

4. ALGORİTMA

Algoritma sözcüğü, [Özbekistan](#)'ın [Harezmi](#), bugünkü [Türkmenistan](#)'ın [Hive](#) kentinde 9. Yüzyılda doğmuş olan Ebu Abdullah Muhammed İbn Musa el [Harezmi](#) (İranlı Musaoğlu Horzumlu Mehmet)'den gelir. Problem çözümü için genel kurallar oluşturmuştur. Araplar "alherizmi" olarak kullanmışlardır. [Cebir](#) alanındaki algoritmik çalışmalarını "Hisab el-cebir ve el-mukabala" isimli kitaba dökerek matematiğe çok büyük bir katkı sağlamıştır. Bu kitabı dünyanın ilk cebir ve ilk algoritma kitabıdır. Latince çevirisi Avrupa'da çok ilgi görmüştür. Avrupalılar "algorizm" sözcüğünü "[Arap](#) sayıları kullanarak aritmetik problemler çözme kuralları" manasında kullanırlar. Bu sözcük daha sonra "algoritma"ya dönüşmüş ve bugünkü genel kapsamda kullanılmaya başlanmıştır.

Tanım :

Tam anlamı ile izlendiğinde yapılması istenen bir iş için oluşturulmuş sınırlı bir KOMUT kümesidir.

ALGORİTMANIN İNCELENMESİ

1. Algoritmaların işlenmesi için gerekli **donanım**
2. Algoritmaların tanımlayıcısı olan **programlama dilleri**
3. Algoritmaların **temel prensipleri**
4. Algoritmaların **analizi**

ALGORİTMALARIN TAŞIMASI GEREKEN ÖZELLİKLER

1. Her algoritma sıfır veya daha fazla sayıda GİRDİ (INPUT) içerir
2. Her algoritmanın en az bir ÇIKTI (OUTPUT)'u vardır
3. Algoritmayı oluşturan her adımın anlamı AÇIK ve BİRDEN FAZLA ANLAM İFADE ETMEYECEK şekilde olmalıdır (definiteness)
4. Algoritmaların tüm seçenekleri (sonuca giden alternatif yolları) SINIRLI ADIMDAN sonra sona erebilmelidir (finiteness)
5. Algoritmaların her adımı BASİT DÜZEYDE olmalıdır ve tek bir şey ifade etmelidir (clear & tells one thing)
6. Algoritma çözüm getirdiği problem ile ilgili ortaya çıkabilecek her durumu (alternatifi/seçeneği) içermelidir
7. Bir algoritma benzer problemler için genellenebilmelidir (iyi bir programlama için gerekli)

Çoğu zaman bir programa yönelik olarak hazırlanabilecek birden fazla algoritma söz konusudur. Ancak bunlardan bir tanesi daha etkin olabilir, yani daha az bellek gerektirecek ve daha az işlem adımı ile çalışır ve dolayısı ile daha hızlı çalışabilecektir. Etkin algoritma hahazırlayabilmek, deneyim gerektiren bir durumdur.

AKIŞ ŞEMALARI

Bir algoritmanın grafiksel ifadesine akış şeması denir. Akış şemasında kullanılan şekiller

, yani grafik semboller genel olarak standarttır. Akış şemasını , bir bilgisayar dili şeklinde kodlamak oldukça kolaydır. Akış şeması hazırlanırken, genelde herhangi bir bilgisayar veya programlama dili göz önüne alınmaz. Akış şeması bir problemin çözümüne yönelik hazırlanır ve çözüm için gerekli işlem adımlarını içerir. Ancak bazen akış şeması belirli bir programlama dili göz önüne alınarak da hazırlanabilir.

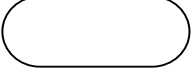


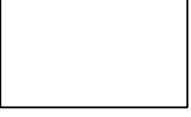

Akış şemaları iki amaçla hazırlanır

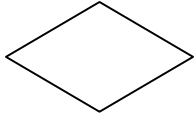
1. Program akış şeması
2. Sistem akış şeması

Bir akış şeması, kullanılan sembollerin akış yönünü gösteren oklarla birbirine bağlanması ile oluşur. Oklar akış şemasının akış yönünü belirler.

ANSI (American National Standards Institution) standartlarına göre akış şemalarında kullanılabilecek sembollerin anlamları aşağıda görülmektedir.

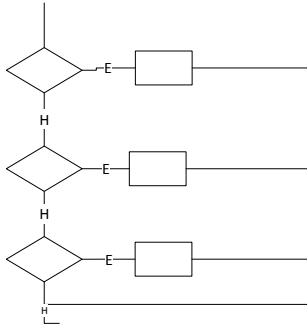
Pseudocode : Program tasarımında kullanılan basitleştirilmiş bir programlama dilini andıran bir gösterimdir

Akış şema sembolleri	Sembollerin anlamı	BASIC code
	BAŞLA/DUR	STOP
	GENEL VERİ GİRİŞ/ÇIKIŞ (INPUT/OUTPUT)	READ/DATA
	KLAVYE VERİ GİRİŞİ (KEYBOARD DATA ENTRY)	INPUT
	İŞLEM	[LET]
	FONKSİYON/SUBROUTINE	FUNC/SUB



İKİLİ KARAR

IF <koşul> THEN ELSE ...



ÇOKLU KARAR

SELECT CASE <koşul>

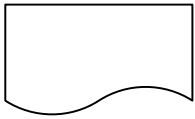
CASE <koşul1> : ...

CASE <koşul2> : ...

....

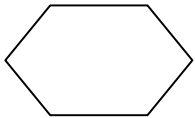
CASE ELSE : ...

ENDCASE



YAZICI ÇIKTISI/DOCUMENT

PRINT

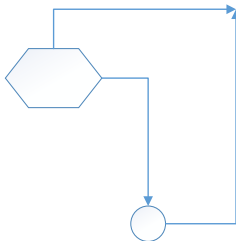


DÖNGÜ (LOOP)

FOR ... NEXT

REPEAT ... UNTIL <koşul>

WHILE <koşul> ... DO ... REPEAT

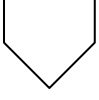


AKIŞ YÖNÜ (FLOW DIRECTION)

GOTO etiket/satırno



SAYFA İÇİ BAĞLAÇ/CONNECTOR (ON-PAGE REFERENCE)



SAYFA DIŞI BAĞLAÇ/CONNECTOR (OFF-PAGE REFERENCE)



VERİ TABANI (DATABASE)



DIŞ VERİ (EXTERNAL DATA)

Not : KARAR sembolü ve BAŞLA/DUR sembolleri hariç, her sembolün bir giriş, bir çıkış yönü vardır. KARAR sembolünün bir giriş iki çıkışı, BAŞLA sembolünü sadece bir çıkışı, DUR sembolünün sadece bir girişi vardır.

PROBLEMİN ANALİZİ

1. Problem tam olarak anlaşılmalıdır (understanding)
2. Bir çözüm metodu seçilir veya geliştirilir (developing)
3. Çözüm süreci adım adım tanımlanır (describing)
4. Algoritma programlama diline dönüştürülür ve program test edilir (verified, debugging)
5. Algortimanın geçerliliği test edilir (validating)

YAZILIM GELİŞTİRMENİN AŞAMALARI

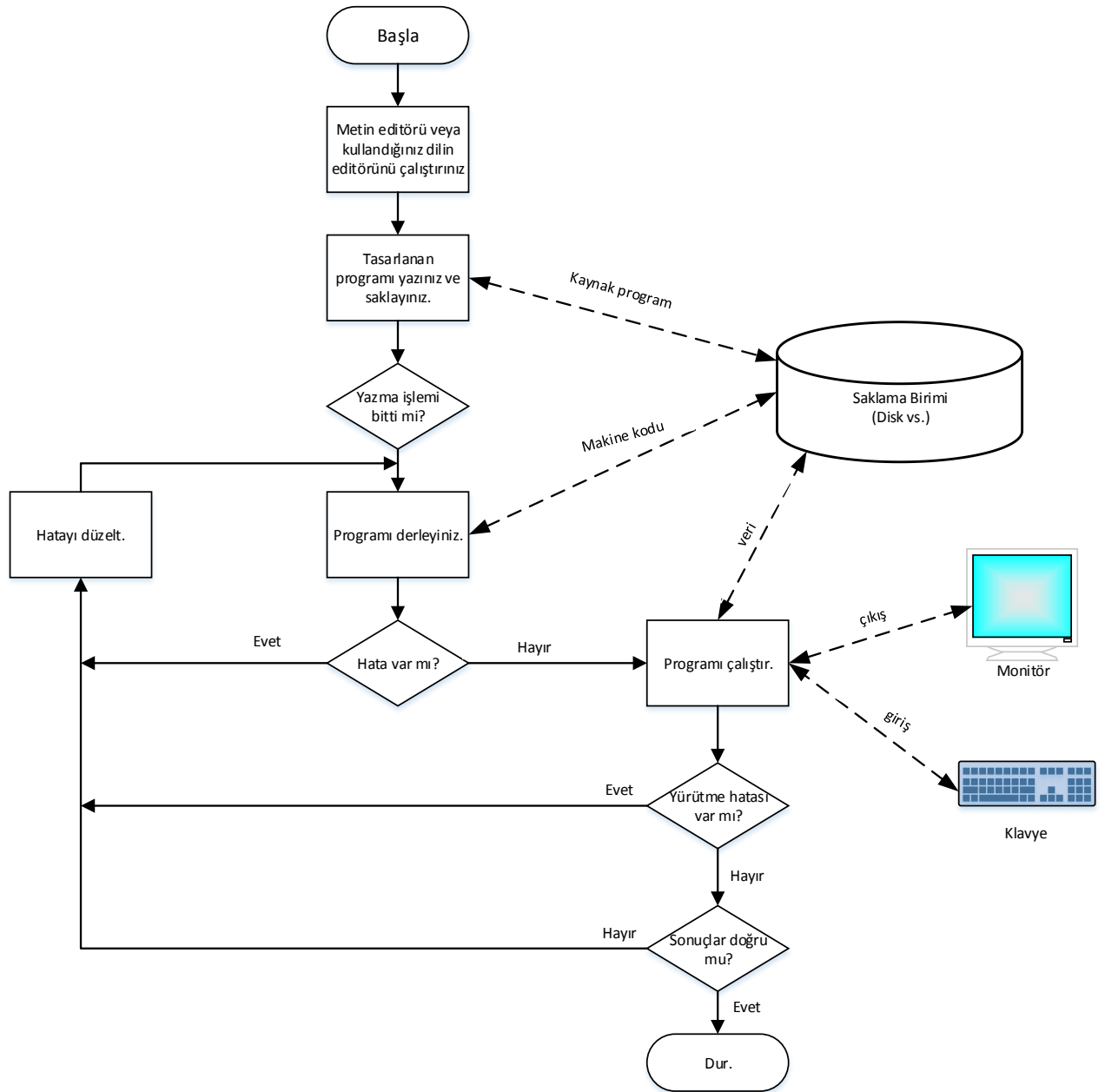
Programlama bilgisayar ile bir problemi çözme prosesi olarak tanımlanabilir. Program yazımı aşağıdaki aşamaları içerir:

