2. BÖLÜM

Problem Çözme ve Algoritmalar

Problem Çözme

Problem Çözme Tekniği (Descartes'e göre):

1. Doğruluğu kesin olarak kanıtlanmadıkça, hiçbir şeyi doğru olarak kabul etmeyin; tahmin ve önyargılardan kaçının.
2. Karşılaştığınız her güçlüğü mümkün olduğu kadar çok parçaya bölün.
3. Düzenli bir biçimde düşünün; anlaşılması en kolay olan şeylerle başlayıp yavaş yavaş daha zor ve karmaşık olanlara doğru ilerleyin.
4. Olaya bakışınız çok genel, hazırladığınız ayrıntılı liste ise hiçbir şeyi dışarıda bırakmayacak kadar kusursuz ve eksiksiz olsun.

Problem çözme sırası

1. **Problemi anlama** (Understanding, Analyzing),
2. **Bir çözüm yolu geliştirme** (Designing),
3. **Algoritma ve program yazma** (Writing),
4. **Tekrar tekrar test etme** (Reviewing)

Problem Çözme

Bir problemi çözmek için yazılacak programda, genel olarak, aşağıdaki yazılım geliştirme aşamaları uygulanmalıdır.

Yazılım Geliştirme Aşamaları

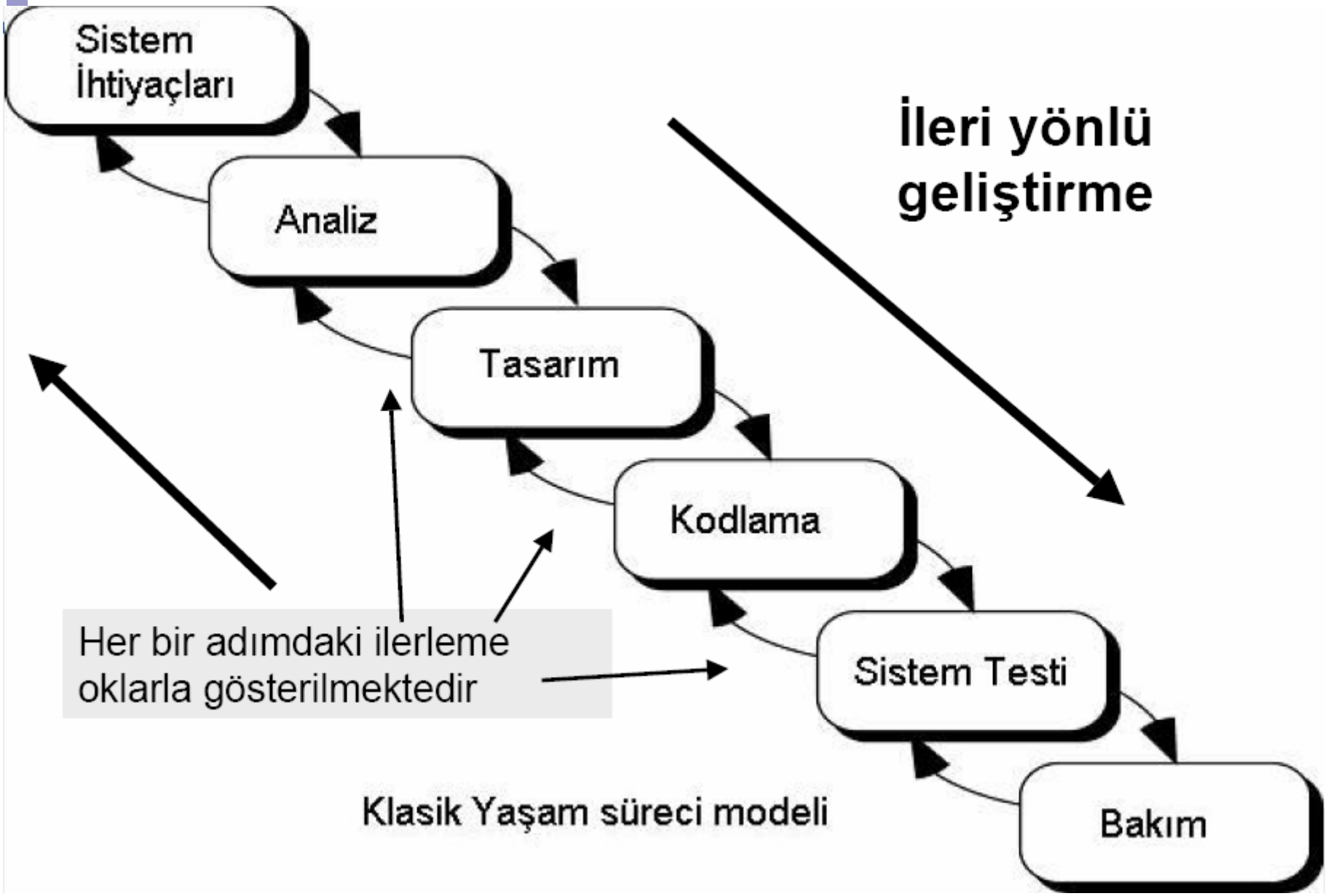
1. Problemin Analizi: Problemin tam olarak ne olduğunun anlaşılmasıdır. Bu nedenle, problemin çözümünden neler beklendiği ve yaratacağı çözümün girdi ve çıktılarının neler olacağı kesin olarak belirlenmelidir.
2. Tasarım: Problemi çözmek için kullanılacak çözüm adımlarını gösteren bir liste yapılması gereklidir. Bir problemin çözüm adımlarını gösteren bu listeye algoritma denir. Böyle bir liste tasarlanırken, ilk önce problemin ana adımları çıkarılır; daha sonra her adım için, gerekiyorsa, daha ayrıntılı bir çözüm tasarlanır.
3. Kodlama: Kağıt üzerinde geliştirilen algoritma, programcının tercih ettiği bir programlama diline çevrilir.

Problem Çözme

4. Test etme: Program değişik girdiler ile çalıştırılarak ürettiği sonuçlar kontrol edilerek test işlemi gerçekleştirilir. Sonuçlar beklendiği gibi ise , programın doğru çalıştığı kanıtlanmış olunur; değilse doğru çalışmayan kısımları tespit edilerek düzeltilir.

5. Belgeleme: Bütün bu çalışmaların belli bir dosyalama sistemi içinde belgeler halinde saklanmasının sağlandığı aşamadır.

6. Bakım: Programın güncel koşullara göre yeniden düzenlenmesini içeren bir konudur. Oluşan hataların giderilmesi,, yeni eklemeler yapılması ya da programın teknolojisinin yenilenmesi gibi işlemlerdir.



Klasik Yaşam süreci modeli

Problem Çözme

Bir problem çözülürken biri algoritmik, diğeri herustic(sezgisel) olarak adlandırılan iki yaklaşım vardır.

Algoritmik yaklaşımda, çözüm için olası yöntemlerden en uygun olanı seçilir ve yapılması gerekenler adım adım ortaya konulur.

Herustic yaklaşımda ise, çözüm açıkça ortada değildir. Tasarımcının deneyimi, birikimi ve o andaki düşüncesine göre problemi çözecek bir şeylerin şekillendirilmesiyle yapılır. Program tasarımcısı, algoritmik yaklaşımla çözemediği, ancak çözmek zorunda olduğu problemler için bu yaklaşımı kullanır.

