

İstanbul Üniversitesi

Açık ve Uzaktan Eğitim Fakültesi



Algoritma ve Programlamaya Giriş Editörler

İÇİNDEKİLER

- 1. TEMEL KAVRAMLAR**
- 2. PROGRAMLAMAYA GİRİŞ**
- 3. KARAR YAPILARI**
- 4. DÖNGÜLER**
- 5. ALT PROGRAMLAR, FONKSİYONLAR**
- 6. DİZİLER**
- 7. ARAMA VE SIRALAMA ALGORİTMALARI**
- 8. GENEL UYGULAMALAR**

1. TEMEL KAVRAMLAR

Giriş

Algoritma, programlamanın temelini oluşturur. Nasıl ki geçmişte ve günümüzde yabancı dil bilmek bir gereklilik ise artık programlama dillerini bilmek de aynı şekilde önemli bir gereklilik olarak birçok iş alanında karşımıza çıkmaktadır. Dolayısıyla programlama ve temelinde yatan algoritma mantığını bilmek önemlidir.

1.1. Algoritma Kavramı

Algoritma temelde, bir problemin çözümü için gerekli olan adımlar serisi olarak tanımlanır (Althoff, 2022). Farklı araştırmacılar tarafından yapılan tanımlamalar hemen hemen benzer bir tanımı içermektedir. Bir taraftan algoritma, açık bir şekilde tanımlanmış ve sonlu işlem basamakları şeklinde tanımlanırken (Çobanoğlu, 2014) diğer yandan belirli bir işin veya problemin sonucunu elde etmek amacıyla ardışık şekilde uygulanacak adımları ve koşulları ortaya koyan komutlar dizisi şeklinde de açıklanabilir (Çölkesen, 2011). Öte yandan algoritmaya, girdiyi çıktıya dönüştüren bir dizi hesaplama adımıdır da denebilir (Cormen et al., 2022). Farklı tanımlamalar olsa da hemen hemen hepsi aynı kapiya çıkmaktadır.

Herhangi bir hesaplama sorunu, belirli bir sırada bir dizi eylem gerçekleştirerek çözülebilir. Dolayısıyla bir problemin çözülmesi veya başka bir deyişle bilgisayar tarafından işlenmesi için işlemlerin belirlenmesi ve işlem sırasının kontrol altına alınması gereklidir. Yürüttülecek eylemler yani gerçekleştirilecek işlemler ve bu eylemlerin/islemlerin yürütülme sırasına algoritma denir (Deitel & Deitel, 2015). Burada önemli olan ister bilgisayar bilimlerinde ister gündelik hayatı olsun, probleme uygun adımların seçilmesi ve hangi sıra ve/veya koşullar dahilinde hangi adının uygulanacağı bilinmesi gerekliliğidir.

Althoff (2022), algoritmanın kesinlik (*definite*), etkinlik (*effective*), sonlu olma (*finite*) ve doğruluk (*correctness*) özelliklerinin olması gerektiğini aktarmıştır. Buna göre:

- Algoritmanın, süreç adımları kesin, açık ve özlü şekilde ifade edilmelidir, anlatılanlardan tek bir anlam çıkmalıdır.
- Algoritmanın, her bir adımı, bir problemi çözmek için etkin olmalıdır. Sonuçla ilgisi olmayan adımların algoritmada yeri yoktur.
- Belli bir sayıda adım gerçekleştirildikten sonra algoritmanın çalışması durmalıdır, algoritma kendi içinde sonsuz döngüye düşmemelidir.
- Algoritma, verilen bir girdi için her zaman aynı çıktıyi vermelii ve bu çıktı problemin doğru çözümü olmalıdır. Kisaca algoritma deterministik bir yapıya sahiptir denilebilir.

Algoritma, programlama aşamasına geçmeden önce, hangi işlemlerin hangi sıra ile yapılacağını net bir şekilde ortaya konmasını sağlayan bir alt yapı sağlar. Bir algoritmanın tasarımını bilgisayarın temel çalışma prensibine yani “girdi-islem-çıktı” (*input-process-output*) modeline paralel olarak düşünülebilir. Bilgisayara girdi olarak sunulan veri, birtakım işlemlerden geçerek bilgisayarın donanım ve yazılım aygıtları tarafından işlenir, sonucunda ise kullanıcıya bir çıktı döndürülür. Bilgisayarın çalışma prensibi basit olarak bu şekilde tanımlanabilir. Buna göre bilgisayar üzerinde belli işlemleri yapması için tasarlanacak olan bir programın da bu prensibe uygun bir tasarıma sahip olması gereklidir. Bu uygun tasarımın oluşturulması da en temelinde algoritmanın yani program adımları ve işlemlerinin düzgün bir şekilde tanımlanması ile mümkündür. Dolayısıyla algoritma hazırlama, programlamaya geçmeden önce yapılması gereken önemli bir adımdır. Bahsedildiği gibi adımların belirlenmesi, özel koşullar, tekrarlayan işlemler ve işlemlerin sırası programlama yani kodlama aşamasına geçmeden belirlenmeli ve net bir şekilde ortaya konmalıdır.

1.2. Algoritma Hazırlama

Herhangi bir programlama dilinde bir yazılımın geliştirilebilmesi için bir problemin ortaya konmuş olması ve adımlarının tanımlanmış olması gereklidir. Yukarıdaki kısımda belirtildiği gibi algoritma hazırlama, programlamadan temelini oluşturur. Doğrudan programlama aşamasına geçmek, herhangi bir hata ortaya çıktığında buna çözüm bulmayı zorlaştıracaktır. Oysa net olarak tanımlanmış problem ve işlem adımlarından başlayarak yola çıkıldığında programlama aşaması da kolaylaşacaktır. Algoritmalar sayesinde tanımlanan adımlar, farklı şekillerde ifade edilebilir (Vatansever, 2011):

- Sözlü olarak ifade etme
- Sözde/kaba kod (*pseudo code*) ile ifade etme
- Akış diyagramı (*flow chart*) ile ifade etme

Sözlü ifadede, problem ve adımlar düz metin ile ifade edilir (Vatansever, 2011). Kaba kod, algoritmanın, programlama dili, konuşma dili ve matematiksel ifadeler kullanılarak karma bir şekilde ifade edildiği ifade biçimidir (Çölkesen, 2017). Akış diyagramı ile ise algoritma, görsel simge veya sembollerle ifade edilir (Çobanoğlu, 2014). Uluslararası Standartlar Teşkilatı yani International Organization for Standardization - ISO'ya göre de benzer bir tanımlama yapılmıştır: Akış çizelgesi; işlemleri, verileri, akışı, ekipmanı vb. temsil etmek için sembollerin kullanıldığı, bir problemin tanımının, analizinin veya çözüm yönteminin grafiksel bir temsilidir (ISO, 1985). Bu yöntemler bir algoritmanın adımlarını ve işlem sırasını ortaya koymak için kullanılır. Akış diyagramı, görsel bir dil olarak tanımlanabilir (Erwig, 2017).

Bir işlem serisinin **sözlü** olarak ifade edilmesine günlük hayattan örnekler verilebileceği gibi benzer adımlar bilgisayardaki bir programa yönelik olarak da tasarlabilir. Örneğin;

BAŞLA

Uyan.

Duş yap.

İşe gitmek için kıyafetlerini giy.

Evden çıkış.

BİTİR

Bu örnekte bir kişinin her sabah takip ettiği günlük rutinini bir algoritmanın adımları gibi, sözlü olarak ifade etmek mümkündür. Çünkü bu rutin, belli sıralı adımları ve sonlu eylemleri ifade etmektedir. Dolayısıyla nasıl ki bir kişinin günlük rutini adım adım ifade edilebiliyorsa, bir programın çalışma mantığı da aynı şekilde adımlara bölünerek ifade edilebilir. Yani bilgisayarda gerçekleştirilecek bir işlemin de benzer şekilde ifade edilmesi mümkündür. Örneğin dikdörtgenin çevresinin hesaplanması için izlenmesi gereklili olan adımlar **sözlü** olarak şu şekilde ifade edilebilir:

BAŞLA

Kısa kenar uzunluğunu gir.

Uzun kenar uzunluğunu gir.

Kısa kenar ve uzun kenarı topla ve iki ile çarp.

Çıkan sonucu yazdır.

BİTİR

Dikdörtgenin çevresinin hesaplanması problemi **kaba kod** ile de benzer şekilde ifade edilebilir. Bu ifade şekli, herhangi bir programlama dilinde yazılmamış olmakla birlikte programlamaya geçiş aşamasını kolaylaştırın bir nitelik taşımaktadır. Kaba kod, herhangi bir programlama dilinin yazım şeklinin (*syntax*) katı ayrıntıları ile ilgili endişelenmeye gerek kalmadan algoritma geliştirilmesine yardımcı olan, formal olmayan, bilgisayar tarafından çalıştırılmayan bir dildir (Deitel & Deitel, 2015). Aslında bir programlama dilinde yazmaya çalışmadan önce bir programı düşünmemize yardımcı olur (Deitel & Deitel, 2015). Yani burada, sözel ifadeden biraz daha programlama mantığına yakın ancak herhangi bir dilde yazılmayan bir ifade şeklinde sözcüklerle ifade edilmektedir. Kaba kod, sözlü ifade ile programlama dilindeki ifade arasında bir geçiş sağlar.

Kaba kodun net bir standarı yoktur, kitabı akışı içinde görüleceği gibi aynı algoritma değişik kaba kodlarla ifade edilebilir. Örneğin kullanıcının veri alacağı zaman OKU, GİRDİ, KULLANICIDAN AL gibi

kalıpların tercih edilmesi ya da “Kullanıcı x sayısını girer” gibi ifadeler algoritmaları farklılaştırmaz, deterministik özelliğini bozmadır. Sonuçta amaç kullanıcından bir sayı alınması gerekliliğini ifade etmektedir. Bu ifade şekli ile programlama kalıplarına tam olarak uyulmadığından değişik şekillerde de ifade edilebilir. Yukarıdaki örneğin **kaba kod** ile yazılımı şu şekillerde olabilir:

BAŞLA	BAŞLA
A değerini OKU Kullanıcıdan A değerini al	
B değerini OKU Kullanıcıdan B değerini al	
$C = (A + B) * 2$ Hesapla $C = (A+B) \times 2$	
C değerini YAZ YAZ C	
BİTİR	BİTİR

Bu örnekler aynı zamanda **akış diyagramları** ile de ifade edilebilir. Akış diyagramı, simgesel ve görsel olarak adımların tasarılanmasını sağlar. Akış diyagramlarında kullanılan görsel şekiller, uluslararası standart olarak kabul edilen simgesel ifadelerden oluşmaktadır. Burada ISO tarafından sunulan standartlar güncel olarak kullanılmaktadır. En son 2019 yılında gözden geçirilmiş olan ISO 5807:1985 (ISO, 1985) başlıklı doküman, halen akış diyagramlarındaki simgesel ifadelerin standardını oluşturur. Görsel bir ifade şekli, problemin ve çözüm yönteminin, daha kolay anlaşılmasını ve takip edilmesini sağlar. Görsel ifadenin standartlaşırılmış elemanları mevcuttur. Buna göre aşağıdakiler (Tablo 1) algoritmanın simgesel gösteriminde temel olarak kullanılabilecek elemanların başlıklarını oluştururlar (ISO, 1985):

Tablo 1: Akış Diyagramı Elemanları

Başlangıç ve bitiş ile ilgili semboller: Sürecin başlangıcını ve bitişini ifade etmek amacıyla kullanılan sembollerdir.

Başlangıç/Bitiş



Veri ile ilgili semboller: İşlem için kullanılacak veya işlemler sonucunda elde edilecek verinin ifade edilmesi sağlamak amacıyla kullanılan sembollerdir.

Girdi/Çıktı



Çıktı



Veritabanı



Süreç ile ilgili semboller: Süreç ile ilgili fonksiyonların, yapılacak işlemlerin veya bir koşula bağlı olan işlemlerin ifade edilmesi sağlamak amacıyla kullanılan sembollerdir.

İşlem



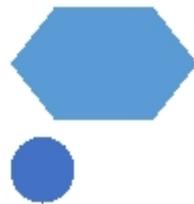
Alt Program



Karar



Sayılı Döngü



Sayaç

Bağlantı sembollerleri: Veri akışının hangi yönde olacağını ifade eden semboldür

Çizgi



Yukarıda bahsedilen akış diyagramı elemanlarına daha yakından bakalım. Algoritma, başla ve bitir elemanları ile başlatılır ve sonlandırılır. Önemli olan bu iki elemandan bir algoritma akış diyagramında birer tane olmasıdır. Yani döngüler veya farklı elemanlar defalarca kullanılabilseler de başlama ve bitirme sembollerini sadece birer kez kullanılabılır.

Veri ile ilgili semboller, sisteme veri girişinde ve/veya yapılan işlemler sonucunda oluşan değerlerin kullanıcıya bildiriminde kullanılır. Girdi simbolü kullanıcıdan herhangi bir medya aracı (klavye, tarayıcı, mikrofon vb.) üzerinden alınan ve sisteme girişi sağlanan verilerin algoritma girişini sağlar. Diğer yandan çıktı simbolü de işlemler sonucunda elde edilen değerlerin çıktı birimleri üzerinden (ekran, yazıcı, hoparlör vb.) kullanıcıya iletilmesini gerçekleştirir. Eğer algoritmda veritabanından veri alma işlemi veya veritabanına veri kaydetme işlemi gerçekleştiriliyorsa diyagramda eleman olarak veritabanı simbolü kullanılır. Her algoritmda bu sembollerin hepsi olacak diye bir kural yoktur, bunların hiçbirini, bazılarını veya hepsi olabilir, her simbol (algoritmanın ihtiyacına göre) birden fazla defa da kullanılabılır. Bu elemanları kullanmadan algoritma veri girişini veya algoritmdan veri çıkışını yapılamaz.

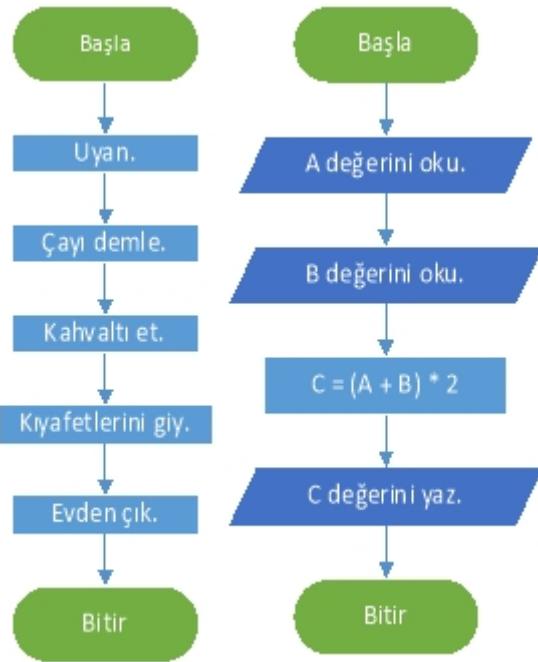
Süreç ile ilgili semboller ise algoritmanın esas bel kemiğini oluşturan kısmıdır. En sık kullanılan süreç elemanı işlem simbolüdür. Yapılan her işlem, hesaplama, eylem veya adım bu eleman ile gösterilir. Örneğin bir toplama işlemi yapılacaksa bu eleman kullanılır veya uyanma eylemi bu eleman ile gösterilir. İşlem elemanı bir eylemi veya hesaplamayı ifade etmeye yarar, belli değişken değerlerini kullanarak başka değişkenlerin değerlerinin atanmasını sağlar.

Karar elemanı, koşullu yapıları ifade etmek için kullanılır. Eğer bir durumun kontrolünün yapılması isteniyorsa, görsel eleman olarak bu eleman tercih edilir. Diğer sembollerden sadece birer çizgi çıkabilenken, karar elemanında farklı olarak birden fazla çıkış olabilir. Şartlara bağlı olarak algoritma bu çıkışlardan duruma uygun olanından devam eder. Karar yapıları üçüncü bölümde (3. KARAR YAPILARI) ele alınmış ve aktarılmıştır. Öte yandan karar elemanı döngülerdeki döngü şartının kontrolünde de kullanılır. Bu konu, dördüncü bölümde (4. DÖNGÜLER) daha detaylı olarak açıklanacaktır. Altıgen şeklindeki sayılı döngü elemanı ise tekrar sayısı önceden belli olarak tekrarlayan döngülerde kullanılır. *for* döngüsü ile çalışılacaksa bu elemandan yararlanılır, sayacın artmasını ifade etmek için ise daire şeklinde eleman kullanılabilir. *while* yapısı için karar elemanı ve sayıç tanımlamasından yararlanılır.

Alt program simbolü ise programın farklı yerlerinde kullanılmak üzere oluşturulan program parçaları yerine kullanılır. Alt programlar aslında program içinde programcıklar olarak özetlenebilirler. Bu programların birer başlangıçları ve bitişleri vardır ve aynı programlar gibi bu iki nokta arasında sıralanmış işlemlerden oluşurlar. Bu alt programlara da bu simbol kullanılarak ulaşılabilir. Sembolün içine erişilmek istenen alt programın adı ve iletilecek parametreler gibi detaylar yazılır. Bu konuya beşinci bölümde (5. ALT PROGRAMLAR, FONKSİYONLAR) detaylıca değinilecektir.

Cizgi ise tüm bu elemanlar arasındaki akışın yönünü ve ilişkisini ifade eder. Burada dikkat edilmesi gereken, bitiş elemanı hariç tüm elemanlara sadece bir çizginin bağlanabildiğiidir. Aynı şekilde sadece bir karar elemanından birden fazla çizgi çıkıştır ve başka bir elemana bağlanabilir; işlem gibi diğer elemanlardan çıkabilecek çizgi sayısı tekdir. Bu kurallara uyulmadığı takdirde algoritmanın yapısı bozuk olur ve bu algoritmayı kullanarak programlama işlemi gerçekleşmez. Buna ek olarak bir de *for* döngüsünde kullanılan sayıç artırımı görselinde farklı sayıda çıktı gösterilebilir.

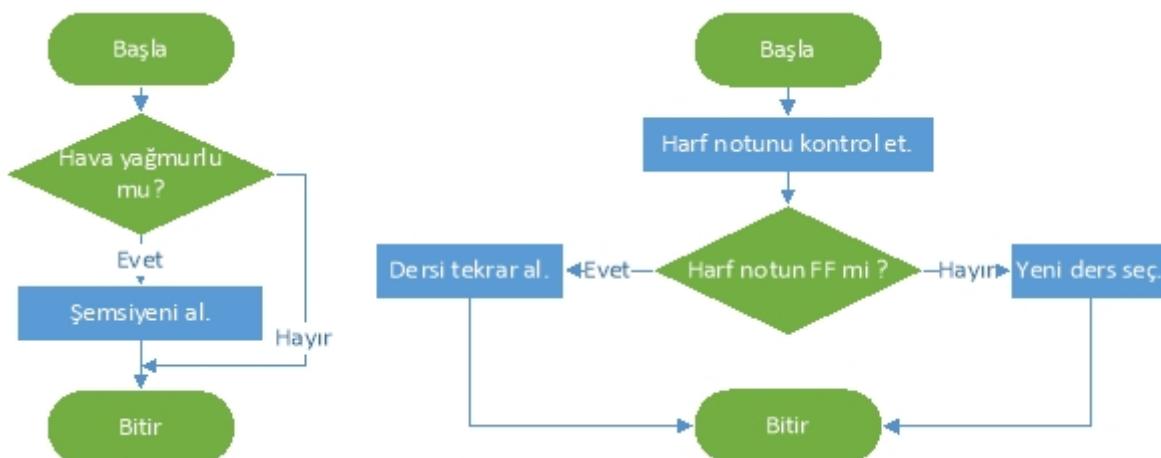
Sabah rutinine ve dikdörtgenin çevresinin hesaplanmasına dair örnekler akış diyagramı elemanları ile şu şekilde ifade edilebilir:



Tüm semboller ile aslında algoritmanın programcı tarafından kolayca anlaşılması amaçlanmaktadır. Bir algoritma, ele aldığı probleme veya çözüm yöntemine bağlı olarak üç farklı yapıda olabilir. Deitel et al. (2014) da bu üç farklı kontrol yapısından bahsetmiştir. Bunlar **sıralı yapı** (*sequence structure*), **seçimli yapı** (*selection structure*) ve **tekrarlı yapı** (*repetition structure*) olarak isimlendirilebilirler.

Sıralı yapıda her bir adım ardışık bir şekilde gerçekleştirilmektedir. Yani en basit haliyle iki işlem birbirine sıralı bir şekilde bağlanır (Erwig, 2017). Örneğin bir sabah rutininde “çayı demleme” ve “kahvaltı etme” eylemleri ardışık şekilde gerçekleştirilebilir. Bu iki işlem arasında, herhangi bir özel adım, belli duruma bağlı olarak aradaki adımları atlama veya bir adımın birden fazla kez tekrarlanması gibi olaylar söz konusu değildir.

Farklı koşullara göre yapılan işlemlerde ise **seçimli** veya **koşullu** bir yapıdan söz edilir. Yani istenen bir koşulun sağlanması durumunda, bir işleminden sonra hangi işlemin geleceği belirlenir. Burada yine günlük rutin örneğinden ilerlenebilir. “Eğer saat 7:00 ise uyan.” gibi bir koşulun varlığı, saat 7:00’den evvel kişinin uyanmayacağı, 7:00’de ise, belki bir alarm yardımı ile uyanacağını ifade eder. Aynı durum “Eğer su kaynadiysa çayı demle.” gibi bir koşul üzerinden de örneklenirilebilir. Su kaynamadan çay demleme eylemi gerçekleştirilmeyecektir, çayın demlenmesi işlemi, suyun kaynamasına bağlıdır. Aşağıda havanın yağmurlu olma koşuluna bağlı olarak çalışan bir algoritma görülmektedir. Hava yağmurlu ise kişi şemsiyesini yanına alacaktır. Diğer karar yapısı örneğinde ise koşulun sağlanıp sağlanmamasına göre iki farklı işlem gerçekleştir. Harf notu FF ise öğrencinin dersi tekrarlaması gereklidir, eğer FF değilse yeni bir ders seçebilir. Karar yapılarından daha detaylı olarak 3.bölümde bahsedilecektir.



Bir algoritmada sıralı işlemlerin yanı sıra, bazen belli bir grup işlemin tekrar etmesi gereklidir. Bu şekilde tekrarlayan eylemlerin varlığında **tekrarlı** bir yapıdan bahsedilir. Tekrarlı bir yapıda belli bir koşula bağlı

olarak, işlemlerin istenen sayı kadar tekrarlanması sağlanır. Burada örnek olarak mevsimlerin döngüsü düşünülebilir. Dört mevsim sürekli olarak tekrar etmektedir. Belli süreler içerisinde mevsimlerin dönüştüğünü görürüz ve bu yıllık döngüler halinde gerçekleşir. Benzer şekilde örneklendirmek gerekirse, kıyafetlerini giyme eylemi kıyafetlerin ütülénip ütülénmemesi ile bağlantılı olabilir. Giyilecek her parça kıyafet ütülénene kadar ütü yapma eylemi tekrarlanacaktır. Örneğin; kişi, gömlek, pantolon, ceket ve kazak olmak üzere dört parça kıyafet giyecekse, ütü yapma eylemi her parça kıyafet için bir kere, toplamda dört kere tekrarlanacaktır. Bilgisayarda da bunlara benzer şekilde bazı işlemler belli sayılarda tekrarlayabilekmektedir. Ortalama kontrolü yapan bir programda, ortalamanın 50'ye eşit veya 50'den küçük olma durumu kontrol edilebilir. Buna göre 50'den küçük olan durumda kişinin dersi tekrarlaması istenebilir. Ya da kitabın sonuncu sayfası gelene kadar kitabı okunmaya devam edilmesi gerekir. Aşağıdaki görsel bu örneği temsil etmektedir. Tekrarlı yapılar, **döngülerle (loop)** ifade edilir ve döngülerden daha detaylı olarak 4.bölümde bahsedilecektir.



1.3. Algoritma Analizi

Bir algoritma analiz edilirken önemli olan nokta, algoritmanın ne kadar verimli çalıştığınıdır. Burada bir algoritmanın yapısının zaman, bellekte kapladığı yer ve işlemci performansı maliyeti açısından ele alınması söz konusudur (Çölkesen, 2017). Diğer yandan algoritmayı analiz etmek, algoritmanın ihtiyaç duyduğu bellek, bant genişliği veya enerji tüketimi gibi kaynakların değerlendirilmesi olarak düşünülebileceği gibi çoğu zaman hesaplama süresini temel alır (Cormen et al., 2022). Burada yürütme zamanı (*running time*), herhangi bir programlama dilinde yazılan bir algoritmanın bilgisayar tarafından yürütülme süresini ifade eder. Ancak değerlendirme için bu tek başına yeterli olamamaktadır. Birçok kaynakta sıkılıkla karşımıza çıkan **büyük O notasyonu** (*Big O notation*) algoritmaların değerlendirilmesinde önemli bir kavram olarak görülmektedir.

