## ALT BAŞLIKLAR



## ÜNİTE HAKKINDA

Programcılar problemleri çözmektedirler. Bu problem bazen bir sayının asal sayı olup olmadığını ya da iki gezegenin çarpışması durumunda ortaya çıkması muhtemel gazları tespit etmek gibi özel bir konuda da olabilmekte, bazen de bütün süpermarketlerin ürün satışında barkodlu sistemin uygulanmaya başlaması gibi daha genel bir konuda da olabilmektedir.

Problemlerin çözüm aşamaları da bağlantılı olarak çok karmaşıklaşabilmektedir. Bu yüzdendir ki biz programcılar, kafamızda az-çok tasarladığımız çözüm aşamalarını daha belirgin hale getirip problemlerin algoritmalarını kurarız. Kurduğumuz bu algoritmaya bağlı olarak oluşturduğumuz algoritma adımlarını da numaralandırarak alt alta yazarız.

Yalnız oluşturduğumuz bu algoritma adımları, problemlerin çözüm aşamalarını basitleştirerek daha kolay anlaşılabilir hale getirmesi gerekirken, alt alta satırlar halinde yazı yığınlarına da dönüşebilmektedir ve özellikle bir dilde (örneğin Türkçe) yazıldıkları içinde yeterince algılanabilir olmamaktadırlar.

İşte bunlar ve benzeri nedenlerden ötürü biz programcılar algoritmalarımızı, geometriksel şekillerle algoritma adımlarını evrenselleştiren akış diyagramlarına dönüştürmekteyizdir.

Bu ünitede akış diyagramlarını işlemeye çalışacağız.

## ÖĞRENME HEDEFLERİ

Bu ünite bitiminde;

- Akış diyagramları nedir? Tanımlayabilecek,
- Akış diyagramlarında kullanılan şekiller/sembolleri tanıyabilecek,
- Karar yapılarını kullanarak kendi döngülerinizi oluşturabilecek ve
- Problemlerin çözüm aşamalarını akış diyagramları şeklinde ifade edebileceksiniz.

## ÜNİTEYİ ÇALIŞIRKEN

Akış diyagramları konusunda, problem çözme becerilerinizin iyice artması açısından bol örnek çözmek çok önemlidir. Onun için ders notlarında verilen örneklerle yetinmeyip, değişik kaynaklardan ve internet ortamından da bu konuyla ilgili örneklere çalışabilirsiniz.

Ayrıca programlamanın standartlarından sayılabilecek belli başlı algoritmaları da şimdiden incelemeye başlayabilirsiniz (Sıralama algoritmaları gibi).

## ANA METİN

# 3. Akış Diyagramları

## 3.1. Akış Diyagramı Nedir?

Önceki ünitelerde program yazmaya başlamadan önce "programın algoritmasının kurulması" gerektiğinden bahsedilmişti. Algoritma da, problemin çözümü adına üretilmiş "işlem basamakları" idi.

Bu işlem basamaklarını üretmek / üretebilmek bir anlamda problemi de çözebilmek demekti. Çünkü,

**!** Eğer bir problemin çözümüne dair işlem basamakları oluşturulabiliyorsa, o problem çözülmüş demektir. Geriye sadece, herhangi bir programlama dilinde, işlem basamaklarını bilgisayarın anlayabileceği şekle dönüştürmek kalmıştır.

Bu dönüştürme işlemini kolaylaştırmak için, işlem basamaklarımızı oluştururken bilgisayar diline yakın bir dille yazmanın önemine de değinilmişti.