1. Problemi anlama
2. Çözüm metodunu planlama
3. Uygun algoritmalar, akış şemaları, vb. kullanarak metotların geliştirilmesi
4. Bir programlama dili kullanarak komutların kodlanması
5. Komutların bilgisayarın anlayabileceği şekle dönüştürme
6. Programın testi ve hataların ayıklanması
7. Programın oluşturulmasında kullanılanların dokümantasyonu

Bir program geliştirirken programcının vaktinin bölünmesi şöyle olmaktadır:

Problem tanımı ve çözümü	%40
Kodlama	%20
Hataların ayıklanması ve dokümantasyon	%40

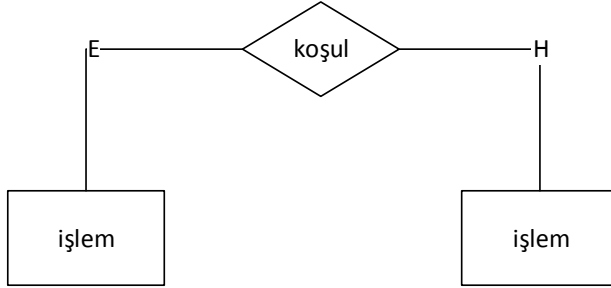
Bir program geliřtirmenin ařamalarının akıř diyagramı ile gsterimi



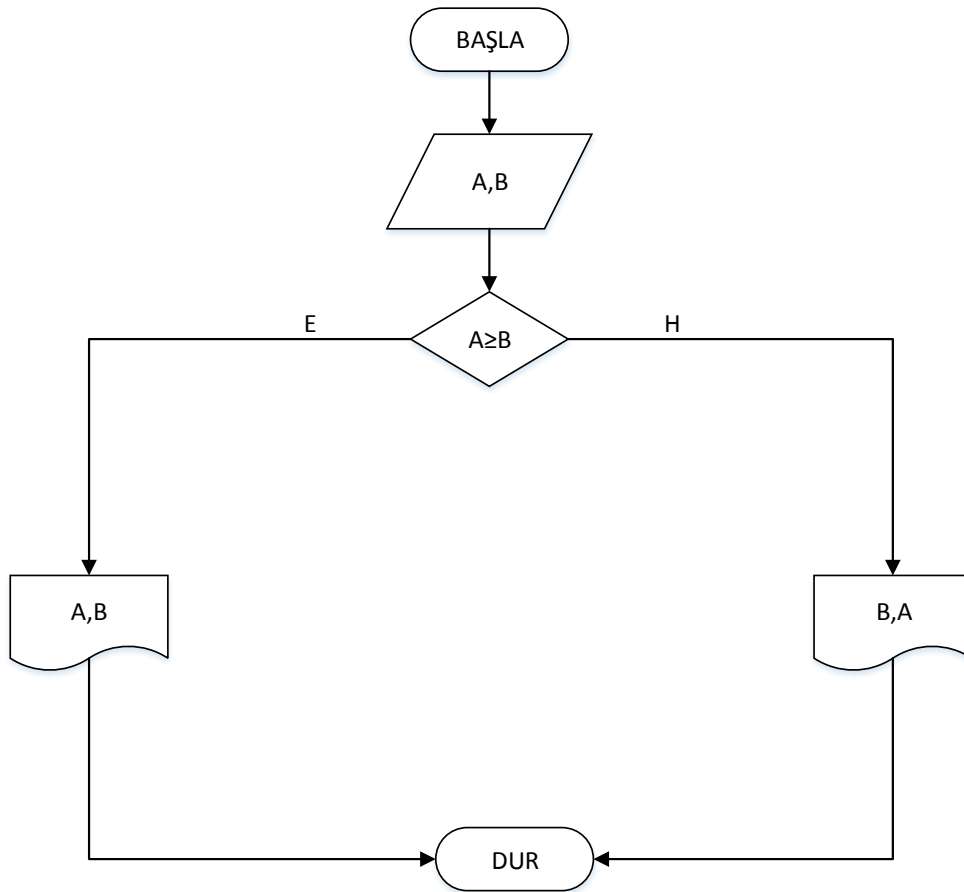
KARAR İFADESİ VE İŞLEMLERİ

İf-then-else :

İf <koşul> then koşul sağlandığında yapılacak işlem else koşul sağlanmadığında yapılacak işlem



Örnek (şe2) : Girilen A ve B gibi iki sayısal değer karşılaştırılmakta ve karşılaştırma sonucuna göre ilgili sonuçlar yazdırılmaktadır.



```
READ A,B
IF A>B THEN PRINT A,B ELSE PRINT B,A
DATA 10,20
STOP
```

Örnek 1 (ş3) : Bilgisayara üç değer girilmekte ve bunların en büyüğü elde edilip yazdırılmaktadır.

ALGORİTMANIN ANALİZİ

Girdi : A, B, C

İşlem : A>B evet ise A>C evet ise ENB=A hayır ise ENB=B

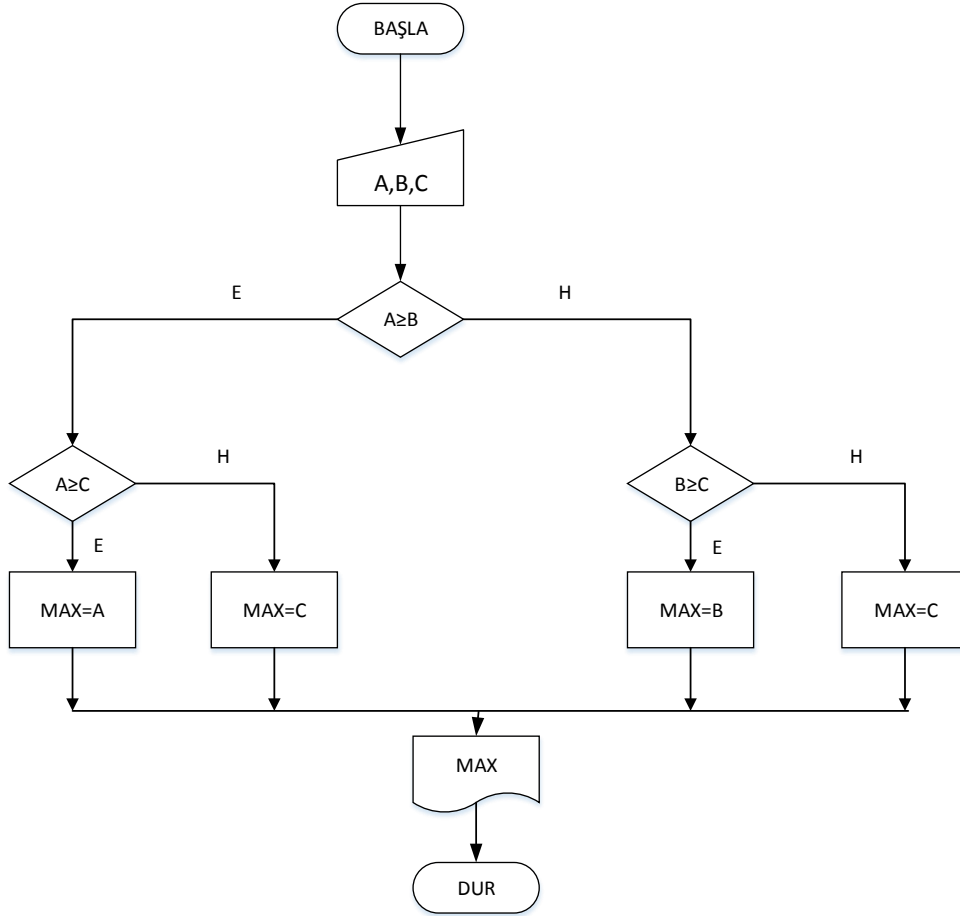
Hayır ise B>C evet ise ENB=B hayır ise ENB=C

Çıktı : ENB

ALGORİTMANIN TESTİ

İz tablosu :

Sem.No.	A	B	C	ENB	A>B	A>C	B>C



INPUT A,B,C

IF A>=B THEN IF A>=C THEN MAX=A ELSE MAX=C

ELSE IF B>=C THEN MAX=B ELSE MAX=C

PRINT MAX

STOP

NOT : MAX değişkeni kullanmadan program yaz , avantaj ve dezavantajı tartış.

Örnek 2 : 10 adet verilen sayının aritmetik ortalaması,

2.a) İşlem : $(x_1 + x_2 + \dots + x_{10})/10$

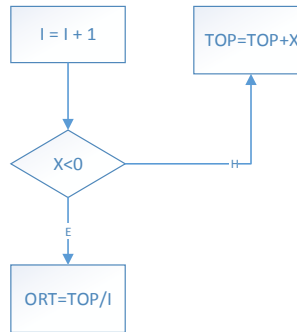
Çözüm : tek bir INPUT kullanarak X'leri okut.