Büyük O gösterimi veya notasyonu, algoritmaların verimliliğinin ifade edilmesinde kullanılır (Mailund, 2021). Bu notasyon, bir algoritmanın zaman veya bellek gereksinimlerinin, işlenecek veri kümesinin eleman sayısı (n) artıkça nasıl arttığını açıklayan matematiksel bir gösterimdir (Althoff, 2022). Yani veri kümesinin büyüklüğünün algoritmanın performansına olan etkisi ifade edilmeye çalışılır. Bunu yaparken de hem zaman hem de alan maliyetlerine olan etkiye bakılır. Veri kümesi büyülüğündeki artışın, bellek kullanımına dair gereksinim artışını gösteren kavrama **alan karmaşıklığı** (*space complexity*) adı verilir (Althoff, 2022; Çölkesen, 2017). **Zaman karmaşıklığı** (*time complexity*) ise bir algoritmanın n sayısı büyükçe tamamlaması gereken maksimum adım sayısını ifade eder (Althoff, 2022). Yani veri kümesindeki eleman sayısı artışının, algoritmanın zaman maliyetine olan etkisinin nasıl değişeceğidir (R. Çölkesen, 2017). Büyük O notasyonu ile her iki kavramın da ifade edilmesi ve algoritmanın, veri boyutları büyükçe performansının ne kadar karmaşıklığının ortaya konması mümkün olur. Bu bölümde, zaman karmaşıklığı ilgili kavramlar ele alınmış ve her bir algoritmanın zaman karmaşıklığı ilgili bölümlerde verilmiştir.

Örnek vermek gerekirse, bir sınıfındaki öğrenciler bir veri kümesi ve öğrenci sayısı da eleman sayısı olabilir. Buna göre öğrencilerin ortalamasının hesaplanması öğrenci sayısına bağlıdır. Buradaki zaman karmaşıklığı $O(n)$ olacaktır. Öğrenciler arasından belli biri aranırsa bu durumda tüm sınıfındaki kişilerin gözden geçirilmesi gerekecektir. Yine zaman karmaşıklığı aynı değerde olur. Okuldaki öğrencilerin isminin tek tek yazdırılması ve ortalamalarının hesaplanması gerekiyorsa, bu durumda öğrenci sayısının iki katı kadar

işlemler tekrarlanacak olmasına rağmen $2n$ ile n arasında çok büyük fark olmadığından zaman karmaşılığı yine $O(n)$ olarak gösterilecektir. Diğer yandan iki döngünün iç içe girdiği ve eleman sayısının karesi kadar işlem yaptığı durumlarda zaman karmaşılığı $O(n^2)$ olacaktır. Diğer durumların aksine, eleman sayısı kaç olursa olsun tek işlem yapılıyorsa, bu durumda zaman karmaşılığı $O(1)$ olacaktır.

Zaman karmaşılığının anlaşılabilmesi için en iyi, en kötü ve ortalama durum senaryolarından bahsedilmesi gereklidir (Althoff, 2022; Çölkesen, 2017): **En iyi durum** (*best-case complexity*), algoritmanın en iyi durumda, yani en hızlı çalıştığı durumu ifade eder. **En kötü durum** (*worst-case complexity*), algoritmanın çalıştığı en kötü durumu ifade eder. **Ortalama durum** (*average complexity*) ise, algoritmanın gösterdiği ortalama performansı ifade eder. Algoritmalar karşılaştırılırken genellikle ortalama durum karmaşılığını üzerinden karşılaştırma yapılır, daha detaylı analizler için en iyi ve en kötü durumlar karşılaştırılabilir. Bir örnek vermek gereklidir, bir arama algoritmasında aranan elemanın dizinin ilk olması en iyi durumu temsil ederken, aranan elemanın dizinin içinde olmaması en kötü durumu ifade eder (Çölkesen, 2017). En iyiden (en verimli) en kötüye (en az verimli) doğru sıralanmış olarak, zaman karmaşılığı büyük O gösterimi aşağıdaki şekillerde ifade edilebilir (Tablo 2):

Algoritmanın çalışma performansı büyük O notasyonunun büyüklüğüne göre değerlendirilebilir. Girdi sayısı arttıkça, başka bir deyişle veri setinin boyutu arttıkça, algoritmaların performansları etkilenebilmektedir. Burada tabloda belirtildiği gibi en iyi zaman karmaşılığı sabit zamanda çalışan algoritmalarla aittir. Yani veri boyutuna göre çalışma performansı değişiklik göstermemektedir. Karmaşılığın çok fazla olduğu bir durum ideal bir durum değildir, veri boyutu arttıkça karmaşılık örneğin üssel bir şekilde artırsa algoritmanın performansının kötüleştiği yani verimsizleştiği anlamına gelir.

Tablo 2: Büyük O Notasyonu

Karmaşıklık Büyüklüğü Tanımı	Büyük O Gösterimi	Örnek İşlem / Örnek Algoritmalar
Sabit zaman (constant time)	Algoritmanın zaman karmaşılığı, veri setinin büyüklüğüne bağlı olarak değişmez. En verimli olan zaman karmaşılığı $O(1)$ sabit zamandır.	Aritmetik işlemler veya dizideki bir elemana erişim
Logaritmik zaman (logarithmic time)	Algoritmanın zaman karmaşılığı, veri boyutunun logaritması ile orantılı olarak artar.	$O(\log n)$ İkili arama
Doğrusal zaman (linear time)	Algoritmanın zaman karmaşılığı, veri boyutuyla doğru orantılı büyür.	$O(n)$ Dizide eleman arama Doğrusal arama
Log-lineer zaman (log-linear time)	Algoritmanın zaman karmaşılığı, veri boyutunun logaritmik ve lineer zaman karmaşılığının bir kombinasyonu (çarpımı) olarak büyür.	$O(n \log n)$ Hızlı sıralama Birleştirilmeli sıralama
Kuadratik zaman (quadratic time)	Algoritmanın zaman karmaşılığı, veri boyutunun karesiyle doğru orantılı olarak artar.	$O(n^2)$ Kabarcık sıralama Yerleşirmeli sıralama
Kübik zaman (cubic time)	Algoritmanın zaman karmaşılığı, veri boyutunun kübik boyutuyla doğru orantılı olarak artar.	$O(n^3)$ Üç değişkenli denklem çözme
Üssel zaman	Algoritmanın zaman karmaşılığı, veri boyutuyla bağıntılı şekilde üssel olarak artar.	$O(c^n)$ Tüm alt kümeleri bulma

*(exponential
time)*

Bölüm Özeti

Bu bölümde programlama aşamasına temel sağlayan algoritma kavramı, algoritma hazırlama aşamalarından ve elemanlarından bahsedilmiştir.

Birinci bölümde algoritma kavramı ve algoritmada olması gereken özellikler ele alınmıştır. Algoritma, bir problemin çözüm yolunu veya adımlarının ifade edilmesi şeklinde tanımlanabilir. Algoritmanın; kesinlik, etkinlik, sonlu olma ve doğruluk özelliklerine sahip olması gereklidir. Bir algoritmanın adımları tartışmaya yol açmadan, net ve kesin bir şekilde ifade edilebilir. Problem çözümü için, belirlenen adımlar etkin olmalıdır. Algoritma sonsuza kadar çalışamaz dolayısıyla bir noktada sonlanacak şekilde tasarlanmalıdır. Aynı zamanda bir algoritma her çalıştırıldığında aynı çıktıyı vermelidir. Bunlar algoritmanın olmazsa olmaz özellikleridir.

İkinci bölümde algoritmanın farklı ifade şekillerinden bahsedilmiştir. Adımların ifade edilmesi için üç farklı gösterim şekli mevcuttur. Bir algoritma; sözlü olarak, sözde veya kaba kod ile veya akış diyagramı ile ifade edilebilir. Bulardan akış diyagramında kullanılan elemanlar detaylı olarak verilmiştir. Algoritma içerisinde kullanılmış her yapıya uygun bir eleman mevcuttur. Örneğin döngülerin ifade edilmesi için ayrı bir eleman kullanılırken, işlem veya eylemlerin ifade edilmesi için farklı bir eleman kullanılır. Akış diyagramı tasarlarken dikkat edilmesi gereken önemli noktalardan biri, bir akış diyagramında sadece bir başlangıç ve bitiş elemanı olabileceğidir. Her elemandan sadece bir çizgi çıkış başka bir elemana bağlanabilir; öte yandan her bir elemana sadece bir çizgi girebilir. Burada sadece döngü yapılarında veya sayıç ifadesinde farklılık görülmektedir.

Üçüncü başlıkta ise algoritmanın analizi ile ilgili bilgiler sunulmuştur. Büyük O notasyonu, algoritmanın karmaşıklığını ifade etmek için kullanılan bir kavram olarak ele alınmış, farklı algoritmaların hangi değer ile ifade edilebileceğine dair örnekler verilmiştir.

Kaynakça

- Althoff, C. (2022). *The self-taught computer scientist : the beginner's guide to data structures & algorithms*. John Wiley & Sons, Inc.
- Çobanoğlu, B. (2014). *Algoritma Geliştirme ve Veri Yapıları* (5th ed.). Pusula Yayıncılık.
- Çölkesen, R. (2011). *Algoritma Geliştirme Veri Yapıları* (2nd ed.). Papatya Yayıncılık Eğitim.
- Çölkesen, R. (2017). *Algoritmalar*. In T. R. Çölkesen & O. Aliefendioğlu (Eds.), *Bilgisayar Bilimine Giriş* (pp. 201–247). Papatya Yayıncılık Eğitim.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms* (4th ed.). The MIT Press.
- Deitel, P., Deitel, A., & Deitel, H. (2014). *Visual Basic 2012 How to Program* (6th ed.). Pearson.
- Deitel, P., & Deitel, H. (2015). *Java How to Program - Late Objects Version* (10th ed.). Pearson.
- Erwig, M. (2017). *Once upon an algorithm : how stories explain computing*. MIT Press.
- ISO. (1985). ISO - ISO 5807:1985 - Information processing — Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts. <https://www.iso.org/standard/11955.html>
- Mailund, T. (2021). *Introduction to Computational Thinking: Problem Solving, Algorithms, Data Structures, and More*. Apress. <https://doi.org/10.1007/978-1-4842-7077-6>

Ünite Soruları

Soru-1 :

Aşağıdakilerden hangisi algoritmayı ifade etmek için kullanılan yöntemlerden değildir?

(Çoktan Seçmeli)

- (•) - Sözde kod
- (•) - Sözlü kod
- (•) - Akış diyagramı
- (•) - Kaba kod
- (•) - Pseudo kod

Cevap-1 :

Sözlü kod

Soru-2 :

Hangisi büyük O notasyonu ile ölçülen bir kavramdır?

(Çoktan Seçmeli)

- (•) - çalışma süresi
- (•) - işlem miktarı
- (•) - algoritma türü
- (•) - zaman karmaşıklığı
- (•) - değişken sayısı

Cevap-2 :

zaman karmaşıklığı

Soru-3 :

Aşağıdaki hangisi veritabanını ifadede kullanılır?

(Çoktan Seçmeli)



Cevap-3 :



Soru-4 :

Karar yapılarını ifade etmek için hangi eleman kullanılır?

(Çoktan Seçmeli)





Cevap-4 :



Soru-5 :

Aşağıdakilerden hangisi işlemleri veya eylemleri ifade etmek için kullanılır?

(Çoktan Seçmeli)



Cevap-5 :



Soru-6 :

Akış diyagramının hangi elemanında birden fazla çıktı olabilir?

(Çoktan Seçmeli)

- (•) - Karar yapıları
- (•) - Fonksiyonlar
- (•) - Başlangıç
- (•) - İşlem
- (•) - Hiçbiri

Cevap-6 :

Karar yapıları

Soru-7 :

Bir akış diyagramında birden fazla başlangıç ve bitiş elemanı kullanılabilir mi?

(Çoktan Seçmeli)

- (•) - Kullanılabilir.
- (•) - Kullanılamaz.
- (•) - Döngü yapısı ile birlikte kullanılabilir.
- (•) - Koşullu yapıda kullanılabilir.
- (•) - Alt programlarda kullanılabilir.

Cevap-7 :

Kullanılamaz.

Soru-8 :

Aşağıdakilerden hangisi bir algoritmada olması gereken özelliklerinden değildir?

(Çoktan Seçmeli)

- (•) - Sonlu olma
- (•) - Etkinlik
- (•) - Verimlilik
- (•) - Kesinlik
- (•) - Doğruluk

Cevap-8 :

Verimlilik

Soru-9 :

Aşağıdakilerden hangisi bir algoritmada olabilecek kontrol yapılarından değildir?

(Çoktan Seçmeli)

- (•) - Sıralı yapı
- (•) - Seçimli yapı
- (•) - Tekrarlı yapı
- (•) - Koşullu yapı
- (•) - Sonlu yapı

Cevap-9 :

Sonlu yapı

2. PROGRAMLAMAYA GİRİŞ

2.1. Temel Veri Tipleri

Programlama konusuna başlarken ilk önce veri tiplerinden bahsedilmesi gereklidir. Veri tipleri belirli kategoriler altında toplanmıştır ve veri tipi tanımlarında yapılacak yanlış tercihler programın hızını etkilemenin yanı sıra program hatalarına ve hatta çökmelerine bile sebep olabilmektedir. Bu yüzden program yazarken veri tiplerinin seçimi esnasında çok dikkatli olunması gerekmektedir. Programlama dillerinde veri tipleri dört kategoride incelenebilir (Veritabanları dillerine geçildiğinde bu sayı artacaktır.). Bu bölümde kullanılan veri tipleri Visual Basic (VB) ve C # dillerine göre anlatılmıştır, bu tipler diğer dillerde ad ve kapladıkları boyut açısından ufak tefek değişiklikler gösterebilirler.

2.1.1. Sayısal Veri Tipleri

Sayısal veri tipleri ondalıklı sayılar ve tam sayılar olmak üzere iki grupta incelenebilir.

2.1.1.1. Tam Sayılar

Kesirli olmayan veri tiplerine tam sayı veri tipleri denir. Bu veri tipleri çoğu programlama dilinde işaretli veya işaretsiz olarak ikiye ayrılırlar. İşaretli olanlar negatif veya pozitif değerler tutabilirlerken işaretsiz olanlar sadece pozitif değerleri saklayabilmektedir. Aritmetik işlemlerde tam sayı veri tipleri diğerlerine göre çok daha hızlı işlenirler, bu yüzden uygun olan her yerde bu veri tiplerini kullanmak isabetli olacaktır. Tam sayı veri tipleri programlama dillerinde büyülüklüklerine göre farklı isimler alabilmektedirler.

İşaretli bir küçük tam sayı saklanmak isteniyorsa en iyi tercih *sbyte*'tir. Bu veri tipi ile -128'den 127'ye kadar tüm sayılar saklanabilir. İşaret önemli değilse o zaman *byte* tipi veri kullanmak tercih edilebilir. Bu veri tipinde 0'dan 255'e kadar olan sayılar tutulabilir. Daha büyük bir sayı saklanmak isteniyorsa ve sayı işaretli ise sırasıyla *ushort*, *uint* ve *ulong* tipleri, sayı işarette sahip değilse de *short*, *int* ve *long* veri tipleri tercih edilir. Veri tipinin tutabildiği sayı boyutunun büyümesiyle programın çalışma hızı yavaşlar. Örneğin bir program *int* veri tipiyle *long*'a göre daha hızlı çalışırken *short* veri tipine göre de daha yavaş çalışmaktadır. Aşağıdaki tabloda (Tablo 3) Visual Basic ve C# programlama dillerindeki tam sayı veri tipleri ve boyutları görülebilir.

Tablo 3: Veri Tipleri

Veri Tipi	Aralık	İşaret	Boyut
<i>sbyte</i>	-128 - 127	İşaretli	8 bit
<i>byte</i>	0 - 255	İşaretsiz	8 bit
<i>short</i>	-32.768 - 32.767	İşaretli	16 bit
<i>ushort</i>	0 - 65.535	İşaretsiz	16 bit
<i>int</i>	-2.147.483.648 - 2.147.483.647	İşaretli	32 bit
<i>uint</i>	0 - 4.294.967.295	İşaretsiz	32 bit
<i>long</i>	-9.223.372.036.854.775.808 - 9.223.372.036.854.775.807	İşaretli	64 bit
<i>ulong</i>	0 - 18.446.744.073.709.551.615	İşaretsiz	64 bit

Tam sayı değerlerini aralıklarına göre kullanmak önemlidir. Örneğin il kodları bir değişkende tutulacaksa en büyük il kodu 81 olduğuna göre bu değişkene en uygun veri tipi *byte* olacaktır, *short*, *int* ve *long* veri tipleri büyük gelecektir. Burada *byte* yerine örneğin *long* kullanıldığında her veri için 8 bit yerine 64 bit yer rezerve edilecektir, bu da hem sabit diskte yer kaybına, hem de programın çalışması esnasında RAM'deki fazla yer rezervasyonu sebebiyle performans düşüşüne sebep olacaktır. Diğer yandan TC kimlik numarasını saklamak için tek çare, değişkeni *long* olarak tanımlamaktır. Diğer yandan sadece pozitif değerlerin tutulacağı bir

değişken için işaretli bir veri tipini kullanmak, gereksiz yere kaynağın yarısını kullanmayı yani israfı (veri başına bir bit) beraberinde getirecektir.

Tercihler sonucunda çıkan farklar küçük gibi görünse de veri kümesi büyündükçe sorun büyümektedir. Örneğin bir öğrencinin belli bir değerini tutan bir değişkende bir bayt bir şey ifade etmezken, üniversitede bulunan 80.000 öğrenci için bu rakam 80 kbayt'a çıkmaktadır. Başka alanlarda da bu israf olduğu takdirde, bahsi geçen rakam, performansı zorlayacak noktalara rahatlıkla gelecektir. Bu yüzden verilerin saklanmasında kullanılacak veri tipine karar verirken, beklenen en büyük ve en küçük değerleri düşünüp ona göre veri tipini seçmek doğru olacaktır.

2.1.1.2. Ondalıklı Sayılar

Ondalıklı veri türleri, hem tam sayı hem de kesirli bölümleri olan sayıları temsil eden veri türleridir. Programlama dillerine bağlı olarak değişseler de bu türlerin en belli başlı örnekleri *decimal*, *single* ve *double* veri tipleridir. Bu türlerin hepsi işaretli veri türleridir. Eğer bir değişken kesir içeriyorsa bu türlerden biriyle tanımlanmalıdır.

Ondalık sayı tüm içeriğiyle, kesin bir şekilde saklanmak isteniyorsa *decimal* tip tercih edilmelidir. *Decimal* sayıların ikili bir tamsayı değeri ve değerin hangi bölümünün ondalık kesir olduğunu belirten bir tam sayı ölçeklendirme faktörü vardır. Para değerleri için de bu tür kullanılabilir. *Decimal* tipinin diğer ondalık veri tiplerine göre avantajı değerlerin tam olarak saklanabilmesiken diğer yandan sabit disk üzerinde daha fazla yer kapladıklarından hem program çalışma hızı hem de veri saklama kapasitesinin kullanım konusunda dezavantajlıdır. 128 bitlik bu veri tipi en fazla 29 anlamlı basamağı destekler ve mümkün olan en büyük ve küçük sayılar $+/-7.9228162514264337593543950335 \times 10^{28}$ dir. Sıfır olmayan en küçük *decimal* sayı da 0,00000000000000000000000000000001 olarak söylenebilir.

Decimal türünde tanımlama yaparken *p* (toplam basamak sayısı) ve *q* (toplam ondalık basamak sayısı) değerleri kullanılır. Bu değerlerin, hem gereksiz yer kaybından sakınacak hem de işlevlerinin kaybolmayacağı şekilde tanımlanması gerekmektedir. Örneğin saklanacak en yüksek değerinin 156,3098 sayısı olabileceği bir değişken, *decimal(7,4)* şeklinde deklare edilmelidir.

Single ve *double* veri türleri değerleri tam saklamayan, belli bir duyarlılığa sahip ondalık sayılardır. *Decimal* veri tipine göre daha büyük sayıları daha küçük boyutlarda saklayabilirken doğalarından kaynaklı olan yuvarlamalar sebebiyle sayının tam değerini tutamazlar. Bu türler daha hızlıdır ve daha az bellek gerektirir, ancak yuvarlama hataları içerirler. Bu sayılar, *mmm* değerlerinin mantis (anolamlı basamaklar) ve *eee* değerlerinin de üs (10'un gücü) olduğu *mmmEEE* olarak ifade edilirler.

Bir örnek ile anlatmak gerekirse, 1234567890123456789012345678 gibi bir sayı *single* veri tipinde 1,234567E+27 şeklinde bir sayıya dönüşür ve kalan 21 basamak 0 olarak eklenir. Bu, sayının daha küçük bir alanda kaydedilmesini sağlarken aynı zamanda kesin değerini kaybetmesine sebep olur. Bu kayıp adı geçen örnekte 72/100.000.000 gibi bir hata oranına tekabül etmektedir. Diğer yandan aynı sayı sabit diskte 128 bit yerine 32 bit tuttuğundan yer kazancı yüzde 75'tir. Bu yüzden bahsi geçen kayıp göz ardı edilebilir. Bu tür veri tipleri, verileri daha verimli bir şekilde depolayabilirler ve bellek tüketiminden tasarruf edilmesini sağlarlar.

Single veri tipi negatif değerler için $-3,4028235E+38$ ile $-1,401298E-45$ arasında, pozitif değerler içinse $1,401298E-45$ ile $3,4028235E+38$ arasında değişen, işaretli, 32 bit, tek duyarlıklı ondalık sayıları tutar. Diğer yandan *double* veri tipi negatif sayılar için $-1,79769313486231570E+308$ ile $-4,9406564584124$, pozitif sayılar içinse $4.94065645841246544E-324$ ile $1.79769313486231570E+308$ arasında değişen, işaretli, 64 bit, çift duyarlıklı ondalık sayıları tutar. İki tip veri de gerçek bir sayının yaklaşık bir halini depolar. *Double* veri tipi, tüm ondalık sayı veri tiplerinin en az duyarlılığından diğer yandan en verimlidir denilebilir. Buna rağmen *double* tiplerle yapılan işlemler tam sayılar ile yapılanlar kadar hızlı değildir.

2.1.2. Karakter Veri Tipleri

Programlama dillerinde yazdırılabilir ve görüntülenebilir karakterler için karakter veri tipleri bulunmaktadır. Bu türlerin en yaygın iki örneği *char* ve *string* tipleridir. Bunların her ikisi de Unicode karakterlerle

(Karakterler için oluşturulmuş bir metin standartıdır.) ilgilenirken, *char* veri tipi tek bir karakter, *string* ise belirsiz sayıda karakter içerir.

Bilgisayarlar aslında karakterleri değil sadece sayıları tanırlar. Bu yüzden aslında karakter veri tipleri içerisindeki değerleri sayı olarak saklar ve kullanırlar. Yaygın olarak kullanılan Unicode karakterler, ASCII karakterlerinin diğer çeşitli alfabe harfleri, para birimi simgeleri, kesirleri, matematiksel ve teknik simgeleri vb. ile genişletilmiş halidir. Bu listede örneğin “A” harfi 65 sayısı ile ifade edilirken “c” harfi de 99 sayısı ile ifade edilmektedir. Tüm ASCII listesine ulaşmak için eke (EK-1) bakılabilir.

Char veri tipi iki baytlık (16 bit) Unicode karakterdir. Bir değişken sadece bir karakter saklayacaksı bunu *char* olarak tanımlamak gereklidir. Eğer değişken daha fazla sayıda karakter içerecekse o zaman *string* tipi kullanılmalıdır. *String* veri tipi, 2 milyara yakın (2^{31}) karakter içerebilir, içerdeki her karakter için de 2 bayt yer tutar. Örneğin “Yönetim Bilişim Sistemleri” tamlamasında toplam 26 harf vardır ve bu tamlama sabit diskte 52 bayt alan kaplar.

2.1.3. Diğer Veri Tipleri

Diger veri tiplerinin belli başlılarından biri *boolean* veri türüdür. Bu veri tipi bir bit yer kaplar ve doğru/yanlış, evet/hayır, açık/kapalı gibi durumlar için kullanılır. İçerisinde 0 (*False*) veya 1 (*True*) şeklinde, iki durumdan birini içerir. Erkek/kadın gibi iki seçenekli bir veri saklanacaksa bunu 0 kadın, 1 erkek şeklinde dönüştürüp *boolean* olarak depolamak çok mantıklı bir hareket olacaktır.

Bir başka veri türü de *date* şeklinde ifade edilen hem tarih hem de zaman bilgisini tutan 64 bitlik veri tipidir. Her artış, Gregoryen takvimde 1. yılın 1 Ocak'ının başlangıcından (12:00) bu yana geçen 100 nanosaniyeyi (100 nanosaniye = 10^{-7} saniye = 0,0000001 saniye) temsil eder, kısaca gün, ay, yıl, saat, dakika, saniye bilgileri nanosaniye duyarlığında saklanır. Örneğin “01.01.2022 18:30:15,0000001” bu veri tipinde saklanan bir veridir. Bu veri tipi değişik programlama dillerinde farklı kullanım şekilleri gösterir.

Nesne veri tipi gibi program dillerine de bağlı olan birçok veri türü daha vardır. Algoritmaya giriş konusunda bu tür veri tiplerine ihtiyaç olmayacağından başka veri tipine detaylı olarak değinilmeyecektir.

