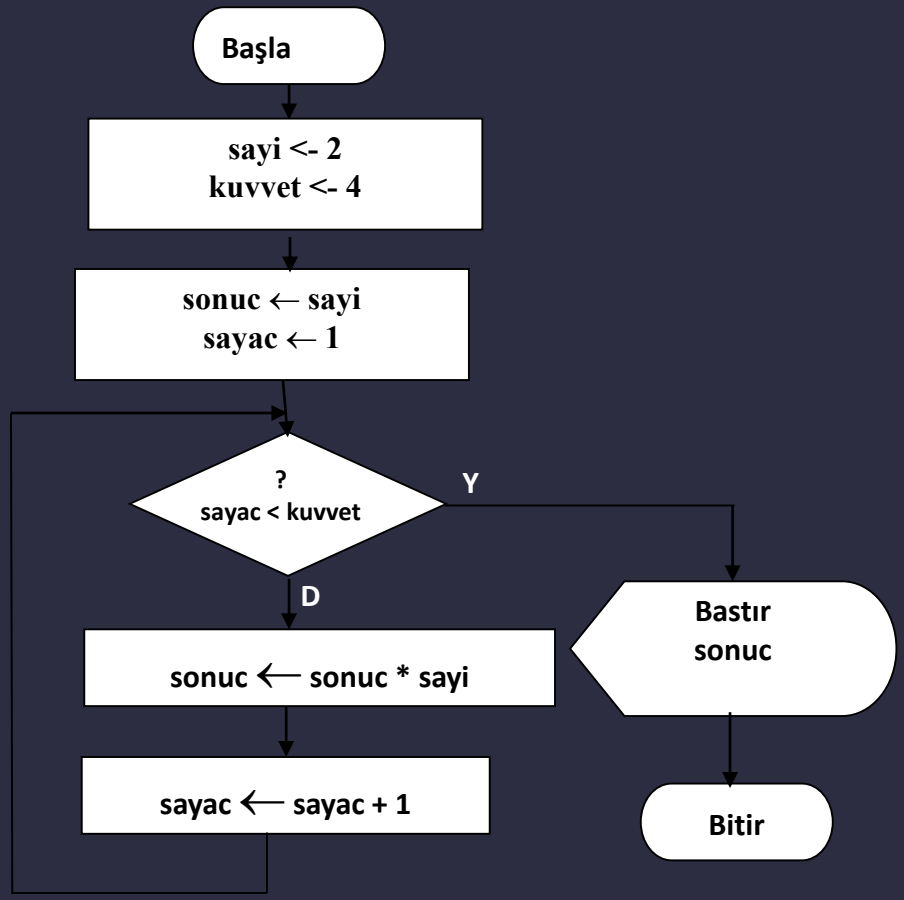
Örnek 9

- 2^4 sayısını döngü yöntemi ile hesaplayan bir yazılım geliştirin.

Örnek 9

Adım 1: sayi ← 2
Adım 2: kuvvet ← 4
Adım 3: sonuc ← sayi
Adım 4: sayac ← 1
Adım 5: Dongu sayac < kuvvet
sonuc ← sonuc * sayi
sayac ← sayac +1
Adım 8: *Bastır* sonuc

Örnek 9


Başla
sayi <- 2
kuvvet <- 4
sonuc ← sayi
sayac ← 1
?
sayac < kuvvet
Y
D
sonuc ← sonuc * sayi
sayac ← sayac + 1
Bastır
sonuc
Bitir

Örnek 10

- Sırayla 3 sayı alıp, en büyük sayıyı bulan programı yazınız.

Örnek 10

- Adım 1: *Giriş* N1
- Adım 2: Max ← N1
- Adım 3: *Giriş* N2
- Adım 4: *Eğer (*N2>Max)
  Max = N2
- Adım 5: *Giriş* N3
- Adım 6: *Eğer (*N3>Max)
  Max = N3
- Adım 7: *Bastır* “En büyük sayı: ”, Max

5, 7, 3 girişleri için:

	N1	N2	N3	Max	N2>Max	N3>Max
Adım 1:	5	?	?	?	?	?
Adım 2:	5	?	?	5	?	?
Adım 3:	5	7	?	5	T	?
Adım 4:	5	7	?	7	T	?
Adım 5:	5	7	3	7	F	F
Adım 6:	5	7	3	7	F	F
Adım 7:	Bastır → En Büyük sayı **7**					

Örnek 11

- N adet sayı girişi alıp, en büyüğünü bastıran bir program yazınız. Sayı girişleri teker teker olacaktır.

