Program tasarım aşamaları şu şekilde özetlenebilir:

1- Çözmek istenilen problemin tüm detaylarıyla belirlenmesi.
2- Belli işlemleri yaptırmak üzere programa dışarıdan yapılacak girişlerin ve programın ürettiği çıkışların tespit edilmesi.
3- Programı oluştururken kullanılacak algoritmanın tasarlanması.
4- Algoritmanın MATLAB ifadelerine çevrilmesi.
5- Programın MATLAB'da test edilmesi.

MATEMATİKSEL VE MANTIKSAL OPERATÖRLER

Program dallandıran bir çok yapıda, işlemler, sonucun "doğru" (1) veya "yanlış" (0) olması durumları ile kontrol edilir. MATLAB'da 'doğru' veya 'yanlış' ifadeleri ile sonuçlar üreten iki çeşit operatör vardır. Bunlar matematiksel ve mantıksal operatörlerdir.

C veya Pascal dillerinde karşılaştığımız "boolean", "integer" veya 'logical' tipteki tanımlamaları MATLAB' da yapmaya gerek yoktur. MATLAB, yanlış bir sonuç için 0 değerini ve doğru bir sonuç için 1 değerini kullanır.

Matematiksel Karşılaştırma Operatörleri

Bu operatörler iki değişkenin değer bakımından karşılaştırmasını yaparlar ve üretilen sonucun doğruluğu (1) veya yanlışlığı (0) durumunda, değişik sonuçlar üretirler.

Genel kullanımları a_1 `islem` a_2 şeklindedir. Burada a_1 ve a_2, aritmetik değerler, değişkenler veya karakter dizileri olabilir. "`islem`" ise, söz ettiğimiz matematiksel kıyaslama operatörlerinden biri olabilir. Eğer a_1 ve a_2 arasındaki ilişki operatörün belirttiği şekilde ise işlem, 1 değerini alır. Eğer operatörün belirttiği durumdan farklı bir durum söz konusu ise işlem 0, değerini alır.

Operatörler	İşlevleri
==	Eşittir
~=	Eşit değildir
>	Büyüktür
>=	Büyük veya eşittir
<	Küçüktür
<=	Küçük veya eşittir

Bazı işlemler ve ürettikleri sonuçlar

İşlem	Sonuç
`5<6`	1
`5<=6`	1
`5==6`	0
`5>6`	0
`6<=6`	1
`'A'<'Z'`	1

Tabloda son örnek olarak görünen `'A'<'Z'` örneğinde, işlemin 1 cevabını üretmesinin nedeni, tırnak içi (string) verilen harflerin, alfabetik öncelik sıralarına göre değerlendirilmesidir. Eğer tırnak içinde verilen ve eşit uzunluklarda olan iki kelimenin kıyaslanması istenirse, MATLAB, bunları her bir eleman, bir diğerine karşılık gelecek şekilde kıyaslar ve buna göre tırnak içi kelimelerle aynı eleman sayısına sahip, 0 ve 1'lerden oluşan bir cevap matrisi üretir. Aynı şekilde, bir dizinin bir skaler ile de kıyaslanması gerekebilir. Örneğin; $a=\begin{bmatrix} 3 & 0 \\ -5 & 4 \end{bmatrix}$ şeklinde bir matris ve `b=0` ise `a>b`

ifadesi; $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ cevabını üretecektir. Tırnak içi ifadelerde olduğu gibi, bir matrisi bir diğer matrisle karşılaştırırken, her bir eleman kendisine karşılık gelen diğer matristeki elemanla karşılaştırılır. Yani karşılaştırılacak matrisler eşit sayıda satır ve sütuna sahip olmalıdırlar. Örneğin; $a=\begin{bmatrix} 3 & 5 \\ -7 & 2 \end{bmatrix}$ ve $b=\begin{bmatrix} 5 & 9 \\ -3 & 0 \end{bmatrix}$ ise a>=b ifadesi, $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ şeklinde bir cevap matrisi üretecektir.