2.2. Değişkenler ve Sabitler

Program işleyişinde kullanılan değerler değişkenlerin içinde tutulurlar ve gerektiğiinde işlemlerde kullanılırlar. Değişkenler programın çalışması esnasında RAM'e (geçici hafıza) kaydedilirler ve yapılan değişiklikler burada gerçek zamanlı olarak güncellenir. Bir değişken; ad, adres, veri tipi ve içerik olmak üzere dört parçadan oluşur.

Değişkenin adı (Tanımlayıcı olarak da geçer.), o değişkeni diğerlerinden ayırmaya yarar. Programcı değişkene istediği gibi isim verebilir ama değişken adının içinde sakladığı değerlere çağrılmış yapacak şekilde seçilmesi, programın anlaşılabilirliği açısından önemlidir. Değişken isimleri harf (İngilizce A-Z), rakam ve alt çizgi içerebilirler. İsim, rakamla başlayamaz, sadece harf veya alt çizgiyle başlayabilir. Kullanılan harflerin büyük veya küçük harf olmaları programlama diline göre fark eder veya etmez. Örneğin C# dilinde “elma” ile “ELMA” ayrı değişkenleri ifade ederken VB dilinde aynı değişkeni ifade ederler.

Değişkenlerin adresi, o değişkenlerin hafıza üzerinde kaydedildikleri yeri bildirir. Tüm değişkenler tanımlıklarında bilgisayar tarafından otomatik olarak bir adrese atanırlar ve bu adres üzerinden okunurlar. Programcinin değişkenle işi bittiğinde değişkeni temizleyerek hafızadaki bu adresi boşaltması mümkündür. Bazı programlama dilleri özel komutlarla programcinin bu adresleri belirlemesine ya da bu adreslere erişip doğrudan adres bilgisini kullanarak işlem yapmasına izin verirler. Bu tarz işlemler bir yandan yapılanın normalden hızlı bir şekilde yapılmasına olanak sağlarken, diğer yandan da içerdeği tehlikeden dolayı iyi programlama bilgisine sahip olmayı gerektirir. Buradaki tehlike, yanlış bir adreste yanlış bir işlem yaparak belki de bilgisayar işletim sistemini bir daha açılmaz hale getirmektir.

Değişkenlerin veri tipleri önceki bölümde (2.1. Temel Veri Tipleri) anlatılan veri tiplerinden birinin seçilmesiyle belirlenir. Bunun için değişkenin ne verisi saklayacağı önemlidir. Eğer değişkende tam sayı

tutulacağsa tam sayı tiplerinden biri seçilirken, ondalık sayı tutulacağsa ondalık veri tiplerinden biri seçilir. Burada önemli olan seçilen tipin değişkene atanacak değer aralığına uygun, en verimli veri tipi olmalıdır.

Her değişkenin içine programın değişik aşamalarında değerler atanır ve bu değerler en başta daha değişken tanımlanırken atanabilecekleri gibi program akışı içerisinde de atanabilirler. Değer atamanın bir limiti yoktur yani değişkenin içeriği defalarca değiştirilebilir. Örneğin bir satırda, “**b=28**” yazarsak *b* değişkenine 28 değerinin ataması yapılır. Burada önemli olan atanın değerin, değişkenin veri tipine uygun olması ve atanacak değerin eşitliğinin sağında, değeri alacak değişkenin de eşitliğinin solunda yer almazıdır.

Her programlama dilinde değişkenlerin yanı sıra sabitler diye adlandırılabilceğimiz özel bir değişken tipi daha bulunmaktadır. Sabitlerin değeri bir kez, tanımlama sırasında atanır ve bir daha değiştirilemez. Genelde KDV oranı, vergi oranı, para birimi gibi program boyunca değişimeyecek değerlerin kullanımında tercih edilirler. *Para_Birim* isimli sabite “**TL**” değeri atandığında, bu içerik hep aynı kalacaktır ve *Para_Birim* isimli sabit her zaman bu değerle kullanılacaktır. Buna bir örnek daha verilebilir, *Pi* isimli sabite “**3,14**” değeri atandığında bu içerik hep aynı kalacaktır ve pi sayısının değişme ihtimali de bulunmadığından yanlış bir durumun ortaya çıkması program boyunca söz konusu olmayacağıdır.

Değişkenlerle ilgili belirtilmesi gereken önemli bir not da değişkenlerin mutlaka program başında tanımlanmaları gereğidir ve hatta bunu yapmamak çoğu program dilinde hata mesajı alınmasına sebep olacaktır. Bu tanımlama şekli programdan programa değişir. Tanımlarken değişkenin adı ve tipi belirtilir, bunun yanı sıra eğer bir ilk değer alması gerekiyorsa o değerin ataması da yapılır. Örneğin *a* isimli bir tam sayı değişkeni kullanılcaksa ve bu değişkenin ilk değeri de 5 olacaksas programın başında *int* veri tipine sahip bir *a* değişkeni tanımlanır ve *a*'nın ilk değeri de 5 olarak atanır. Aşağıdaki tabloda (Tablo 4) bu tanımlamalarla ilgili bazı örnekler görülmektedir.

Tablo 4: Değişken Örnekleri

Tanımlama	VB	C #
<i>a</i> isimli bir string tanımlama	Dim <i>a</i> As String	string <i>a</i> ;
<i>b</i> isimli bir string tanımlayıp ilk değerini “Ali” olarak atama	Dim <i>b</i> As String = "Ali"	string <i>b</i> = "Ali";
maas isimli bir tam sayı tanımlama (değeri en fazla 100000 olabilir)	Dim <i>faiz</i> As Integer	int <i>faiz</i> ;
<i>faiz</i> isimli bir tam sayı tanımlama ve ilk değerini 15000 atama (değeri en fazla 100000 olabilir)	Dim <i>faiz</i> As Integer = 15000	int <i>faiz</i> = 15000;

2.3. Temel İşlemler ve Operatörler

Programlamadaki en temel işlemler, aritmetiksel işlemleridir. Matematiksel ve benzeri birçok işlem operatörleri yardımı ile yapılır. Operatörler, değişkenler üzerinde çeşitli değişiklikler yapmamızı sağlarlar. Birçok operatör, programlama dilleri arasında ortak kullanılırken, bazı işlemler için bazı programlama dilleri farklı operatörler kullanır. Bir kısım programlama dilinde diğerlerinde olmayan operatörlere de rastlanabilir.

Operatörlerden bahsedildiğinde akla ilk olarak kodlarda en çok kullanılan aritmetik operatörler gelir. Bunlar, değişkenler ve veri tipleri üzerinde toplama, çıkarma, çarpma, bölme, atama, üs alma, mod alma gibi cebirsel işlemleri yapmamızı sağlayan operatörlerdir. Bu operatörler çoğu programlama dilinde ortaktır. Aşağıdaki tabloda (Tablo 5) aritmetik operatörlerin iki dil ailesindeki karşılıkları gösterilmiştir (buradaki örneklerde $x=7$, $y=2$ olarak alınmıştır).

Tablo 5: Aritmetik Operatörler

İşlem	Açıklama	VB	C #	Sonuç
Toplama	İki değeri toplar.	x + y	x + y	9
Cıkarma	Bir değeri diğerinden çıkarır.	x - y	x - y	5
Çarpma	İki değeri çarpar.	x * y	x * y	14
Bölme	Bir değeri diğerine böler.	x / y	x / y	3.5
Tam Sayı Bölme	Bir değeri diğerine bölüp tam sayı döndürür.	x \ y	--	3
Üs alma	Bir değerinin üssünü alır.	x ^ y	--	49
Mod alma	Bölmeye kalanı döndürür.	x mod y	x % y	1
Arttırma	Değişkenin değerini bir arttırır.	x++	x++	8
Arttırma	Değişkenin değerini bir artırır.	++x	++x	8
Azaltma	Değişkenin değerini bir eksiltir.	x--	x--	6
Azaltma	Değişkenin değerini bir eksiltir.	--x	--x	6

Bir başka operatör tipi de karşılaştırma operatörleridir. Bu operatörler özellikle daha sonra anlatılacak olan kontrol yapılarında ve döngülerde kullanılmaktadırlar (Tablo 6). Karşılaştırma operatörleri, iki değişken arasında belirli bir koşula bağlı olarak akışın hangi yönde ilerleyeceğine karar vermede kullanılan operatörlerdir. Bu operatörlerin kullanımı sonucunda *TRUE* (Doğru) veya *FALSE* (Yanlış) değerleri elde edilir. Aşağıdaki tabloda bu operatörler gösterilmiştir (Buradaki örneklerde $x=10$, $y=20$ olarak alınmıştır.).

Tablo 6: Karşılaştırma Operatörleri

İşlem	Açıklama	VB	C#	Sonuç
Eşittir	İki değerin eşit olup olmadıklarını kontrol eder. Eğer eşitlerse sonuç "Doğru" olur.	x = y	x == y	FALSE (Yanlış)
Eşit değildir	İki değerin eşit olup olmadıklarını kontrol eder. Eğer eşitler değilse sonuç "Doğru" olur.	x <> y	x != y	TRUE (Doğru)
Büyükür	Soldaki değerin sağdaki değerden büyük olup olmadığını kontrol eder. Eğer büyükse sonuç "Doğru" olur.	x > y	x > y	FALSE (Yanlış)
Küçüktür	Soldaki değerin sağdaki değerden küçük olup olmadığını kontrol eder. Eğer küçükse sonuç "Doğru" olur.	x < y	x < y	TRUE (Doğru)
Büyükür veya eşittir	Soldaki değerin sağdaki değerden büyük veya eşit olup olmadığını kontrol eder. Eğer büyük veya eşitse sonuç "Doğru" olur.	x >= y	x >= y	FALSE (Yanlış)
Küçüktür veya eşittir	Soldaki değerin sağdaki değerden küçük veya eşit olup olmadığını kontrol eder. Eğer küçük veya eşitse sonuç "Doğru" olur.	x <= y	x <= y	TRUE (Doğru)

Sıradaki operatör tipi, programın işlemesini sağlayan atama operatörleridir. Bu operatörler sayesinde, yapılan hesaplamalar değişkenlere atanır, bunun sonucunda da programlar işlevlerini yerine getirirler. Aşağıdaki tabloda (Tablo 7) bu operatörler gösterilmiştir (Buradaki örneklerde $x=7$, $y="aa"$ ve $z="bb"$ olarak alınmıştır.).

Tablo 7: Atama Operatörleri

İşlem	Açıklama	VB	C#	Sonuç
=	Basit atama operatörüdür, sağdaki değeri soldakine atar.	x = 5	x = 5	x = 5 x'in değeri 5 olur.
+=	Ekleyp atama operatörüdür, sağdaki değeri soldakine ekleyp çıkan değeri soldaki değişkene atar.	x += 2	x += 2	x = x + 2 x'in değeri 9 olur.
-=	Çıkarıp atama operatörüdür, sağdaki değeri soldakinden çıkarıp çıkan değeri soldaki değişkene atar.	x -= 2	x -= 2	x = x - 2 x'in değeri 5 olur.
*=	Çarpıp atama operatörüdür, sağdaki değeri soldakiyle çarpıp çıkan değeri soldaki değişkene atar.	x *= 2	x *= 2	x = x * 2 x'in değeri 14 olur.
/=	Bölüp atama operatörüdür, soldaki değeri sağdakine bölüm çıkan değeri soldaki değişkene atar.	x /= 2	x /= 2	x = x / 2 x'in değeri 3.5 olur.
\=	Tam sayı olarak bölüm atama operatörüdür, soldaki değeri sağdakine bölüm çıkan değerin tam sayı kısmını soldaki değişkene atar.	x \= 2	--	x = x \ 2 x'in değeri 3 olur.
^=	Üs alma operatörüdür, soldaki değerin sağdaki kadar kuvvetini alıp soldaki değişkene atar.	x ^= 2	--	x = x ^ 2 x'in değeri 49 olur.
&=	Metin birleştirme operatörüdür, sağdaki değeri soldaki değerin arkasına ekleyp soldaki değişkene atar.	y &= z	y += z	y = y & z y = y + z y'nin değeri "aabb" olur.
%=	Mod alıp atama operatörüdür. Soldaki değeri sağdakine bölüm kalan değerini soldaki değişkene atar.	--	x %= 2	x = x % 2 x'in değeri 1 olur.

Bu bölümde son olarak mantıksal operatörlere değinmek gereklidir. Mantıksal operatörler, sonucu *boolean* olarak çıkan önermelerin aralarında işlem yapılmasını ve bu işlemlerin mantıksal doğruluk tablolarına göre birleştirilmesini sağlar. Aşağıdaki tabloda (Tablo 8) bu operatörler gösterilmiştir, buradaki örneklerde *x* değişkeni *TRUE* (Doğru), *y* değişkeni ise *FALSE* (Yanlış) olarak alınmıştır. Bunu daha detaylı açıklamak gerekirse *a* değeri 2 iken, *x* önermesi (*a*=2), *y* önermesi ise (*a*=5) gibi düşünülebilir.

Tablo 8: Mantıksal Operatörler

İşlem	Açıklama	VB	C#	Sonuç
AND	Her iki önerme de doğruysa, çıktı doğru olur. Önermelerden bir bile yanlışsa çıktı yanlış olur.	x AND y	x && y	FALSE (Yanlış)
OR	Önermelerden biri bile doğruysa çıktı doğru olur. Önermelerin her ikisi de yanlışsa çıktı yanlış olur.	x OR b	x y	TRUE (Doğru)
NOT	Önerme doğruysa çıktı yanlış olur, önerme yanlışsa çıktı doğru olur.	NOT x	! x	TRUE (Doğru)

2.4. Sıralı İşlemler ve Basit Algoritmalar

Değişkenler, değişkenlerin içerebilecekleri veri tiplerini ve operatörleri gördükten sonra sıra bu değişkenleri kullanarak programlarda yapılabilecek temel işlemlere gelmiştir. Öncelikle belirtilmelidir ki, bilgisayar programları aksine bir komut yapısı kullanılmadıkça (kontrol yapısı, döngü vb.) sırayla çalışırlar, yani bir satırдан sonra ardından gelen satır, onun da ardından bir sonraki satır yürütülür ve bu böyle devam eder. Örneğin, eğer bir programda 500 satır varsa, sıralı komut dışında bir komut olmadığı takdirde bu satırların hepsi arkaya arkaya çalıştırılırlar.

BAŞLA
a = 5
b = 4
c = a*b
BİTİR

Yukardaki örnekte önce a değişkenine 5 değeri atanmakta, ardından da b değişkenine 4 değeri atanmaktadır. Bundan sonra a ve b değişkenlerine başka bir sayı atanana veya programın ilgili kısmı bitene kadar bu iki değer hafızada bu şekilde saklanacaktır. Dördüncü adımda c değişkenine a ve b 'nin çarpımları atanmıştır, yani şu anki değerlerle, c değişkeni 20 değerine eşitlenmiştir. İleride c değişkeni değiştirilmeden kullanılırken 20 olarak düşünülmelidir.

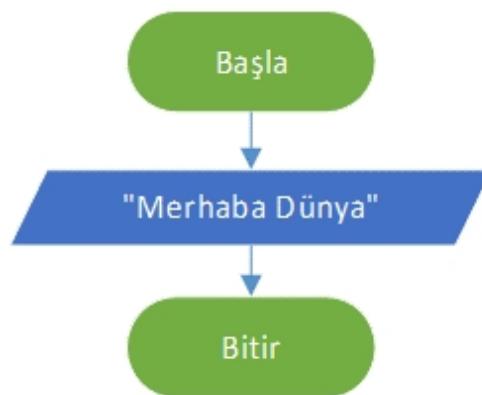
Belirtilmesi gereken bir başka nokta da, artırma için kullanılan iki artı işaretinin sağında olduğunda önce o satırda işlem yapılır sonra değer artırılır. İki artı işaretinin solundaysa önce değer artırılır sonra satırda işlem yapılır. Bunlar azaltma için kullanılan iki eksiz işaretinde de geçerlidir. Bununla ilgili olarak aşağıdaki örnek programa bakılabilir. Örnekte, programın her satırından sonra, değerleri değişen değişkenlerin içerikleri sağda gösterilmiştir:

BAŞLA	a Değeri	b Değeri	c Değeri
.....			
a=3	3		
b=a++	4	3	
c=++a	5		5
.....			
BİTİR			

Programlar parçasını oluşturdukları sistemler gibi girdi ve çıktılara sahiptirler. Girdiler program içindeki algoritma ile işlenerek çıktıya dönüşürler. Çıktısı olmayan program olmaz, eğer bir program çıktı üretmiyorsa o program sadece kendi için çalışıyordu ve boşuna kaynak tüketiyordur. Değişik dillerde farklı girdi ve çıktı komutları kullanılır. Girdi için *scanf*, *input* gibi kullanıcıdan klavye vasıtasyyla değer alan komutlar kullanılırken, çıktılar için *writeln*, *printf*, *print* gibi ekrana değer bastıran komutlar vardır. Aşağıda bazı sıralı algoritma örneklerini açıklamalarıyla görebilirsiniz:

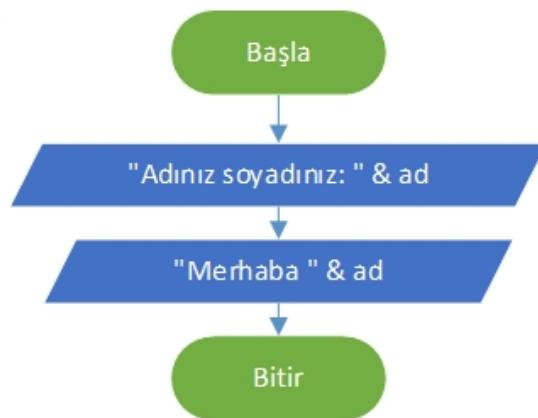
- a) Aşağıdaki programda kullanıcıdan adı ve soyadı alınarak ekran'a "Merhaba Dünya" yazdırılmaktadır.

BAŞLA
YAZ ("Merhaba Dünya")
BİTİR



- b) Aşağıdaki programda kullanıcidan adı ve soyadı alınarak ekrana “Merhaba” ve kullanıcının ismi yazdırılmaktadır.

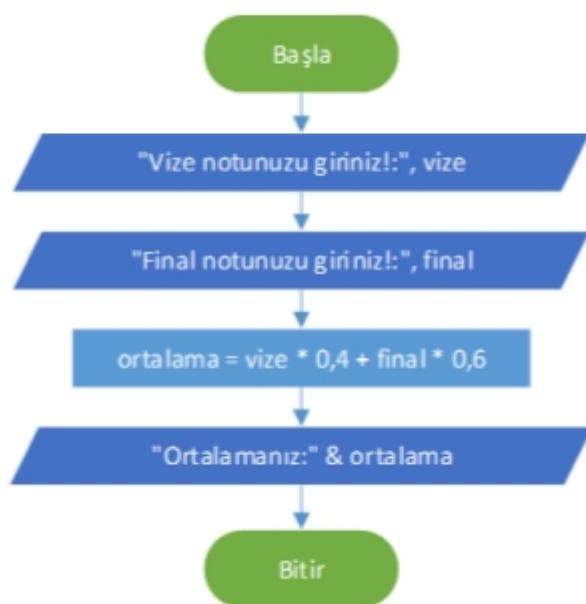
BAŞLA
OKU ("Adınız soyadınız:", ad)
YAZ ("Merhaba " & ad)
BİTİR



- c) Aşağıdaki programda kullanıcidan dersle ilgili vize ve final notları alınarak ağırlıklı ortalaması hesaplanıp ekrana yazdırılmaktadır (Vize ağırlığı %40, Final ağırlığı %60 olarak belirlenmiştir.).

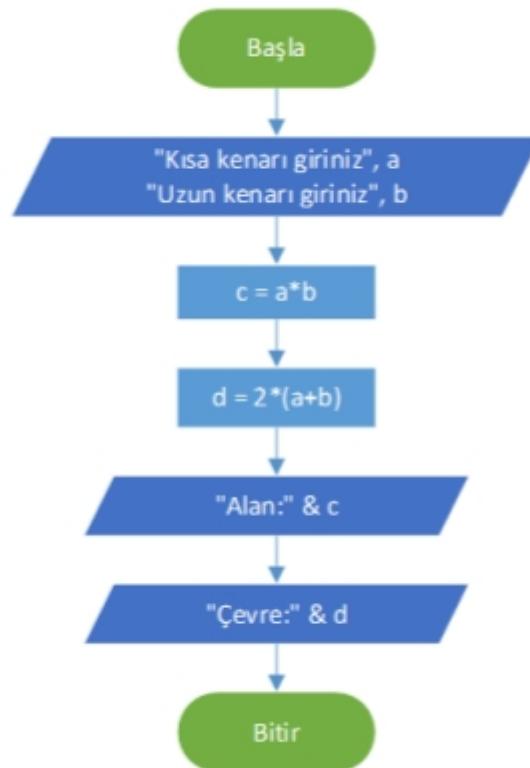
d)

BAŞLA
OKU ("Vize notunuzu giriniz!", vize)
OKU ("Final notunuzu giriniz!", final)
ortalama=vize*0,4+final*0,6
YAZ ("Ortalamanız:" & ortalama)
BİTİR



Aşağıdaki programda kullanıcıdan bir dikdörtgenin kısa ve uzun kenarlarının uzunluğu alınarak dikdörtgenin alanı ve çevresi hesaplanıp ekrana yazdırılmaktadır.

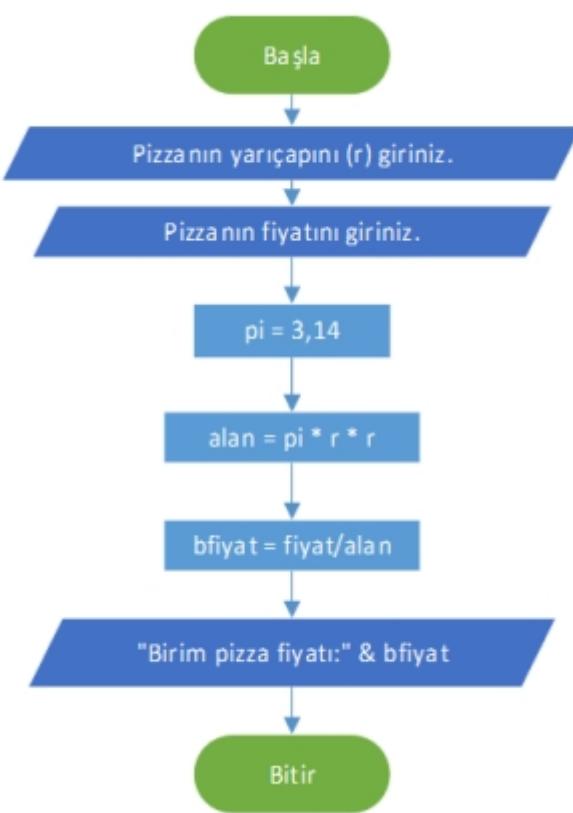
BAŞLA
OKU ("Kısa kenarı giriniz!", a)
OKU ("Uzun kenarı giriniz!", b)
c=a*b
d=2*(a+b)
YAZ ("Alan:" & c)
YAZ ("Çevre:" & d)
BİTİR



e) Aşağıdaki programda kullanıcıdan bir pizzanın yarıçapı ve toplam fiyatı alınarak pizzanın birim cm^2 fiyatı hesaplanıp ekrana yazdırılmaktadır.

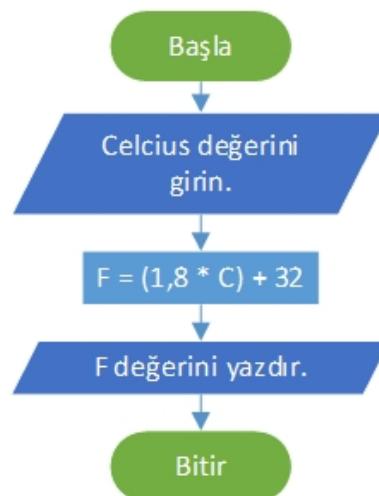
f)

BAŞLA
OKU ("Pizzanın yarıçapını giriniz", r)
OKU ("Pizzanın fiyatını giriniz", fiyat)
pi = 3,14
alan = pi*r*r
bfiyat = fiyat/alan
YAZ ("Birim pizza fiyatı: " & bfiyat)
BİTİR



Celsius cinsinden girilen sıcaklık değerini 1,8 ile çarpıp 32 ekleyerek Fahrenheit cinsine çeviren ve görüntüleyen program aşağıda yer almaktadır.

BAŞLA
Celsius (C) cinsinden sıcaklık değerini OKU
$F = (1,8 * C) + 32$
YAZ (F)
BİTİR



g) Aşağıdaki programda girilen iki sayının toplamını hesaplanmakta ve sonuç ekrana yazdırılmaktadır.

h)

BAŞLA
OKU ilk sayı
OKU ikinci sayı
$Toplam = ilk\ sayı + ikinci\ sayı$
YAZ (toplam)
BİTİR



Çalıştığı saat başına ücret alan bir çalışanın ücretini hesaplayıp ekrana yazdırılan program aşağıda yer almaktadır.