2.b) $(I < 10)$ kontrolü ile

2.c) $(I < N)$ kontrolü ile

2.d) başlangıçta bilinmeyen sayı için

Çözüm: $0 < X < 100$ ise, durdurmak amaçlı X 'e -1 gibi negatif bir sayı girmek



Örnek 3 : $y = ax^2 + bx + c$ denkleminin köklerini bulan prog., $x_{1,2} = (-b \pm \sqrt{\Delta})/2a$, $\Delta = b^2 - 4ac$

Örnek 4 : $ax + by = c$ ve $dx + ey = f$ iki bilinmeyenli denklem sisteminin x,y için ortak çözümü

Örnek 5 : N adet sayının aritmetik ortalaması

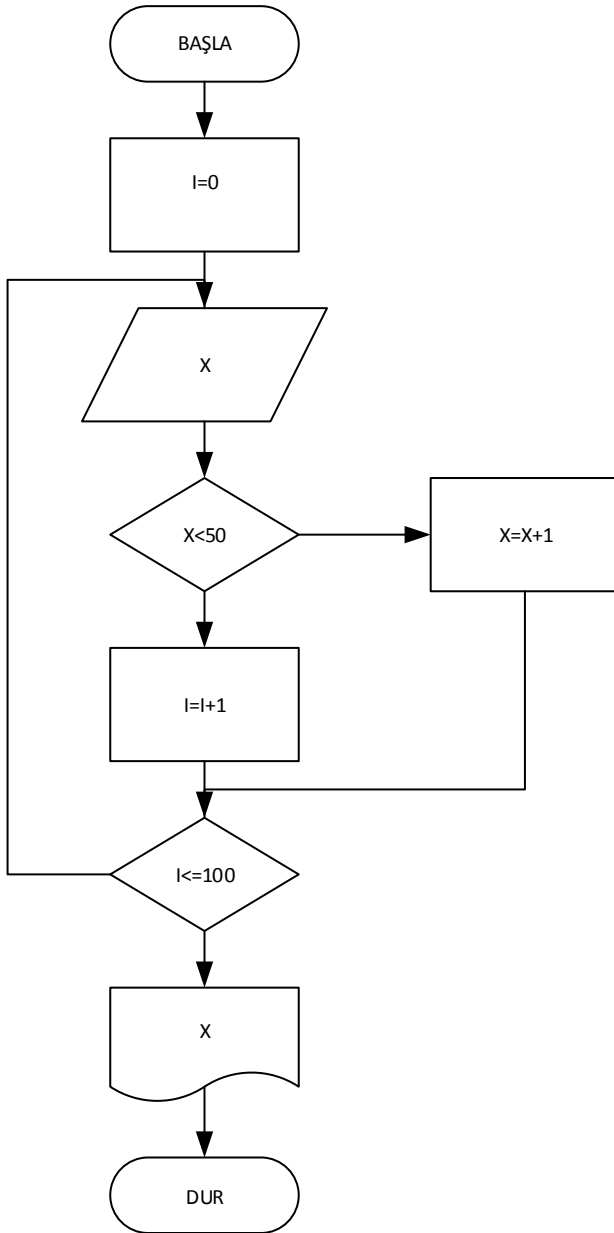
Örnek 6 : N adet sayının geometrik ortalaması

Örnek 7 : N! hesabı

Örnek 8 : NC_n (kombinasyon hesabı)

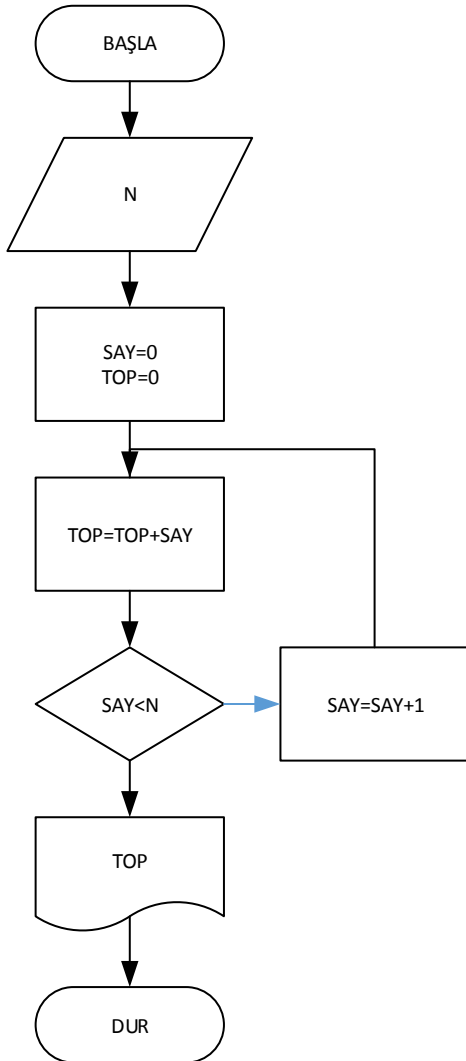
Örnek 9 : NP_n (Permütasyon hesabı)

Örnek : 100 Öğrencinin başarı notu giriliyor, 50 altında olan, kalan öğrenci sayısını bulan program.



```
I=0
GIT1 :
INPUT X
IF X<50 THEN X=X+1 : GOTO GIT
    ELSE I=I+1
GIT2 :
IF I<=100 THEN GOTO GIT1
    ELSE PRINT X
STOP
```

Örnek : Birden, girilen N sayısına kadar sayıların toplamını bulan program



INPUT N

SAY=0 : TOP=0

DEVAM :

TOP=TOP+SAY

IF SAY<N THEN SAY=SAY+1 : GOTO DEVAM

ELSE PRINT TOP

STOP

case ... end :

işlem akışında, hangi koşul sağlanırsa, o koşulu takip eden (gerektirdiği) işlemleri yapar ve işlem bloğunun sonundaki end ile biten case bloğunu terk eder.

Case

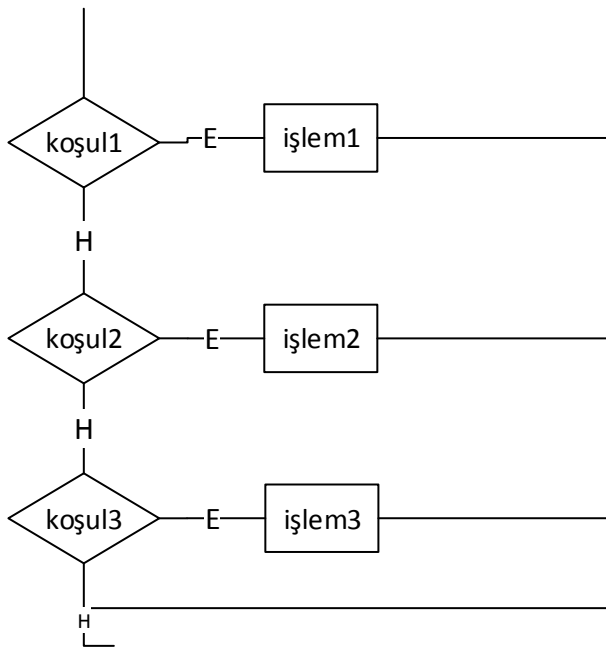
<koşul-1> : işlem-1

<koşul-2> : işlem-2

...

<koşul-n> : işlem-n

end case



veya duruma dayalı bu karar ifadesi,

case

: <koşul1> : işlem1

: <koşul2> : işlem2

: else : işlem3

end

şeklinde de kullanılabilir.

Örnek : Bir öğrencinin 100 üzerinden girilen notunu 4 üzerinden A,B,C,D,F sistemine dönüştüren program parçası.

DNOT : dersten aldığı not

BNOT : başarı not grubu (A,B,C,D,F)

Case DNOT

: DNOT >=85 : BNOT="A"

: DNOT < 85 and DNOT>=70 : BNOT="B"

: DNOT< 70 and DNOT>=60 : BNOT="C"

: DNOT<60 and DNOT>=50: BNOT="D"

: DNOT<50 : BNOT="F"

end case

BASLA :

INPUT DNOT

SELECT CASE DNOT

CASE (DNOT>=85 AND DNOT<=100) : PRINT "A"

CASE (DNOT<85 AND DNOT>=70) : PRINT "B"

CASE (DNOT<70 AND DNOT>=60) : PRINT "C"

CASE (DNOT<60 AND DNOT>=50) : PRINT "D"

CASE (DNOT<50 AND DNOT>=0) : PRINT "F"

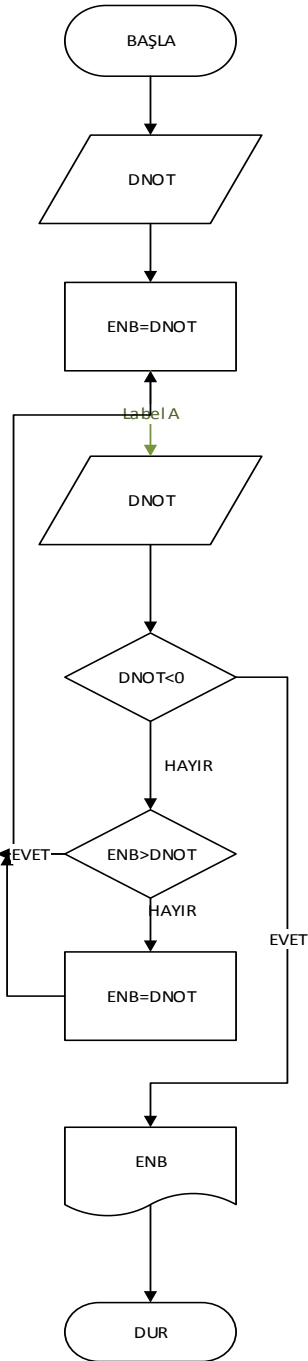
ELSE CASE : PRINT "girilen not yanlış"

GOTO BASLA

ENDCASE

STOP

Örnek : Başlangıçta sayısı belli olmayan öğrencilere ait sınav notları içinde en büyüğünü bulup yazdıran programın akış şeması. Negatif bir sayı girildiğinde program sonlanıyor.



Sem.No.	NOT	MAX	NOT<0	MAX>NOT

```

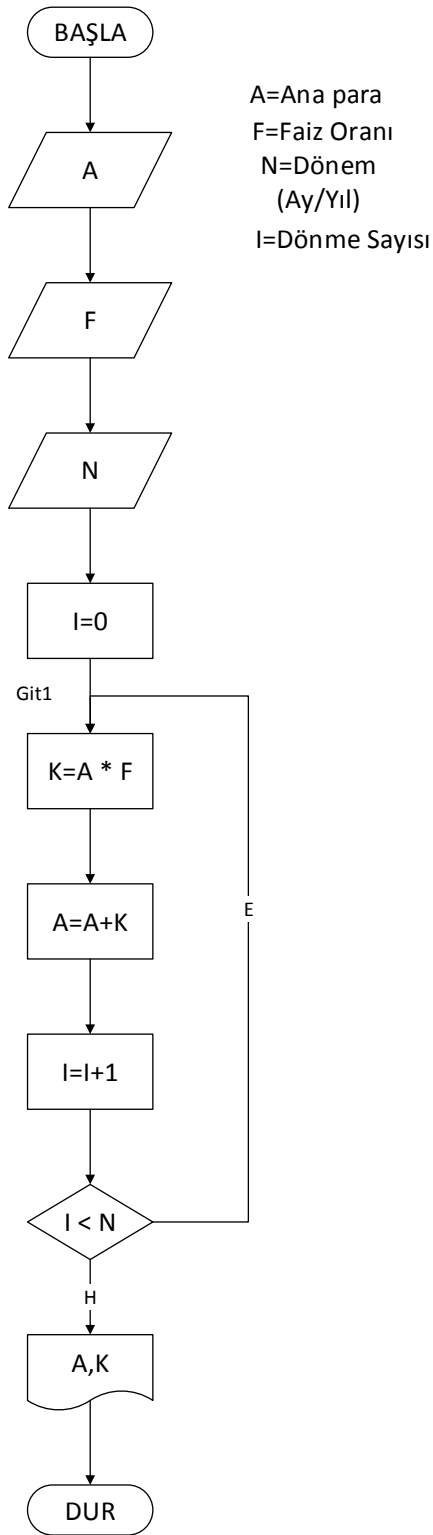
BASLA
INPUT DNOT
A :
LET ENB=DNOT
INPUT DNOT
IF DNOT<0 THEN PRINT ENB : STOP
      IF ENB>DNOT THEN GOTO A
      ENB=DNOT
GOTO A

```

Not : Akış şemasında 3. Sembol MAX = 0 olarak kullanılırsa, 2. Girdi sembolüne gerek kalmaz.