Algoritmik Yaklaşım

Algoritma, herhangi bir sorunun çözümü için izlenecek yol anlamına gelmektedir. Çözüm için yapılması gereken işlemler hiçbir alternatif yoruma izin vermeksizin sözel olarak ifade edilir. Diğer bir deyişle algoritma, verilerin, bilgisayara hangi çevre biriminden girileceğinin, problemin nasıl çözüleceğinin, hangi basamaklardan geçirilerek sonuç alınacağının, sonucun nasıl ve nereye yazılacağının sözel olarak ifade edilmesi biçiminde tanımlanabilir.

Algoritma hazırlanırken, çözüm için yapılması gerekli işlemler, öncelik sıraları gözönünde bulundurularak ayrıntılı bir biçimde tanımlanmalıdırlar.

Örnek 1: Verilen iki sayının toplamının bulunmasının algoritması aşağıdaki gibi yazılır.

Adım 1 – Başla
Adım 2 – Birinci Sayıyı Oku
Adım 3 – İkinci Sayıyı Oku
Adım 4 – İki Sayıyı Topla
Adım 5 – Dur

Algoritmik Yaklaşım

Algoritmalar iki farklı şekilde kağıt üzerinde ifade edilebilirler;

1. Pseudo Code (*Kaba Kod* veya *Yalancı Kod* veya *Sözde Kod*), bir algoritmanın yarı programlama dili kuralı, yarı konuşma diline dönük olarak ortaya koyulması/ tanımlanmasıdır. Bu şekilde gösterim algoritmayı genel hatlarıyla yansıtır.

2. Akış şeması, algoritmanın görsel/şekilsel olarak ortaya koyulmasıdır. Problemin çözümü için yapılması gerekenleri, başından sonuna kadar, geometrik şekillerden oluşan simgelerle gösterir.

Sözde (Pseudo) Kod

Sözde programlar, doğrudan **konuşma dilinde** ve programlama mantığı altında, **eğer, iken** gibi koşul kelimeleri ve **> = <** gibi ifadeler ile beraber yazılır. İyi bir biçimde yazılmış, sözde koddan, programlama diline kolaylıkla geçilebilir.

Örnek: Verilen bir sıcaklık derecesine göre suyun durumunu belirten bir sözde program yazınız.

•Örnek Giriş/Çıkış

–Bu Program, Sıcaklığa göre suyun durumunu gösterir

–Su, Buz, Buhar

–----------

–Lütfen derece cinsinden sıcaklığı giriniz: 140

–BUHAR elde edeceksiniz.

Sözde (Pseudo) Kod

İstenilen programın Pseudo Kodu:

1. Program açıklama mesajı yaz.
2. Kullanıcın sıcaklığı girmesi için bir uyarı mesajı yaz.
3. Girilen Sıcaklığı Oku.
5. Eğer Sıcaklık < 0 ise Durum="Buz"
6. Eğer Sıcaklık>= 100 ise Durum="Buhar"
7. Değilse Durum ="Su"
8. Sonucu Yaz.

Akış Diyagramları (Şemaları)

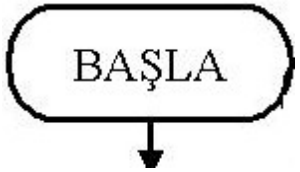
Algoritmanın, görsel olarak simge ya da sembollerle ifade edilmiş şekline “akış şemaları” veya FLOWCHART adı verilir. Akış şemalarının algoritmadan farkı, adımların simgeler şeklinde kutular içine yazılmış olması ve adımlar arasındaki ilişkilerin ve yönünün oklar ile gösterilmesidir.

Programın saklanacak esas belgeleri olan akış şemalarının hazırlanmasına, sorun çözümlenmesi sürecinin daha kolay anlaşılır biçime getirilmesi, iş akışının kontrol edilmesi ve programın kodlanmasının kolaylaştırılması gibi nedenlerle başvurulur. Uygulamada çoğunlukla, yazılacak programlar için önce programın ana adımlarını (bölümlerini) gösteren genel bir bakış akış şeması hazırlanır. Daha sonra her adım için ayrıntılı akış şemalarının çizimi vardır.

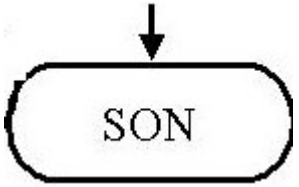
En basit şekliyle dikdörtgen kutulardan ve oklardan oluşur. Akış şeması sembolleri ANSI (American National Standards Institute) standardı olarak belirlenmiş ve tüm dünyada kullanılmaktadır.

Akış Diyagramları (Şemaları)

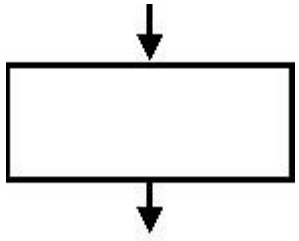
Her simge, yapılacak bir işi veya komutu gösterir. Akış şemalarının hazırlanmasında aşağıda yer alan simgeler kullanılır.



Bir algoritmanın başladığı konumu göstermektedir. Tek çıkışlı bir şekildir.

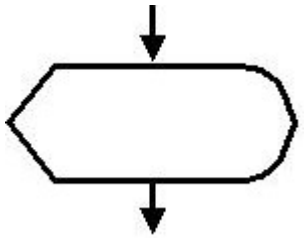


Bir algoritmanın bittiği konumu göstermektedir. Tek girişli bir şekildir.

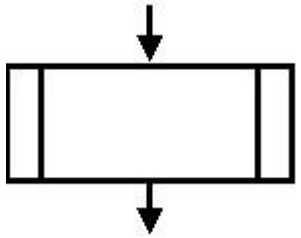


Bir algoritmada aritmetik işlem yapılmasını sağlayan şekildir. Bu dörtgen kutu içerisine yapılmak istenen işlem yazılır. Tek girişli ve tek çıkışlı bir şekildir.

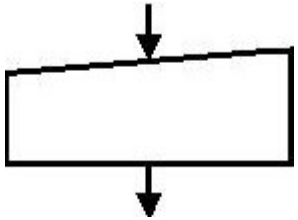
Akış Diyagramları (Şemaları)



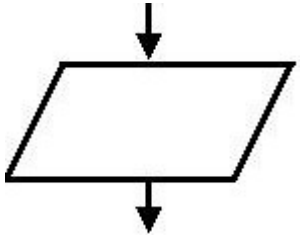
Algoritmada bir bilginin ekrana yazılacağı konumu gösteren şekildir. Ekrana yazılacak ifade ya da değişken bu şekil içerisine yazılır.



Bir algoritmada başka bir yerde tanımlanmış bloğun yerleştiği konumu gösteren şekildir. Kutu içerisine bloğun adı yazılabilir.

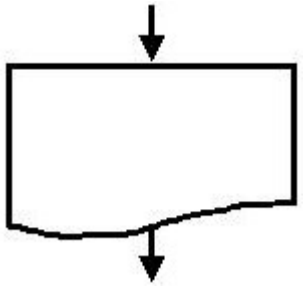


Klavyeden Bilgisayara bilgi girilecek konumu belirten şekildir. Girilecek bilginin hangi değişkene okunacağını kutu içerisine yazabilirsiniz.