BAŞLA
OKU Çalışma saati
OKU Saatlik ücret
Toplam Ücret = Çalışma saati * Saatlik ücret
YAZ (Toplam Ücret)
BİTİR



Programlarda, özellikle döngülerde, bazı değişkenlerin işlem adedi gibi sayılması gereken değerleri tutması beklenmektedir. Bu tarz amaçlarla kullanılan değişkenlere genelde **sayaç** (*counter*) ismi verilir. Diğer yandan bazı değişkenlerin de iki ihtimalli bir durum belirten (açık/kapalı, satıldı/satılmadı, işlendi/işlenmedi vb.) değerleri tutması beklenir, bunlara da genel olarak **işaret** (*flag*) adı verilir. Bu tip değişkenlerle genelde döngü kontrolü yapılmaktadır. Bu iki tür değişken bir sonraki bölümde (4. DÖNGÜLER) detaylı olarak ele alınacaktır.

Programlardaki önemli problemlerden biri de iki değişkenin tuttukları değerleri birbirleriyle değiştirmektir. Özellikle sıralama algoritmaları gibi algoritmalarda bu durum sık sık karşımıza çıkmaktadır. Örneğin a ve b değişkenlerinin içeriklerini değiştirmek için önce ($a=b$) sonra ($b=a$) yapıldığı takdirde iki değişkende de b 'nin ilk değeri olacaktır (Once a 'ya b değeri atanacak, sonra b 'ye a yani eski b tekrar atanacak). Bu yüzden, bu gibi durumlarda geçici bir değişken kullanmak yerinde olacaktır, bu geçici değişken de genelde *temp* (İngilizce *temporary*, geçici demektir.) olarak adlandırılır. Sayaçlar ve değişken değerlerinin birbirlerine aktarılması ile ilgili örnekler aşağıda verilmiştir.

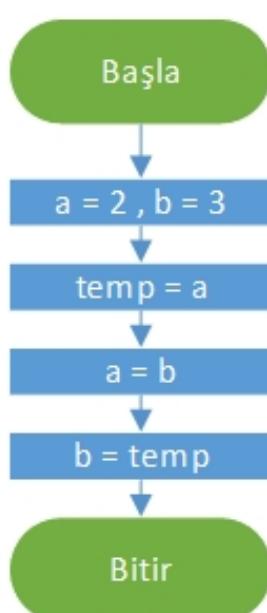
- i) Aşağıdaki program sayaç vasıtısıyla kullanıcının girdiği renk adedini tutmaktadır.

BAŞLA
sayaç=0
OKU ("Bir renk giriniz", r1)
sayaç++
OKU ("Bir renk giriniz", r2)
sayaç++
OKU ("Bir renk giriniz", r3)
sayaç++
OKU ("Bir renk giriniz", r4)
sayaç++
YAZ (sayaç & " adet renk girdiniz")
BİTİR



- j) Aşağıdaki programda ilk olarak a'ya 2, b'ye 3 değeri atanmıştır. temp isimli geçici değişken kullanılarak a ve b değişkenlerinin içerikleri değiştirilmiş, a 3, b ise 2 olmuştur.

BAŞLA
a=2
b=3
temp=a
a=b
b=temp
BİTİR



EK-1

ASCII KARAKTER KODLARI TABLOSU

KOD	CHAR	KOD	CHAR	KOD	CHAR	KOD	CHAR	KOD	CHAR	KOD	CHAR
000	(nul)	043	+	087	W	138	É	185	¶	221	█
001	(soh)	044	,	088	X	139	Ó	186		222	█
002	(stx)	045	-	089	Y	140	Ó	187	₩	223	█
003	(etx)	046	.	090	Z	141	í	188	₪	224	á
004	(eot)	047	/	091	[142	f	189	Ј	225	þ
005	(enq)	048	0	092	\	143	≈	190	Ј	226	ȝ
006	(ack)	049	1	093]	144	…	190	Ј	226	ȝ
007	(bel)	050	2	095	-	145	È	191	Ղ	227	π
008	(bs)	051	3	096	,	146	Δ	192	Լ	228	Σ
009	(tab)	052	4	097	a	148	Ú	193	Լ	229	σ
010	(if)	053	5	098	b	149	º	194	Տ	230	μ
011	(vt)	054	6	099	c	150	·	195	Ւ	231	τ
012	(np)	055	7	100	d	151	º	196	—	232	Փ
013	(cr)	056	8	101	e	152	Ö	197	+	233	ø
014	(so)	057	9	103	g	154	Ü	198	Ւ	234	Ω
015	(si)	058	:	104	h	155	-	198	Ւ	235	δ
016	(dle)	059	:	105	i	156	£	199		236	∞
017	(dc1)	060	<	106	j	157	\$	200	Լ	237	∅
018	(dc2)	061	=	107	k	158	§	201	Ր	238	ε
019	(dc3)	062	>	108	l	159	§	202	Լ	239	∩
020	(dc4)	063	?	109	m	160	:	203	Ր	240	≡
021	(nak)	064	@	110	n	161	Ü	204	Ւ	241	±
022	(syn)	065	A	111	o	162	·	205	=	242	≥
023	(etb)	066	B	112	p	163	Ò	206	‡	243	≤
024	(can)	067	C	114	q	164	□	207	—	244	[
025	(em)	068	D	115	r	165	▫	208	—	245]
026	(eof)	069	E	116	s	166	ğ	209	—	246	÷
027	(esc)	070	F	117	t	167	TL	210	—	247	≈
028	(fs)	071	G	118	u	168	—	211	—	248	o
029	(gs)	072	H	119	v	169	—	212	—	249	“
030	(rs)	073	I	120	w	170	Ω	213	—	250	-
031	(us)	074	J	121	x	171	▫	214	—	251	√
032	sp	075	K	122	y	172	0	215	—	252	∩
033	!	076	L	123	z	173	·	216	‡	253	≥
034	í	077	M	124	{	174	¤	217	—	254	■
035	#	078	N	125	}	175	¤¤¤	218	—	255	
036	\$	079	O	126	~	176	¤¤¤	219	—		
037	%	080	P	127		177	¤¤¤	220	—		
038	&	081	Q	128	ç	178	¤¤¤				
039	é	082	R	129	ü	179	—				
040	(083	S	131	É	180	—				
041)	084	T	132	‰	181	—				
042	*	085	U	133	‡	182	—				
		086	V	134	À	183	—				
				135	ç	184	—				
				136	í						
				137	í						

Kaynak: https://www.emo.org.tr/ekler/fe4e15b3924b1a7_ek.doc?tipi=34&turu=X&sube=0

Bölüm Özeti

Bu bölümde programlamaya giriş için gerekli olan temel bilgilerin verilmesi amaçlanmıştır. Buna göre öncelikle temel veri tiplerinden bahsedilmiştir. Veri tipleri; sayısal veri tipleri, karakter veri tipleri ve diğer veri tipleri olarak üç farklı başlık altında ele alınmıştır.

Sayısal veri tipleri kendi içinde tam sayılar ve ondalıklı sayılar olmak üzere ikiye ayrılır. Bunların da farklı alt tipleri mevcuttur. Tam sayı veri tipleri, saklanmak istenen verinin türüne göre *sbyte*, *byte*, *short*, *ushort*, *int*, *uint*, *long*, *ulong* veri tipleri ile ifade edilebilir. Bu veri tiplerinden bazıları işaretli bazılarıysa işaretsiz olarak tanımlama yapılmasına izin verir. Yani bazılarında pozitif ve negatif sayısal değer tanımlanabilirken bazılarında sadece pozitif değerler tanımlanabilir. İşaretli olan tam sayı tipleri *sbyte*, *short*, *int*, *long*, işaretsiz olan tam sayı veri tipleri *byte*, *ushort*, *uint*, *ulong* veri tipleridir. Bu farklı veri tiplerinin bellekte kapladığı

alan ve programın çalışmasına olan etkisi fark etmektedir. Sayısal veri tiplerinden ondalıklı sayılar da birden farklı şekilde ifade edilebilir. *decimal*, *single* ve *double* ondalıklı sayıları tanımlarken kullanılabilen veri tipleridir. Bunlar ondalıklı sayıları farklı şekillerde tuttuğu için her birinin birbirine göre hassasiyeti ve bellekte kapladığı alan farklılık gösterir.

İkinci başlıkta ele alınan karakter veri tipi ise yine iki farklı veri türünü içerir. Bunlardan *char* veri tipi tek bir karakter tutarken, *string* veri tipi ise belirsiz sayıda karakterden oluşur. Bu karakterlerin sayısal birer değer olarak bilgisayar tarafından algılanması ve işlenebilmesi için her bir karakterin sayısal karşılığı uluslararası bir standart çerçevesinde belirlenmiştir. Bu karakterler ASCII tablosu ve bu tablo dışındaki farklı değerlerden oluşan Unicode karakterleri ile temsil edilir. Bir sonraki başlıkta yer alan diğer veri tiplerinde ise *boolean* değerler ele alınmıştır. Bu veri tipi ile iki seçimi bir yapı doğru/yanlış, evet/hayır gibi durumlar ifade edilir.

Değişkenler ise ad, adres, veri tipi ve içerik olmak üzere dört parçadan oluşan ve her programda mutlaka kullanılan elemanlardır. Değişkenler, bahsedilen farklı veri tiplerindeki değerleri alırlar. Değişkenlerin değeri programın yürütülmesi esnasında değiştirilebilir. Bir değişkene isim verilirken, ismin, değişkenin aldığı değer ile ilgili olarak mantıklı bir çağrı yapması gereklidir. Tabi bu yapılrken değişken isminin başlayamayacağı sayısal değerler gibi değerlerin kullanılmamasına dikkat etmek ve değişkeni mutlaka programın başında tanımlamak gereklidir. Sabit değerler ise değişkenlerden farklı olarak programda kullanılmak üzere bir kez tanımlanır ve program süresince değeri değiştirilemez.

Bir sonraki konu başlığında temel işlem ve operatörlerden bahsedilmiştir. Programlama dilinden diline, ifade şekilleri farklılık gösterebilse de her dilde temel mantık aynıdır. Temel aritmetik işlemlerin yapılmasını sağlayan operatörler, sonucu *true/false* olarak döndüren karşılaştırma operatörleri, bir değişkene değer atanmasını sağlayan atama operatörleri ve mantıksal karşılaşmalar yapılmasını sağlayan mantıksal operatörler mevcuttur.

En son bölümde ise ardışık olarak gerçekleşen sıralı işlemler ve basit algoritmalar ele alınmıştır. Sıralı işlemler, koşul ve döngü yapılarının olmadığı ardışık olarak gerçekleşen yapılardır ve bu bölümde sıralı işlemler içeren algoritmalar verilmiştir.

Ünite Soruları

Soru-1 :

Aşağıdakilerden hangisi tam sayı veri tiplerinden değildir?

(Çoktan Seçmeli)

- (•) - sbyte
- (•) - short
- (•) - int
- (•) - sint
- (•) - ulong

Cevap-1 :

sint

Soru-2 :

Aşağıdakilerden hangisi karakter veri tipindedir?

(Çoktan Seçmeli)

 - Char - strng - byte - letter - stg**Cevap-2 :**

Char

Soru-3 :**Aşağıdakilerden hangisi aritmetik işlem operatörü değildir?**

(Çoktan Seçmeli)

 - toplama - çıkarma - kesişim kümesi oluşturma - mod alma - üs alma**Cevap-3 :**

kesişim kümesi oluşturma

Soru-4 :**Aşağıdakilerden hangisi mantıksal işlem operatörlerindendir?**

(Çoktan Seçmeli)

 - AND NOT - OR AND - NOT - OR NOT - NOT OR**Cevap-4 :**

NOT

Soru-5 :**Aşağıdakilerden hangisi karşılaştırma operatörüdür?**

(Çoktan Seçmeli)

- (•) - Eşittir
- (•) - küçük veya büyütür
- (•) - true
- (•) - false
- (•) - eştir

Cevap-5 :

Eşittir

Soru-6 :**Karakterler hangi tablo referans alınarak tanımlanmaktadır?**

(Çoktan Seçmeli)

- (•) - char Tablosu
- (•) - akış tablosu
- (•) - ASCII
- (•) - AXCI
- (•) - Karakter tablosu

Cevap-6 :

ASCII

Soru-7 :**Aşağıdakilerden hangisi sayısal ve karakter tipindeki verilerin dışındaki veri tiplerindendir?**

(Çoktan Seçmeli)

- (•) - string
- (•) - boolean
- (•) - float
- (•) - short
- (•) - char

Cevap-7 :

boolean

Soru-8 :

Basit algoritma yapısında işlemler birbirini nasıl takip eder?

(Çoktan Seçmeli)

- (•) - Ardışık olarak
- (•) - Bazı adımları atlayarak
- (•) - Bazı adımları tekrarlayarak
- (•) - Sondan başa doğru
- (•) - Hiçbiri

Cevap-8 :

Ardışık olarak

Soru-9 :

AND mantıksal operatöründe sonucun doğru (TRUE) döndürülmesi için aşağıdakilerden hangi koşulun sağlanması gereklidir?

(Çoktan Seçmeli)

- (•) - Durumlardan birinin doğru olması
- (•) - Durumlardan en az birinin doğru olması
- (•) - Her iki durumun doğru olması
- (•) - Her iki durumun yanlış olması
- (•) - Hiçbiri

Cevap-9 :

Her iki durumun doğru olması

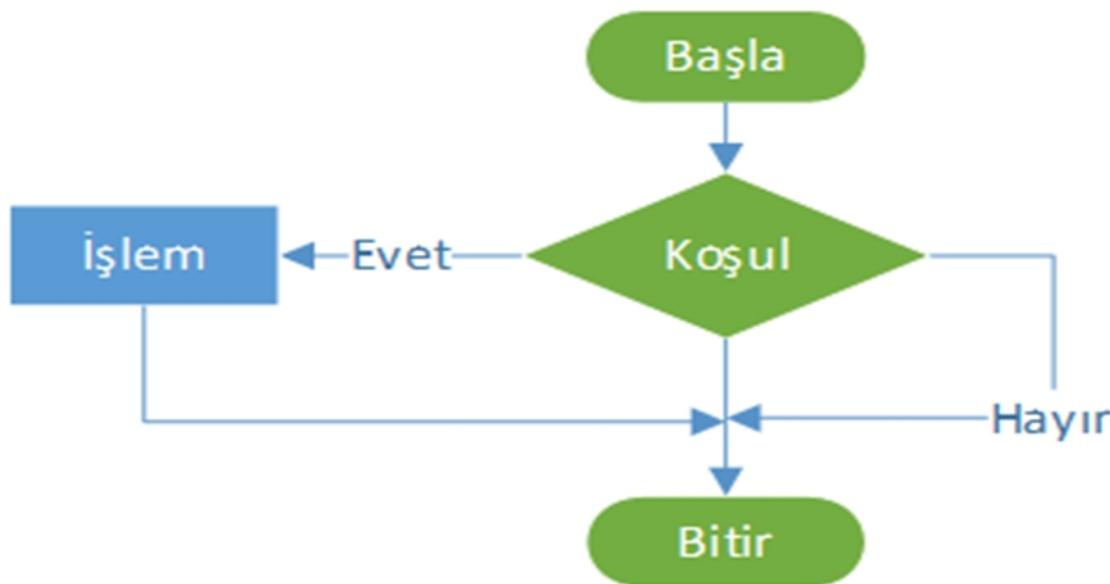
3.KARAR YAPILARI

3.1. Karar Yapıları

Karar yapıları/koşullu yapılar, bir program akışının belli bir koşul veya koşullara bağlı olarak değiştmesini sağlar. Böylece bir koşula bağlı olarak programa hangi işlemin gerçekleştirileceğine karar verilir (Çobanoğlu, 2014). Yani bir programa işlem adımları her zaman birbirlerini takip etmek zorunda değildirler. Ardışık olarak birbirini takip eden adımlar yerine bir duruma veya koşula göre problemin çözümünde veya işlem akışında farklı bir adıma atlanabilir. Burada bir koşulun sağlanması, bir durumun kontrol edilmesi veya bir karar verilmesi söz konusudur. Bu yapılar aynı zamanda seçimli yapı olarak da adlandırılabilirler.

3.1.1. Tek Seçimli Yapı (*if* Yapısı)

Karar yapılardan ilki *if* yapısıdır. *if* yapısı, bir koşulun sağlanması yani doğru (*true*) olması durumunda bir eylemin gerçekleştirilmesini sağlar, koşulun sağlanmaması durumunda ise bir şey yapılmaz (Deitel & Deitel, 2015). Burada tek seçimli bir yapıdan bahsetmek gerekir. Sadece tek bir koşula bağlı olarak değişen işlem sırası söz konusudur. Tek seçimli karar yapılarının temel gösterimi şu şekilde ifade edilebilir:



if deyimine günlük hayatdan örnek vermek gereklidir:

- Eğer 18 yaşını doldurduysan ehliyet alabilirsin.
- Eğer yemek yediysen ilacını al.
- Eğer vize ve final ortalaman 50'den düşükse dersten kalırsın.

Tüm bu örneklerde bir koşul ve ardından gelen bir eylem mevcuttur. Koşulun sağlanmaması durumunda farklı bir işlem gerçekleştirilmemektedir. İlkörnekte, kişi 18 yaşını doldurduysa ehliyet alabilir, yaşı yetmiyorsa ehliyet alamaz yani bu koşul sağlanana kadar ehliyet alma eylemini gerçekleştiremez. Koşulun sağlanmaması durumunda farklı bir şey yapılmaz. İkinciörnekte, kişi yemeğini yediye sonra ilacını alması gerekmektedir, yemeğini yemediyse ne yapması gereği söylenmez. Üçüncüörnekte ise, iki sınav notunun ortalamasının 50'den düşük olması durumunda kişinin bir dersten kalacağı belirtilmiştir. İki sınavın ortalaması 50'den düşükse dersten kalacaktır, büyük veya 50'ye eşitse dersten geçecektir ancak tek koşullu yapıda bu durum belirtilmez. Burada sadece ortalamanın 50'ye göre küçük mü büyük mü olduğunu kontrolü yapılır, bunun sonucuna göre mevcut ortalama 50'den düşükse kişinin kaldığı sonucu ortaya konar, büyükse farklı bir eylem gerçekleştiriliip gerçekleştirilmeyeceği belirtilmez. Koşul, not ortalamasının

değerine bağlıdır. Çift seçimli yapılarda ise koşulun sağlandığı ve sağlanmadığı durumlarda farklı işlemlerin takip edilmesi sağlanır. Bu yapıdan bir sonraki konu başlığı altında bahsedilmiştir. Tek seçimli yapıya örnek olarak kişinin girdiği şifre doğruysa sisteme giriş yapmasına izin verilen program aşağıda verilmiştir:

BAŞLA
OKU Şifre
EĞER Şifre doğruysa sisteme giriş yap.
BİTİR



if deyimi, farklı programlama dillerinde farklı şekillerde yazılabilir ancak temel mantığı yukarıda anlatıldığı gibidir. Burada unutulmaması gereken nokta, her programlama dilinde gösterimlerin ve yazım şekillerinin değişebildiğiidir. Bu duruma kitabın 2. bölümünde de değinilmiştir. Temel mantık tüm dillerde benzerdir. Örneğin C# dilinde *if* komutunun koşulu normal parantez () içine alınırken şarta bağlı komutlar süslü paranteze {} alınmaktadır. Diğer yandan VB'de *if* deyimi ile birlikte *then* ve döngünün sonlandırılması için *end if* deyimleri birlikte kullanılır. Bu yapı aşağıda da görüleceği üzere tek satırda gösterilebileceği gibi alt alta satırlar halinde de gösterilebilir. Kodun rahat okunabilmesi için, kesin kural olmasa da genel uygulama, komutların uygun şekilde girintili olarak yazılmasıdır.

- **If [koşul] Then**

[işlem 1] → Koşul sağlandığında devam edilecek işlem

End If

- **If [koşul] Then [işlem 1] End If**

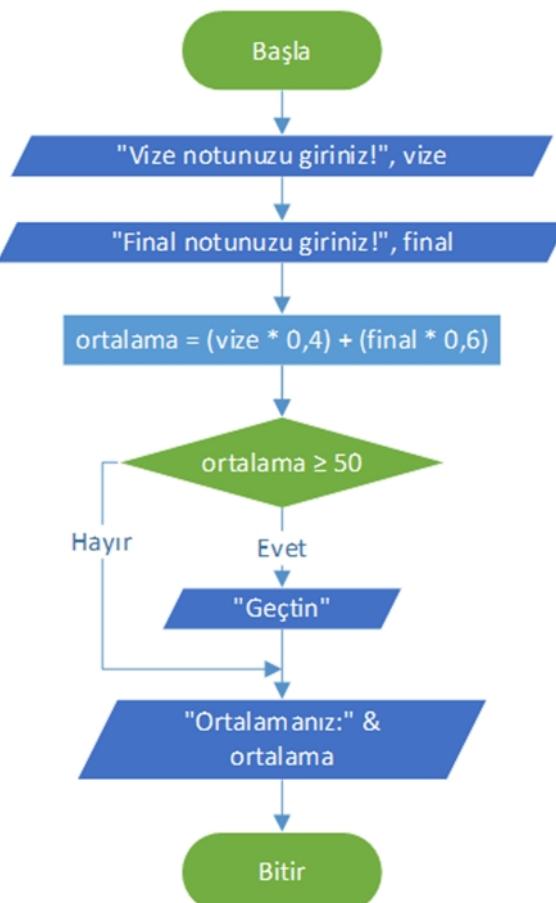
Saat 7:00 olduğunda uyanma eyleminin gerçekleşmesi gerektiğine dair bir örnek VB dilinde aşağıdaki gibi yazılabılır.	x sayısının 10 ile karşılaştırıldığı örnek aşağıda verilmiştir.
If (Saat = 7:00) Then Uyan End If	If (x ≥ 10) Then YAZ ("x 10'dan büyüktür.") End If

Tek seçimli *if* ile ilgili bazı örnekler aşağıda görülebilir:

a) Kullanıcıdan vize ve final notlarını alan ve bu notlara göre ortalama hesaplayan programın kodu aşağıda verilmiştir. Program, hesaplamadan sonra kullanıcının not ortalaması 50'den yüksekse "Geçtin" şeklinde bir çıktı vermektedir ve ortalamasını da ekrana yazdırmaktadır.

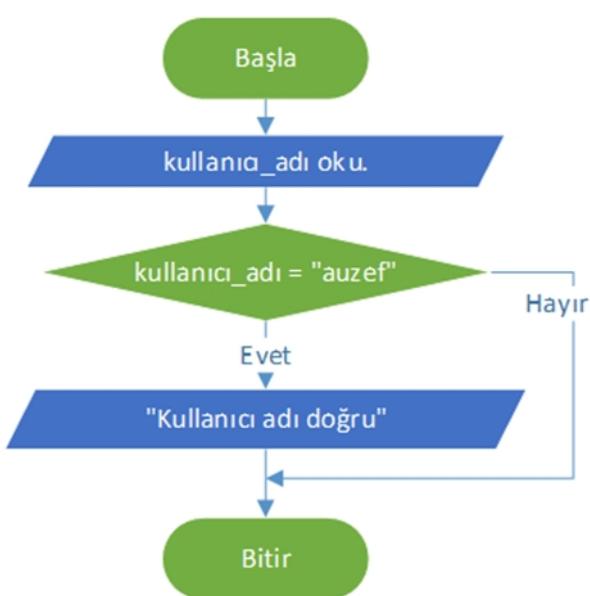
b)

BAŞLA
OKU ("Vize notunuzu giriniz!", vize)
OKU ("Final notunuzu giriniz!", final)
ortalama=vize*0.4+final*0.6
IF (ortalama \geq 50) THEN
YAZ ("Geçtin")
END IF
YAZ ("Ortalamanız:" & ortalama)
BİTİR



Kullanıcının isim girmesini ve girilen ismin doğruluğunu kontrol eden programda kullanıcının girdiği isim doğruysa ekrana mesaj döndürülür.