Örnek : Basit faiz hesabı

I.	A	F	N	K	I<N



```

BASLA

INPUT A

INPUT F

INPUT N

I=0

GIT1 :

K=A*F

A=A+K

I=I+1

IF I<N THEN GIT1

      ELSE PRINT A,K

STOP
  
```

Bilgisayarda matematiksel operatörlerin öncelikleri

1. Bir matematiksel bilgisayar işleminde öncelikle parantez içindeki işlemler tamamlanır. Diğer bir ifadeyle parantez () aşağıda belirtilen işlem önceliklerini değiştirmek için kullanılır. Ancak birden çok iç içe parantezler kullanılıyor ise açılan parantez sayısı ile kapanan parantez sayısı aynı olmasına dikkat edilmelidir. İşlem sırası en içteki parantezden başlar giderek dış parantez işlenir.
2. Üs alma işareti (** veya ^)
3. Sayı değerini negatife çevirme işareti (-)
4. Çarpma işareti (* veya .) ve bölme işareti (/ veya :)
5. MOD bölme ile kalan hesaplama
6. Toplama ve çıkartma işareti (+ ve -)
7. Aynı özellikli iki işlem operatörü aynı işlem içerisinde yer alıyorsa öncelik soldaki işlem operatöre aittir.
8. Bir işlem içerisinde iki işlem operatörü arka arkaya gelemmez.
9. Bilgisayarda atama ifadesi olarak kullanılan = veya ← operatörlerinin solunda daima İŞARETSİZ ve sadece BİR değişken yer alabilir.

(A+B = C+D olamaz, -B=C+D olamaz, A=B+C olur)

NOT : Yukarıdakiler Cebirsel ifadeleri bilgisayar işlemlerine şekline dönüştürürken dikkat edilmesi gereken kurallardır.

Örneğin ; $y=ax^2+bx+c$ denkleminin $y=0$ için kökleri olan $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ cebirsel ifadesinin bilgisayar programında yazımı

$$X1 = -B + (B^2-4*A*C)^{0.5}/(2*A)$$

$$X2 = -B - (B^2-4*A*C)^{0.5}/(2*A)$$

şeklinde olur, başka bir örnek aşağıda verilmiştir.

$$\sqrt[3]{x\sqrt{x}} + \frac{x-y}{c} + 1 \quad (X*X^{0.5})^{(1/3)}+(X-Y)/C+1$$

İşlem sırası :

3 2 1 4 2 1 4 3 7 5 6 8

5 (bölme)

6 7

Not : Kök içi işlemlerini de $(\sqrt{a})=(a)^{(1/2)}$ gibi parantez olarak ifade etmek kod yazmada kolaylık sağlar. Birden çok kök veya parantez iç içe ise, kod yazımına en içten başlamak kolaylık sağlar.

$$\frac{a + b + c}{2}$$

$$a + \frac{b + c}{2}$$

$$3(a^2 + b^2)$$

$$(a + \frac{b}{c})^{2d}$$

$$3^{ab^2}$$

$$\sqrt[3]{a + \sqrt{a^5\sqrt{a} + \sqrt{a}}}$$

İlişkisel (karşılaştırma-karar) operatörleri

<= veya =< Küçük eşittir

>= veya => Büyük eşittir

< Küçüktür

> Büyüktür

<> veya >< Eşit değil

= Eşittir

Mantıksal işlem operatörleri

işlem öncelikleri

Değil	'	—	NOT (complement)	1
Ve	.	^	AND (conjunction)	2
Veya	+	v	OR (disjunction)	3
			XOR (exclusive OR)	4
			EQV (equivalence)	5
			IMP (implication)	6

X	Y	NOTX	X AND Y	X OR Y	X XOR Y	X EQV Y	X IMP Y
+	+	-	+	+	-	+	+
+	-	-	-	+	+	-	-
-	+	+	-	+	+	-	+
-	-	+	-	-	-	+	+

Genel olarak işlem öncelikleri ise aşağıdaki sıradadır.

1. Fonksiyonlar
2. Aritmetiksel işlemler (kendi içindeki işlem öncelikleri ile)
3. İlişkisel işlemler
4. Mantıksal işlemler (kendi içindeki işlem öncelikleri ile)

BİLGİSAYARDA KULLANILAN VERİ TÜRLERİ

1. MAlfabetik karakterler (a – z ve A – Z arası v kombinasyonları)
2. Sayısal karakterler (0 – 9 arası ve kombinasyonları)
3. Özel karakterler (#, @, \$, %, & , ...)

Değişken

Programın her çalıştırılmasında, farklı değerler alan bilgi/bellek alanlarıdır.

Sabit

Programdaki değeri değişmeyen ifadelerle “sabit” denir. “İsimlendirme kuralları”na uygun olarak oluşturulan sabitlere, sayısal veriler doğrudan; alfa sayısal veriler ise tek/çift tırnak içinde aktarılır.

Bilgisayar programlarında kullanılan veri türleri

1. Alfabetik karakterler (A,a,B,b,C,c,...), genellikle “ ” içerisinde değişkene atanan ifadelerdir. Bu tırnak içindeki ifadeler sayısal olması halinde, bu sayılar ile matematiksel işlem yapılamaz.
2. Sayısal karakterler (0,1,2,3,...), matematiksel işlem yapılabilen ifadelerdir.
3. Özel karakterler (#,@,&!,%,...), her bilgisayar dilinde o dile özgü olarak tanımlanmış özel amaçlarla kullanılmak üzere rezerve edilmiş sembollerdir, sadece tanımlı oldukları amaçlar için kullanılırlar ve değişken/fonksiyon adlarında kullanılamazlar.

Veri yapıları bilgisayar ortamında verilerin etkin olarak saklanması ve işlenmesi için kullanılan yapılardır.

Karakter, Tamsayı ve Gerçek Sayı gibi değişkenler temel veri yapısı olarak kabul edilir. Karakterler bir araya gelerek sözcükler (string), sayılar bir araya gelerek dizileri (array) oluşturur.

Seçilen veri yapısı algoritmanın doğru ve etkin çalışabilmesi için önemlidir.

- 0100 0010 0100 0001 0100 0010 0100 0001
- 4 2 4 1 4 2 4 1
- Yukarıdaki bit dizisi;
 - Karakter dizisi (string) ise (ASCII): B A B A
 - BCD (Binary Coded Decimal) ise: 4 2 4 1 4 2 4 1
 - 16-bit tam sayı ise: 16961 16961
 - 32-bit tam sayı ise: 1111573057
 - 32-bit gerçek sayı ise: 0.4276801x10⁶⁶

Tamsayılar

8 bit char, byte
16 bit short int, short, ShortInt, Int16
32 bit int, integer, Int32

64 bit Karakterler dizileri (string)

ASCII Her karakter 8 bit (28 = 256 farklı karakter)
Unicode Her karakter 16 bit (216 = 65536 farklı karakter)
long int, long, LongInt, Int64

Ondalık (Gerçek) Sayılar

16 bit half (IEEE 754-2008)
32 bit single, float
64 bit double, real (Pascal)
128 bit quad

Array : dizinler, tek/çok boyutlu dizinler

Örnek : Bir sınıftaki 50 öğrenciye ait bilgisayar ve matematik notu girilmektedir. Bilgisayar notu 50 ve daha yüksek olanlar ile matematik notu 50 ve daha yüksek olanların sayısını ayrı ayrı bulduran algoritma.

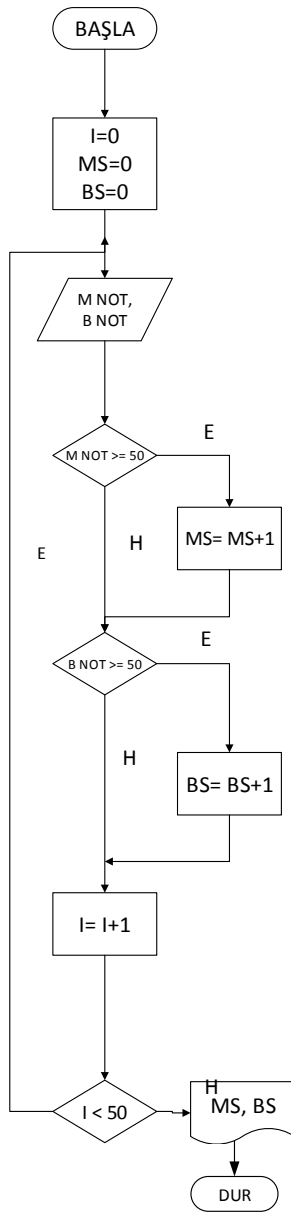
Girdi : M NOT , B NOT

Diğer değişkenler : I : sayaç

İşlem : M NOT \geq 50 ise MS ARTTIR

B NOT \geq 50 ise BS ARTTIR

Çıktı : MS VE BS



NO	I	MS	B S	MNOT	BNOT	MNOT \geq 50	BNOT \geq 50	I<50

Örnek : Bir sınıfta 50 öğrenciye ait matematik ve bilgisayar notları girilmektedir. Bilgisayar veya matematik notu 50 ya da daha büyük olan öğrenci sayısını bulup yazdıran programın akış şemasını yazınız.