Eşitlik durumlarında verilen işaret iki adet eşittir (==) işaretinden oluşur. Oysa değişken atamalarında kullandığımız eşittir (=), bir tanedir. Bu ikisi birbirlerinden farklı operatörlerdir. "==" operatörü, kıyaslama durumlarında kullanılır ve mantıksal bir sonuç üretir. "Eşit ise", "eşit midir?" şeklindeki durumlarda kullanılır. = işareti ise, bir değişkene bir değer atamada kullanılır. Örneğin MATLAB komut penceresinde; `3=5` yazdığımızda; program hata üretir. Oysa `3==5` yazdığımızda bu "3, 5'e eşit midir?" anlamına gelir, kıyaslama yanlıştır ve MATLAB bu durum için 0 cevabını üretir. Yeni başlayanlar için bir karşılaştırma durumunda tek eşittir (=) işareti kullanmak, sık yapılan bir hatadır.

```
>> 3==5
ans =
     0
>> 3=5
??? 3=5
     |
Error: Missing operator, comma, or semicolon.
```

Sık yapılabilecek diğer bir hata da karşılaştırma operatörlerinin aritmetik operatörlerden daha sonra değerlendirildikleri durumunu ihmal etmektir. Yani parantezlerden yararlanılmadığı durumlarda bile aritmetik işlemler, öncelikle yapılır.

```
2+8 > 5+9
(2+8) > (5+9)
```

Bu iki durumda da MATLAB'ın üreteceği cevap 0'dır.

MATLAB'da, yukarıdaki tabloda verilen matematiksel karşılaştırma operatörleri ile aynı işleve sahip bazı hazır fonksiyonlar tanımlanmıştır. Bu işlemciler ve kullanım şekilleri, aşağıdaki tabloda örneklerle birlikte verilmiştir.

İşlemciler	Karşılıkları	Örnek	
eq	==	eq(3,3)=1	eq(3,5)=0
ne	~=	ne(3,5)=1	ne(3,5)=0
gt	>	gt(3,5)=0	gt(5,3)=1
ge	>=	ge(3,5)=0	ge(5,3)=1
lt	<	lt(3,5)=1	lt(5<3)=0
le	<=	le(3,3)=1	le(3,5)=0

== ve ~= Operatörleri Hakkında Bazı Bilgiler

"Eşit midir?" karşılaştırma operatörü olan `==` , karşılaştırılan iki ifadenin aynı değerlere sahip olması durumunda 1 cevabını, farklı değerlere sahip olması durumunda ise 0 cevabını üretir. Benzer şekilde eşit değildir operatörü `~=`, karşılaştırılan iki ifadenin aynı değerlere sahip olması durumunda 0 cevabını, farklı değerlere sahip olması durumunda ise 1 cevabını üretir. Bu iki karşılaştırma operatörü, sabit nümerik değerleri ve tırnak içi ifadeleri karşılaştırmada güvenilir olsa da, bazı ifadelerin karşılaştırılmalarında şaşırtıcı sonuçlar üretebilmektedirler. Yuvarlama farklılıklarından dolayı teorik olarak birbirine eşit iki sonuç eşit değilmiş gibi değerlendirilebilir. Örneğin;

```
>> x=0;
>> y=sin(pi);
```

şeklinde yapılan iki değişken ataması takiben yapılan `x==y` karşılaştırmasının sonucu olarak 1 cevabı beklenirken, 0 cevabı alınır. MATLAB, bu iki değeri farklı olarak algılamıştır. Çünkü MATLAB'da `sin(pi)` değeri 0'a eşit değildir. `sin(pi)`, yuvarlamadaki farklılıktan ötürü $1.224646799147353 \times 10^{-16}$ değerine sahiptir ve 0'a eşit değildir. Yani teorik olarak birbirine eşit iki değer, aslında farklı sayılara tekabül etmektedir.

```
>> x=0;
>> y=sin(pi);
>> x==y
ans =
     0
```

Karşılaştırmak istenilen iki ifadenin gerçek değerleriyle kıyaslanması isteniyorsa, bu karşılaştırmada farklı bir yol izlenebilir. Örneğin, eğer 10^{-15}'ten daha küçük sayılar sıfır kabul edilecekse, önceki örnekte kullanılan `x` ve `y` sayılarının mutlak farkının 10^{15}'ten daha küçük olup olmadığı sorgulanabilir ve yuvarlama hatalarına rağmen doğru bir sonuç elde edilebilir.

```
>> abs(y-x)<10E-15
ans =
      1
```

Mantıksal Operatörler

Bu operatörler, bir veya iki mantıksal anlamı olan ve mantıksal bir sonuç üreten operatörlerdir. Üç çift mantık operatörü vardır. Bunlar; "AND", "OR" ve "XOR (exclusive or)" dur. Bu operatörler çiftlidir ve diğer bir mantık operatörü tekli yapıya sahip olan "NOT" operatörüdür. Çiftli yapıdaki operatörlerin genel kullanımı `a islem b` şeklinde iken tekli bir operatör olan "NOT"un genel kullanımı `islem a` şeklindedir. Burada `a` ve `b` değişkenler iken `islem`, aşağıdaki mantık operatörlerinden biridir. `a` ile `b`'nin arasındaki ilişki operatörün belirttiği şekilde ise sonuç 1, eğer değilse sonuç 0 olur. MATLAB'da 0 harici bütün değerler, mantıksal 1 olarak kabul edilirler.