BAŞLA
OKU kullanıcı_adı
IF (kullanıcı_adı = "auzef") THEN
YAZ ("Kullanıcı adı doğru.")
END IF
BİTİR

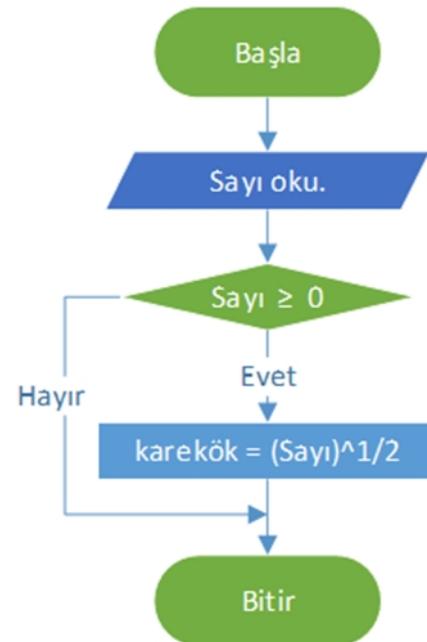


c) Kullanıcı tarafından girilen sayının karekökünü hesaplayan program aşağıdadır.

```

BAŞLA
OKU Sayı
IF (Sayı ≥ 0) THEN
    karekök = (Sayı) ^1/2
END IF
BİTİR

```

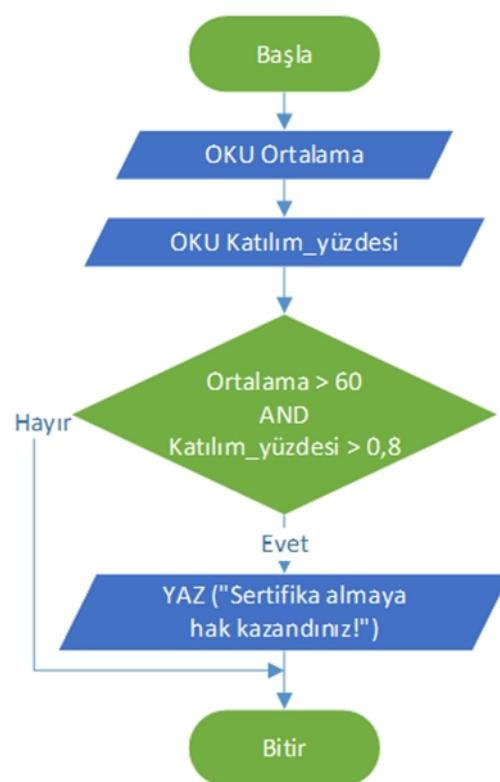


- d) Sınav ortalaması 60’ın ve katılım oranı yüzde 80’in (0,8) üzerinde olan öğrenciler sertifika almaya hak kazanmaktadır. Bunu hesaplayan program aşağıdadır.

```

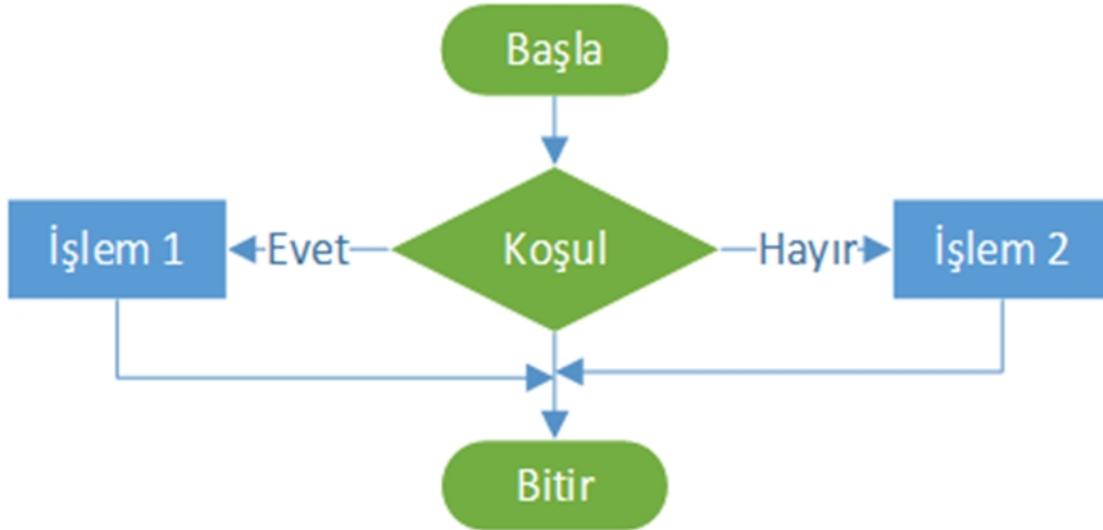
BAŞLA
OKU Ortalama
OKU Katılım_yüzdesi
IF (Ortalama > 60) AND
    (Katılım_yüzdesi > 0,8) THEN
    YAZ ("Sertifika almaya hak
        kazandınız!")
END IF
BİTİR

```



3.1.2. Çift Seçimli Yapı (*if...else* Yapısı)

if..else yapısı çift seçimli bir yapı sağlar. Yani bu yapıda bir koşulun sağlanması (doğru olması) durumunda bir eylem, sağlanmaması (yanlış olması) durumunda da başka bir eylem gerçekleştirilir (Deitel & Deitel, 2015). Kisaca koşula bağlı olarak gerçekleştirilen iki farklı işlem mevcuttur. Çift seçimli bir yapının temel gösterimi aşağıdaki gibi olabilir:



Çift seçimli yapılarda, tek seçimli yapıdan farklı olarak koşulun sağlanmaması durumunda farklı bir eylem uygulanır. *if...else* deyimine günlük hayattan örnek vermek gerekirse:

- Eğer hava 20 derecenin altındaysa kaloriferi, 20 derenin üzerindeyse klimayı çalıştır.
- Eğer finalden aldığı harf notu FF ise dersi tekrarlaman gereklidir değilse başka bir ders seçebilirsin.
- Eğer keyfin yerindeyse arkadaşlarıyla buluş değilse kendine zaman ayır ve evde film izle.

İlk örnekte, kaloriferin veya klimanın çalıştırılması hava sıcaklığına bağlı olarak hangisinin çalıştırılacağına karar verilir. Yani bu durumda yapılan kontrolün doğru veya yanlış olmasına bağlı olarak farklı işlemlerin yapılması sağlanır. Bir sonraki örnekte de öğrencinin dersten aldığı harf notu kontrol edilir. Eğer harf notu FF ise öğrenci dersi tekrarlamak zorundadır ancak harf notu bu koşulu sağlamıyorsa, dersten geçmiş demektir ve bu durumda yeni bir ders seçmesi mümkündür. Üçüncü örnekte bir kişinin keyfi yerindeyse arkadaşlarıyla buluşması değilse evde kendine zaman ayırması ve film izlemesi eylemleri gerçekleştirilir. Aşağıda kaba kodu ve akış diyagramı verilen örnekte ise bir alarmın çalışma durumuna göre iki farklı eylem gerçekleşmektedir. Alarm çalıysa kişinin uyanması ve o güne ait iş programını kontrol etmesi gereklidir, çalışmadiysa uyumaya devam eder.

BAŞLA
EĞER alarm çalıysa uyan ve iş programını kontrol et.
EĞER alarm çalmadıysa uyumaya devam et.
BİTİR



Bu yapıda kontrol koşulunun sağlandığı ve sağlanmadığı durumlarda iki farklı eylemin gerçekleştirildiği görülebilmektedir. Daha önce bahsedildiği gibi dilden dile gösterim şekilleri veya ifadeler/deyimler değişiklik gösterebilir ancak önemli olan temel mantığın nasıl kurulduğudur. Bu yapının VB dilindeki ifadesi de aşağıdaki gibidir. Burada *if* deyimindeki koşulun sağlanmadığı durumda yapılacaklar *else* deyimi kullanılarak gösterilir.

- **If [koşul] Then**

[işlem 1] → Koşul sağlandığında devam edilecek işlem

Else

[işlem 2] → Koşul sağlanmadığında devam edilecek işlem

End If

Aşağıda hem gerçek hayatın bir örneğin hem de matematiksel bir hesaplamanın VB dili kullanılarak nasıl kaba kod olarak gösterilebileceği görülebilir.

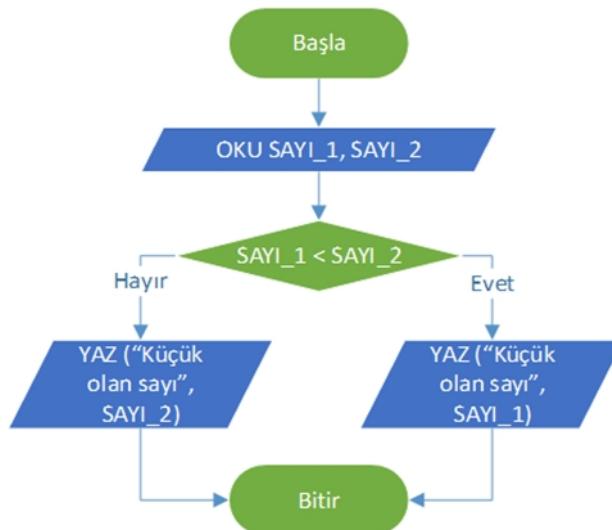
Duruma göre mesaj yazdırın bir algoritma örneği aşağıda verilmiştir.	x sayısının 10 ile karşılaştırıldığı bir örnek aşağıda verilmiştir.
If (Saat = 7:00) Then ("Günaydın!!") Else ("Daha sabah olmadı.") End If	If ($x \geq 10$) Then YAZ ("x 10'dan büyük veya eşittir.") Else YAZ ("x 10'dan küçük veya eşittir.") End If

Konu ile ilgili farklı örnekler aşağıda görülebilir:

a) Girilen iki sayıdan küçük olanı (eşit olmalarına dikkat etmeksiz) yazdırın programın kaba kodun aşağıda görülebilir. Program, kullanıcıdan iki sayı girmesini ister ve bunları birbiri ile karşılaştırarak küçük olan sayıyı ekrana yazdırır.

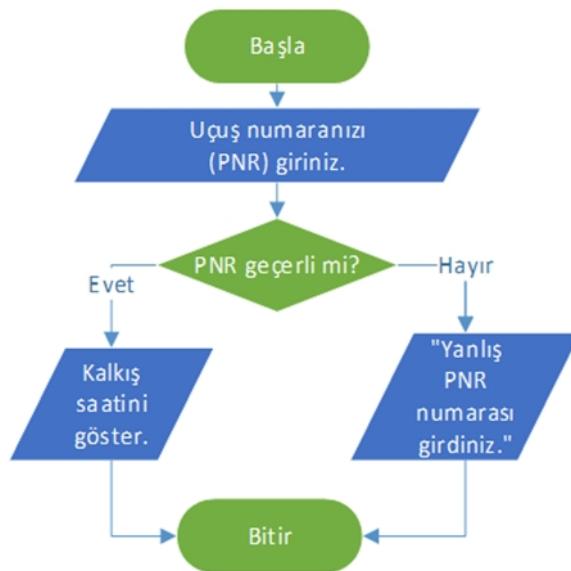
b)

BAŞLA
OKU (SAYI_1, SAYI_2)
IF (SAYI_1 < SAYI_2)
YAZ ("Küçük olan sayı", SAYI_1)
ELSE
YAZ ("Küçük olan sayı", SAYI_2)
END IF
BİTİR



Girilen uçuş numarasının (PNR) geçerli olup olmadığını kontrol eden ve sonuca göre ilgili mesajı ekrana yazdırın program aşağıda verilmiştir.

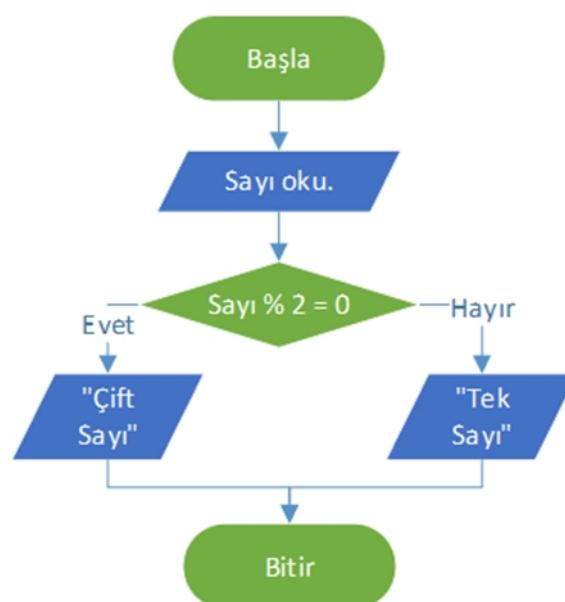
BAŞLA
OKU PNR
IF PNR geçerliyse THEN
YAZ (Kalkış saatı)
ELSE
YAZ ("Yanlış PNR numarası girdiniz.")
END IF
BİTİR



c) Kullanıcıdan bir sayı girmesi istenir. Girilen sayının çift mi tek mi olduğunu yazdırın algoritma örneği aşağıda verilmiştir.

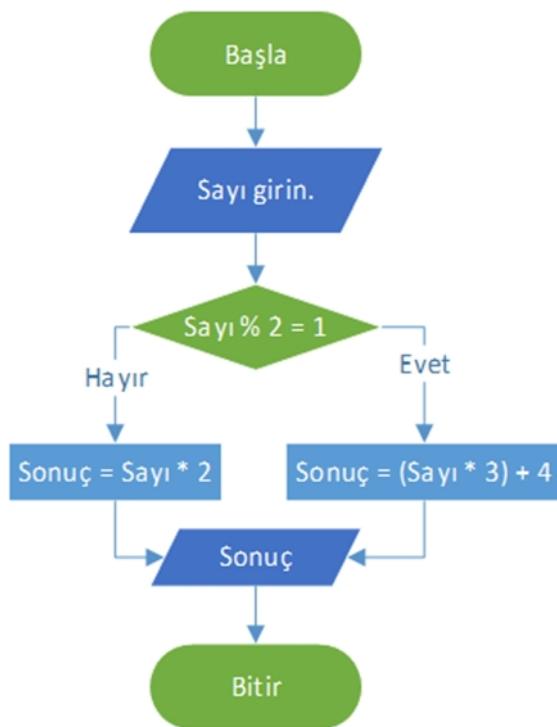
d)

BAŞLA
OKU Sayı
IF (Sayı % 2 = 0) ise THEN
YAZ ("Çift sayı")
ELSE
YAZ ("Tek sayı")
END IF
BİTİR



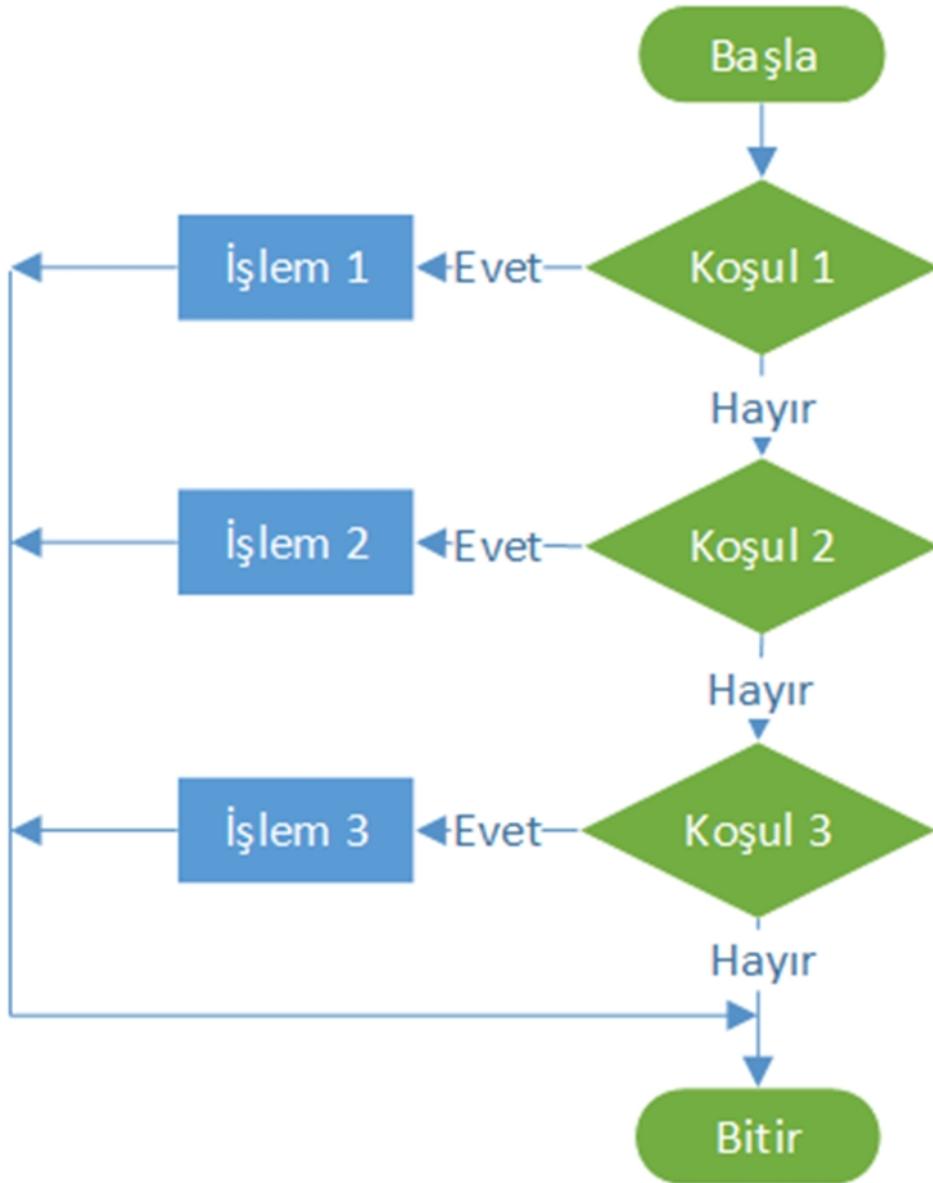
Girilen sayı tek ise 3 ile çarpılıp sayıya 4 eklenir, çift ise 2 ile çarpılır. Çıkan sonucu ekrana yazdırın program aşağıdadır.

BAŞLA
OKU Sayı
IF (Sayı % 2 = 1) ise THEN
Sonuç = (Sayı * 3) + 4
ELSE
Sonuç = Sayı * 2
END IF
YAZ (Sonuç)
BITİR



3.1.3. Çok Seçimli Yapı (*if...elseif* Yapısı)

Çok seçimli karar yapısı birden fazla koşulu bünyesinde barındıran bir yapı ile istenilen koşulun kontrolünün yapılmasını sağlar, böylelikle birden fazla durum kontrol edilmiş olur (Deitel & Deitel, 2015). Bu yapıda birden fazla *if...elseif* ifadesi kullanılır.



Cocukluğumda, tek seçenekli yapıdan farklı olarak koşulun sağlanmaması durumunda farklı bir eylem uygulanır. *if...elseif* deyimine günlük hayatdan örnek vermek gereklidir:

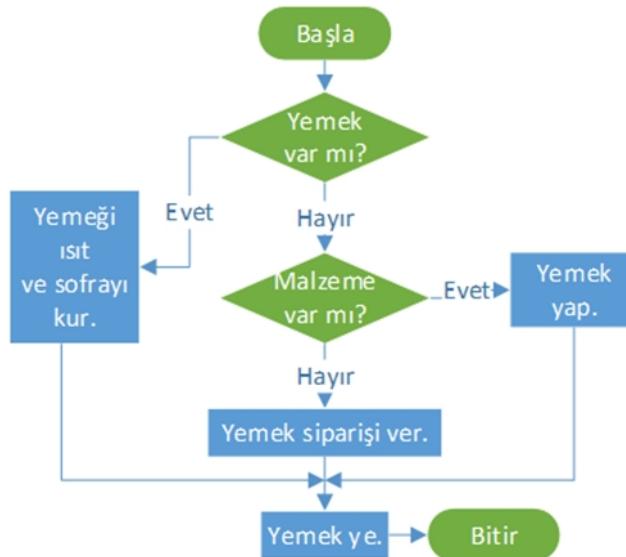
- Eğer mevsimlerden yaz ise mayo satın al, değilse havanın durumunu kontrol et. Hava soğuksa kazak satın al, değilse yağmur durumunu kontrol et. Hava yağmurluysa yağmurluk satın al.
- Eğer kaldığın ders varsa yaz okuluna yazılı. Kalmadıysan bütçene uygun bir uçak biletini satın almadığını kontrol et, bilet bulursan tatil planı yap.

İlk örnekte kişinin bir seyahate çıkacağını varsayıyalım. Buna göre seyahatten önce satın alacağı ürünler değişiklik gösterecektir. Mevsimlerden yaz ise kişinin mayo satın alması gereklidir. Mevsim yaz değilse hava gideceği yerdeki durumunu kontrol eder ve hava soğuksa kendisine kazak satın alması gereklidir. Hava soğuk değilse bir kontrol daha yapılır ve havanın yağmurlu olup olmadığı kontrol edilir. Yağmur varsa yağmurluk satın alması gereklidir, değilse bir şey yapması gerekmeyen program sonlanır. İkinci örnekte bir öğrencinin derslerinden geçip geçmediği kontrol edilir. Eğer derslerinden geçemediyse yaz okula yazılır. Geçtiyse arkadaşları ile tatil planı yapmak üzere bütçesine uygun bir uçak biletini satın almadığını kontrol eder. Bilet bulursa tatil planı yapar bulamazsa program sonlanır.

Yukarıda görülebileceği gibi çok seçenekli yapıda birden fazla koşullu yapıdan söz edilmektedir. İlk kontrolün ardından koşul sonunda gidilen iki seçenekten birinde (burada olumsuz olanda) bir başka *if* bloğuna gidilmektedir. Program, bu şekilde birbirini takip eden koşulların sonucusuna vardığında, önceki koşulların hiçbirini sağlamadı demektir. Örneğin, evde yemek olup olmadığına dair kontolle ilgili olarak önce yemeğin

varlığına bakılır. Yemek varsa ısıtılar ve sofra hazırlanır. Eğer yemek yoksa malzeme kontrolü yapılır ve varsa yemek yapılır yoksa sipariş verilir. Her durumda algoritma yemek yeme işlemi ile sonlanır. Bu örnekle ilgili kaba kod ve akış diyagramı aşağıda görülebilir.

BAŞLA
EĞER yemek var ise
Yemeği ısıt ve sofrayı kur.
EĞER yemek yoksa
EĞER malzeme varsa
Yemek yap.
EĞER malzeme yoksa
Yemek siparişi ver.
Yemek ye.
BİTİR



Bu örnekteki yapı VB dilinin yapısına uyarlandığında tek bir *else* deyimi yerine birden fazla *elseif* yer alacaktır, böylece birden fazla koşul kontrolü yapılabilir. Ayrıca programcılığın kaliteli olması açısından her zaman bir *else* deyiminin olması gereklidir. Böylelikle gözden kaçan bir durumda yanlış sonuçlar çıkması engellenmiş olur. VB'de çok seçimli yapının kurulması için aşağıdaki temel yapı kullanılır:

- **If [koşul 1] Then**

[işlem 1] → Koşul 1 sağlanlığında devam edilecek işlem

ElseIf [koşul 2] Then

[işlem 2] → Koşul 2 sağlanlığında devam edilecek işlem

ElseIf [koşul 3] Then

[işlem 3] → Koşul 3 sağlanlığında devam edilecek işlem

....

Else → Hiçbir koşul sağlanmadığında yapılacak işlem

End If

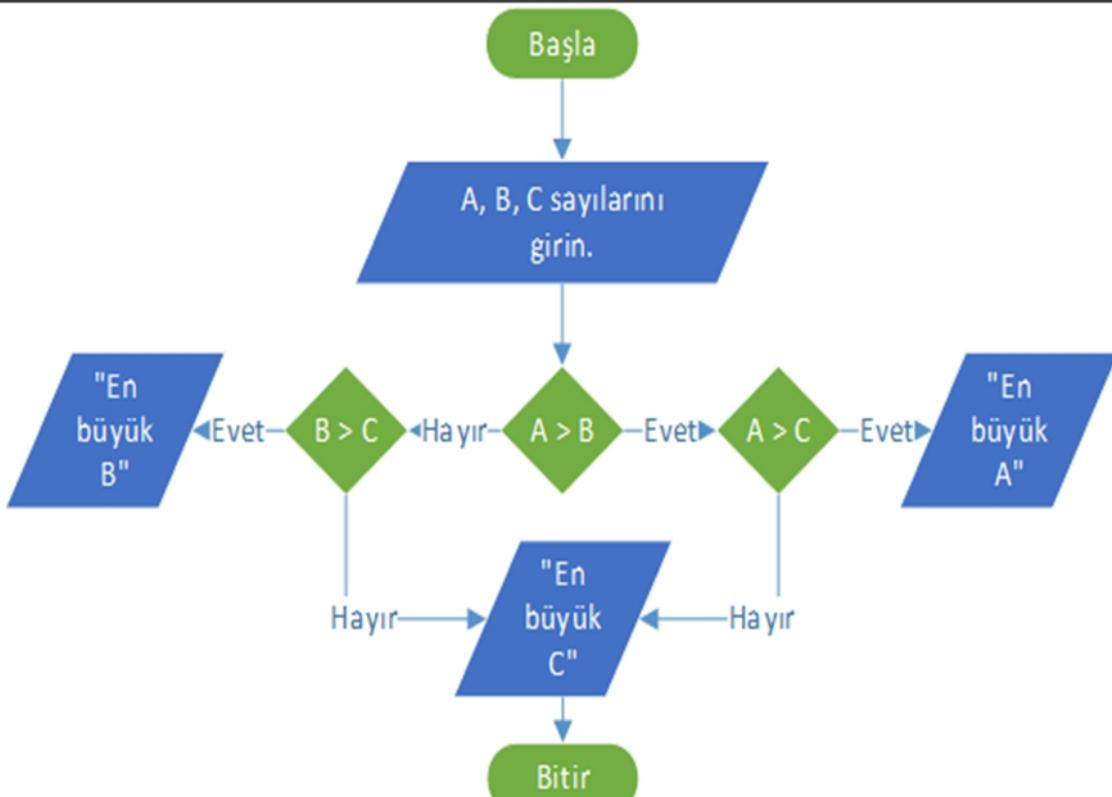
Aşağıda hem gerçek hayattan bir örneğin hem de matematiksel bir hesaplamanın VB dili kullanılarak nasıl gösterilebileceği görülebilir.