Girdi : Matematik Notu: M Not

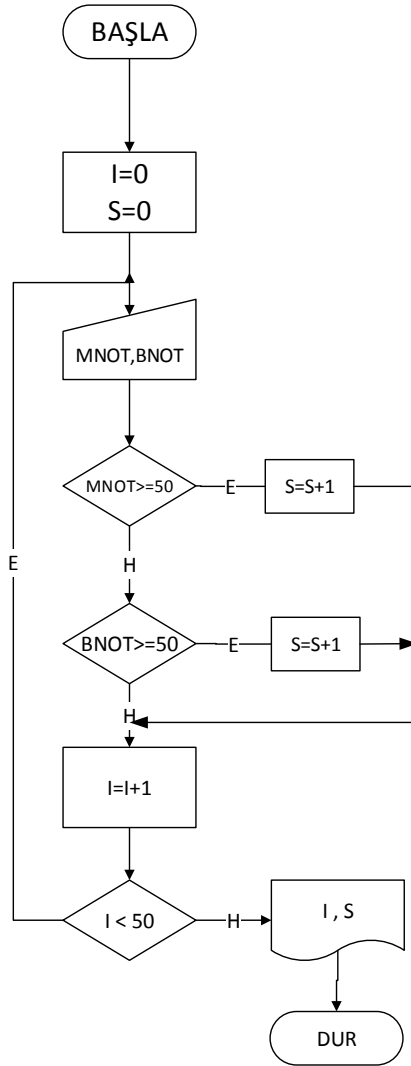
Bilgisayar Notu: B Not

İşlem : (M Not veya B Not) \geq 50 olan öğrenci sayısı

Çıktı : S

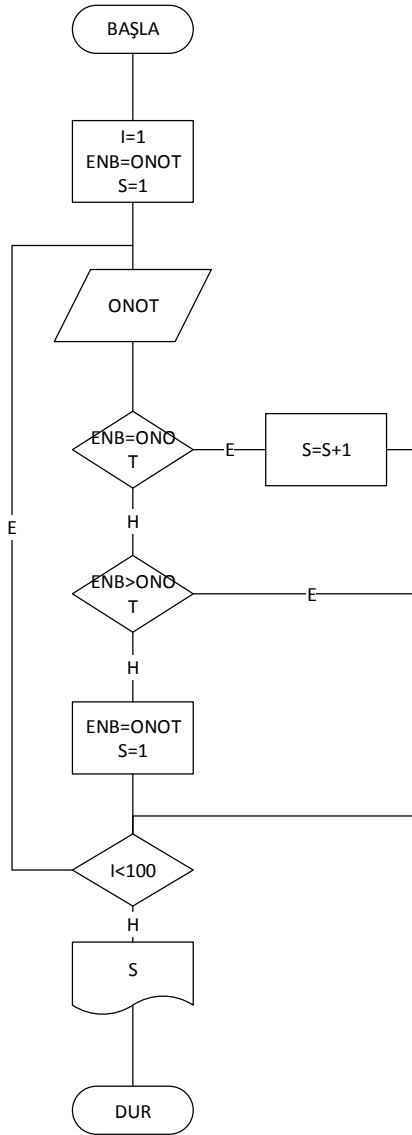
diğer değişkenler I: Sayaç

S: \geq 50 olan öğrenci sayısı

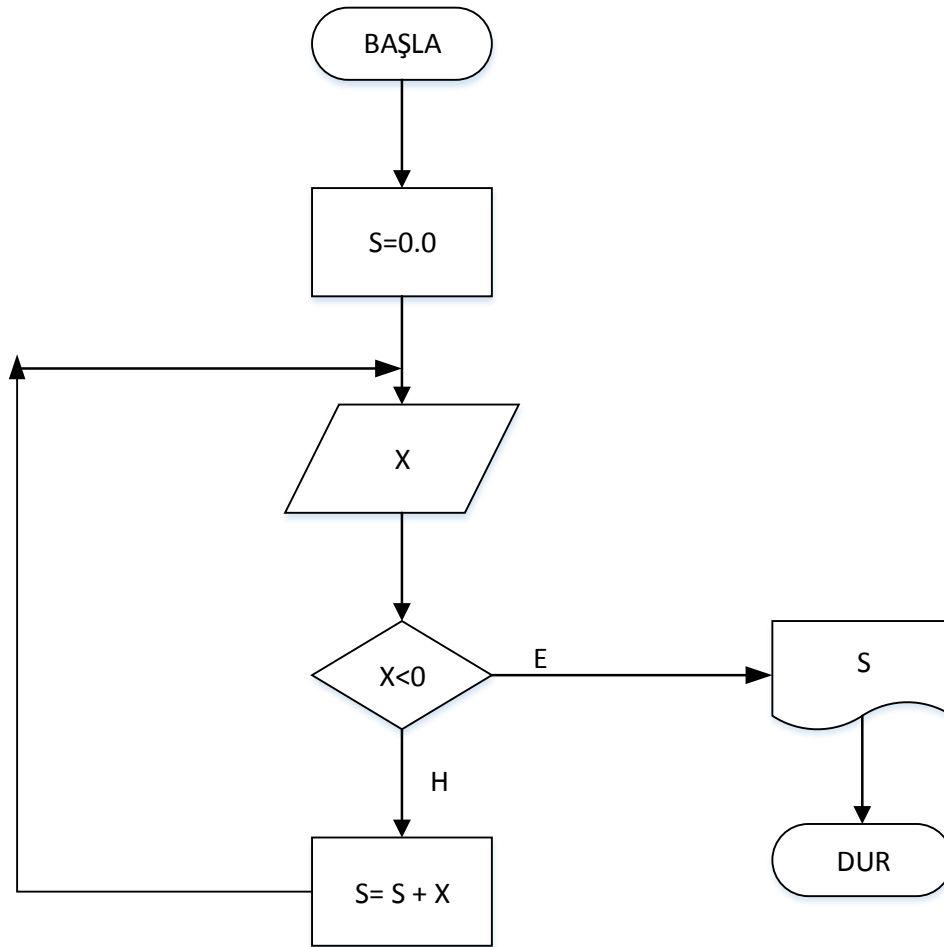


I	M NOT	B NOT	S
1	50	40	1
2	50	60	2
3	30	20	2
4	60	20	3
47			
48			
49			
50			

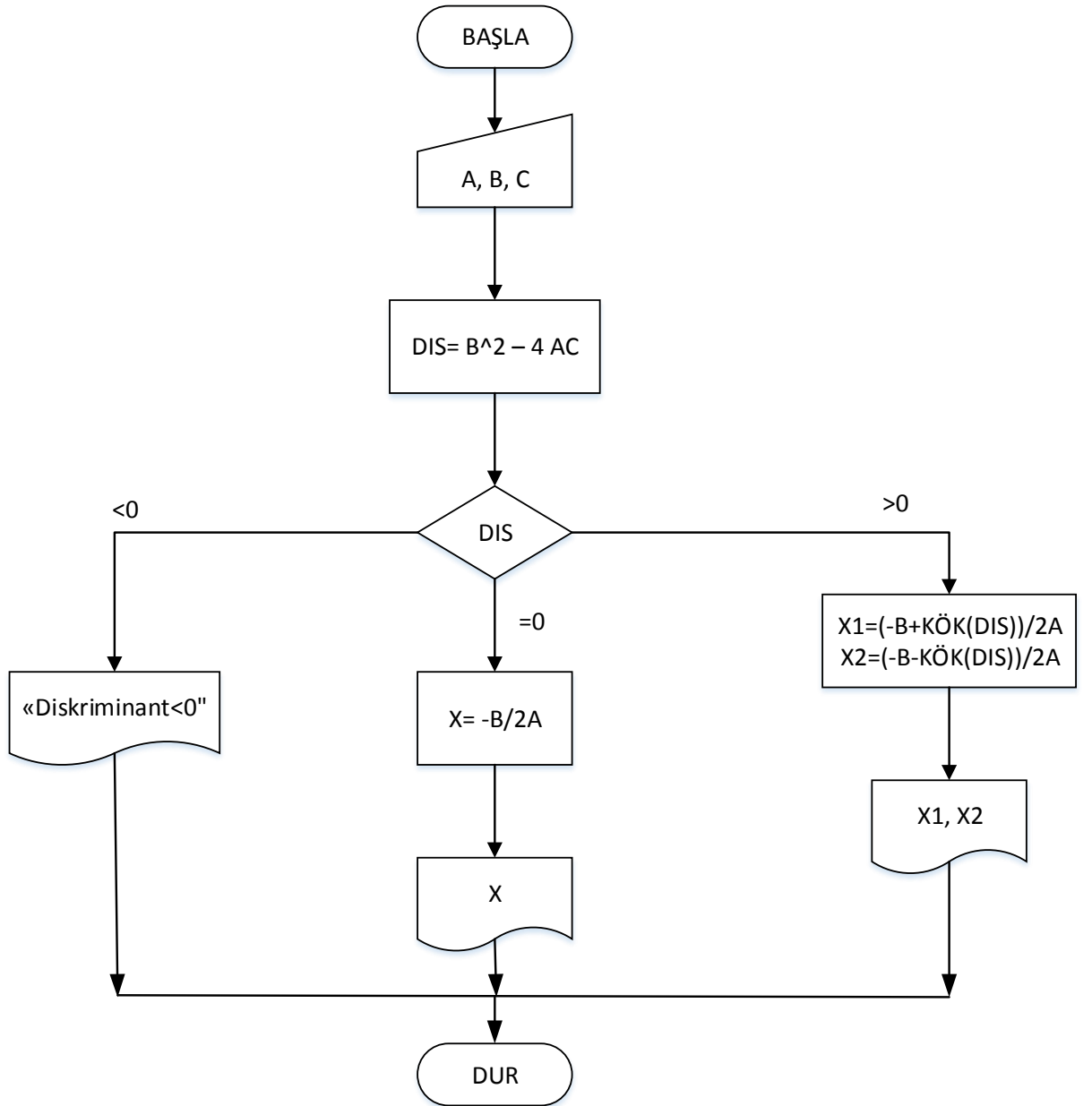
Örnek : 100 öğrenci içinde kaç kişinin en yüksek nota sahip olduğunu bulan program.