Operatörler	İşlevleri
&	Mantıksal ve
|	Mantıksal veya
xor	Özel mantıksal veya
~	Mantıksal değil

`xor` mantıksal deyimi, `xor(X,Y)` şeklinde kullanılır. X veya Y'den sadece birisi 0'dan farklı ise sonuç 1, yani doğrudur. Eğer X ve Y aynı anda 0'a eşitse veya her ikisi de aynı anda 0'dan farklı ise sonuç 0, yani yanlış olur.

Örneğin, `X=3` ve `Y=0` için `xor(X,Y)=1` olur.
`X=0` ve `Y=0` için `xor(X,Y)=0` olur.
`X=5` ve `Y=5` için `xor(X,Y)=0` olur.

Bu operatörlere ait sonuçlar ve operatörlerle gerçekleştirilebilecek ihtimallere ait sonuçlar, özet halinde aşağıdaki doğruluk tablosunda verilmiştir.

Girişler a	b	VE a&b	VEYA a|b	ÖZEL VEYA xor(a,b)	DEĞİL ~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

MATLAB'da mantıksal karşılaştırma operatörleri, aşağıdaki tabloda verilen hazır fonksiyonlarla birlikte de kullanılabilirler.

İşlemciler	Karşılıkları	Örnek	
and	&	`and(1,0)=0`	`and(1,1)=1`
or	|	`or(0,0)=0`	`or(1,0)=1`
not	~	`not(5)=0`	`not(0)=1`

Mantıksal operatörler, skaler değerli bir vektör ile bir matrisin karşılaştırılmasında kullanılabilirler. Örneğin $a=\begin{bmatrix}1 & 5\\ 4 & 6\end{bmatrix}$ ve b=0 ise a&b işlemi, $\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$ cevap matrisini üretir. Mertebeleri aynı olduğu sürece mantıksal operatörleri iki matrise de uygulamak mümkündür. Fakat farklı mertebelerdeki matrislerin karşılaştırılmaları istenildiğinde, program hata verir.

A matrisi $\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$ ve B matrisi $\begin{bmatrix}1 & 1\\ 0 & 0\end{bmatrix}$ olmak üzere A|B operasyonu $\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}$ sonucunu üretir.

Matematiksel ve mantıksal operatörlerde işlem öncelik sırası, aşağıdaki gibi olur:

1- Öncelikle tüm aritmetik işlemler, arzulanan sıraya göre yaptırılır.
2- Bundan sonra <, <=, >, >=, ==, ~= gibi matematiksel kıyaslamalar kontrol edilir.

3- Diğer mantıksal operatörlere göre, değil (~) mantıksal operatörünün önceliği vardır.
4- Son olarak, `& ve |` gibi mantıksal işlemler yürütülür.

Örnek:

```
deger1=1
deger2=-10
deger3=0
```

olmak üzere

```
~deger1 = 0                    deger1 & deger2 | deger3 = 1
deger1 | deger3 = 1            deger1 & (deger2|deger3) = 1
deger1 & deger3 = 0            ~(deger1 & deger3) = 1
```

Yukarıdaki örnekte kullanılan `&` ve `|` mantıksal operatörleri aynı işlem öncelik sırasına (hiyerarşiye) sahiptirler. Bu nedenle parantez kullanmak veya kullanmamak durumlarında elde edilen sonuçlar arasında fark yoktur.

```
deger1 > deger2 & deger3 =  0
deger1 > (deger2 & deger3) = 1
```

Yukarıda geçen örnekte ise ikinci durumda kullanılan parantez "`deger2 & deger3`" ifadesinin öncelikle değerlendirilmesini sağlamıştır. Bu nedenle iki durum için elde edilen cevaplar birbirlerinden farklıdır.