Duruma göre farklı eylemler gerçekleştirilmesini sağlayan program örneği aşağıda verilmiştir.	x sayısının 10 ile karşılaştırıldığı bir örnek aşağıda verilmiştir.
<pre>If Su kaynadiysa Then Çay demle ElseIf Kahvaltlıklar tamamsa Then Masayı hazırla Else Sipariş ver End If</pre>	<pre>If (x = 10) Then YAZ ("x 10'a eşittir.") ElseIf (x > 10) Then YAZ ("x 10'dan büyüktür.") Else YAZ ("x 10'dan küçüktür.") End If</pre>

Çok seçimli karar yapısına farklı örnekler verilebilir:

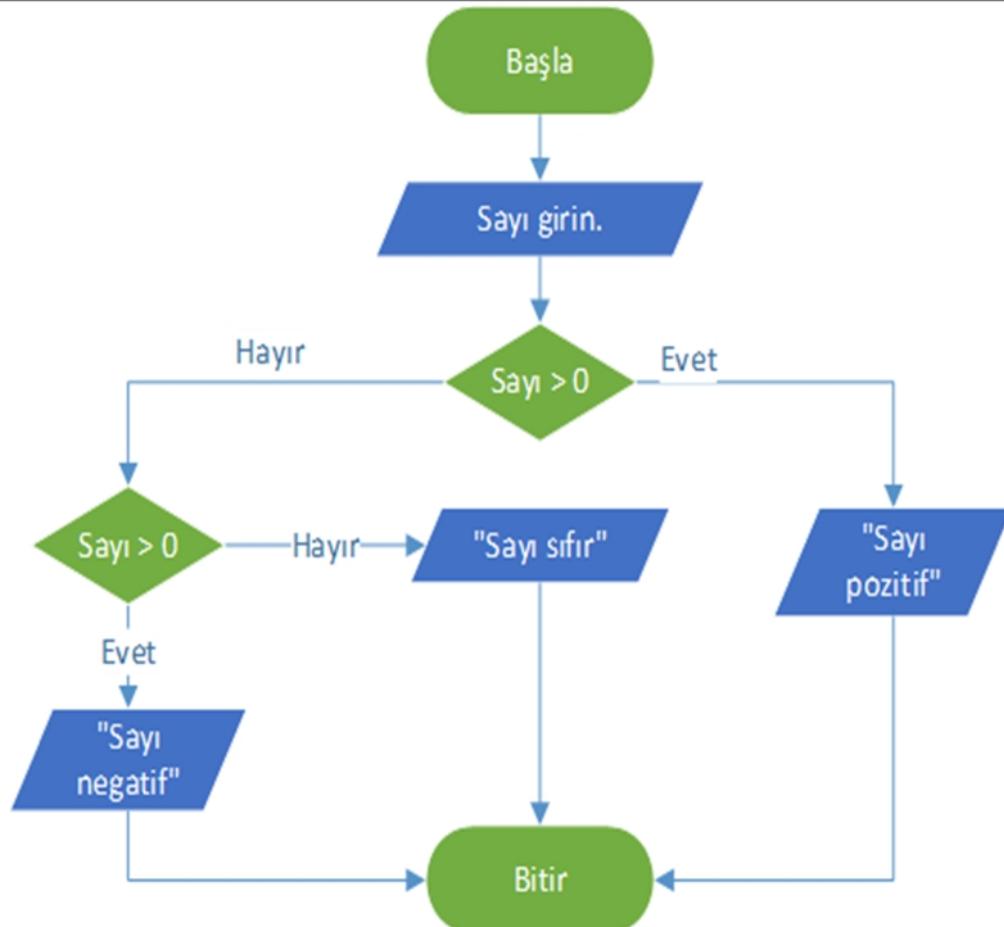
- e) Girilen 3 sayıdan en büyük olanını yazdırın programın kaba kodu aşağıda verilmiştir (Bu programda eşitliklere dikkat edilmemektedir).

BAŞLA
Üç sayı (A, B, C) girin.
IF (A > B) THEN
IF (A > C) THEN
YAZ ("En büyük A")
ELSE
YAZ ("En büyük C")
END IF
ELSEIF (B > C) THEN
YAZ ("En büyük B")
ELSE
YAZ ("En büyük C")
END IF
BİTİR



- f) Girilen sayının pozitif mi, negatif mi yoksa sıfır mı olduğunu yazdırın programın algoritması aşağıdaki gibidir.

BAŞLA
OKU Sayı
IF (Sayı > 0) THEN
YAZ ("Sayı pozitif")
ELSEIF (Sayı < 0) THEN
YAZ ("Sayı negatif")
ELSE
YAZ ("Sayı sıfır")
END IF
BİTİR



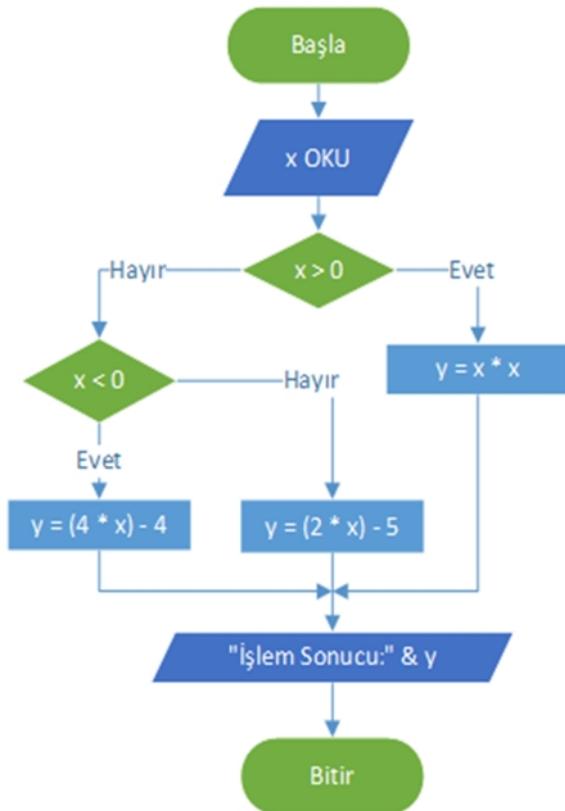
g) x 'in aldığı farklı değerlere göre $f(x)$ fonksiyonunun değerini hesaplayan algoritma aşağıdaki verilmiştir:

$$f(x) = x^2 \ (x > 0)$$

$$f(x) = 2*x - 5 \ (x = 0)$$

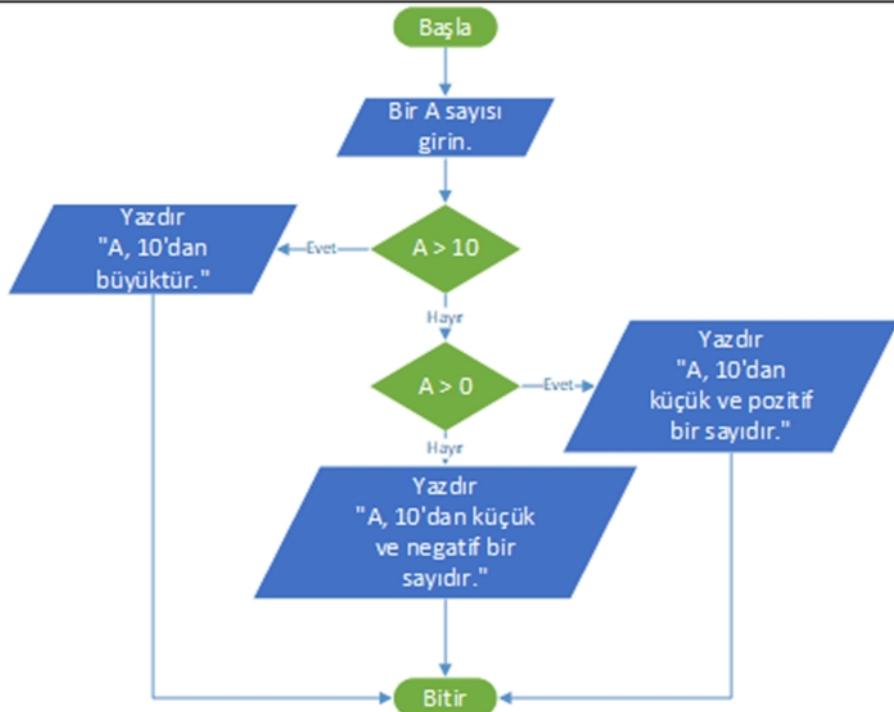
$$f(x) = 4*x - 4 \ (x < 0)$$

BAŞLA	Çıktı:
OKU x	
IF (x > 0) THEN	
y = x * x	
ELSEIF (x < 0) THEN	
y = (4 * x) - 4	
ELSE	
y = (2 * x) - 5	
END IF	
YAZ ("İşlem sonucu:" & y)	
BİTİR	



- h) Kullanıcı tarafından girilen bir A sayısının 10'dan büyük olup olmadığını ve negatif veya pozitif olma durumunu kontrol eden program aşağıda verilmiştir.

BAŞLA
Bir A sayısı girin.
IF (A > 10) THEN
YAZ ("A, 10'dan büyüktür.")
ELSEIF (A > 0) THEN
YAZ ("A, 10'dan küçük ve pozitif bir sayıdır.")
ELSE
YAZ ("A, 10'dan küçük ve negatif bir sayıdır.")
END IF
BİTİR



Cok seçimli yapıdan bahsederken *select...case* deyimine degenmemek olmaz. Bazen bir algoritma, değişken veya ifadenin alabileceği her bir farklı değer için ayrı ayrı test edilen bir dizi karar içerir; algoritma daha sonra bu değerlere göre farklı eylemler gerçekleştirir (Deitel et al., 2014). Burada fayda sağlayabilecek çok seçimli yapıya VB'de *select...case* denir. Farklı programlama dillerinde bu yapı *switch...case* şeklinde de ifade edilmektedir. Bu yapının akış diyagramı ile gösterimi çok seçimli yapı ile aynıdır. Bu yapının VB'deki kullanımı aşağıdaki gibidir:

- **Select Case** [test edilen değişken]

Case 1

[işlem 1] → Koşul döndürülecek sonuç

Case 2

[işlem 2] → Koşul döndürülecek sonuç

Case 3

[işlem 3] → Koşul döndürülecek sonuç

...

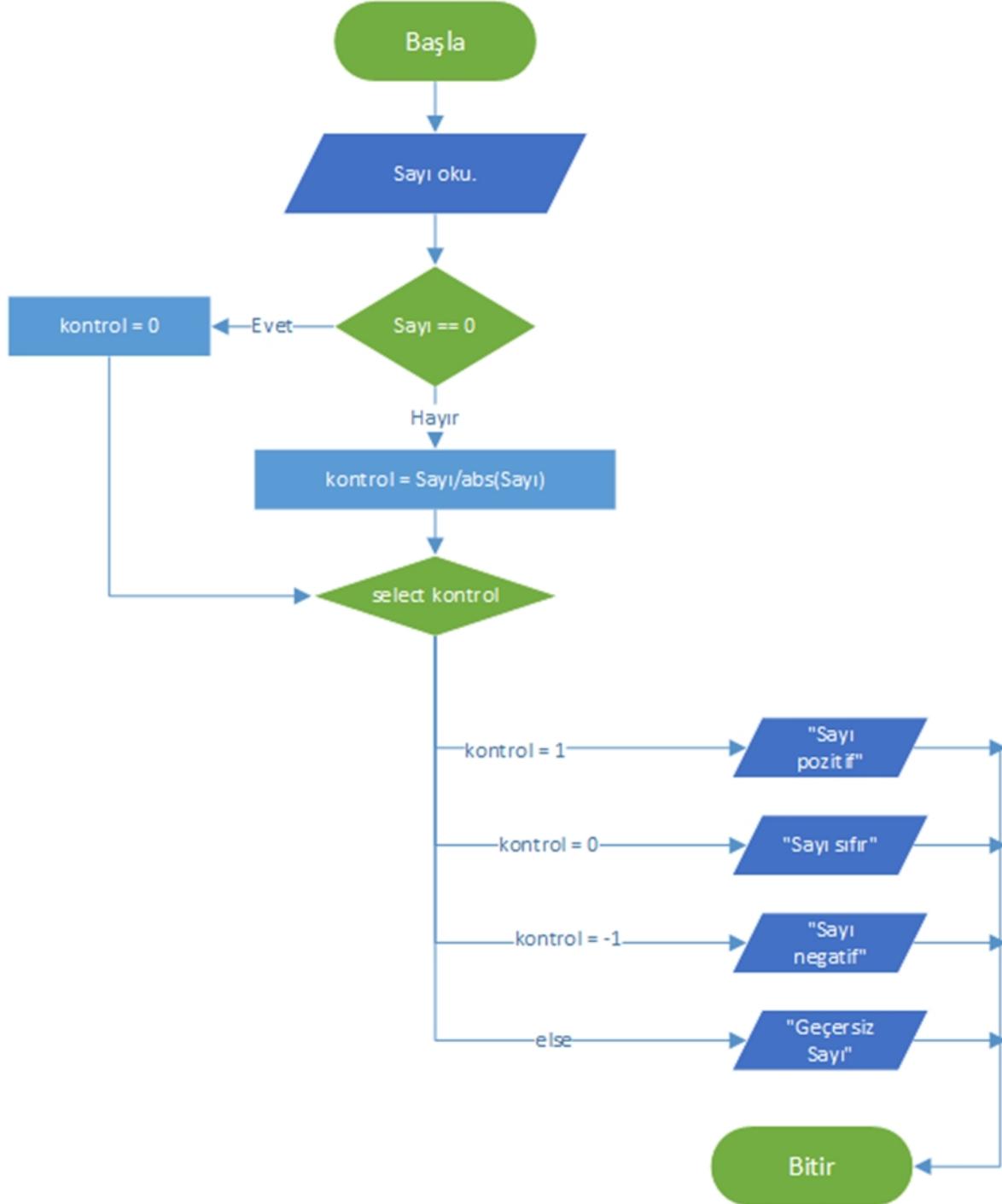
End Select

Ders notuna göre farklı sonuçları gösteren bir algoritma örneği aşağıda verilmiştir.	Risk durumuna göre bir çıktı veren algoritma aşağıda verilmiştir.
<pre>Select Case (Ders Notu) Case 40 YAZ ("FF") Case 60 YAZ ("CC") Case 80 YAZ ("BA") Case 100 YAZ ("AA") Case Else YAZ ("Not girilmedi.") End Select</pre>	<pre>(risk = 50) Select Case risk Case 0 YAZ ("Risk düşüktür") Case 50 YAZ ("Risk ortadır") Case 100 YAZ ("Risk yüksektir") Case Else YAZ ("Bilinmiyor") End Select</pre>

Çok seçimi karar yapısı için *select...case* kullanılarak oluşturulan örnekler aşağıda verilmiştir:

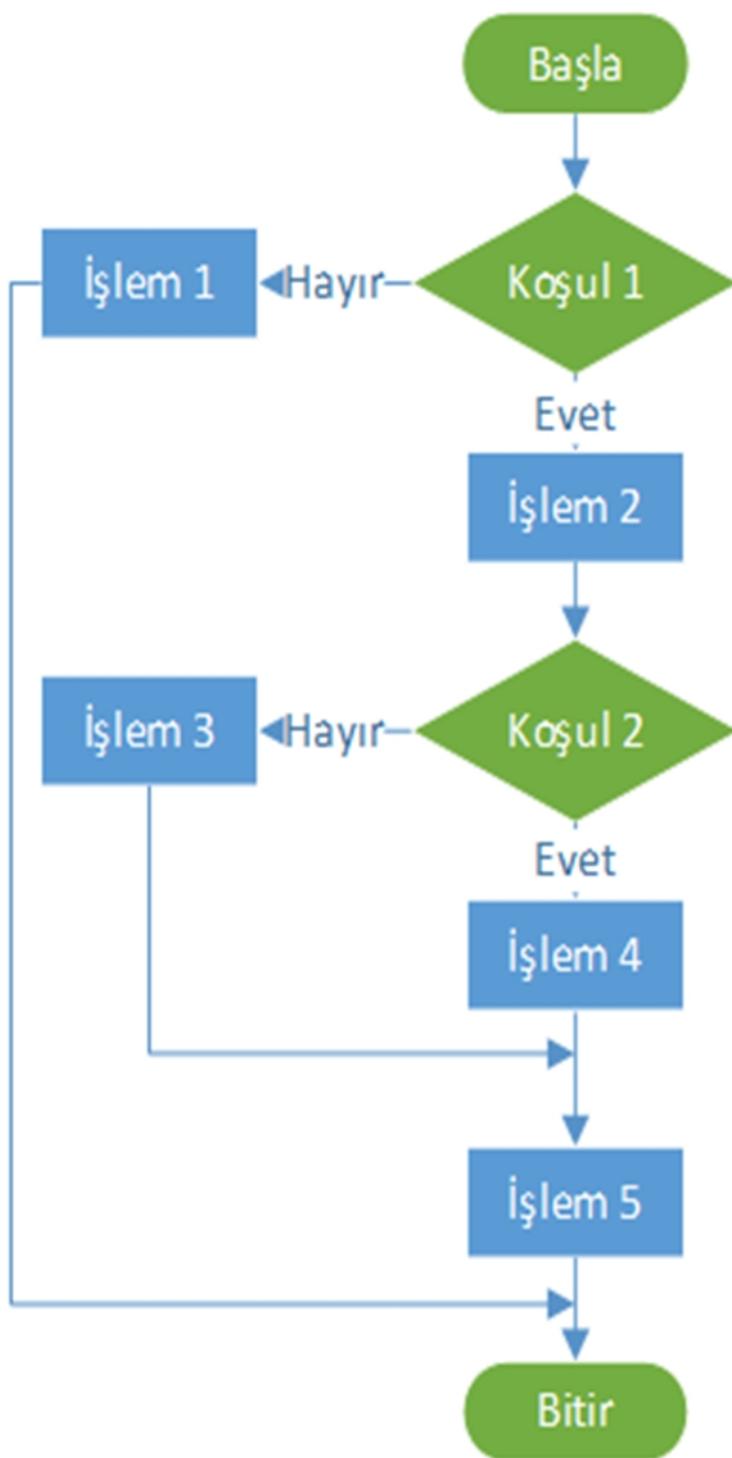
- Verilen sayının pozitif mi, negatif mi yoksa sıfıra mı eşit olduğunu yazdırın program aşağıdadır. Kontrol aşamasında kullanılan abs fonksiyonu, mutlak değerin elde edilmesi için kullanılmıştır (örneğin $\text{abs}(-5)=5$ ve $\text{abs}(5)=5$).

BAŞLA
OKU sayı
IF (sayı = 0) THEN
kontrol = 0
ELSE
kontrol = sayı/abs(sayı)
END IF
SELECT CASE kontrol
CASE 1
YAZ ("Sayı pozitif")
CASE 0
YAZ ("Sayı sıfır")
CASE -1
YAZ ("Sayı negatif")
CASE ELSE
YAZ ("Geçersiz Sayı")
END SELECT
YAZ (y)
BİTİR



3.1.4. İç İçe Seçimli Yapı (nested if...else... if Yapısı)

Bu seçim yapısında birden fazla *if* yapısı birlikte ve iç içe bir şekilde kullanılır. Yani bir koşulun kontrolü sağlanır ve bu koşulun doğru olması yani sağlanması durumunda bir sonraki adımda başka bir koşulun kontrolü yapılır. İç içe seçimli bir yapının temel gösterimi aşağıdaki gibi olabilir. Burada mevcut bir *if* deyiminin içinde bir başka *if* deyimi bulunmaktadır. Kullanılan her *if* deyimi kadar *end if* olması gereği unutulmamalıdır.



Daha önceki bölümlerdeki gibi, günlük hayattan örnekler iç içe koşullu yapılara da uyarlanabilir. Bunlara örnek vermek gereklidir:

- Yarın sınavın varsa ders çalış ve yemek siparişi ver. Eğer sınav yoksa ailenle bir saat vakit geçir ardından spor salonunun programını kontrol et. Uygun bir ders varsa spor salonuna git ve spor yap, yoksa evde kal ve yemek siparişi ver.
- Eğer doktora randevunun olduğu gün bugünse, randevuna yetişmek için evden çıkmalısın. Randevun bugün değilse arkadaşınla buluş. Randevun bugünse evden çıktıktan sonra otobüs saatlerini kontrol et. Otobüsü kaçırmışsan taksi çağır.

İlk örnekte kişinin yarın sınavı varsa, ders çalışması gereklidir ve bu kişi yemek siparişi verir. Eğer ertesi gün sınav yoksa önce ailesiyle vakit geçirir ve sonrasında spor salonuna gitmek üzere spor salonunun ders programını kontrol eder. Kendi uygun bir ders bulursa spor salonuna gider ve spor yapar, uygun bir ders yoksa evde kalır ve yemek siparişi verir. Yani ders çalışmak için evde kaldığı ve spor sonrasında eve döndüğü iki durumda da bu eylemler yemek siparişi ile sonlanır. Spor yapma eylemi ve bunun kontrolünün

sağlandığı ikinci karar yapısı olan sınavın olup olmaması durumuna bağlı olarak gerçekleşir. Sınav varsa spor eylemi hiçbir koşulda gerçekleşmez. İkinci örnekte ise ilk koşul olan doktor randevusunun bugün olma durumu gerçekleşmeden diğer eyleme geçilmez. Randevu bugün değilse kişi arkadaşı ile buluşur. Bugünse evden çıkar ve otobüs saatlerini kontrol eder. Otobüsü kaçırılmışa taksi çağrıır. Otobüs saatlerini kontrol etme ve taksi çağrıma işlemleri randevunun olup olmama durumuna bağlıdır.

Aşağıda detaylandırılmış olan örnekte ise öncelikli olarak o günün iş günü olup olmadığı kontrol edilir. Eğer iş günü ise kişinin hazırlanması ve sonrasında trafiği kontrol etmesi gereklidir. Köprüde trafik varsa vapur yoksa araba ile işe gidecektir. Her iki durumun ardından kahve alma eylemi gerçekleşecektir. Eğer iş günü değilse alarmı ertelemesi ve uyumaya devam etmesi ile algoritma sonlanacaktır.

BAŞLA
EĞER iş günü ise
İşe gitmek için hazırlan.
Trafığı kontrol et.
EĞER köprüde trafik varsa
Vapura bin.
EĞER köprüde trafik yoksa
Arabayla git.
Kahve al.
EĞER iş günü değilse alarmı ertele
VE uyumaya devam et.
BİTİR



Bu önekteki yapı da VB dilinin yapısına uyarlanabilir. Bu yapıda temel olarak *if* ve *else* deyimleri kullanılmakla beraber bir *if* deyiminin altında farklı koşul yapıları olduğu görülür. VB'de çok seçimi yapıının kurulması için aşağıdaki temel yapı kullanılır:

```

• If [koşul1] Then
  If [koşul 2] Then
    [işlem 1] → Koşul 1 ve 2 sağlanlığında devam edilecek işlem
  Else
    [işlem 2] → Koşul 1 sağlanıp 2 sağlanmadığında devam edilecek işlem
  End If
  Else
    If [koşul 3] Then
      [işlem 3] → Koşul 1 ve 3 sağlanlığında devam edilecek işlem
    Else
      [işlem 4] → Koşul 1 sağlanıp 3 sağlanmadığında devam edilecek işlem
    End If
  End If

```

Aşağıda hem gerçek hayattan bir örneğin hem de matematiksel bir hesaplamanın VB dili kullanılarak nasıl kaba kod olarak gösterilebileceği görülebilir.

18 yaşında olup olmama durumuna göre ehliye alma kontrolünü yapan program aşağıdadır.	x ve y sayılarını sırasıyla 10, 20 ve birbirleriyle karşılaştırılan program aşağıda verilmiştir.
<pre> If 18 yaşına geldiysen Then Sürücü kursuna yazıl. If Ehliyet sınavına girdiysen Then Sürüş sınavı için ders al. Else Sınav başvurusu yap. End If Else 18 yaşına gelmeyi bekle. End If </pre>	<pre> If (x > y) Then If (x > 10) Then YAZ ("x 10'dan büyük ve y'den büyuktur.") Else YAZ ("x 10'dan küçük veya eşit ve y'den büyük.") End If Else If (y ≤ 20) Then YAZ ("y 20'den küçük veya eşit ve x'ten büyük.") Else YAZ ("y 20'den büyük x'ten büyuktur.") End If End If YAZ ("Başka bir sayı giriniz.") </pre>

Bölüm Özeti

Bu bölümde karar yapılarının aktarılması amaçlanmıştır. Karar yapıları, seçimi yapılar veya koşul yapıları olarak da adlandırılabilir. Bu doğrultuda her bölümde farklı karar yapıları aktarılmış ve farklı örnekler verilerek konunun pekiştirilmesi hedeflenmiştir.

İlk bölümde tek seçimi yapı ele alınmıştır. Bu yapıda bir koşulun kontrolü sağlanır ve buna göre bir işlem yürütülür ancak koşulun sağlanmaması durumunda ne yapılacağı belirtilmez. İkinci bölümde ele alınan çift seçimi yapıda ise bir koşulun kontrolü sağlanır, koşulun sağlandığı ve sağlanmadığı durumlarda farklı eylemler gerçekleştirilir.