Öyle ki; bilgisayara verilecek iki sayıyı toplayıp, sonucu ekrana yazacak program için oluşturulan işlem basamakları,

1. Sayıları oku
2. Sayıların toplamlarını hesapla
3. Toplamlarını ekrana yaz

şeklinde olduğunda, bu işlem basamaklarını herhangi bir programlama dilini kullanarak bilgisayara aktarmak, aşağıdaki şekilde oluşturulan işlem basamaklarını bilgisayara aktarmaktan daha zor olacaktır. Oysaki işlem basamakları,

1. Başla
2. A sayısını oku
3. B sayısını oku
4. C = A + B
5. C sayısını yaz (ekrana yaz)
6. Dur

şeklinde oluşturulduğunda, her bir algoritma adımı, ilgili programlama dilinin karşılık gelen uygun komutlarıyla çok kolay bir şekilde bilgisayara aktarılabilmekteydi.

Fakat burada programlama dillerinden bağımsız olarak oluşturmaya çalıştığımız işlem basamakları, her ne kadar bilgisayar diline yeteri kadar yakınlaştırılmış olsa da bu sefer de işlem basamakları, algılama açısından biz insanlar için yeteri kadar evrensel değildir.

Bu şu anlama gelmektedir: Tamam, yukarıdaki 6 satırlık algoritma adımı problemi iyi bir şekilde çözüyor olabilir, bir "Türk'te" bu satırları okuduğu zaman çözümü anlıyor olabilir fakat neden işlem basamakları biz insanlar için daha anlaşılır, daha algılanabilir yani neden daha evrensel olmasın?

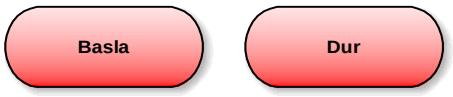
✔ İşte bu amaçla, algoritma adımlarını daha evrensel bir hale getirmek için kullanılan grafiksel sembollere daha doğrusu algoritma adımlarının grafiksel sembollerle ifade edilmesine "Akış Diyagramları" denmektedir.

Şimdi bu akış diyagramlarını oluşturmada kullanılan grafiksel sembolleri ve anlamlarını görelim.

## 3.2. Akış Diyagramlarında Kullanılan Şekiller / Semboller (Vatansever, 2002)

Akış diyagramlarında kullanılan temel şekiller aşağıda verilmektedir. Bu şekiller kaynaklarda bazen değişik biçimlerde gösterilebilmektedir. Fakat genellikle şekillerin genel hatları korunmaktadır. Örneğin çıktı işlemleri için kullanılan şeklin biçimleri değişik olsa da genellikle sağ alt köşesi gedik durumdadır.

### 3.2.1. Başla / Dur İşlemleri

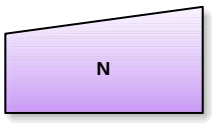



Kenarları yuvarlatılmış dikdörtgen veya elips şekli ile gösterilir. İşlem basamakları için "Başla" ve "Dur" işlemleri, standarttır. Öyle ki her akış diyagramı; "Başla" şekli ile başlar, "Dur" şekli ile biter. Diğer bütün işlem basamakları bu iki şekil arasında yer almaktadır.

❗"Başla" şekli tek çıkışlı bir şekildir.

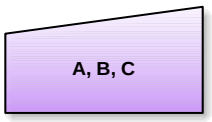
### 3.2.2. Bilgi Girişi İşlemleri

✔ Girdi kavramı, kullanıcıdan bir bilgi alma anlamına gelir. Örneğin; kullanıcıya adını soran ve "Merhaba Selin!" şeklinde çıktı üreten bir program düşünün. Örnekteki isim sorma işlemine "girdi" adı verilir. (Köseoğlu, 2005)




Sol üst köşesi gedik olan bir dörtgen şekli ile gösterilir. Bilgisayara dışarıdan bilgi girişini temsil eder. Bu şeklin içine, dışarıdan girilen bilginin aktarılacağı değişkenin ismi yazılır. Örneğin, eğer şeklin içinde 'N' yazıyorsa ve program çalıştığında '10' girilirse, bu N=10 olacaktır anlamına gelmektedir.