&& ve || İşlemcileri

Bazı mantıksal denetlemelerin kısa yoldan gerçekleştirilmesi için kullanılan bu işlemcilerden `&&` mantıksal işlemcisi, `ifade_1` ve `ifade_2`, birer mantıksal ve skaler sonuç üreten işlemler olmak üzere, `ifade_1 && ifade_2` biçiminde bir genel kullanıma sahiptir. Örnek:

```
>> b=0;
>> b&&a
ans =
     0
```

Burada `b` değişkenine önceden 0 değeri verildiği için, `a` değeri ne olursa olsun (tanımlanmamış bir değişken olsa bile) sonuç 0 olur. MATLAB bu yüzden a değişkeninin değerini önemsemeyerek mantıksal değerlendirmeyi kısa yoldan yapar. Eğer b=1 olarak verilmiş ve a değeri tanımlanmamış olsaydı bu durumda MATLAB a değerinin tanımlanması gerektiğini belirtecekti.

Aynı mantıksal değerlendirme `&` işlemcisi ile yapılsaydı, MATLAB aşağıdaki hatayı verirdi.

```
>> b=0;
>> a&b
??? Undefined function or variable 'a'.
```

Bir tane `&` işlemcisinin kullanıldığı bu durumda ise MATLAB, `a` değişkeninin tanımlanması gerektiğini ifade eden bir hata mesajı verir.

Genel kullanımı "`ifade_1 || ifade_2`" şeklinde olan `||` işlemcisi de, `ifade_1` ile değerlendirilen durumun 1 cevabını üretmesi durumunda `ifade_2` durumunu değerlendirmeden 1 cevabını üretir. MATLAB, sadece `ifade_1` ile belirtilen durumun 0 cevabını üretmesi halinde `ifade_2`'yi denetleyecektir. Aşağıda `||` işlemcisinin kullanımıyla ilgili örnekler verilmiştir.

```
>> a=1;
>> x=a||b
x =
     1

>> a=0;
>> x=a||b
??? Undefined function or variable 'b'.
```

`all` ve `any` İşlemcileri

MATLAB'da, mantıksal ve matematiksel kıyaslama operatörlerinin kullanılmalarıyla elde edilen sonuçlarla üretilen matrisler için genel kullanımları `all(dizi)` ve `any(dizi)` biçiminde olan iki mantıksal operatör daha tanımlıdır. `all(dizi)` şeklindeki bir kullanımda, `dizi` elemanlarının ancak tamamının 1 olması halinde 1 cevabı üretilir. En az bir elemanın 0 olduğu bir durumda üretilen cevap 0 olur. `any(dizi)` şeklindeki bir kullanımda ise, ancak `dizi` elemanlarının tamamının 0 olması durumunda 0 cevabı üretilir. `dizi` elemanlarının en az bir tanesinin 1 olduğu durumlarda ise, üretilen cevap 1 olacaktır. Aşağıda, bu iki operatörün bir `a` dizisine uygulanışı örnek olarak verilmiştir.

```
>> a=[1 1 1 1];
>> all(a)
ans =
     1

>> a=[1 1 1 0];
>> all(a)
ans =
     0

>> a=[1 0 1 0];
>> any(a)
ans =
     1

>> a=[0 0 0 0];
>> any(a)
ans =
     0
```

Mantıksal Fonksiyonlar

MATLAB; test ettikleri şartın doğru olması durumunda 1 ve yanlış olması durumunda 0 sonucunu üreten bir dizi mantıksal fonksiyona sahiptir. Bu fonksiyonlar, diğer karşılaştırma operatörleriyle beraber kullanılabilirler ve program dallarına ilişkin kontrolü ve döngülerin oluşturulmasını sağlayabilirler.

Fonksiyon	Özelliği
`ischar(a)`	a karakter dizisi ise 1, değilse 0
`isnumeric(a)`	a sayısal bir dizi ise 1, değilse 0
`isempty(a)`	a boş dizi ise 1, değilse 0
`isinf(a)`	a belirsiz bir sayı ise 1, değilse 0 (1/0 gibi)
`isnan(a)`	a belirsiz bir ifade ise 1, değilse 0 (0/0 gibi)

ŞARTLI İFADELER

Dal yapıları, program kodlarından istenilenleri seçen ve onları işleten, istenilen kodları ise değerlendirme dışı bırakabilen MATLAB ifadelerinden oluşur. Bu dal yapıları `if`, `switch` ve `try-catch` yapıları ile oluşturulur.