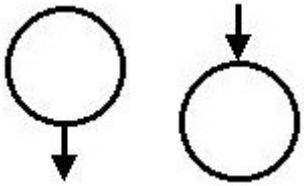


Giriş Çıkış komutunun kullanılacağı yeri belirler ve kutu içerisine hangi değişkeni ve OKUma mı YAZ mı yapılacağını belirtmeniz gerekir.

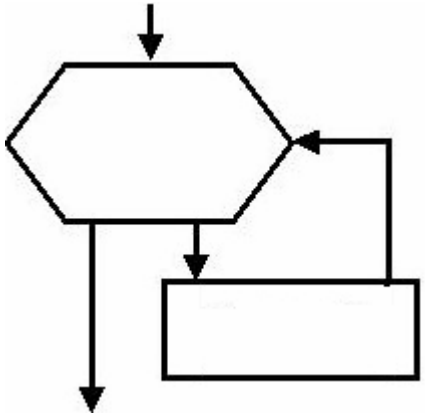
Akış Diyagramları (Şemaları)



Bilginin Yazıcıya yazılacağı konumu gösteren şekildir.

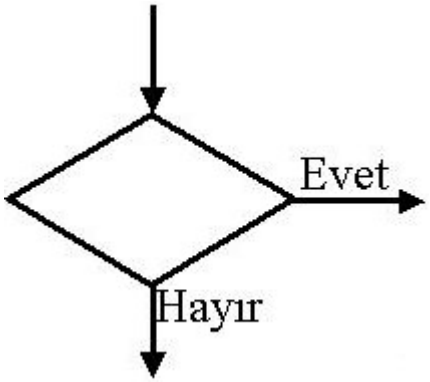


Bir algoritmanın birden fazla alana yayılması durumunda bağlantı noktalarını gösteren şekildir. Tek girişli veya tek çıkışlı olarak kullanılırlar.



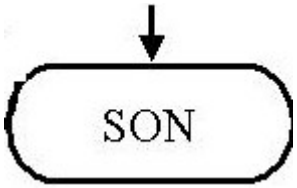
Birçok programda; belirli işlem blokları ardışık değerlerle veya bazı koşullar sağlanıncaya kadar tekrarlanır. Bu tekrarlamalı işlemler döngü olarak isimlendirilir. Döngü şeklinin içine; döngü (çevrim, kontrol) değişkeni, başlangıç değeri, bitiş değeri ve adımı yazılır.

Akış Diyagramları (Şemaları)



Bir algoritmada bir kararın verilmesini ve bu karara göre iki seçenekten birinin uygulanmasını sağlayan şekildir. burada eşkenar dörtgen içerisine kontrol edilecek mantıksal koşul yazılır. Program akışı sırasında koşulun doğru olması durumunda "Evet" yazılan kısma Yanlış olması durumunda "Hayır" yazılan kısma sapılır. Tek girişli ve çift çıkışlı bir şekildir.

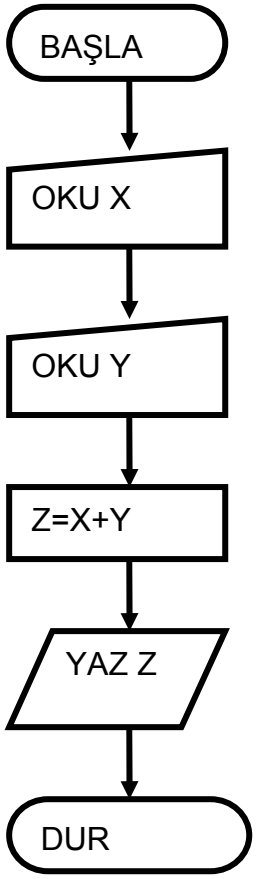
Akış Çubuğu



Programın bittiği yer ya da yerleri gösteren bir şekildir.

Akış Diyagramları (Şemaları)

Klavyeden girilen İki sayının toplamını hesaplayıp yazan pseudo kod ve akış şemasını hazırlayınız.



(X: Birinci sayı, Y: İkinci sayı, Z: toplam)
A1 : Başla
A2 : Klavyeden oku X
A3 : Klavyeden oku Y
A4 : Hesapla Z = X + Y
A5 : Yaz Z
A6 : Dur

Algoritmalarda Kullanılan Operatörler

- İşlemleri belirten sembollere bilgisayar dilinde “operatör” denir.
- Algoritmada kullanılan operatörler Tabloda verilmiştir.

Matematiksel İşlem oper	
Üs alma	^
Çarpma	*
Bölme	/
Toplama	+
Çıkarma	-
Tam ve onda ayırma	.
Mantıksal İşlem Operatörleri	
Değil	‘
Ve	.
Veya	+

Karşılaştırma Operatörleri	
Eşittir	=
Eşit değildir	<>
Küçüktür	<
Büyüktür	>
Büyük eşittir	>=
Küçük eşittir	<=
Genel İşlem Operatörleri	
Aktarma	=
Parantez	()

Algoritmalarda Kullanılan Terimler

- Tanımlayıcı
- Değişken
- Sabit
- Aktarma
- Sayaç
- Döngü
- Ardışık Toplama
- Ardışık Çarpma

Tanımlayıcı

- Programcı tarafından oluşturulur.
- Programdaki değişkenleri,sabitleri, kayıt alanlarını, özel bilgi tiplerini vb adlandırmak için kullanılan kelimeler
- Tanımlayıcılar, yerini tuttukları ifadelere çağrışım yapacak şekilde seçilmelidir.
- İngiliz alfabesindeki A-Z veya a-z arası 26 harften
- 0-9 arası rakamlar kullanılabilir
- Sembollerden sadece alt çizgi (_) kullanılabilir.
- Tanımlayıcı isimleri harfle veya alt çizgiyle başlayabilir.
- Tanımlayıcı ismi,rakamla başlayamaz veya sadece rakamlardan oluşamaz.

Algoritmalarda Kullanılan Terimler

Değişken

- Programın her çalıştırılmasında, farklı değerler alan bilgi/bellek alanlarıdır.
- Değişken isimlendirilmeleri, yukarıda sayılan tanımlayıcı kurallarına uygun biçimde yapılmalıdır.

Örneğin ;

Bir ismin aktarıldığı değişken ; **ad**
Bir isim ve soy ismin aktarıldığı değişken; **adsoyad**
Ev telefon no sunun aktarıldığı değişken; **evtel**
Ev adresinin aktarıldığı değişken; **evadres**
İş adresinin aktarıldığı değişken; **isadres**

Sabit

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

Algoritmalarda Kullanılan Terimler

Örnek Algoritma

Başla
Bir isim giriniz(A)
“İlk algoritmama Hoş geldin” mesajı (B)
B VE A yı Yaz
Dur

- Yukarıdaki algoritma, dışarıdan girilen bir A değişkeni ile B sabitini birleştirip ekrana yazar.