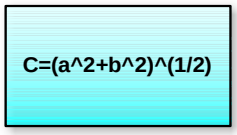
Bilgi girişi şeklinin içine, birden fazla değişken ismi de yazılabilir




Yukarıdaki şekildeki gibi giriş değişkenleri programlama dili ile kodlanıp çalıştırılırsa; işlem (program akışı) bu satıra geldiğinde klavyeden '3', '11', 'Teknik' ifadeleri girilirse, yapılan işlem A=3, B=11 ve C='Teknik' atamalarına eşdeğer olacaktır.

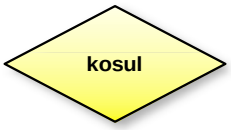
### 3.2.3. İşlem Şekli


Düz dikdörtgen şekli ile gösterilir. Programın işlenmesi sırasında yapılacak işlemleri ifade etmek için kullanılan şekildir. İçine formüller aynen yazılır. İşlem akışı buraya geldiğinde, şeklin içinde yazılı işlem gerçekleştirilir. Birden fazla işlem; aynı şekil içinde, aralarına virgül konularak veya alt alta yazılarak gösterilebilir.



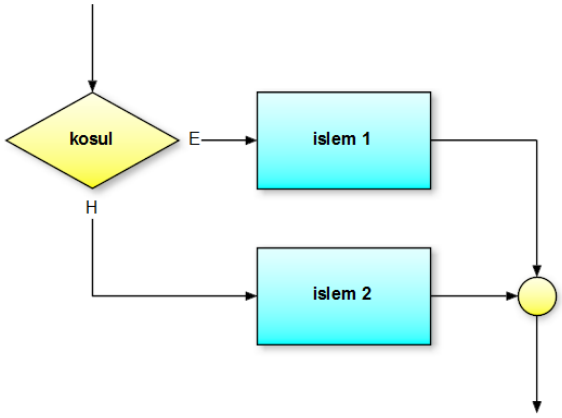

Yukarıdaki örnekte; işlem akışı bu şekle gelince, program $c = \sqrt{a^2 + b^2}$ işlemini yapar. Formüldeki 'a' ve 'b' değişkenleri daha önceki adımlarda girilmiş olan değerlerdir.

### 3.2.4. Karar (Karşılaştırma) İşlemleri



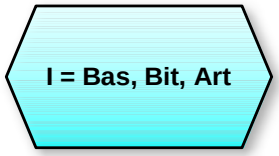
Baklava dilimi şeklindeki bir dörtgenle gösterilir. Karar verme (kontrol etme, karşılaştırma) işlemlerini temsil eden şekildir. Oluşması istenen koşul veya koşullar, aynen şeklin içine yazılır.

Karşılaştırma işlemlerinin en genel kullanım biçimi aşağıdaki şekilde görülmektedir.




Burada; eğer koşul doğru ise (Evet) 'İşlem 1' gerçekleştirilirken, yanlış ise (Hayır) 'İşlem 2' gerçekleştirilir. İşlemlerden sonra, program akışı bağlantı noktasında birleşerek alt satırlara gider. İşlem; sadece koşulun doğru (veya yanlış) olduğu durumda varsa, diğer işlem kutusunu çizmeye gerek yoktur.

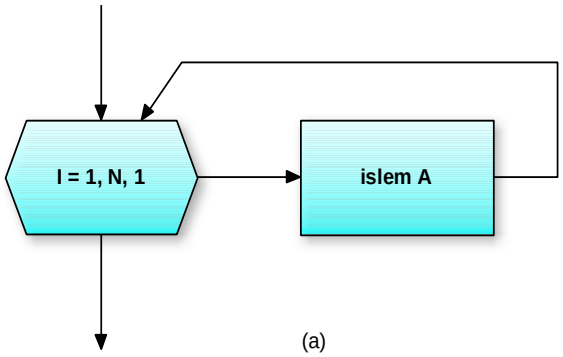
### 3.2.5. Döngü İşlemleri




Birçok programda, belirli işlemler farklı yerlerde aynen tekrarlanır. Bu tekrarlama işlemi 'döngü' (çevrim) olarak isimlendirilir. Döngü şeklinin içine; kontrol (döngü, çevrim) değişkeni, başlangıç değeri, bitiş değeri ve artımı yazılır. Yazım formatı:

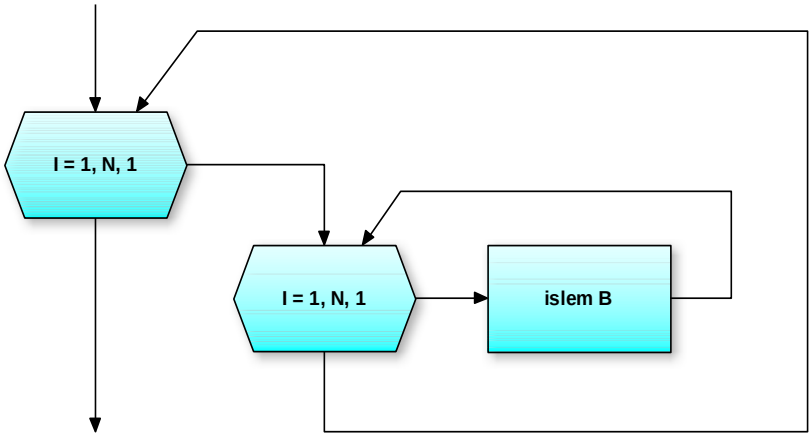
"Kontrol Değişkeni = Başlangıç Değeri, Bitiş Değeri, Artım" şeklindedir. Döngüler iki şekilde oluşturulabilir:

a. Artan Döngü: Başlangıç değeri, bitiş değerinden küçüktür (artım değeri pozitiftir)
b. Azalan Döngü: Başlangıç değeri, bitiş değerinden büyüktür (artım değeri negatiftir)




(a)

İç içe birden fazla döngü kurulabilir. Döngüler kapatılırken, ilk önce en içteki döngünün tamamlanması gerekir. Yukarıdaki şekilde (a) tek döngü vardır ve 'İşlem A', "N" defa tekrarlanacaktır. Aşağıdaki şekilde (b) ise iç içe iki döngü bulunmaktadır. Önce en içteki döngü (J) tamamlanmakta ve 'İşlem B', N.N=$N^2$ defa tekrarlanmaktadır.




(b)

**!** Döngü sembollerinde program akışı, sembole mutlaka yukarıdan giriş yapmalıdır. Sağ tarafından döngü içi işlemlere devam eder. Döngü içi işlemler bittiğinde ise program akışı döngü sembolünün alt kısmından yoluna devam eder.

**!** İç içe döngülerde dıştaki döngünün her bir aşaması için, içteki döngü bütün aşamalarını tamamlamaktadır. Dolayısıyla içteki döngü, genellikle çok yorulan(!) döngü olur.

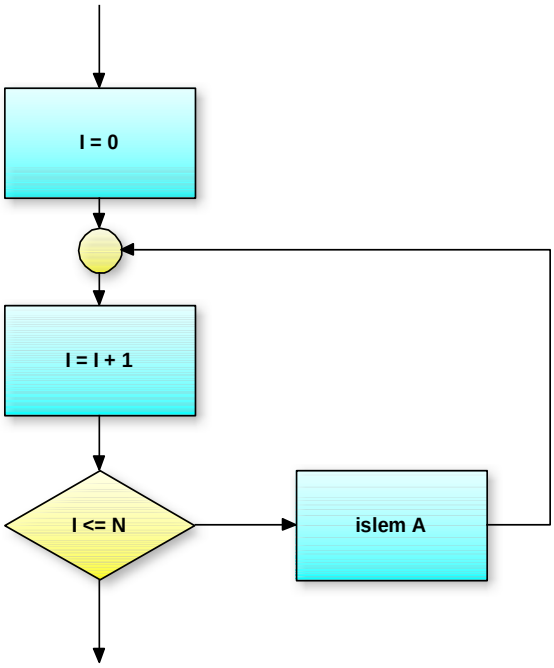
#### 3.2.5.1. Karar Yapılarıyla Döngü Oluşturma

Aşağıdaki şekle dikkat edilecek olursa, akış diyagramlarını oluşturan şekiller arasındaki döngü sembolü kullanılmadan, yalnızca işlem ve karar yapılarıyla "işlem A" nın 'N' defa tekrarlatılması sağlanmıştır.