`if` Şartlı Deyimi

```
if durum1
   ifade_1
   ifade_2
     .....      } 1. ifadeler bloğu
   ifade_n
elseif durum2
   ifade_1
   ifade_2
     .....      } 2. ifadeler bloğu
   ifade_n
else
   ifade_1
   ifade_2
     .....      } 3. ifadeler bloğu
   ifade_n
end
```

`durum1`'in değerlendirilmesi sonucu üretilen cevap `0`'dan farklı bir değere sahipse (`if` şartlı deyimi ile önerilen durum doğru ise) program, 1. blokta bulunan ifadeleri işletir. Bunun dışındaki durumlar için program, `durum2`'yi denetler. `durum2`, eğer `1` cevabını üretirse; MATLAB, 1. bloktaki ifadeleri işletmeden atlar ve 2. bloktaki ifadeleri yürütür. Eğer `durum1` ve `durum2`'nin denetlenmesi sonucu üretilen cevaplar `0` ise, program `else` deyimini takip eden bloktaki ifadeleri işletir. Çünkü

`else` deyimine, ancak `if` ve `elseif` deyimlerinin tamamının `0` cevabını üretmeleri sonucunda başvurulabilir.

Bir `if` yapısı içinde çok sayıda `elseif` deyimine yer verebilmek mümkündür. Eğer gerek duyulmazsa, `elseif` ve `else` deyimlerinin biri veya her ikisi de kullanılmayabilir. Aşağıda, bir `if` şartlı deyimi örneği verilmiştir.

```
x=10
toplam=50
a=5
if a<=5
   x=x+1
toplam=toplam+a
end
```

Yukarıdaki program segmentinde, `a` değişkenine başlangıçta 5 değeri atandığı için, `if` ile kontrol edilen `a<=5` durumu doğrudur ve program aşağıdaki çıkışları üretecektir.

```
x=10
toplam=50
a=5
x=11
toplam=55
```

`if` Yapısının Kullanımı Hakkında Bazı Bilgiler

`if` yapısını değişik şekillerde kullanabilmek mümkündür. Genel olarak, bir `if` ifadesi ile bir `end` ifadesi kullanılır ve bu iki ifade arasına, istenilen sayıda `elseif` deyimi ve bir tane de `else` deyimi yerleştirilebilir. Bu oluşumlar ile, programa istenilen dal yapısı yerleştirilmiş olur.

Aynı zamanda, `if` yapısını iç içe geçmiş deyimler halinde de kullanabilmek mümkündür. Dıştaki `if` deyiminin içtekini kontrol ettiği bu oluşuma, "iç içe geçmiş `if` yapısı" adı verilir. Bu yapı aşağıdaki şekliyle kullanılır.

```
if x>0
   .....
   if y<0
      ......
   end
   ......
end
```

MATLAB'da, tüm `if` şartlı deyimleri, bir `end` ifadesi ile görevlerini bitirirler (kapatılırlar). Verilen örnekte de, ilk `end` ifadesi, `y<0` durumu halinde işletilecek ifadeleri kapatıcı görevdedir. İkinci `end` ifadesi ise, `x>0` durumunun ifadelerini kapatıcı durumdadır. Bu tip yapılar, basit içerikli programlarda yaygın şekilde

kullanılabilirlerse de, büyük programlarda yanlış kullanılmaları durumunda içinden çıkılamaz hatalara sebep olabilirler.

Aşağıdaki örnekte, iç içe geçmiş `if` yapılarıyla ilgili bir örnek verilmiştir.

```
say=6;
a=50;
if say>5
   say=say+1
  if a==50
     say=say+2
  end
  if  a>50
      say=say+3
  end
end
```

Bu programın çalıştırılması aşağıdaki çıkışları üretecektir.

```
say =
     7
say =
     9
```

Bu örnekte birinci `if` deyimi ile kontrol edilen şart doğru olduğu için, `say=say+1` ifadesi işletilmiştir ve bu `if` deyimi ile sağlanan şartın doğru olması, içteki `if` deyimi ile ifade edilen durumun da denetlenmesini sağlamıştır. Eğer ilk `if` deyimi ile kontrol edilen durum yanlış olsaydı, içteki `if` deyimi asla denetlenmeyecekti. İçteki ikinci `if` deyiminin denetlenmesi de ilk `if` deyiminin kontrol ettiği duruma bağlıdır ve içteki birinci `if` deyiminin kontrol ettiği durumdan bağımsızdır.