A değişkeni	*B sabiti*	*Sonuç*
Onur	İlk Algoritmama Hoş geldin	İlk Algoritmama Hoş geldin Onur
Emre	İlk Algoritmama Hoş geldin	İlk Algoritmama Hoş geldin Emre

Algoritmalarda Kullanılan Terimler

Aktarma

- Herhangi bir bilgi alanına, veri yazma; herhangi bir ifadenin sonucunu başka bir değişkende gösterme vb görevlerde “aktarma” operatörü kullanılır.

değişken=ifade

- Değişken yazan kısım herhangi bir değişken ismidir.
- İfade yazan kısımda ise matematiksel,mantıksal veya alfa sayısal ifade olabilir.
- = sembolü, aktarma operatörüdür ve sağdaki ifadenin/işlemin sonucunu soldaki değişkene aktarır. Bu durumda değişkenin eğer varsa bir önceki değeri silinir.

1.işlem: sağdaki ifadeyi gerçekleştir veya sağdaki işlemi yap

2.işlem: Bulunan sonucu soldaki değişkene aktar.

Algoritmalarda Kullanılan Terimler

Sayaç

Programlarda bazı işlemlerin belirli sayıda yaptırılması veya işlenen/üretilen değerlerin sayılması gerekebilir.

say=say+1

Bu işlemde sağdaki ifadede değişkenin eski değerine 1 eklenmekte; bulunan sonuç yine kendisine yeni değer olarak aktarılmaktadır.

Bu tür sayma işlemlerine algoritmada sayaç adı verilir.

Sayacın genel formülü;

Sayaç değişkeni=sayaç değişkeni+adım

Örnek; X=X+3

Örnek; S=S-5

Algoritmalarda Kullanılan Terimler

Örnek

- Aşağıdaki algoritmada 1-5 arası sayılar, ekrana yazdırılmaktadır. 1-5 arası sayıları oluşturmak için sayaç(s=s+1) kullanılmıştır.

1. **Başla**
2. **S=0**
3. **Eğer s>4 ise git 7**
4. **S=S+1**
5. **Yaz S**
6. **Git 3**
7. **Dur**

Eski S	Yeni S	Ekrana Yazılan
0	0+1=1	1
1	1+1=2	2
2	2+1=3	3
3	3+1=4	4
4	4+1=5	5

Algoritmalarda Kullanılan Terimler

Döngü

- Bir çok programda bazı işlemler, belirli ardışık değerlerle gerçekleştirilmekte veya belirli sayıda yaptırılmaktadır. Programlardaki belirli işlem bloklarını, verilen sayıda gerçekleştiren işlem akış çevrimlerine “döngü” denir.

- Örneğin 1 ile 1000 arasındaki tek sayıların toplamını hesaplayan programda T=1+3+5 …. yerine 1 ile 1000 arasında ikişer artan bir döngü açılır ve döngü değişkeni ardışık toplanır.

Algoritmalarda Kullanılan Terimler

Örnek

Aşağıdaki algoritmada, 1 ile 10 arası tek sayıların toplamı hesaplanmaktadır.

1. **Başla**
2. **T=0**
3. **J=1**
4. **Eğer j>10 ise git 8**
5. **T=T+J**
6. **J=J+2**
7. **Git4**
8. **Dur**

DÖNGÜ

Eski J	Eski T	Yeni T	Yeni J
1	0	0+1=1	3
3	1	1+3=4	5
5	4	4+5=9	7
7	9	9+7=16	9
9	16	16+9=25	11
11	-	-	-

Algoritmalarda Kullanılan Terimler

Ardışık Toplama

Programlarda, aynı değerin üzerine yeni değerler eklemek için kullanılır.

Toplam değişkeni=Toplam değişkeni + Sayı

Örnek: Klavyeden girilen 5 sayısının ortalamasını bulan programın algoritması.

1. Başla
2. T=0
3. S=0
4. Eğer S>4 ise git 9
5. S=S+1
6. Sayıyı (A) gir
7. T=T+A
8. Git 4
9. Ortalama=T/5
10. Yaz Ortalama
11. Dur

Algoritmalarda Kullanılan Terimler

Ardışık Çarpma

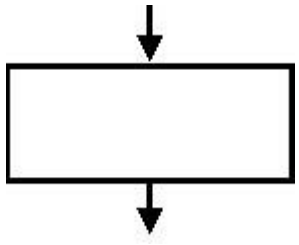
Ardışık veya ardışıl çarpma işleminde; aynı değer, yeni değerlerle çarpılarak eskisinin üzerine aktarılmaktadır.

Çarpım değişkeni=Çarpım değişkeni * Sayı

Örnek: Klavyeden girilen N sayısının faktöriyelini hesaplayan programın algoritmasını yazınız.

1. Başla
2. N sayısını gir
3. Fak=1
4. S=0
5. Eğer S>N-1 ise git 9
6. S=S+1
7. Fak=Fak*S
8. Git 5
9. Yaz Fak
10. Dur

Akış Diyagramlarında Kullanılan Temel Şekiller



Programın çalışması sırasında yapılacak işlemleri ifade etmek için kullanılan şekildir. İçine işlem cümleleri/ifadeleri aynen yazılır. Program akışı buraya geldiğinde, şeklin içerisindeki yazılı işlem gerçekleştirilir. Birden fazla işlem; aynı şekil içinde, aralarına virgül konularak veya alt alta yazılarak gösterilebilir.

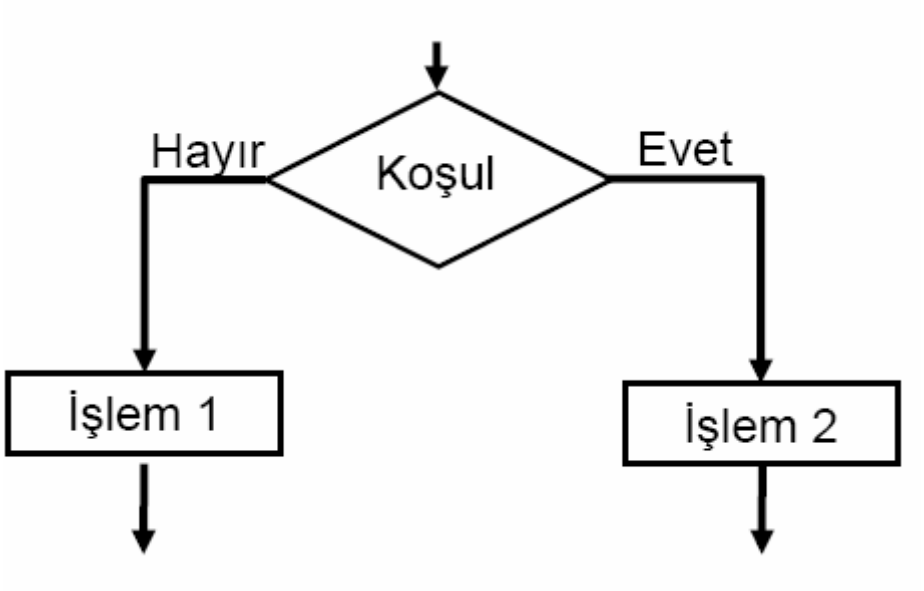
C=(a^2+b^2)^(1/2)

Örnek: İşlem akışı bu şekle gelince, program $c = \sqrt{a^2 + b^2}$ işlemini yapar. İfadedeki ‘a’ ve ‘b’ daha önceki adımlarda girilmiş olan değerlerdir.