Burada ilk işlem kutusunda değer aktarılan 'I' değişkeni, ilk değerini alarak döngünün "kontrol değişkeni" gibi davranmaktadır. Zira bir sonraki işlem kutusunda da değeri bir

arttırılmaktadır. Takî karar yapısı sayesinde bitiş değerine ulaşıp ulaşmadığı sorgulanarak, program akışının tekrar yukarıya yönlendirilmediği sürece.

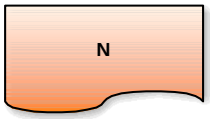
Dolayısıyla 'I' değişkeni istenen değere ulaşıncaya kadar, "işlem A" tekrarlanacaktır.




! Döngü yapıları karar yapıları ile taklit edilebilirken, karar yapılarının yerine geçebilecek başka bir mekanizma mevcut değildir.

### 3.2.6. Çıktı İşlemleri

✓ Çıktı kavramı, en basit anlamıyla bilgisayar ekranına bir şeyler yazarak kullanıcıyı bilgilendirmek anlamına gelmektedir. (Köseoğlu, 2005)




Çıktı işlemleri için kullanılan bu şekilde kaynaklarda çok farklı biçimlerde karşımıza çıkabilmektedir. Fakat genel hatları itibari ile sabittir ve yukarıdaki gibidir. Sağ alt köşesinin gedik olmasından tanınabilir. Ekrana veya yazıcıya bilgi çıkışı için kullanılır. Yazdırılacak olan mesaj ise; tek/çift tırnak içinde çıktı şeklinin içine yazılır. Bir değişken içeriği yazdırılacaksa, şeklin içine değişken ismi yazılır (Yukarıdaki şekilde olduğu gibi). Birden fazla değişken içeriği aralarına virgül konularak tek şekil içinde gösterilebilir.

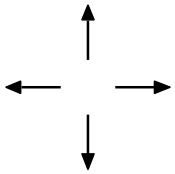
### 3.2.7. Bağlantı İşlemleri


Bağlantı şekli, genel anlamda birleştirici bir noktadır. Genel olarak aşağıdaki amaçlar için kullanılır:

a. Farklı yerlere dallanan işlem akışlarını tekrar toplamak.
b. Akış diyagramı bir sayfaya sığmadığı zaman diğer sayfadaki akış diyagramı ile bağlantı yapmak.
c. Parça parça çizilen akış diyagramları arasında bağlantı yapmak.

**!** Parça parça program yazılması veya diyagramın çizilen yere sığmadığı durumlarda; bağlantı yapılacak uçlara bağlantı şekli çizilir ve içine aynı harf veya rakam yazılır.

### 3.2.8. İşlem Akış Yönleri



Akış diyagramlarında işlem akışının hangi yönde olduğunu gösteren oklardır. İşlem akış yönüne göre uygun olan ok kullanılır.

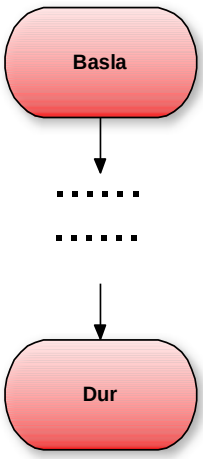
## 3.3. Örnekler

### 3.3.1. Örnek 1

Kullanıcı tarafından "ortalaması" verilen bir öğrencinin geçip-kaldığına karar verip, sonucu ekrana yazan programın akış diyagramını oluşturunuz.