`switch-case` Şartlı Deyimi

"`switch-case`" yapısı, diğer bir dallandırma yapısıdır. Programcının, belli durumlar için sadece belli ifadelerin bulunduğu blokların işletilmesini istediği programlarda kullanılırlar. Bu durumlar, dışarıdan girilen değişkenin değişik özelliklerine göre belirlenirler. "`switch-case`" yapısının genel kullanımı aşağıda verilmiştir.

```
switch (durum)
case (durum1),
 ifade_1
 ifade_2
   .....    } 1. ifadeler bloğu
 ifade_n
case (durum2),
```

```
ifade_1
ifade_2
  .....   } 2. ifadeler bloğu
ifade_n
otherwise,
ifade_1
ifade_2
  .....   } 3. İfadeler bloğu
ifade_n
end
```

Eğer "`switch`" ile belirtilen "`durum`", "`durum1`"i sağlıyorsa veya ona eşitse, "`case (durum1)`"e ait ifadeler işletilir ve program `end` ifadesine giderek görevini tamamlar. Benzer şekilde, "`durum`", "`durum2`" yi sağlıyorsa, "`case (durum2)`" ye ait ifadeler işletilir. "`otherwise`" ifadesinin programda bulunup bulunmaması, isteğe bağlıdır. Eğer hiçbir "`case (durum)`", `switch (durum)` ile örtüşemezse, yani `case` ile belirtilmeyen diğer tüm durumlar için, `otherwise` deyiminin ifadeleri yürütülür.

Aşağıdaki örnekte, klavyeden girilecek 1-10 arasındaki bir tam sayının tek veya çift olması durumu "`switch-case`" şartlı deyimi kullanılarak incelenmiştir.

```
sayi=input('1-10 arasında bir sayı giriniz= ');
switch (sayi)
case   {1,3,5,7,9},
       'sayı tek'
case   {2,4,6,8,10},
       'sayı çift'
otherwise,
       'sayı 1-10 aralığının dışında'
end
```

`try-catch` Şartlı Deyimi

```
try
ifade_1
ifade_2
  .....   } 1. ifadeler bloğu
ifade_n
catch
ifade_1
ifade_2
  .....   } 2. İfadeler bloğu
ifade_n
end
```

Bu yapıda, eğer `try` ile `catch` deyimleri arasında bulunan ifadeler, geçerli birer MATLAB ifadeleri ise, MATLAB sadece bu ifadeleri işletir. Bu durumda `catch` ve `end` arasındaki ifadelerin içerikleri önemli değildir, bu ifadeler MATLAB tarafından denetlenmezler. `try` ile `catch` arasında sırayla işletilen ifadeler arasında, tanımlanmamış bir değişken veya anlamsız bir ifade ile karşılaşıldığı andan itibaren MATLAB, `catch` ile `end` deyimleri arasında bulunan ifadeleri yürütür. Aşağıda, bu yapı ile ilgili bir örnek verilmiştir.

```
a=3;
try
    x=5*a
    y=3*b
catch
    z=a^2
end
```

Bu komutlar işletildiğinde, aşağıdaki çıkışlar alınır.

```
x =
    15
z =
    9
```

Bu örnekte `try` ile `catch` deyimleri arasında yürütülen ifadelerden `x=5*a` ifadesinde kullanılan `a` değişkeni, önceden tanımlanmıştır. Bu yüzden bu ifade, aynen işletilir. Bu deyimler arasındaki diğer bir ifade olan `y=3*b`, önceden tanımlanmamış bir `b` değişkeni bulundurduğu için MATLAB, bu noktadan itibaren `catch` ile `end` arasındaki ifadelerin işletilmesine geçmiştir. Eğer `try` ve `catch` arasında sadece `x=5*a` ifadesi bulunuyor olsaydı, MATLAB asla `catch` ve `end` arasındaki ifadeleri işletmeyecekti.

DÖNGÜLER

Döngüler, programların belli ifadelerinin istenilen sayıda tekrarını sağlarlar. Döngüleri, "`while`" ve "`for`" döngüsü olmak üzere iki temel kısma ayırabilmek mümkündür. Bu iki döngü, ifadeleri tekrar etme tarzları ile birbirlerinden ayrılırlar. "`while`" döngüsünde, kullanıcının arzuladığı duruma ulaşılıncaya kadar yapılacak tekrarlar söz konusudur ve bu tekrarların sayısı önceden belirtilmemiştir. "`for`" döngüsünde ise ifadeler, kullanıcının önceden belirlediği sayıda tekrar edilirler.