Örnek 11

- Adım 1: *Giriş N*
- Adım 2: *Giriş sayi*
- Adım 3: Max ← *sayi*
- Adım 4: sayac ←1
- Adım 5: *Dongu* (sayac < N)

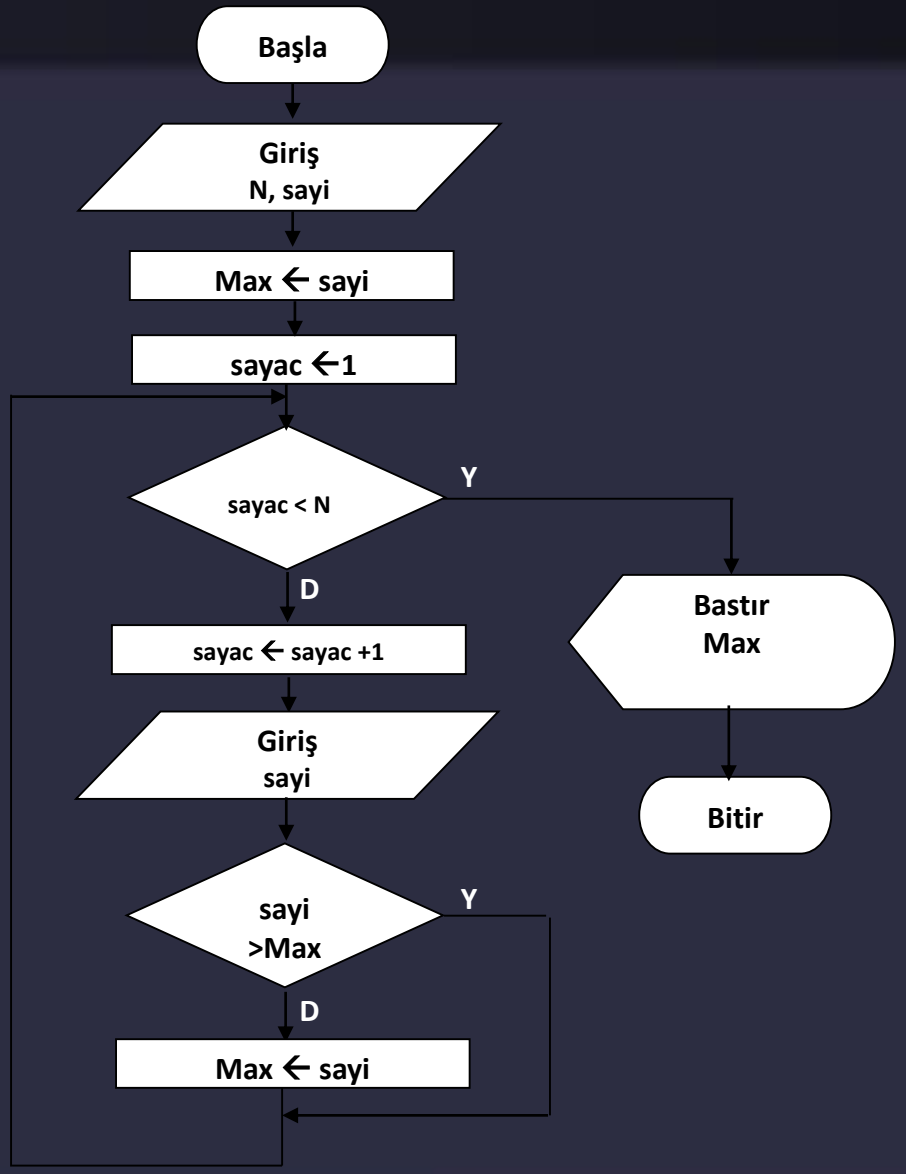
  sayac ← sayac + 1

  Giriş sayi

  Eğer (sayi > Max)

  Max ← sayi
- Adım 6: *Bastır* Max

Örnek 11


Başla
Giriş
N, sayi
Max ← sayi
sayac ←1
sayac < N
Y
D
sayac ← sayac +1
Giriş
sayi
sayi
>Max
Y
D
Max ← sayi
Bastır
Max
Bitir