Akış Diyagramlarında Kullanılan Temel Şekiller

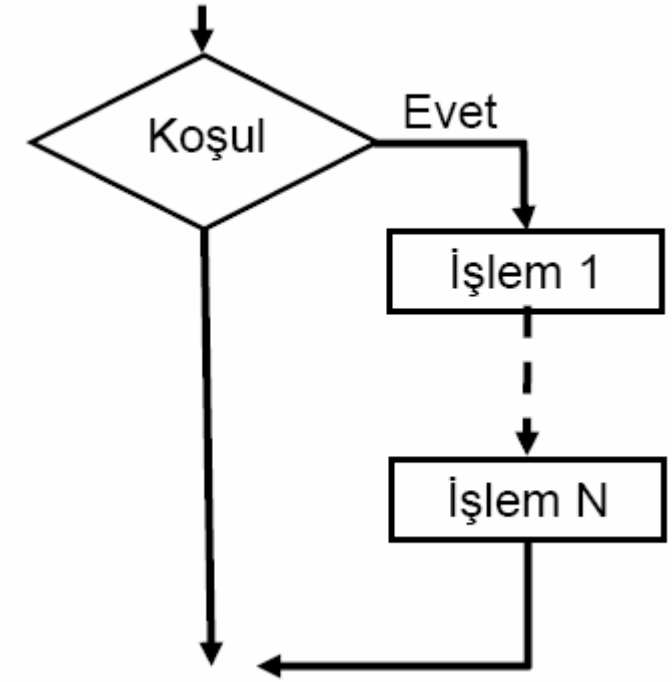
Karar Verme

1. Durum:



a) Koşulun durumuna bağlı olarak 2 farklı işlem vardır.

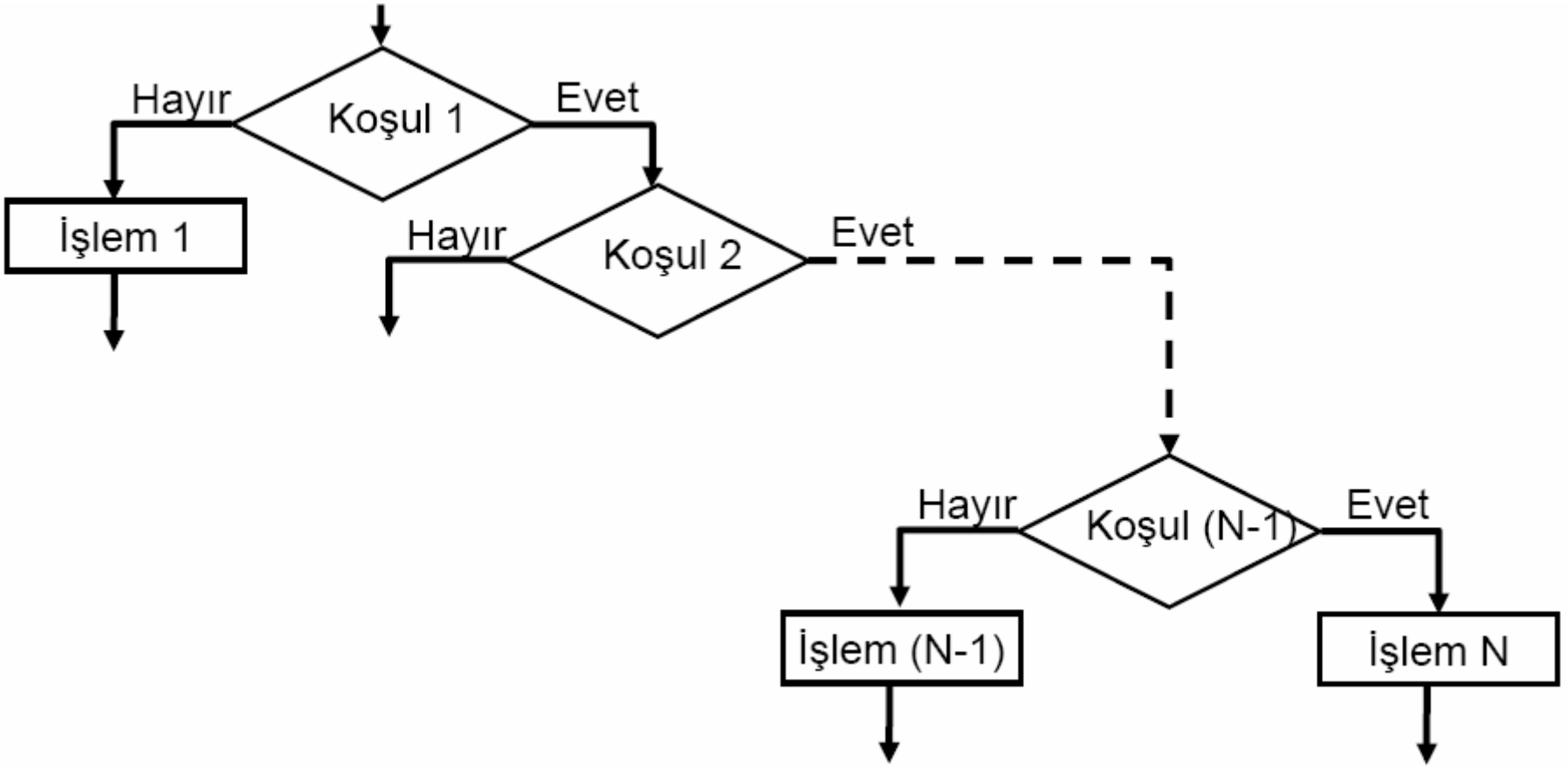
2. Durum:



b) Olumsuz koşulda yapılacak işlem yoktur; olumlu olması durumunda ise N adet işlem yapılacaktır.