Üçüncü bölümde ele alınan çok seçimi karar yapısında kararın gerçekleştiği veya gerçekleşmediği bir durumda farklı bir karar yapısına atlanır. Örneğin bir koşul gerçekleşiyorsa bir işlem yapılır, gerçekleşmiyorsa başka bir koşulun kontrolü yapılır ve buna göre programın devamı gelir. İyi programcılık açısından açısından her zaman bir *else* deyiminin olması önerilmektedir. Böylelikle programda göz önünde

bulundurulmayan bir hatanın ortaya çıkması engellenmiş olur. Çok seçimli yapının bir alternatifi olan bir diğer yapı da bu bölümde ele alınmıştır.

Son bölümde ise iç içe seçimli yapılardan bahsedilmiştir. Bu yapıda ise koşulun sağlanması ve sağlanmaması durumlarında birer eylem gerçekleştirilir. Bunlardan birine bağlı başka bir karar yapısı önmüze çıkar ve program buna göre yürütülür. Burada dikkat edilmesi gereken nokta kullanılan her bir *if* deyimi için aynı sayıda *end if* deyiminin programda yer alması gerektidir.

Kaynakça

Çobanoğlu, B. (2014). Algoritma Geliştirme ve Veri Yapıları (5th ed.). Pusula Yayıncılık.

Deitel, P., Deitel, A., & Deitel, H. (2014). Visual Basic 2012 How to Program (6th ed.). Pearson.

Deitel, P., & Deitel, H. (2015). Java How to Program - Late Objects Version (10th ed.). Pearson.

Walia, R. K. (2022). Algorithm & Flowchart Manual for Students.
<https://courses.minia.edu.eg/Attach/16036flowchart-algorithm-manual.pdf>

Ünite Soruları

Soru-1 :

“Eğer yemek yediysen ilaçını al.” cümlesi ile ilgili aşağıdakilerden hangisi yanlıştır?

(Çoktan Seçmeli)

- (•) - Tek bir koşulun kontrolü yapılır.
- (•) - Koşulun doğru olması durumunda sadece bir işlem yapılır.
- (•) - Koşulun yanlış olduğu durumda ne yapılacağı bellidir.
- (•) - Koşulun yanlış olduğu durumda hangi işlemin yapılacağı belli değildir.
- (•) - İlaç alma eylemi bir koşula bağlıdır.

Cevap-1 :

Koşulun yanlış olduğu durumda ne yapılacağı bellidir.

Soru-2 :

Aşağıdaki boşluğa gelmesi gereken doğru kelime hangisidir?

“Kodun okunurluğunun artması amacıyla koşullu yapıların yazımında kodların _____ yazılması önerilir.”

(Çoktan Seçmeli)

- (•) - birleşik
- (•) - girintili

(•) - ardışık olarak

(•) - iç içe

(•) - sırayla

Cevap-2 :

girintili

Soru-3 :

Koşullu yapıların kullanım amacı ile ilgili olarak aşağıdaki cümlelerden hangisi yanlıştır?

(Çoktan Seçmeli)

(•) - Bu yapılar bazı adımların atlanmasını sağlar.

(•) - Bu yapılar bir seçim yapılmasını sağlar.

(•) - Bu yapılar ardışık işlem yapılmasını zorunlu kılar.

(•) - Bu yapılar bir durumun kontrol edilmesini sağlar.

(•) - Bu yapılar sayesinde istenen bir koşul sağlanır.

Cevap-3 :

Bu yapılar ardışık işlem yapılmasını zorunlu kılar.

Soru-4 :

Bir öğrencinin dersten geçme durumunu ekran'a yazdırın kod parçasında, dersten kalındıysa ekran'a mesaj döndürülür, kalınmadıysa program durur. Bu programda kaç durumun kontrolü yapılır?

(Çoktan Seçmeli)

(•) - 1

(•) - 2

(•) - 3

(•) - 4

(•) - 5

Cevap-4 :

1

Soru-5 :

Öğrencinin notu 50 ise CC, notu 70 ise BB, 90 ise AA değerlerini döndüren bir programla ilgili olarak aşağıdakilerden hangisi doğrudur?

(Çoktan Seçmeli)

- (•) - Programın çıktısı yoktur.
- (•) - Birden fazla koşul kontrolü yapılır.
- (•) - Farklı durumların hepsi aynı sonucu döndürür.
- (•) - Öğrencinin notu boolean tipinde bir değerdir.
- (•) - Öğrenci notu yanlış girilmiştir.

Cevap-5 :

Birden fazla koşul kontrolü yapılır.

Soru-6 :

Aşağıdaki önermede boş bırakılan yer koşullu bir yapıyı ifade etmek için hangi kelime gelmelidir ?

“_____ paketin gelmediyse firmayı arayıp kargonun nerede olduğunu sor.”

(Çoktan Seçmeli)

- (•) - Elseif
- (•) - Çok seçenekli yapı
- (•) - Beklediğin
- (•) - İstediğin
- (•) - Eğer

Cevap-6 :

Eğer

Soru-7 :

Bir programda birden fazla if deyi̇mi bir arada kullanıldığında hangi deyim if deyi̇mi ile aynı sayıda kullanılmalıdır?

(Çoktan Seçmeli)

- (•) - End if
- (•) - elseif
- (•) - else
- (•) - select
- (•) - Hiçbiri

Cevap-7 :

End if

Soru-8 :

İç içe seçimli yapı ile ilgili aşağıdakilerden hangisi kontrol edilir?

(Çoktan Seçmeli)

- (•) - En dıştaki koşul içeridekine bağlı olarak çalışır.
- (•) - Koşulların mutlaka sağlanması gereklidir.
- (•) - Koşulların ikisi de çalışmaz.
- (•) - Koşulların ikisi de aynı anda çalışır.
- (•) - En dıştaki koşula bağlı olarak içerideki koşul çalışır veya çalışmaz.

Cevap-8 :

En dıştaki koşula bağlı olarak içerideki koşul çalışır veya çalışmaz.

Soru-9 :

Girilen sayının 100'e eşit olduğu durumda “Sayı doğru” eşit olmadığı durumda “Sayı yanlış” mesajını döndüren bir programın sözde kodu hangisidir?

(Çoktan Seçmeli)

- (•) - EĞER sayı = 100 ise

YAZ (“Sayı doğru”)

- (•) - EĞER sayı <> 100 ise

YAZ (“Sayı yanlış”)

- (•) - a) EĞER sayı = 100 ise

YAZ (“Sayı doğru”)

EĞER sayı = 100 değilse

YAZ (“Sayı yanlış”)

- (•) - EĞER sayı <> 100 ise

YAZ (“Sayı doğru”)

EĞER sayı = 100 değilse

YAZ (“Sayı doğru”)

- (•) - EĞER sayı - 100 ise

YAZ (“Sayı doğrudur”)

EĞER sayı = 100 değilse

YAZ (“Sayı yanlıştır”)

Cevap-9 :

a) EĞER $\text{say1} = 100$ ise

YAZ (“Sayı doğru”)

EĞER $\text{say1} = 100$ değilse

YAZ (“Sayı yanlış”)

4. DÖNGÜLER

4.1. Döngüler

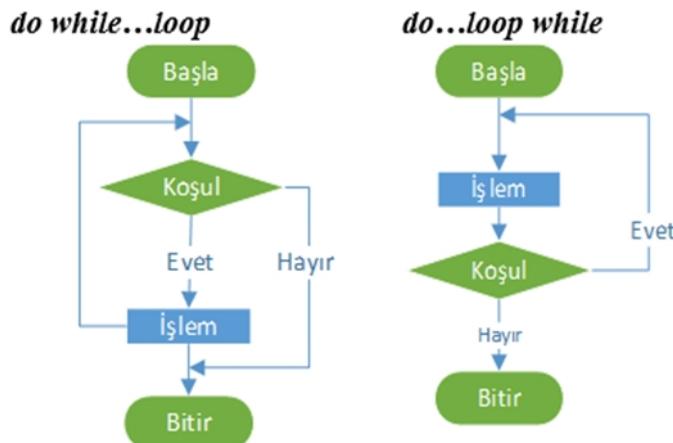
Önceki bölümde bahsedildiği gibi bir programda her zaman tüm işlemler ardışık şekilde yürütülmelidir. Bazan birtakım işlemlerin belli sayıda tekrarlanması gerekebilir. Bu noktada döngü kavramından bahsedilir. Tanım olarak bakıldığında, bir programdaki belli işlemlerin istenilen sayıda gerçekleştirilmesine döngü (*loop*) denir (Vatansever, 2011). Başka bir deyişle döngüler tekrarlı yapılardır (Çobanoğlu, 2014). İşlemlerin tekrar sayısının belirli veya belirsiz olduğuna bağlı olarak bir ayrima gidilir. Tekrarlı yapılarda, tekrar sayısının bir koşula bağlı olduğu yapılar (*do while...loop*) ve tekrar sayısının baştan belli olduğu yapılar (*for...next, for...each*) mevcuttur (Çobanoğlu, 2014). Algoritmanın özelliklerinden bahsettiğimiz kitabin ilk bölümünde (1.2 Algoritma Kavramı) dejindiğimiz gibi, belli adımlar gerçekleştirildikten sonra algoritmanın çalışmasının durması gereklidir. Yani algoritma ve bunun ardından oluşturacağımız programın bir sonu olmalıdır, program sonsuz döngüye girmemelidir. Koşullar tasarlanırken programın sonsuz döngüye girmemesine dikkat edilmelidir.

4.1.1. Tekrar Sayısı Koşula Bağlı Döngüler

Tekrar sayısı koşula bağlı olan döngüler için *do loop...while* komutu kullanılır (Çobanoğlu, 2014). Yani belli bir koşula bağlı olarak bir kod parçası tekrarlanır. Başka bir deyişle, belirtilen koşul doğru olduğu müddetçe, programın bir işlemi veya eylemi tekrarlaması sağlanır (Deitel & Deitel, 2015). Dolayısıyla programın kaç kere tekrarlanacağı en baştan belli değildir, koşula bağlı olarak tekrar sayısı değişir, koşulun sağlanmadığı ana kadar belli bir kod parçası tekrarlanır. Koşul sağlandığı müddetçe istenilen işlemler tekrarlanırken, koşul sağlanmadığında döngü sonlanmış olur.

Tekrar sayısı koşula bağlı olan döngüler, kontrolün başta veya sonda kontrol edilmesiyle istenilen duruma uyarlanabilir. Döngünün bulunduğu kod bloğu çalıştırılmadan önce koşul kontrol edilip, koşul sağlanmadığında döngü çalışmayı bırakıyorsa *do while...loop*; döngünün bulunduğu kod bloğu yürütüldükten sonra koşulun kontrolü sağlanıyorsa *do...loop while* yapısı kullanılır. Bu iki yapı arasındaki temel farklılık kontrolün sonda olması durumunda döngünün en az bir kere çalışıyor olmasıdır (Deitel & Deitel, 2015). Diğer bölgelerde de bahsettiğimiz gibi değişik programlama dillerinde temel mantık aynıdır fakat diller arasında yazım farklılıklarını mevcuttur. Kitaptaki örnekler VB'deki temel gösterim baz alınarak aktarılmıştır.

Temel olarak işlem adımları ve koşul kontrolü aşağıdaki akış diyagramlarındaki gibi ifade edilebilir (Çobanoğlu, 2014):



VB dilinde ise aşağıdaki şekilde bu iki döngü tipinin gösterilmesi mümkündür.

Kullanım şekli: Do {While} koşul [İşlem] → Tekrar edilecek işlem Loop	Kullanım şekli: Do [İşlem] → Tekrar edilecek işlem Loop {While} koşul
20'ye kadar olan her bir sayının 5 katını hesaplayan program örneği iki yapı ile de aşağıda verilmiştir.	
Do While (sayı ≤ 20) sayı = sayı * 5 Loop	Do sayı = sayı * 5 Loop While (sayı ≤ 20)

do while...loop deyimi farklı örnekler ile günlük hayatı uyarlanabilir. Aynı zamanda matematiksel işlemlerin veya hesaplamaların da bu şekilde ifade edilmesi tabii ki mümkündür. Aşağıda bazı örnekler sunulmuştur, devamında ise kaba kod ve akış diyagramları birlikte verilmiştir:

- Bir alışveriş listesinde, alınacak ürünler olduğu müddetçe alışveriş eylemi devam eder. Listedeki alınacak ürünler bittiğinde alışveriş eylemi sonlanır.
- Sofradaki insan sayısı kadar yemek servisi devam edecektir. Yani herkesin yemeği gelene kadar bu işlem tekrarlanacaktır.
- Bir öğrenci dersten kaldıysa, o dersi geçene kadar dersi tekrarlaması gereklidir.

BAŞLA
DO Kitabı oku
LOOP WHILE daha okunacak sayfa olduğu müddetçe
BİTİR



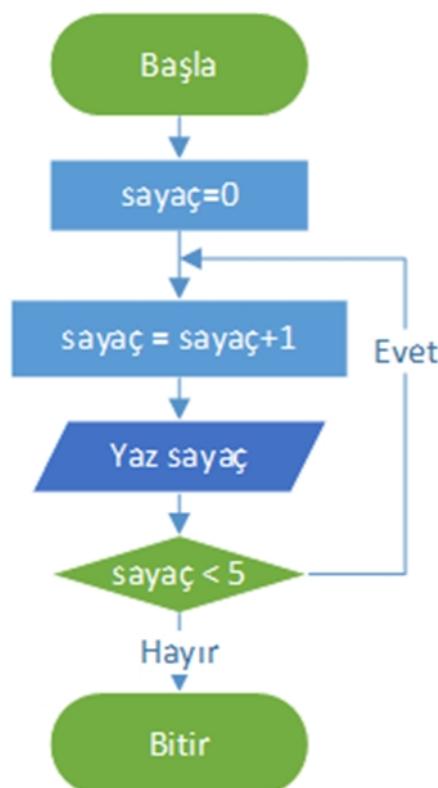
do while...loop döngü yapısı kullanılırken **sayaç (counter)** kullanımından da bahsetmek gereklidir. Sayaç ile ilgili temel açıklamalar (2.4. Sıralı İşlemler ve Basit Algoritmalar) kitabın ikinci bölümünde verilmiştir. Döngünün tekrar sayısının kontrol edilmesi için sayaç değişkeninden yararlanılır. Bir değişkenin her seferde arttırılması veya eksiltilmesi yoluyla tekrar sayısının belirlenmesi sağlanır (Çobanoğlu, 2014). Bu teknik, bir

dizi ifadenin kaç kez yürütüleceğini kontrol etmek için kullanır (Deitel & Deitel, 2015). Örneğin 1'den 5'e kadar olan sayılar sırayla ekrana yazdırılmak isteniyorsa bu tekrarlı işlem sayaç kontrollü bir döngü kullanılarak yapılabilir. Sayaç, burada adımların tekrarlanması görevini görür. Sayacın ifade edilmesi için genellikle i değişkeni kullanılır. Eğer iç içe döngülerden oluşan bir yapı varsa sırasıyla i , j , k değişkenleri sayaç olarak değerlendirilebilir. Döngünün kontrolü amacıyla işaret (*flag*) adı verilen değişkenler de kullanılabilir. Bu değişkenler döngü içerisinde istenen başka koşulların kontrolünün yapılmasını ve buna göre döngünün devam etmesini veya bitmesini sağlar.

do while...loop ve *do...while* yapılarına farklı örnekler verilebilir:

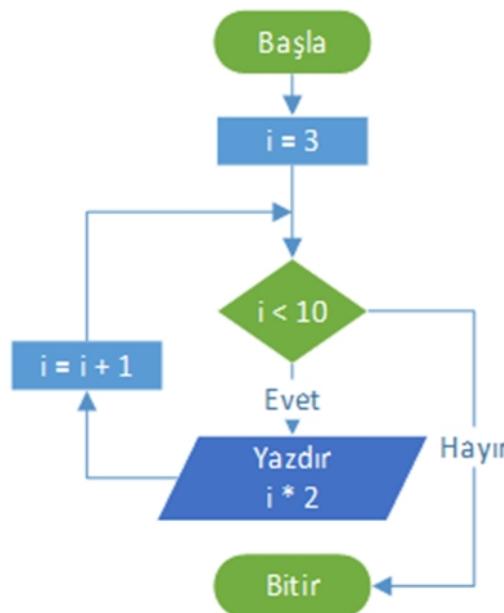
- a) Aşağıdaki program 1'den 5'e kadar olan sayıları ekrana yazdırmaktadır.

BAŞLA
sayaç = 0
DO
sayaç = sayaç + 1
YAZ (sayaç)
LOOP WHILE (sayaç < 5)
BİTİR
Çıktı:
1
2
3
4
5



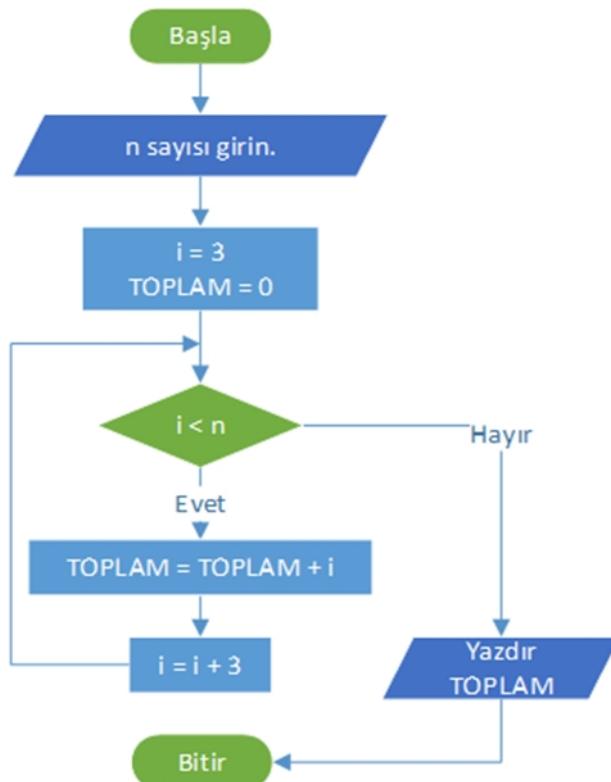
- b) $i=3$ 'ten başlamak üzere 10'a kadar olan sayıların 2 katını ekrana yazdırın program aşağıda verilmiştir.

BAŞLA
$i = 3$
DO WHILE ($i < 10$)
YAZ ($i * 2$)
$i = i + 1$
LOOP
BİTİR
Çıktı:
6
8
10
12
14
16
18



c) i sayısı 3'ten başlamak üzere, i ve n arasındaki üçün katları olan sayıların toplamını hesaplayan program aşağıda verilmiştir.

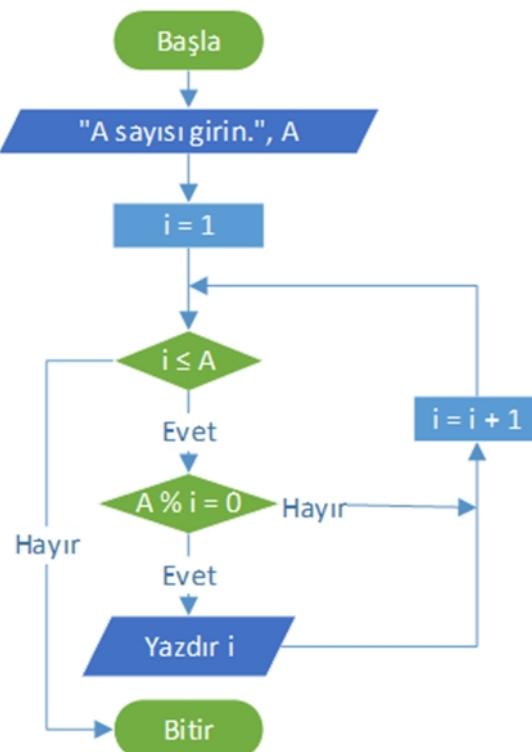
BAŞLA
n sayısını girin.
$i = 3$, TOPLAM = 0
DO WHILE ($i < n$)
TOPLAM = TOPLAM + i
$i = i + 3$
LOOP
YAZ (TOPLAM)
BİTİR
Çıktı:
18
n = 10 olarak seçilmiştir.
$3 + 6 + 9 = 18$



d) Girilen A sayısının tüm tam sayı bölenlerini hesaplayan program aşağıda verilmiştir.

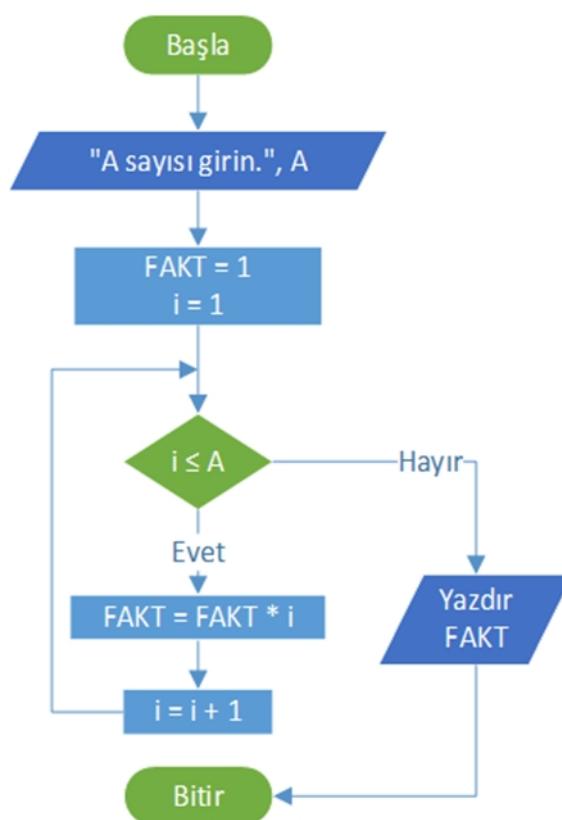
e)

BAŞLA
OKU ("A sayısı girin.", A)
$i = 1$
DO WHILE ($i \leq A$)
IF ($A \% i = 0$) THEN
YAZ (i)
END IF
$i = i + 1$
LOOP
BİTİR
Çıktı:
1
2
5
10
<hr/>
A = 10 girilmiştir.



Girilen A sayısına göre faktöriyel hesaplayan program ($A!$) aşağıda verilmiştir.

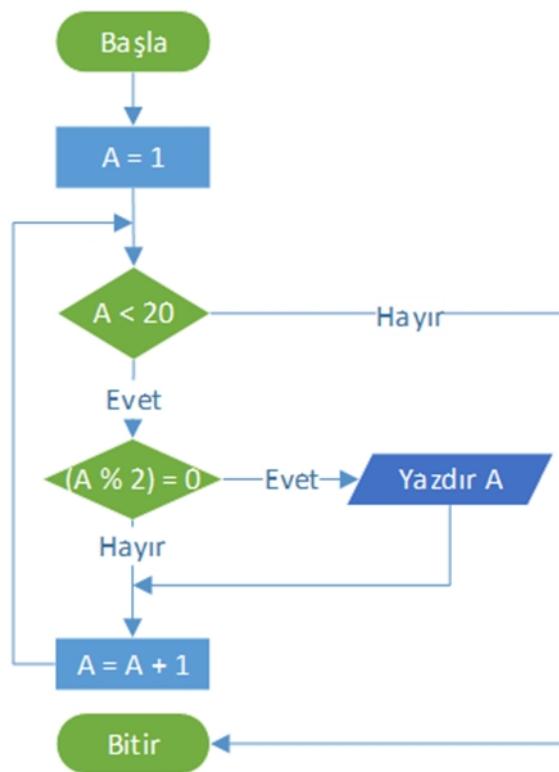
BAŞLA
OKU ("A sayısı girin.", A)
FAKT = 1, $i = 1$
DO WHILE ($i \leq A$)
FAKT = FAKT * i
$i = i + 1$
LOOP
YAZ (FAKT)
BİTİR
Çıktı:
120
A = 5
$5! = 5 * 4 * 3 * 2 * 1 = 120$



f) 1'den 20'ye kadar çift sayıları ekranı yazdıran program aşağıda verilmiştir.

g)

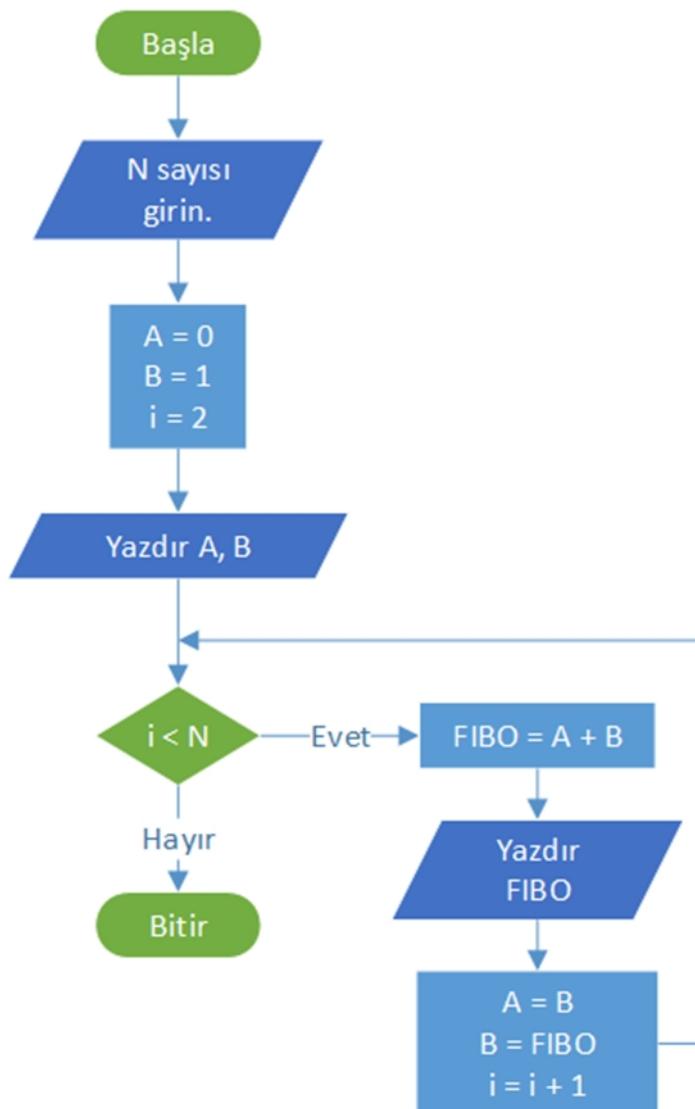
BAŞLA
A = 1
DO WHILE (A < 20)
IF (A % 2 = 0) THEN
YAZ (A)
END IF
A = A + 1
LOOP
BİTİR
Çıktı:
2
4
6
8
10
12
14
16
18



Girilen N sayısına kadar Fibonacci serisini hesaplayan program aşağıda verilmiştir. (Her sayının, kendinden bir ve iki önceki sayının toplanması sonucu oluşan sayı dizisine Fibonacci dizisi adı verilmektedir, 0, 1, 1, 2, 3, 5, 8, 13, 21, ...)