Çözüm:

Bu örnekteki kullanmamız gereken yapıları düşünelim. "Başla" ve "Dur" sembollerinin (yapılarının), mutlaka kullanmamız gereken yapılardan olduğunu artık biliyoruzdur. Bütün program bu iki yapı arasında çalışmaktadır (Aşağıdaki şekilde ki gibi).




Çünkü biliyorsunuzdur, bilgisayarlar sandığımız kadar akıllı değildirdirler. Birileri onlara daha önceden ne zaman başlayıp ne zaman durması gerektiğini tarif etmezse kendi başlarına başlayıp duramazlar (O birileri de biz programcılar olmaktayız).

**!** Programlarımızı başlatan bir yapı olması gerektiği gibi durduran da bir yapı olması gerekmektedir. Eğer ki program akışı bir şekilde "Dur" komutuna ulaşamıyorsa, o program muhtemelen sonsuz bir döngüye girmiş olacak ve hiç durmayacaktır. Böyle bir program artık kullanıcıya cevap veremez duruma geleceğinden, bu durumda biz, o program için kilitlenmiştir deriz.

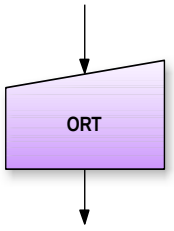
Çözümümüze dönersek; peki ya döngü yapıları? Bu soru için sizce döngü kurmaya gerek var mıdır?

Eğer belli bir düzene göre defalarca yaptırmamız gereken işlemlerimiz olsaydı bu soruya evet cevabını verirdik fakat burada defalarca yaptırılması gereken işlemlerden çok, bir şeylere karar vermeyi gerektirecek durumlar söz konusudur.

Peki, neye karar verdirmemiz gerekmektedir? Öğrencinin geçip-kaldığına…

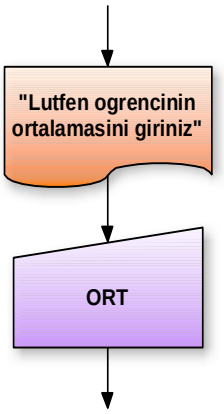
Neye göre geçip-kaldığını belirleyeceğiz? Tabi ki ortalamasına göre…

O halde önce, kullanıcıdan öğrencinin ortalamasını aldırmamız gerekmez mi? Tabi ki evet. Onun için kullanmamız gereken yapı sizce ne olabilir? Aşağıda ki gibi bir sembol neyi ifade eder?




Elbette, kullanıcıdan "ORT" değişkenin aldırılması (okutulması) gerektiğini. Yalnız sizce bu noktada kullanıcının uygun bir mesajla bilgilendirilmesi gerekmez mi? Çünkü bilgisayar kullanırken kendinizi düşünün, yapmak istediğiniz neredeyse her işlem için "şunu yapayım mı?", "bu dosyayı sileyim mi"… gibisinden sorularla bizleri bilgilendirmektedir

değil mi. Bizde kullanıcıya ortalamayı, yani “ORT” değişkeninin alması gereken değeri sormadan önce aşağıdaki şekildeki gibi kullanıcıyı bilgilendirmemiz gerekmektedir.




Kullanıcıyı bilgilendirirken kullandığımız sembole dikkat edin, “çıktı” sembolü.

Kullanıcıdan öğrencinin ortalamasını aldırınca bunu hangi yapılarla sorgulayabiliriz. Tabi ki karar yapılarıyla. Hatırlarsanız döngülerin karar yapılarıyla gerçekleştirilebildiğinden, fakat karar yapılarının yerine başka hiçbir mekanizmanın geçemeyeceğinden bahsetmiştik. Burada da ortalamayı karar yapılarından başka bir yapıyla sorgulamamız, daha doğrusu programı bu sorgulama sonucuna göre istediğimiz bir yere dallandırmak mümkün değildir.