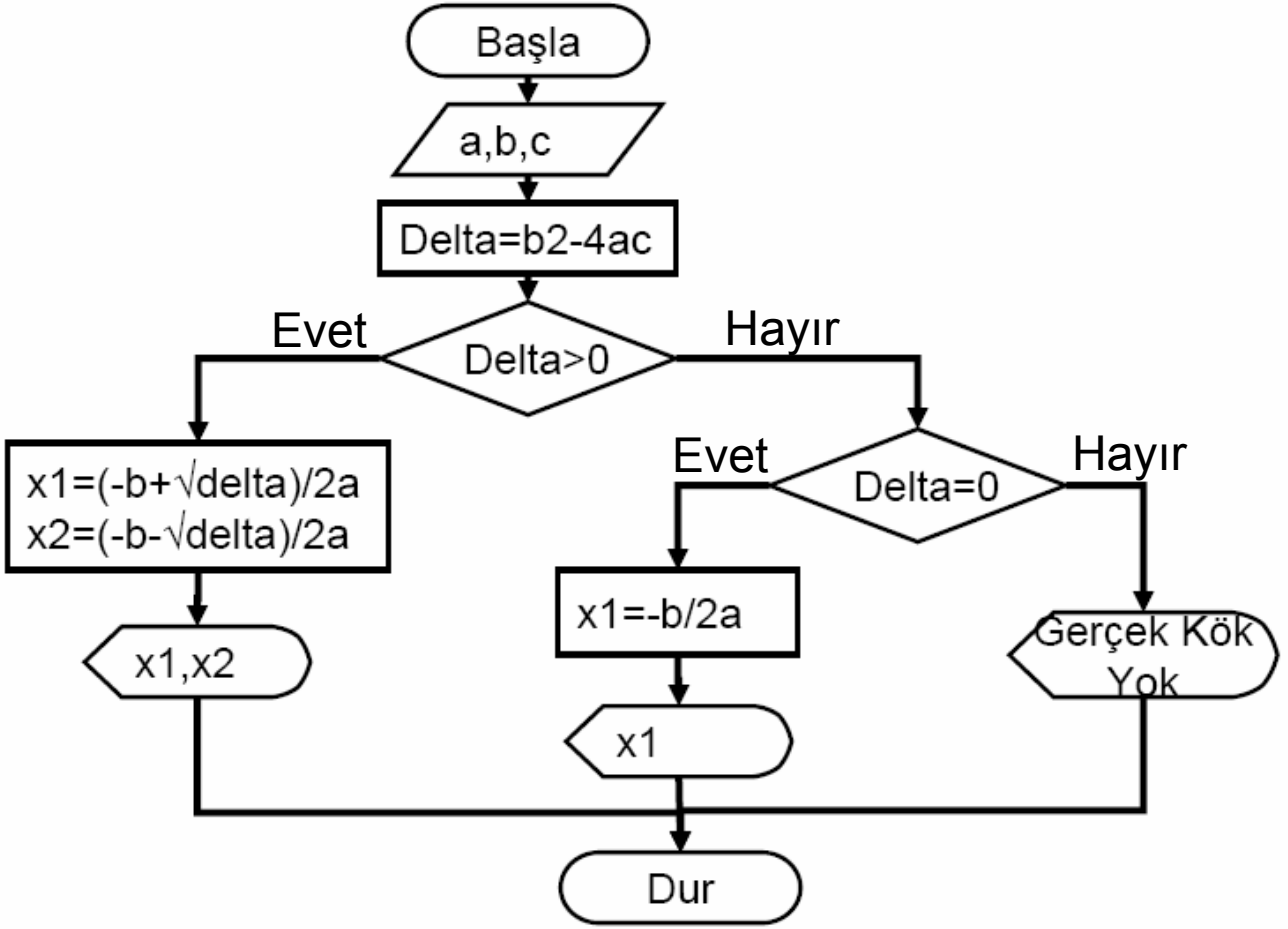
Akış Diyagramlarında Kullanılan Temel Şekiller

Bu yapıyı art arda birden çok kez kullanıp aşağıdaki gibi bir kaşılaştırma dizisi oluşturulabilir.



Akış Diyagramlarında Kullanılan Temel Şekiller

Örnek: ax2+ bx + c = 0 şeklindeki ikinci dereceden bir denklemin köklerini bulan algoritmayı tasarlayıp akış şeması ile gösteriniz.



Akış Diyagramlarında Kullanılan Temel Şekiller

Döngü Yapısı

Bu yapı kullanılırken, döngü sayacı, koşul bilgisi ve sayacın artım bilgisi verilmelidir. Döngü sayacı kullanılmıyorsa sadece döngüye devam edebilmek için gerekli olan koşul bilgisi verilmelidir.

Genel olarak çoğu programlama dilinin döngü deyimleri ;

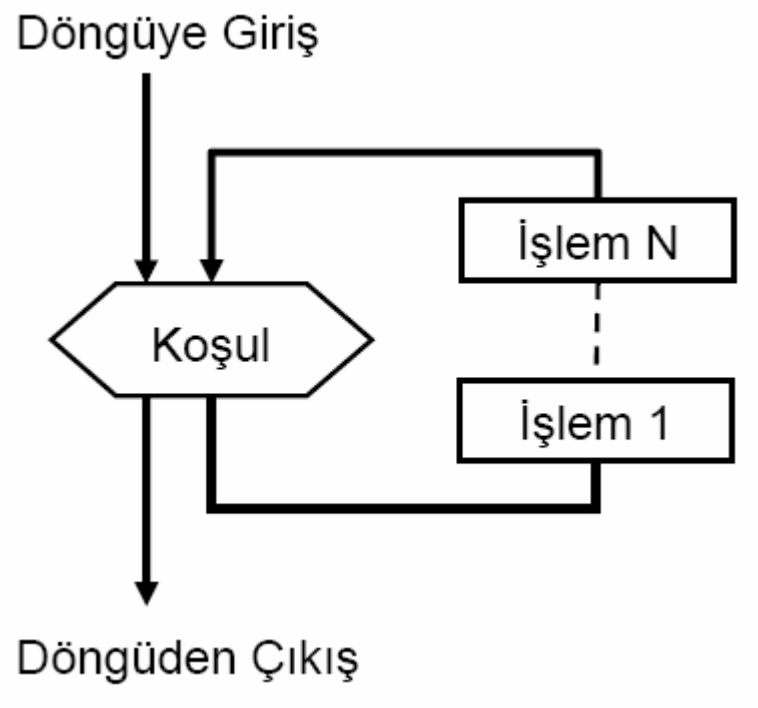
- While
- Do-while
- For

gibi yapılar üzerine kurulmuştur. Farklı dillerde bu yapılara farklı alternatifler olsa da döngülerin çalışma mantığı genel olarak benzerdir.