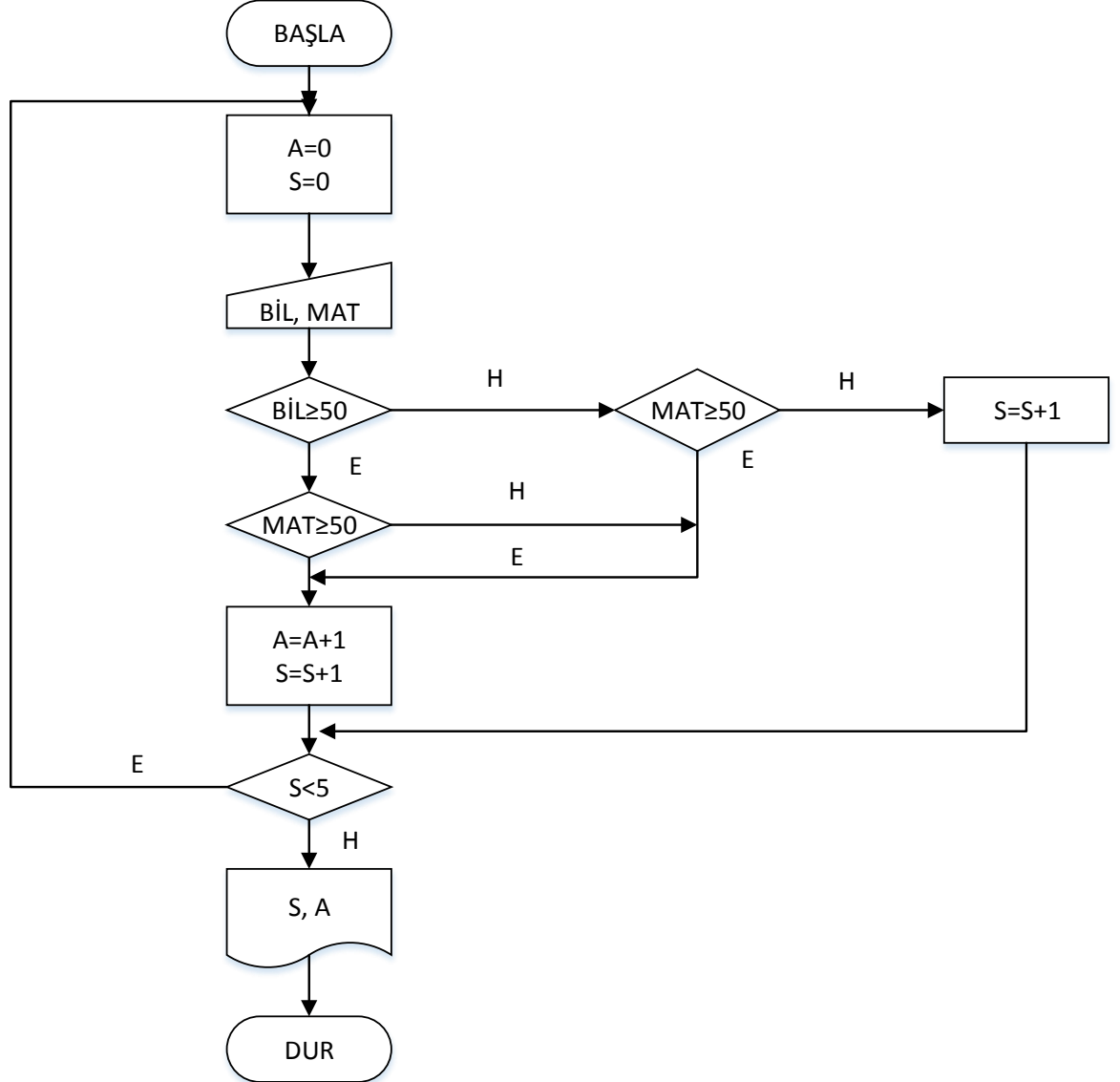
Örnek(): Bilgisayara girilen sayısal değerlerin toplamının elde edilmesi istenmektedir. Girilen değerlerin negatif yani sıfırdan küçük olması durumunda toplama yapılmamakta ve o ana kadar girilen sayıların toplamı yazdırılmakta ve program sona ermektedir. Akış şeması aşağıdaki gibidir.



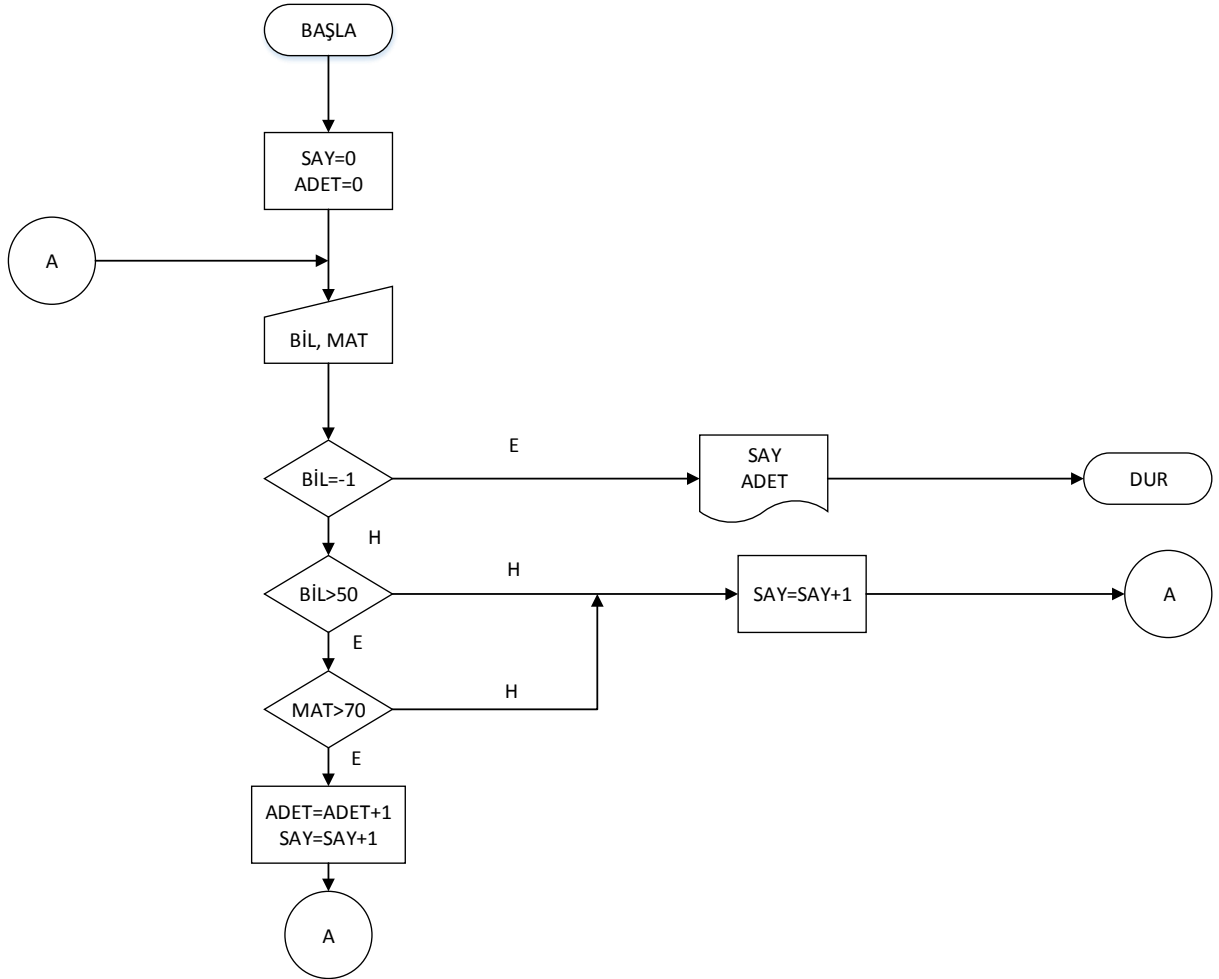
Örnek () : Akış şeması ikinci dereceden bir denklemin köklerini elde edip yazdırmaktadır.



Örnek () : Bir sınıftaki 5 öğrenciye ait bilgisayar ve matematik notları girilmekte, bilgisayar VEYA matematik notu 50’den büyük veya 50’ye eşit olan öğrenci adedinin elde edilip yazdırılması istenmektedir. Bu işlemleri gerçekleştiren akış şeması aşağıdaki gibidir.



Örnek () : Bir sınıftaki öğrencilere ait bilgisayar ve matematik notları girilmekte, bilgisayar notu 50'den VE matematik notu 70'ten büyük öğrenci adedinin elde edilmesi, bu adedin ve toplam öğrenci sayısının yazdırılması istenmektedir. Program bilgisayar notu için -1 değeri girildiğinde yazdırma işlemi gerçekleştirilmekte program durdurulmaktadır. Bu işlemleri gerçekleştiren akış şeması aşağıdaki gibidir.

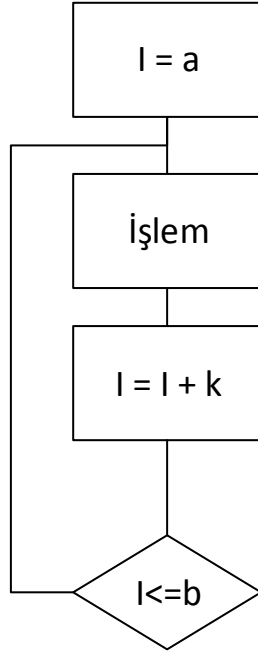


DÖNGÜ (LOOP) /Tekrarlı işlemler

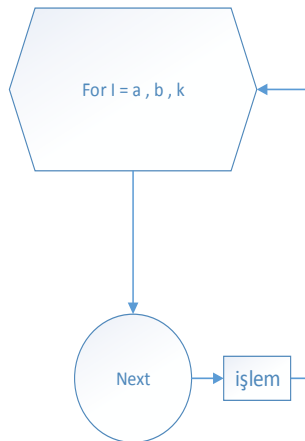
Tekrarlı işlemleri yaptırmak için döngü yapıları kullanılır. Burada 3 çeşit döngü yapısı ele alınacaktır.

a. For-next döngüleri

For-next döngüleri, bu kısma kadar öğrendiğimiz yapılardan aşağıdakilerin yerine geçebilecek şekilde tasarlanmıştır. Birden fazla tekrarlanan işlemleri yapmak için kullanılır.



Aşağıdaki döngü sembolü yukarıdaki sembollerin yerine geçer



I = Sayaç değişkeni, tamsayı

a = sayaç başlangıç değeri, tamsayı

b = sayaç bitiş değeri, tamsayı

k = sayaç artış/azalış değeri, tamsayı

k kullanılmaz ise +1 dir.

For değişken=başlangıçdeğeri **to** bitiş değeri **by/step** artış miktarı

İşlemler

Next

Örnekler :

- çarpım tablosu
- Aritmetik ve geometrik ortalama hesapları
- N faktöryel, Kombinasyon ve permütasyon örnekleri

İç içe birden çok döngü kullanma kuralları :

İçteki döngü sonlanmadan bir dıştaki döngüye geçilemez.

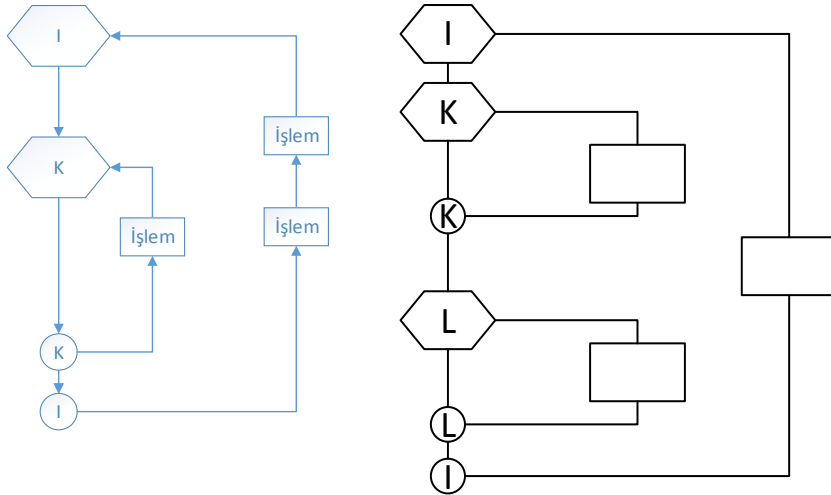
Sayaç değişkenleri farklı olmalıdır

Döngüler birbirini kesmemelidir

Döngü dışından döngü içine, döngü başlangıç deyimi atlanarak doğrudan bir giriş (go to) olamaz ancak içeriden dışarı direk akış (go to ile) olabilir. Bu tür çıkışlarda eğer döngü değişkeni bitiş değerine ulaşmamış ise sayaç/indis değeri bellekte saklı kalır.