`while` Döngüsü

"`while`" döngüsü, önceden ifade edilmiş belli bir durum gerçekleşinceye kadar gereken sayıda tekrar edilen ifadeleri içerir. Bu döngünün genel yapısı aşağıdaki gibidir.

```
while durum
      ifade_1  ┐
      ifade_2  │
        .....  ├ ifadeler bloğu
      ifade_n  ┘
```

```
end
```

Eğer "durum", doğru ise yani 0'dan farklı ise, "while" döngüsünün altında bulunan program kodları işletilir ve program döngüye yeniden başlar. Bu döngü tekrarı "durum" 0 olana kadar devam eder. Program, döngüye tekrar başlayacağı zaman, eğer while ile belirtilen "durum" 0 cevabını üretiyorsa, döngü end ifadesine ulaşır ve program end ifadesinden sonraki ilk ifadeyi işletmeye başlar. Örnek:

```
A=5
while A<10
      A=A+1
end
```

Verilen örnekte, A değişkenine ilk değer olarak 5 sayısı verilmiştir ve A sayısı 10'dan küçük olduğu sürece, A sayısına 1 eklenir. Program, aşağıdaki çıkışları verir.

```
A =
     5
A =
     6
A =
     7
A =
     8
A =
     9
A =
     10
```

for Döngüsü

"for" döngüsü; ifadelerin, kullanıcı tarafından belirlenen sayıdaki tekrarının söz konusu olduğu durumlarda kullanılır. "for" döngüsü genel olarak,

```
for degisken=deger
      ifade_1
      ifade_2   } ifadeler bloğu
        .....
      ifade_n
end
```

şeklindeki yapıya sahiptir. Bu yapıda "degisken", döngü değişkenidir. "deger" ise, döngünün değişkene eşitliğinin kontrol edildiği diğer bir ifadedir. Program akışında, her bir durum için döngü değişkeninin durumu kontrol edilir ve ifadeler bloğunda yer alan komutlar ve ifadeler işletilir. Yani "deger" dizisinin her değeri için bir kez döngü işletilir. "deger" ifadesi, çoğu zaman "ilk_deger:artis_miktarı:son_deger" şeklinde belirtilmiş bir dizidir.

"for" ile "end" arasında kalan ifadeler, döngünün ana gövdesini oluştururlar ve bu ifadeler bloğu döngü değişkeninin aldığı her değer için tekrar edilir.

Aşağıda verilen örneklerde, döngü değişkenine farklı şekillerde yapılmış atamalar yer almaktadır.

```
for i=1:7
    ifade_1
    ifade_2
      .....      ifadeler bloğu
    ifade_n
end
```

Yukarıdaki örnekte, döngüye ait kontrol yapısı, 1×7 boyutunda bir dizidir. Yani döngüye ait ifadeler toplam yedi kez tekrar edilecektir. Döngü değişkeni sırasıyla 1,2,3,4,5,6 ve 7 değerlerini alacak ve bu yedi değer için ifadeler ayrı ayrı işletilecektir. Döngü değişkeni, son olarak 7 değerini alır. Döngü değişkeninin aldığı bu son değeri takiben, döngüyü bitiren "`end`" ifadesinin ardından gelen ilk ifade işletilir. Çünkü değerler matrisinin son elemanı 7'dir. Döngünün işletilmesi bittikten sonra, döngü değişkeni "`i`", kendisine son olarak atanan 7 değerine eşittir.

```
for i=1:2:10
  ifade_1
  ifade_2
    .....      ifadeler bloğu
  ifade_n
end
```

Bu örnekte, döngü değişkeni "`i`"; 1×5 boyutunda bir dizidir ve aldığı değerler; 1,3,5,7 ve 9'dur. Döngü değişkeni, beşinci ve son olarak 9 değerini kazanır. Döngünün beşinci kez ifadeleri işletmesinin ardından, kontrol yapısı "`for`" ifadesine geri gelir. Bu durumda, değerler matrisinin başka sütun elemanı kalmadığı için, döngüyü bitiren "`end`" ifadesinden sonra gelen ilk komutun işletilmesine geçilir. Döngü tamamlandıktan sonra işletilen diğer ifadelerde, "`i`" değişkeni için geçerli olan değer 9'dur.

Aşağıdaki örnekte ise değerler matrisi 2×3 boyutunda bir matristir. "`for`" döngüsünde esas olan değerler matrisinin sütün elemanlarıdır. Yani döngü, toplam 3 kez işletilecektir.

```
for ii = [1 3 5 ; 2 4 6]
  ifade_1
  ifade_2
    .....      ifadeler bloğu
  ifade_n
end
```

Döngü değişkeni "`ii`", sırasıyla; $\begin{bmatrix}1\\2\end{bmatrix}, \begin{bmatrix}3\\4\end{bmatrix}$ ve $\begin{bmatrix}5\\6\end{bmatrix}$ matrislerine eşitlenir. Bu örnekten de anlaşılabileceği gibi, döngü değişkenine matris biçiminde bir değer atayabilmek mümkündür.