Akış Diyagramlarında Kullanılan Temel Şekiller

1. Durum (While)

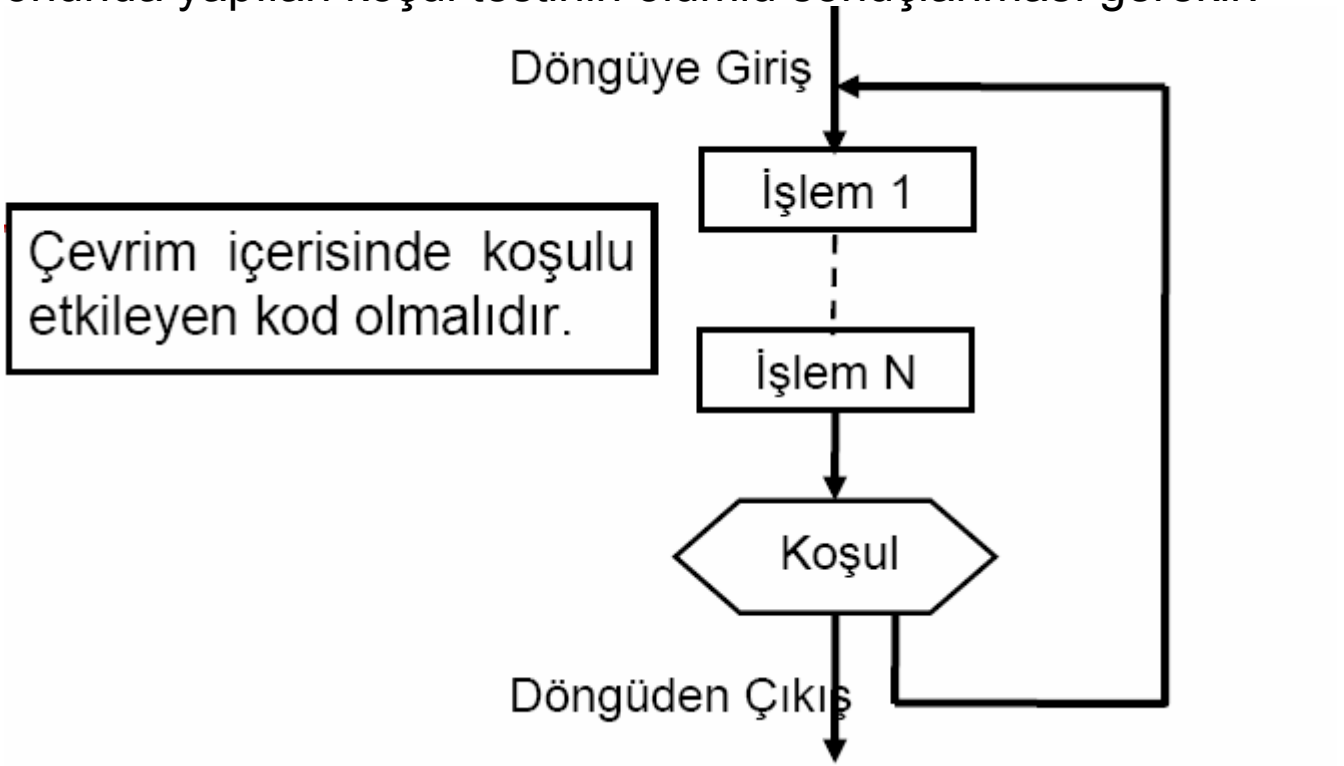
Koşul daha çevrim içerisine girmeden sınanır. Koşul olumsuz olduğunda çerime hiç girilmez ve döngü içerisinde yapılması gerekenler atlanır.



Akış Diyagramlarında Kullanılan Temel Şekiller

2. Durum (Do-While)

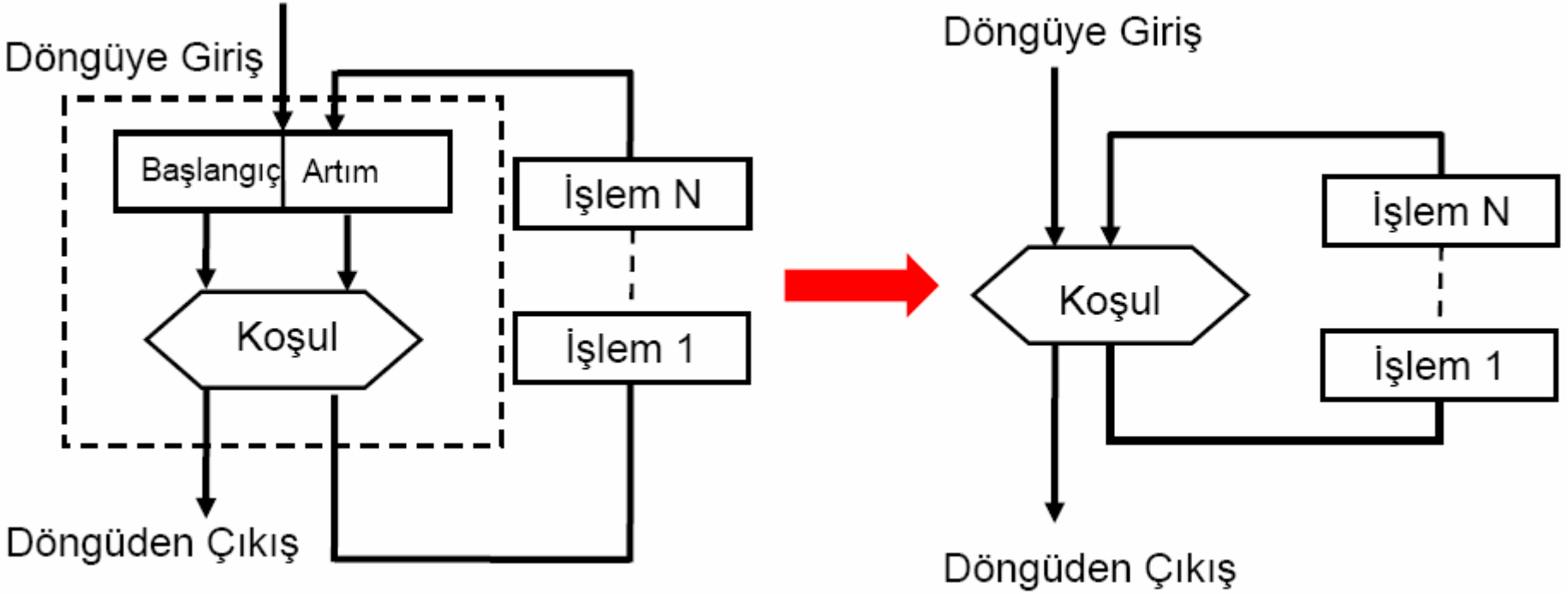
Bu döngü deyiminde, çevrim en az bir defa olmak üzere gerçekleşir. Çünkü koşul sınaması döngü sonunda yapılmaktadır. Eğer koşul sonucu olumsuz ise bir sonraki çevrime geçilmeden döngüden çıkılır. Çevrimin devam edebilmesi için her döngü sonunda yapılan koşul testinin olumlu sonuçlanması gerekir.