Daima içteki döngü dıştaki döngüden hızlı döner, önce içteki döngü tamamlanır sonra dıştaki tamamlanır. (Su veya elektrik sayaç/saatleri gibi)

For-next veya while-do ve repeat-until döngüleri hem kendi içlerinde hem de birinin içinde diğeri olmak kaydı ile iç içe çalışabilirler.



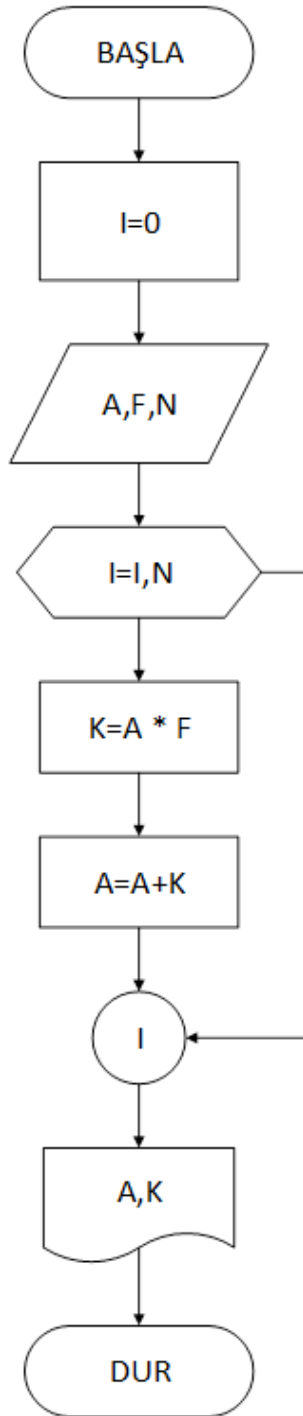
Döngü Sayısı hesaplama :

Bir döngüdeki toplam dönüş sayısı = $(b - a) / k + 1$ şeklinde hesaplanır.

Eğer $(b-a)/k$ tamsayı olarak elde edilmezse, sonucun tam sayı kısmı geçerlidir.

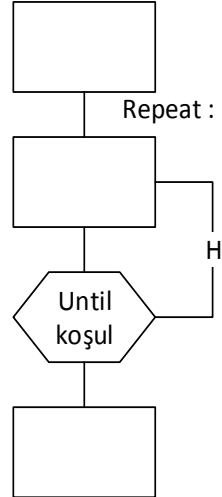
Birden çok iç içe döngülerde toplam döngü sayısı , her döngü sayısının birbiri ile çarpımını sonucu elde edilir

Örnek : Faiz hesabı



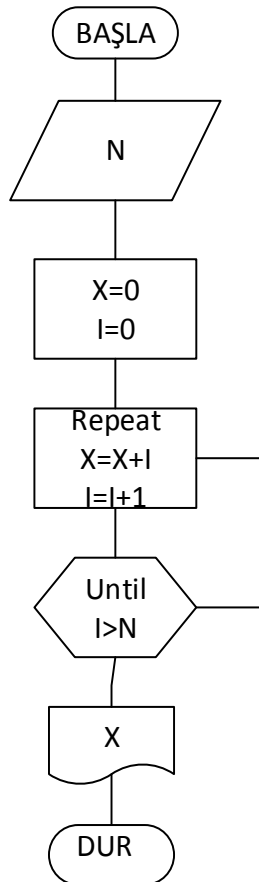
Repeat-until döngüleri

Bu döngüdeki koşul döngü sonunda yer alır. Koşulun hala sağlanıp sağlanmadığına işlemler tamamlandıktan sonra bakılır, eğer koşul sağlanıyorsa döngü dışına çıkılır. Bu döngüler en az bir kez işlenir. Eğer koşul sağlanamaz ise sonsuz döngü oluşur.



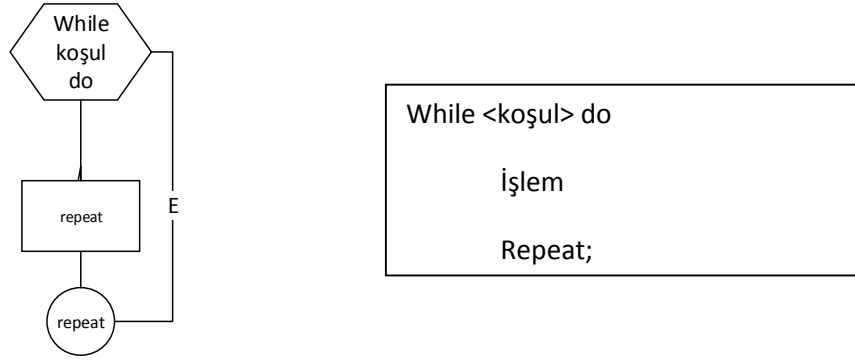
Repeat
İşlem-1
...
İşlem-n
Until <koşul>

Örnek : n adet tamsayının toplamı

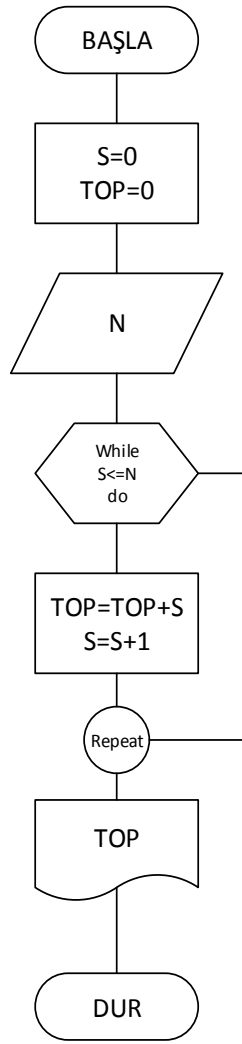


b. **Do-while döngüsü**

Bu tip döngülerde koşul döngü başında yer alır. Koşul doğru olduğu sürece döngü içinde işlemler tekrarlanır, koşul ilk yanlış olduğunda döngüden çıkılır.



Örnek : n adet tamsayı toplamını bulan program.



Örnek : 10 metre yükseklikten bir top yere bırakılıyor, top her zıplayışında bir evvelki yüksekliğin %75 ine erişebiliyor. Topun kaç zıplayış sonunda 1 metrenin altına ineceğini hesaplayan algoritma.