**?** Peki, karar yapılarıyla sorgulamaktan kastımız nedir?

Hatırlarsanız karar yapıları, içlerindeki koşula göre program akışını değiştirebilmekteydiler. O halde biz karar yapısının içine öyle bir koşul yazmalıyız ki, program bizim kontrolümüzde istediğimiz işlemi yapmak üzere dallansın.

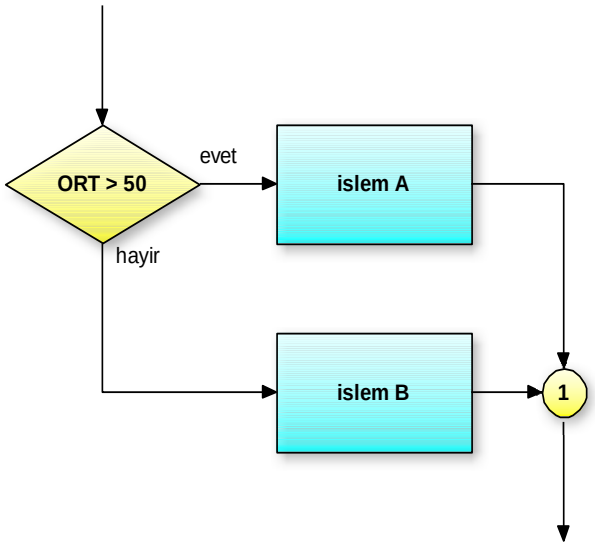
Burada da karşılaştırma operatörlerinden yararlanmamız gerekecektir. Adı üzerinde “karşılaştırma operatörleri”. Kullanıcıdan öğrencinin ortalamasını “ORT” değişkenine okuttuğumuzu varsayalım. “ORT” değişkeninin içindeki değerin 50’den büyük mü küçük mü olduğunu nasıl anlarız. Tabi ki ORT’yi 50 ile uygun operatörü kullanarak karşılaştırarak. Şöyle:

ORT > 50

Sizce bu karşılaştırma işleminin kaç farklı sonucu olabilir; 3 farklı sonucu olabilir mi mesela?. Elbette hayır! Çünkü hatırlayın karşılaştırma işlemlerinde ancak ve ancak iki farklı sonuç mümkündü. “Doğru” ya da “Yanlış”. Mantıken de öyle değil midir? “ORT” değişkeninin içindeki değer ya 50’den büyüktür ya da değildir.

Şimdi sıra bu karşılaştırma işleminden doğan “doğru” ya da “yanlış” sonuçlarına göre programı dallandırmaya geldi. Yukarıda, karar yapılarının içine bir koşul aldığından bahsetmiştik ve koşullarımızın da “doğru” ya da “yanlış” sonuçlar üretebilecek türden olması gerektiğini söylemiştik. Görüyorsunuz ki taşlar yerine oturmaktadır.

“ORT>50” ifadesi karşılaştırma operatörleri kullanılarak oluşturulmuş bir koşuldur. O halde artık bu kadar sözün üzerine, bu koşulumuzu aşağıdaki şekildeki gibi karar yapımızın içine yerleştirebiliriz.