Akış Diyagramlarında Kullanılan Temel Şekiller

3. Durum (For)

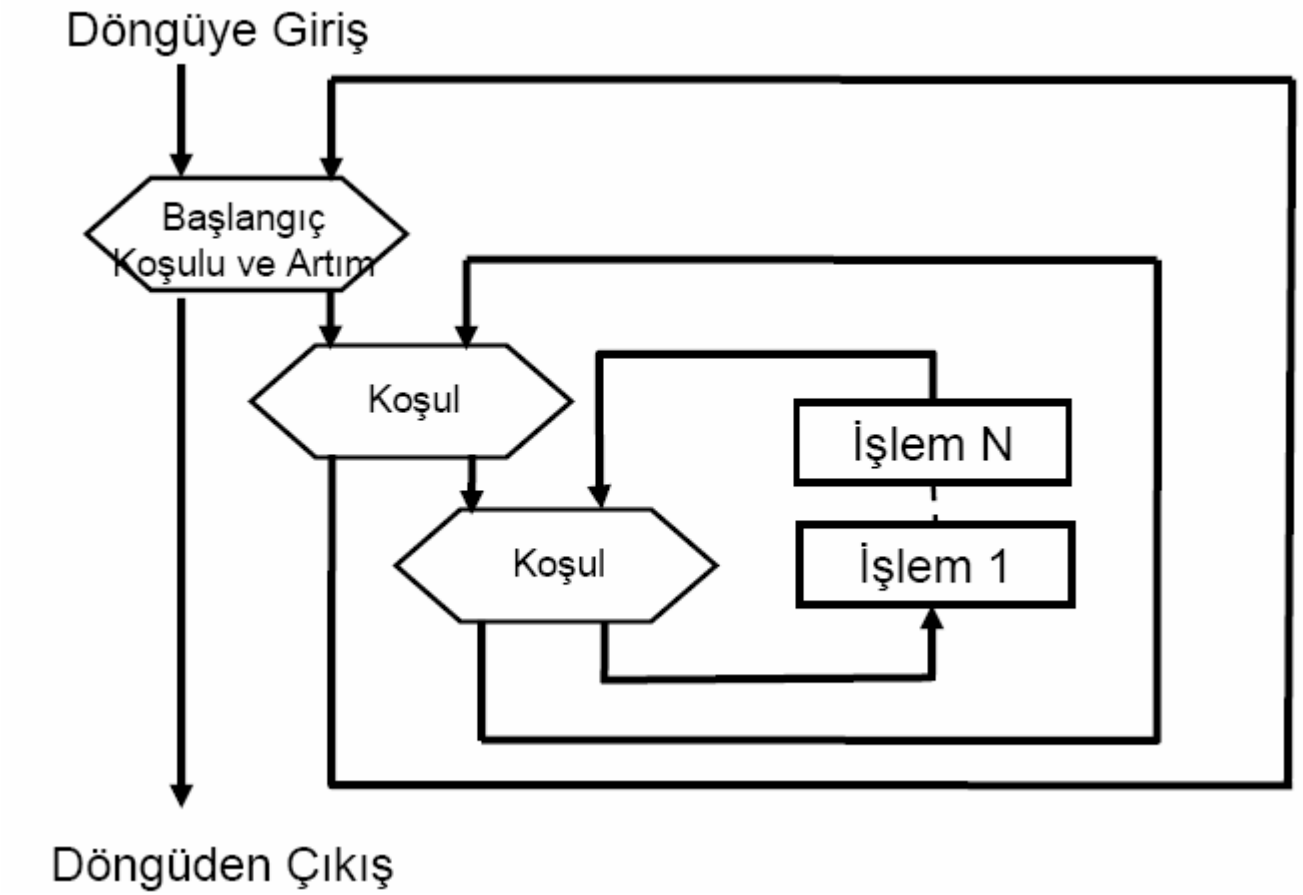
Diğer deyimlerden farklı olarak, döngü sayacı doğrudan koşul parametreleri düzeyinde verilir.
Döngü girmeden önce sayaç değişkenine başlangıç değeri atanmakta ve daha sonra koşula bakılmaktadır. Döngü içerisinde belirtilen işlemler yapıldıktan sonra sayaç değişkeni arttırılmaktadır.



Akış Diyagramlarında Kullanılan Temel Şekiller

İç içe Döngülerin Kullanılması

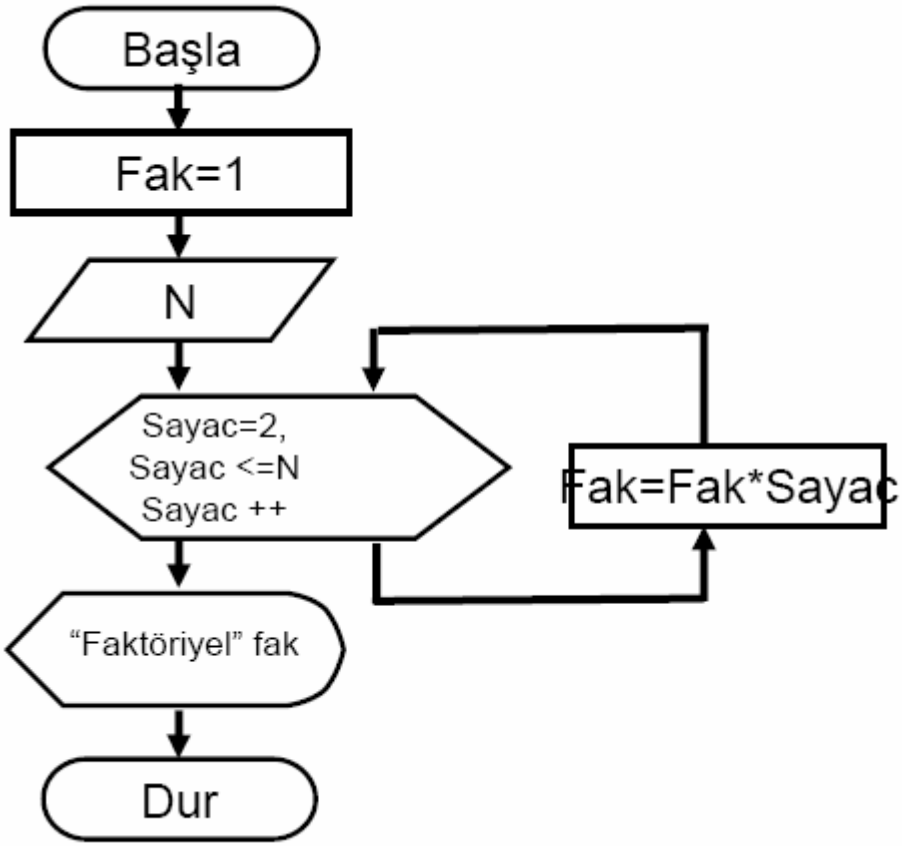
İçiçe döngü kurulurken en önemli unsur, içteki döngü sonlanmadan bir dıştaki döngüye geçilmemesidir. Diğer bir deyişle döngüler birbirlerini kesmemelidir.



En içteki döngü bir dıştaki döngünün her adımında N kez tekrarlanır.

Akış Diyagramlarında Kullanılan Temel Şekiller

Örnek: Klavyeden girilen N sayısının faktöriyelini alan algoritmanın akış diyagramını çiziniz.



- N ile hangi sayının faktöriyelin hesaplanacağı belirlenir ve N çevrimlik bir döngü kurulur.
  - İlk çevrimde 1!, ikinci çevrimde 2! ve sırayla N'inci çevrim sonucunda da N! değeri hesaplanmış olur.
  - Sayac>N koşulu oluştuğunda döngüden çıkılır ve elde edilen sonuç dış ortama aktarılır.

UYGULAMALAR

KAYNAKLAR

1. Rifat Çölkesen, "Veri yapıları ve algoritmalar", Papatya Yayınları, İstanbul, 2002.
2. Fahri Vatansever, "Algoritma geliştirme ve programlamaya giriş", Seçkin Yayınları, Ankara, 2009.
3. Aslan İnan, "MATLAB ve programlama", Papatya Yayınları, İstanbul, 2004.
4. http://www.yildiz.edu.tr/~kunal/bilgisayarbil.html, “Temel Bilgisayar Bilimleri Ders Notları-Ünal Küçük”.
5. Soner Çelikkol, "Programlamaya giriş ve algoritmalar", Akademi Yayınları, İstanbul, 2001.
6. Feridun Karakoç, “Algoritma geliştirme ve programlamaya giriş”, Temel Bilgisayar Bilimleri Ders Notları.
7. Maltepe Üniversitesi, “Programlamanın Temelleri Ders Notları”.
8. www.akademi.itu.edu.tr/buzluca, “Feza Buzluca Bilgisayar Mimarisi Ders Notları”.