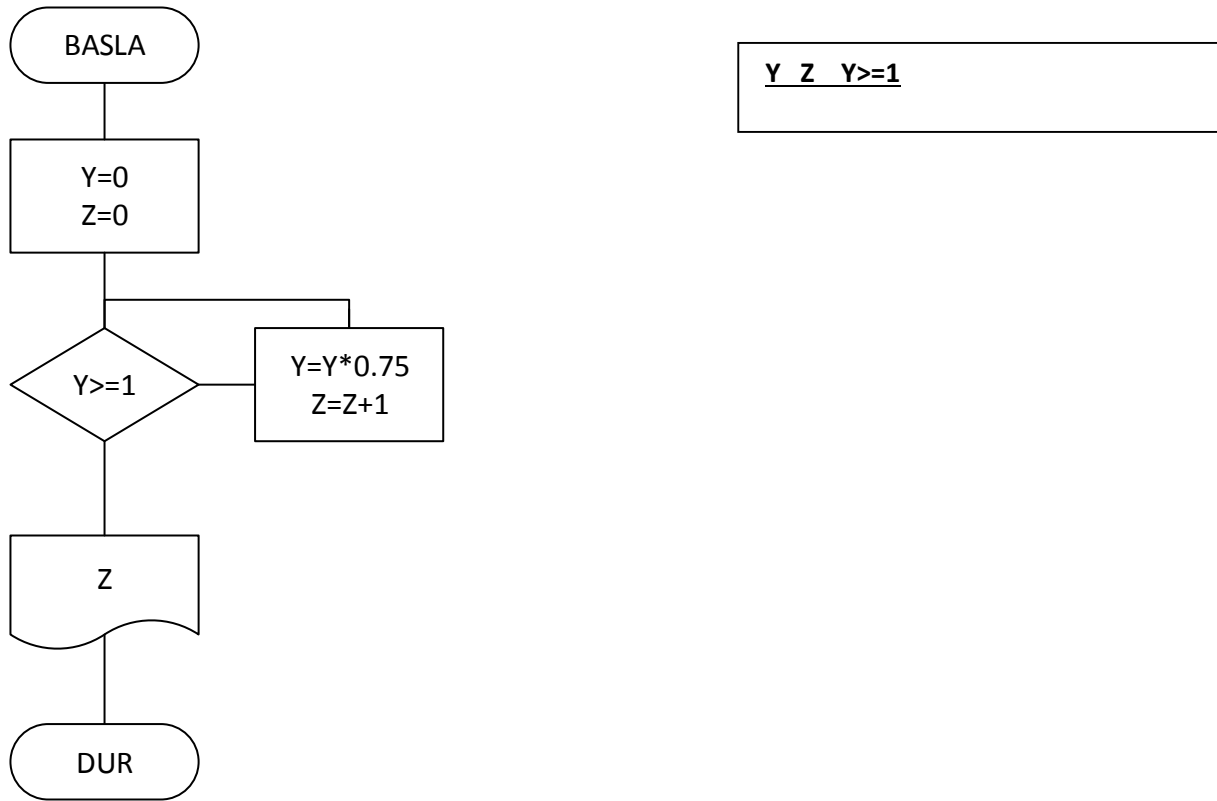
Girdi : yok

Değişkenler : Y ; yükseklik

Z : zıplama sayısı

$Y \geq 1$ evet ise $Y = Y * 0.75$: $Z = Z + 1$

Hayır ise Z yazdır



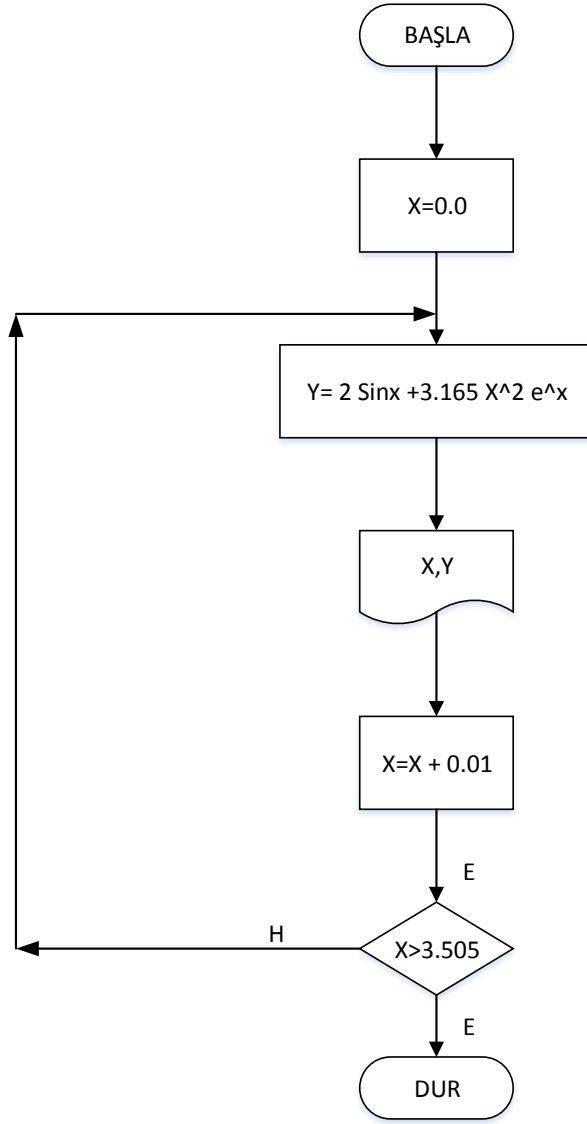
Not : bilgisayara girilen N metre için genelle

Örnek : 1- yukarıdaki algoritmayı repeat-until ile hazırla ve iz tablosu oluştur

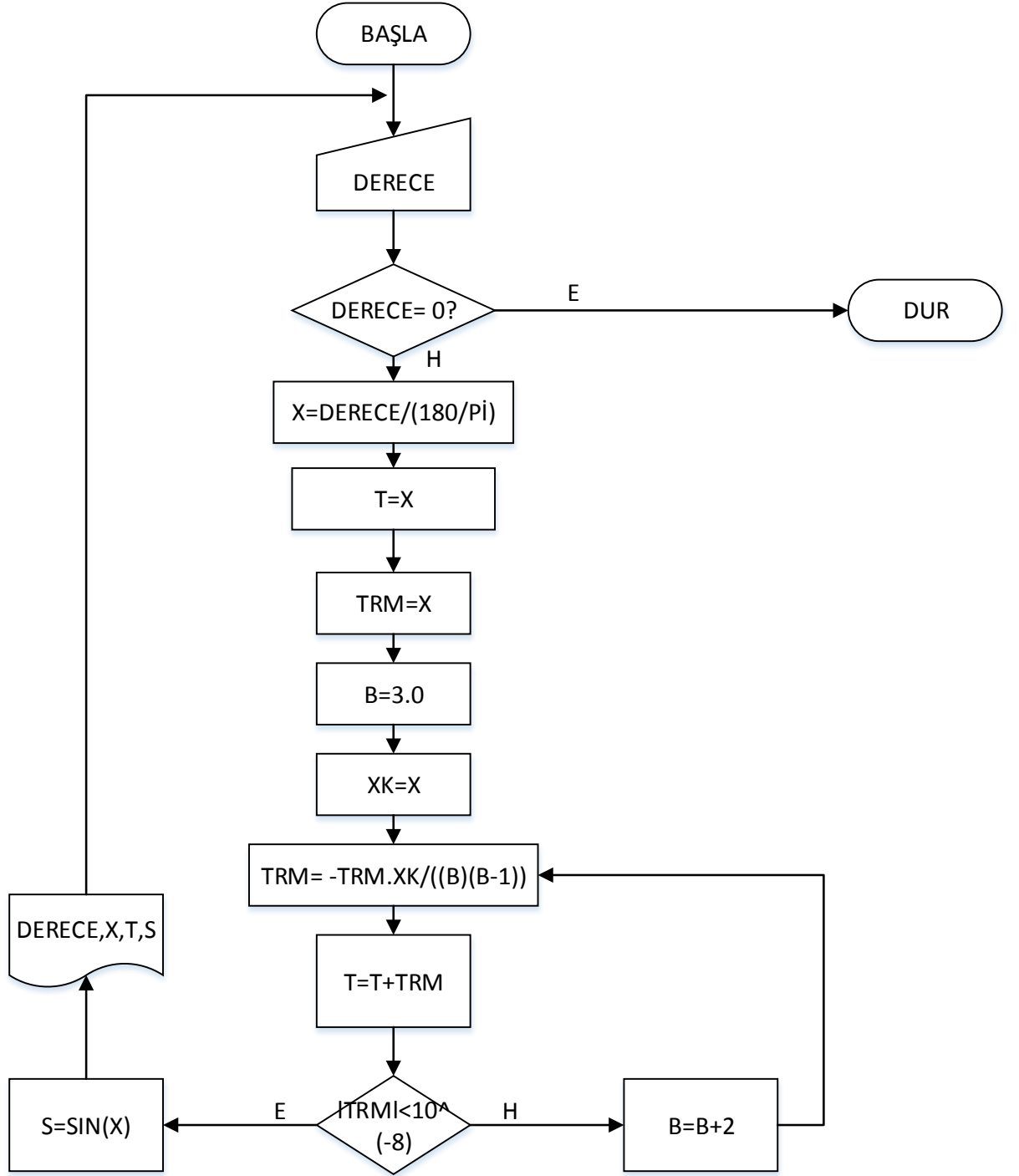
2- yukarıdaki algoritmayı while-do ile hazırla ve iz tablosu oluştur

3- Alınan bir borcun aylık bileşik faiz ödemeleri ile (her taksit= ana paranın bir kısmı ve kalan faizin bir kısmı) olmaz üzere, aylık ödeme miktarına bağlı ödeme sayısını bulan program.

Örnek (şe4) : Akış şeması yardımıyla $x = (0.01) 3.5$ için $y = 2 \sin x + 3.165x^2 e^x$ fonksiyonunun değeri elde dilmekte, x ve y değerleri yazdırılmaktadır.



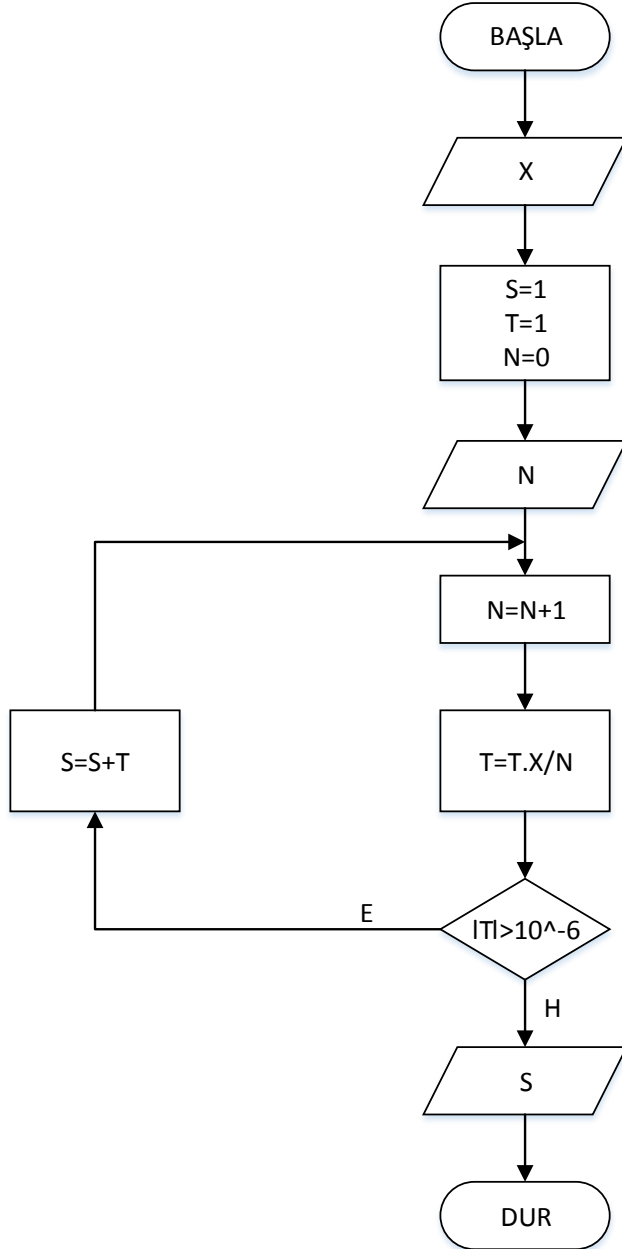
Örnek (ş6) : Sinüs fonksiyonunun Taylor açılımı $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$ gibidir. Akış şeması Taylor seri açılımını kullanarak girilen bir derecenin Sinüsünü elde edip yazdırmaktadır.



Örnek (şe7) : Akış şeması verilen bir x değeri için e^x değerini;

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

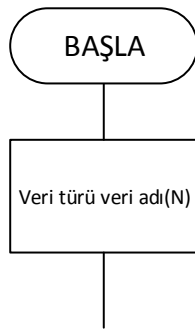
Seri açılımını kullanarak elde etmektedir.



BİRDEN ÇOK BOYUTLU (DİZİNLİ) DEĞİŞKENLER

Birden çok boyutlu değişken kullanımı matematikteki vektör veya matris yapılarına benzer olan yapılarda kullanılır. Çok boyutlu değişkenler kullanılırken aşağıdaki kurallara uyulmalıdır.

1. Boyutlu değişken program içerisinde ilk kullanımından önce, genellikle program başlangıcında programlama diline uygun bir syntax ile tanımlanır.
2. Boyutlu değişkenin tanımlanması aşamasında, değişkenin boyut sayısı ve her boyutta kullanılacak en büyük eleman sayısı tanımlanır
3. Gerektiğinde boyut indisi değişken olarak kullanılabilir.



$X(N) =$

1	2											N
---	---	--	--	--	--	--	--	--	--	--	--	---

$X(M,N) =$

1,1	1,2											1,N
M,1	M,2											M,N

$X(M,N,K,...) =$ ikiden çok boyutlu dizinli veri yapıları da kullanılabilir.

Örnek : bilgisayara girilen N sayısı kadar, X boyutlu değişkenine veri girilen ve aritmetik ortalama , varyansı hesaplayan bir algoritma.

Girdi : N veri sayısı

Değişkenler : X(N) n boyutlu değişken

İşlem : $topx = topx + X(I)$

$Topxkare = topxkare + X(I) * X(I)$

Çıktı : ort, varyans