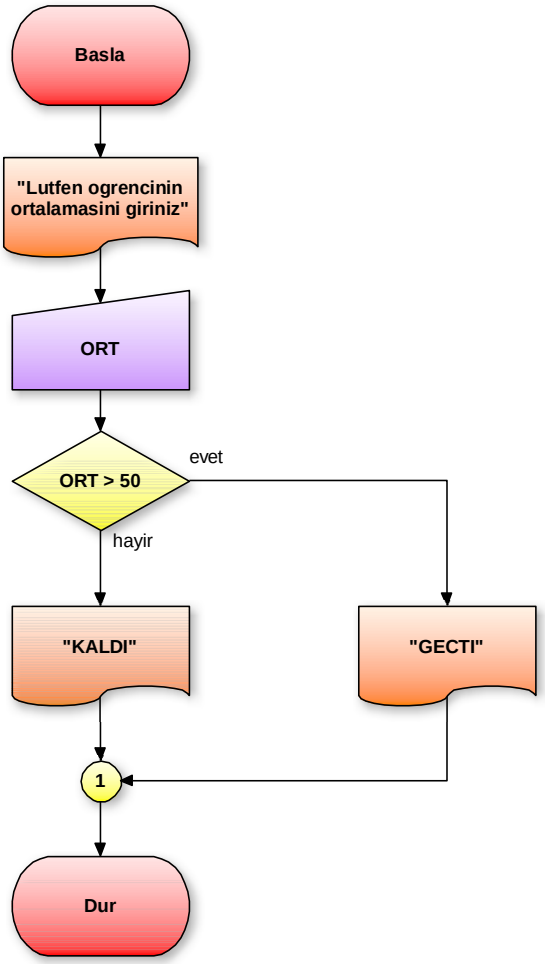

Gördüğünüz gibi programımız bu aşamada “1” numaralı bağlantıya gelmeden önce ikiye ayrılmıştır (dallanmıştır). Yani iki işlemden bir tanesi yapılacaktır. Program akış yönlerine dikkat ederseniz iki işleminde aynı anda gerçekleşme şansı yoktur. Bu durumda aşağıdaki gibi bir tespit yanlış olmayacaktır.

- Eğer ki kullanıcının girdiği ‘ORT’ değeri 50’den büyükse “işlem A” gerçekleştirilecek,
- Değilse de “işlem B” gerçekleştirilecektir.

Ama henüz problemimiz çözülmüş olmadı, çünkü hala işlemler birer muamma. Tamam değer 50’den büyükse işlem yapacak fakat ne işlemi?

Biz ne yapmasını istiyoruz? Ekrana “geçti” veya “kaldı” yazmasını. Onun için karar yapılarından işlemler yerine ekrana yazı yazabilen “çıktı” ları kullanmamız gerekir.

O halde çözümümüzü baştan sona ve birazda tek bir eksen üzerinde çizmeye çalışırsak aşağıdaki gibi olur.




!Yukarıdaki şekilde çıktı sembollerinin içlerindeki sabit mesajların/yazıların kural, gereği tırnak içinde yazıldığına dikkat edin. Tırnak içinde yazılmasaydı, derleyici (programlarımızı yorumlayan program) bu sabit mesajları birer komut veya değişken olarak yorumlamaya çalışacaktı, bu da hataya sebep olacaktı. 'Başla' ve 'Dur' komutlarının ve 'ORT' değişkenin tırnak içinde yazılmadığına dikkat edin.

## ÖZET

Bu ünitede akış diyagramlarından bahsedildi. Akış diyagramları, problemlerin çözümüne dair oluşturduğumuz algoritma adımlarının, daha evrensel olan sembollerle ifade edilmesiydi.

Akış diyagramlarında kullanılabilecek bazı standart semboller vardı. Örneğin bilgi girişi için sol üst köşesi gedik olan dörtgen, bilgi çıkışı için sağ alt köşesi gedik olan dörtgen, karar yapıları için baklava dilimi şeklindeki dörtgen ve döngüler için altıgen şekli gibi…

Bu semboller bağlantı yapıları ve işlem akış yönlerini gösteren oklarla birbirlerine bağlanarak akış diyagramlarımızı oluşturabilmekteydik.

## GÖZDEN GEÇİR

1. Akış diyagramları nedir? Akış diyagramlarına neden ihtiyaç duyulabilir?
2. Akış diyagramlarında kullanılan semboller nelerdir? Açıklayınız.
3. Karar yapılarıyla döngü oluşturmak ne demektir? Açıklayınız.

## KAYNAKLAR


Köseoğlu, K. (2005). *Programcılık Mantığı.* İstanbul: Pusula Yayıncılık.

Vatansever, F. (2002). *Algoritma Geliştirme ve Programlamaya Giriş.* Ankara: Seçkin Yayıncılık.