İç İçe `for` Döngüleri

Bir "`for`" döngüsünü, diğer bir "`for`" döngüsünün ifadeler bloğu içinde kullanabilmek mümkündür. Bu durumdaki döngüler, "iç içe geçmiş `for` döngüleri" adını alırlar.

```
for a=1:3
    for b=1:3
        carpim=a*b;
        fprintf('%d * %d= %d\n' , a,b,carpim);
    end
end
```

Yukarıda verilen örnekte, `a` ve `b` değişkenleri, `[1 2 3]` elemanlarına sahip dizilerdir. `a` değişkeninin 1 değerine eşit olduğu an, `b` değişkeni sırasıyla 1, 2 ve 3 değerlerine eşit olur. Yani `a=1` için `b=1`, `b=2` ve `b=3` olur. Diğer bir deyişle; `[a,b]` matrisi sırasıyla `[1,1]`, `[1,2]` ve `[1,3]` değerlerine eşit olmuş olur. `a=2` için `b` değişkenine yine aynı değerler atanır ve `[a,b]` matrisi, `[2,1]`, `[2,2]` ve `[2,3]` değerlerini almış olur. Bu durumda dıştaki döngü değişkeninin her bir değerine karşılık içteki döngü değişkeninin 3 farklı değer aldığı görülebilir.

Bu örnek için programın ürettiği cevaplar aşağıdaki şekilde olur.

```
1  *  1  =  1
1  *  2  =  2
1  *  3  =  3
2  *  1  =  2
2  *  2  =  4
2  *  3  =  6
3  *  1  =  3
3  *  2  =  6
3  *  3  =  9
```

Aşağıda bununla ilgili diğer bir örnek verilmiştir.

```
for i=1:2
   for j=1:3
   A(i,j)=1/(i+j+1);
    end
end
A
```

Bu örnekte, dış döngü değişkeni `i`; sırasıyla `[1,2]` değerlerini alan bir dizidir. Buna karşılık, iç döngü değişkeni `j`; `[1,2,3]` değerlerini alır. Yani `A(i,j)`

matrisinin elemanları sırasıyla, `A(1,1), A(1,2), A(1,3), A(2,1), A(2,2), A(2,3)` olur.

$$A=\begin{bmatrix} A(1,1) & A(1,2) & A(1,3) \\ A(2,1) & A(2,2) & A(2,3) \end{bmatrix}$$

Döngüye ait ifadeler bloğundan, `A(i,j)` elemanının, `1/(i+j+1)` bağıntısı ile bulunduğu görülmektedir. Bu durumda, program, A matrisinin elemanlarını aşağıdaki şekilde hesaplar.

```
A(1,1) = 1/(1+1+1) = 1/3 = 0.3333
A(1,2) = 1/(1+2+1) = 1/4 = 0.2500
A(1,3) = 1/(1+3+1) = 1/5 = 0.2000
A(2,1) = 1/(2+1+1) = 1/4 = 0.2500
A(2,2) = 1/(2+2+1) = 1/5 = 0.2000
A(2,3) = 1/(2+3+1) = 1/6 = 0,1667
```

Programın MATLAB'da çalıştırılması ile aşağıdaki sonuç üretilir.

```
A =
    0.3333    0.2500    0.2000
    0.2500    0.2000    0.1667
```

`break` İfadesi

"`for`" ve "`while`" döngülerine ilaveten, bir program akışını kontrol edebilmenin diğer bir yolu da "`break`" ifadesini kullanmaktır. "`break`" ifadesini döngü gövdesi içinde kullanmak, döngünün durmasını ve döngüden sonra gelen ilk ifade veya komutun işletilmesini sağlar. Örneğin,

```
for j=2:6
    if j==4
        break
    end
end
fprintf('j= %d\n' , j)
disp('döngü sonlandirildi')
```

Bu ifadeler, programın aşağıdaki çıkışları üretmesini sağlar.

```
j= 4
döngü sonlandirildi
```

Bu örnekte, "`if`" şartlı deyimi, döngü değişkeni 4'e eşit olduğunda devreye girer ve kendisinden sonra gelen "`break`" ifadesinin işletilmesini sağlar. "`break`" komutunun ardından, program döngüden çıkar ve döngüden sonra gelen ilk ifadeyi işletir ve "`döngü sonlandırıldı`" ifadesinin gösterilmesini sağlar.