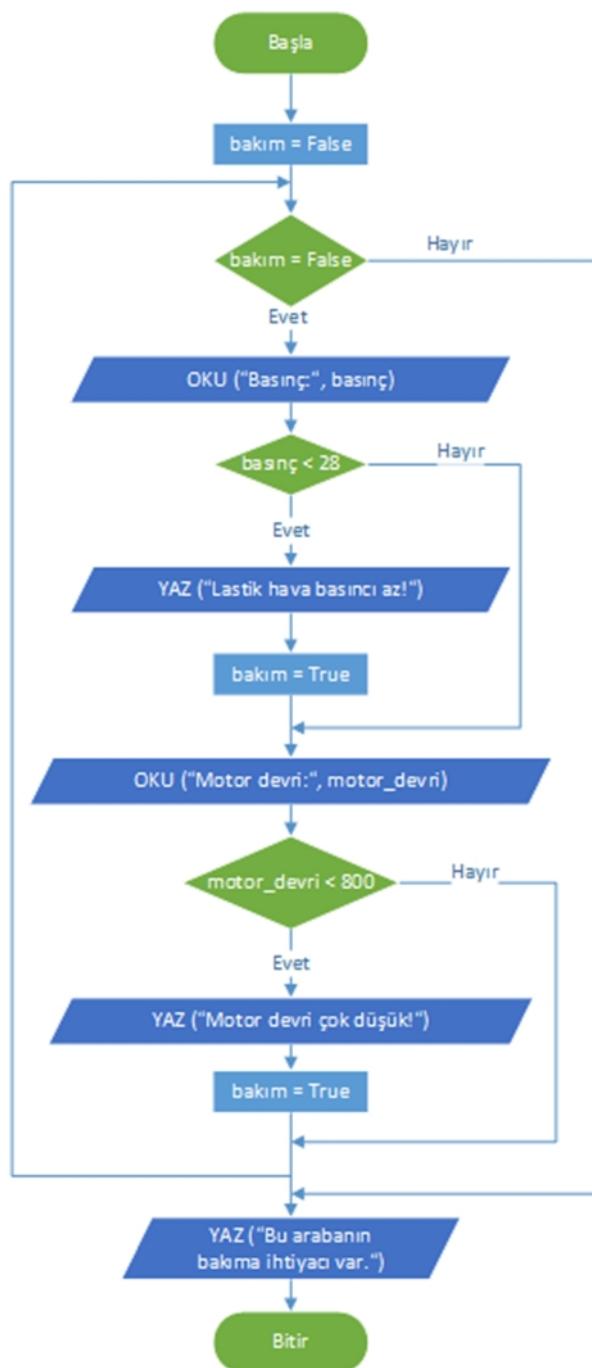
h)

BAŞLA
N sayısını girin.
A = 0, B = 1, i = 2
YAZ (A, B)
DO WHILE (i < N)
FIBO = A + B
YAZ (FIBO)
A = B
B = FIBO
i = i + 1
LOOP
BİTİR
Çıktı:
0 1 1 2 3 5
N = 6 seçilmiştir.



Aşağıdaki programda bir arabanın bakım zamanı ile ilgili kontrol işaret değişkeni kullanılarak yapılmaktadır. İşaret değişkeni ile döngü yapısı kontrol edilmektedir.

BAŞLA
bakım = False
DO WHILE (bakım = False)
OKU ("Basınç:", basınç)
IF (basınç < 28) THEN
YAZ ("Lastik hava basıncı az!")
bakım = True
END IF
OKU ("Motor devri:", motor_devri)
IF (motor_devri < 800) THEN
YAZ ("Motor devri çok düşük!")
bakım = True
END IF
LOOP
YAZ ("Bu arabanın bakıma ihtiyacı var.")
BİTİR

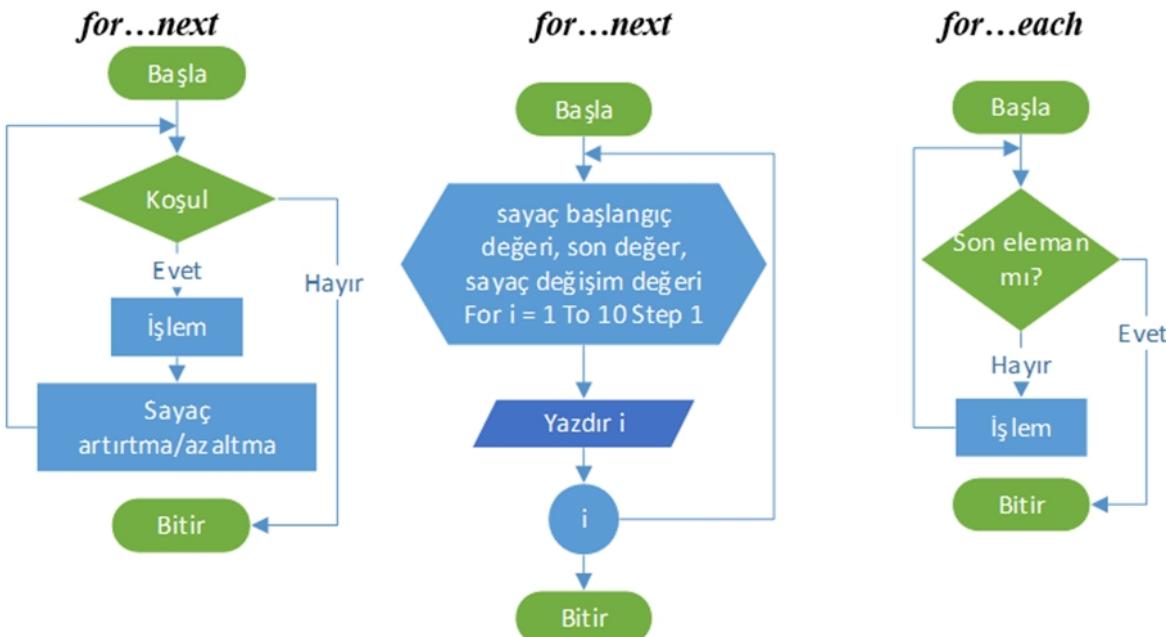


4.1.2. Tekrar Sayısı Belli Döngüler

Tekrar sayısı belli olan döngü yapılarında *for* döngüsünü kullanmak daha kolay olacaktır. Bu yapıda döngünün kaç kere tekrar edeceği baştan bellidir (Erwig, 2017). Belli olan sayı kadar kod bloğundaki işlemler tekrarlanır. *for* döngüsü kullanımını sayıç ile yapılabilecek işlemlerin kolaylaştırılmasını sağlar. Aslında *for* yapısının kullanıldığı durumlar *while* yapısı ile de ifade edilebilir. *do while...loop* yapısında ihtiyaç duyulan sayıç kullanımını *for* yapısında sayıç ve tekrarlama detaylarının tek bir satırda verildiği bir yapıya dönüştürülmüş olur (Deitel et al., 2014). Yani sayacın ayrıca ifade edilmesine gerek kalmadan tek satırda kısa bir kod satırı ile döngü ifade edilir. VB'de bu yapı, *for..next* olarak kullanılır. Bu yapı, bir işlemin belirlenen sayı kadar tekrarlanması sağlar (Deitel et al., 2014). VB'deki kullanımı ile *for each...next* yapısı ise sayıç kullanımına olan ihtiyacı ortadan kaldırarak istenen işlemin, bir kümede mevcut olan verinin herbiri için uygulanmasını sağlar. Bu yapı bir veri kümelerindeki veya dizideki her eleman için belirli işlemlerin tekrarlanması sağlar (Deitel et al., 2014). Örneğin, sınıfındaki her bir öğrenci için ortalamanın hesaplanması gerekmektedir. Herkes için aynı hesaplama işleminin tekrarlanması *for each...next* kullanımına örnek olarak verilebilir.

Bu döngünün temel mantığı ve akış diyagramında gösterim şekli *do while...loop* yapısına bir anlamda benzese de tekrarlanacak işlemin, tekrar sayısının belli olması sebebiyle diğer yapıdan ayrılmaktadır. Örnek

vermek gerekirse işlem, eleman sayısı zaten belli olduğundan son elemana kadar tekrarlanacaktır. Burada kontrol edilen koşul budur, *do while...loop* yapısında ise koşulun kaç kere kontrol edileceği baştan belli değildir. *for* döngüsü aşağıda sunulan, sol taraftaki gösterim ile ifade edilebilir ancak sıklıkla ortadaki altigen eleman ile yapılan gösterim kullanılır. Bu gösterim ile sayaçtaki değişimi ifade etmek için yeni bir işlem kutusu açılması gerekliliği ortadan kaldırılmış olur. Hem *for each...next* gösterimi hem de *for each...next* yapısı *if* yapısının gösterimini andırmaktadır.



for döngüsü ile ilgili önemli olan bir noktayı atlamamak gereklidir. Baştan belirlenen sayıç (indis) değeri, döngünün çalışması esnasında değiştirilmemelidir. Bu değer döngünün başında tanımlanmalı ve döngü boyunca *i* değerine başka bir değer atanmamalıdır. Değerin değiştirilmesi hem döngünün yapısını bozar hem de programın sonsuz döngüye girmesine sebep olabilir.

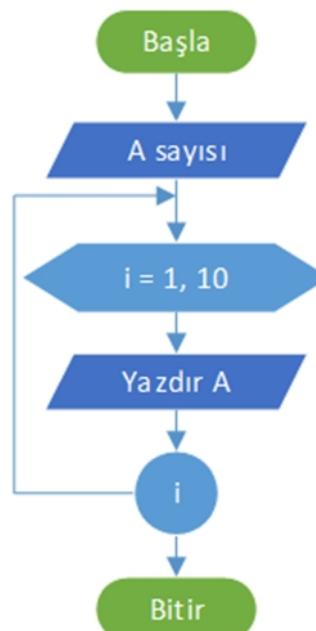
VB'de bu yapının gösterimi aşağıda verilmiştir. Bu gösterimde *step* parametresi de kullanılmaktadır. Ancak sayıç değerini belirten bu parametrenin eğer birer birer değer değişmesi isteniyorsa, bu parametreyi kullanmamız gereklidir. Yani *step* için bir değer girilmemesi bu değerin 1 olduğu varsayılmıştır. Örneklerde sayıç değeri birer artırdığında *step* ifadesi kullanılmamıştır. Sayaç, arttırılarak kullanılabileceği gibi azaltılarak da kullanılabilir. Artış değeri problemin türüne göre programcı tarafından belirlenir.

- **For** (sayaç başlangıç değeri) **To** (tekrar koşulu/son değer) **step** (sayaç değişim değeri)
[İşlem] → **Tekrar edilecek işlem**
Next
- **For Each** (veri kümesindeki her eleman) **In** (veri kümesi)
[İşlem] → **Tekrar edilecek işlem**
Next

Aşağıdaki örnekte ortalama hesaplama işlemi öğrenci sayısı kadar tekrarlanacaktır.	Bu örnekte ise sayıç olarak kullanılan i değeri 1'den başlayarak 20'ye kadar 2'ser artarak yazdırılacaktır.
For (sayac = 1) To (öğrenci_sayıısı) step (1) Ortalama hesapla Next sayıç	For (i = 1) To (20) step (2) YAZ (i) Next i
Aynı örnek <i>for...each</i> yapısı ile de ifade edilebilir.	Sayılar listesindeki her bir sayının yarısı ekrana yazdırılabilir.
For Each (öğrenci) In (öğrenci_listesi) Ortalama hesapla Next	For Each (sayı) In (sayılar_listesi) YAZ (sayı / 2) Next

Aşağıda 1'den başlayarak 10'a kadar olan sayılar sırayla ekrana yazdırılmaktadır. Bunun için i değerinin başlangıç değeri 1 olarak belirlenmiştir. Üst sınır yani for döngüsünün duracağı/sonlanacağı koşul ise i'nin 10 olduğu durumdur, yani 10 sayısı da yazdırılacaktır. for yapısı ayrıca sayı tanımlanması gerekliliğini ortadan kaldırır. Bununla beraber akış diyagramının anlaşılabilirliğinin artması için i'nin ifade edilirken yuvarlak akış diyagramı elemanı kullanılabilir.

BAŞLA
A sayısı gir.
FOR Sayaç (i) = 1 TO 10
A'yı yazdır.
NEXT i
BITİR



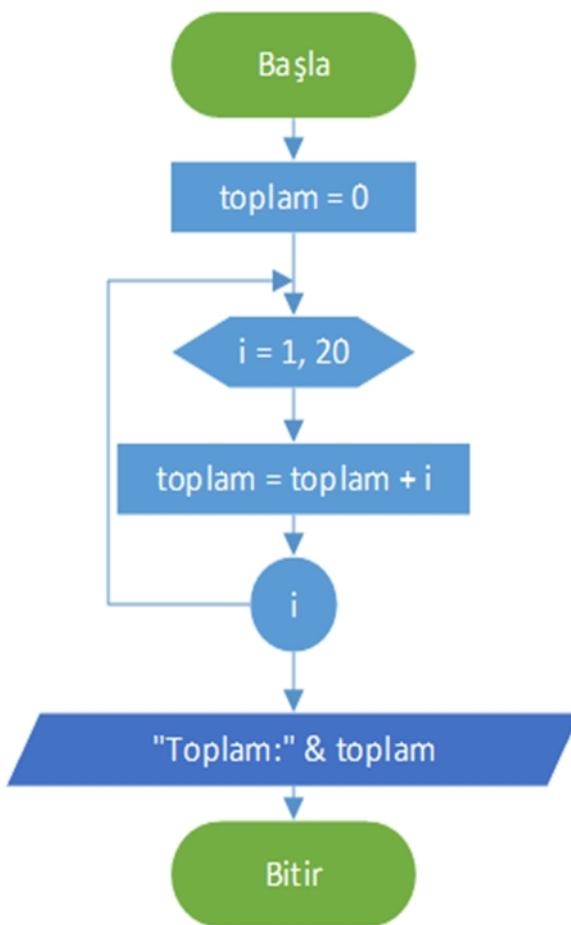
for yapısına farklı örnekler verilebilir:

- 1 ile 20 arasındaki sayıların toplamını yazdırın program aşağıda verilmiştir.

```

BAŞLA
toplam = 0
FOR i = 1 TO 20
    toplam = toplam + 1
NEXT i
YAZ ("Toplam:" & toplam)
BİTİR
Çıktı:
Toplam: 210

```



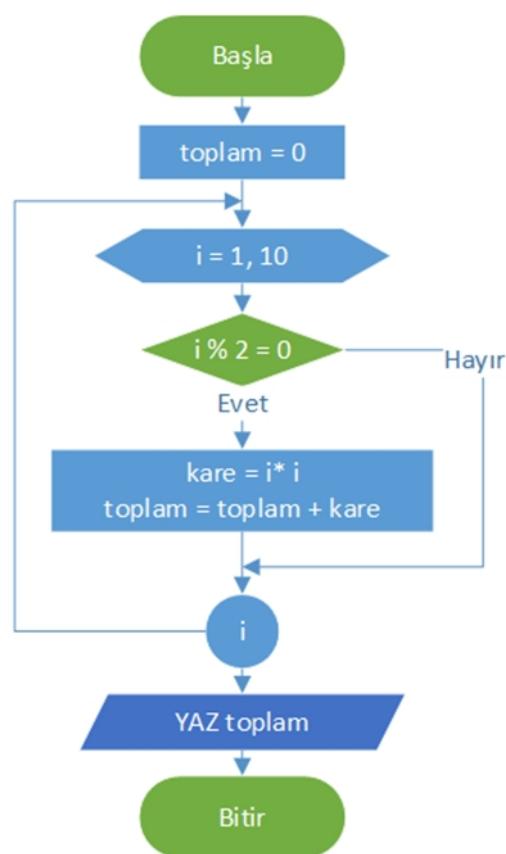
j) 1'den başlayarak 11'den küçük çift sayıların karesini toplayıp ekrana yazdırın program aşağıda verilmiştir.

```

BAŞLA
toplam = 0
FOR i = 1 TO 10
    IF (i % 2 = 0) THEN
        kare = i * i
        toplam = toplam + kare
    END IF
NEXT i
YAZ (toplam)
BİTİR
Çıktı:
220

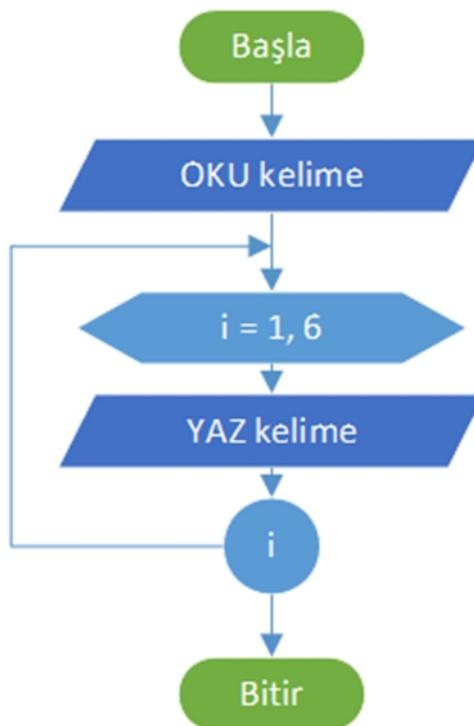
```

Kare	Toplam
2	4
4	16
6	36
8	64
10	100



k) Girilen kelimeyi 6 kere ekrana yazdırın program aşağıda verilmiştir.

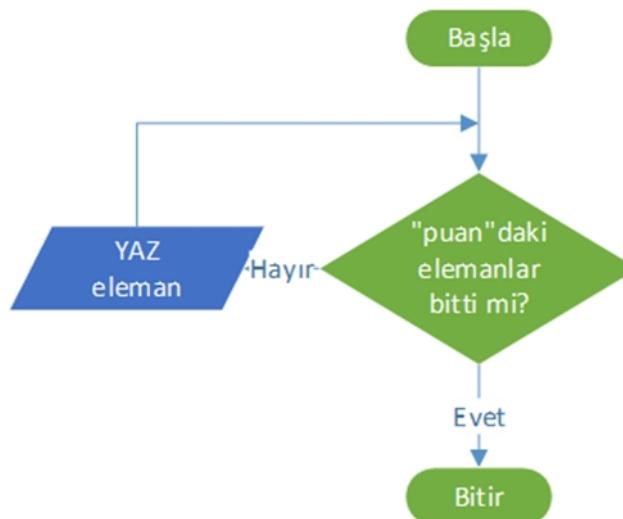
BAŞLA
OKU (kelime)
FOR i = 1 TO 6
YAZ (kelime)
NEXT i
BİTİR
Çıktı:
Algoritma
Algoritma
Algoritma
Algoritma
Algoritma
Algoritma



l) Bir dizideki elemanları yazdırın program aşağıda verilmiştir.

indis 0 1 2 3 4
puan 40 60 90 20 30

BAŞLA
FOR EACH eleman IN puan
YAZ (eleman)
NEXT i
BİTİR
Çıktı:
40
60
90
20
30



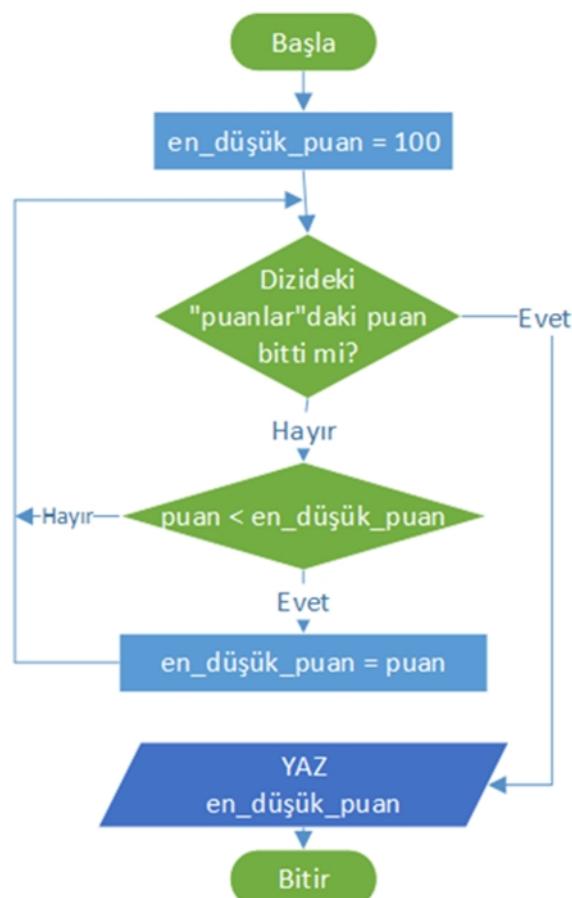
m) Puanlar dizisindeki en küçük elemanı değeri bulan program aşağıda verilmiştir.

indis 0 1 2 3 4
puanlar 40 60 90 20 30

```

BAŞLA
en_düşük_puan = 100
FOR EACH puan IN puanlar
IF (puan < en_düşük_puan)
    THEN
        en_düşük_puan = puan
    END IF
NEXT i
YAZ (en_düşük_puan)
BITİR
Çıktı:
20

```



Bölüm Özeti

Bu bölümde bir programda tekrarlayan eylemlerin ifade edilmesini sağlayan döngü yapıları ele alınmıştır. Döngüler temel olarak tekrar sayısı koşula bağlı döngüler ve tekrar sayısı belli döngüler olmak üzere ikiye ayrırlırlar. Döngülerin yazım şekli de dilden dile farklılık göstermektedir.

Birinci bölümde tekrar sayısı koşula bağlı olan döngüler ele alınmıştır. Bu döngüler “*do while...loop*” ve “*do...loop while*” olarak kendi içinde de çeşitlendirilmiştir. İlkinde kod bloğu çalışır ve eğer koşul sağlanıyorsa istenen eylem tekrarlanır, ikincisinde ise döngünün bulunduğu kod bloğu yürütüldükten sonra koşulun kontrolü sağlanır. Dolayısıyla ikisi arasındaki en önemli fark ikinci yapıda kontrol mekanizması sonda olduğu için istenen eylemin en az bir kez çalışacak olmasıdır. Tekrar sayısı koşula bağlı olarak belirlendiğinden ve koşul sağlanana kadar işlemler sürdürülüğünden tekrar sayısının bu döngülerde baştan bilinmediğini söyleriz. Bu yapı tekrar sayısının kontrol edilmesini sağlayan sayaç ile birlikte kullanılır.

İkinci bölümde ise tekrar sayısı baştan belli olan döngüler ele alınmıştır. Burada “*for...next*” ve “*for each...next*” yapılarından bahsedilmiştir. İşlemler başlamadan önce, işlemin kaç kere tekrar edeceğini bildiğimiz için bu döngü yapısında tekrar sayısının baştan belli olduğunu söyleriz. Bu yapıda da sayaç kullanımı mevcuttur. Ancak ayrıca bir sayaç değişkeni tanımlanması gerekliliği *for* döngüsünün yapısından kaynaklı olarak ortadan kalkar. Kolaylık olması açısından daire şeklindeki akış diyagramı elemanı sayaç artırımı veya azaltımı işlemi için kullanılabilir ve görsel ifadenin daha kolay anlaşılmasına sağlanması.

Ünite Soruları

Soru-1 :

do while...loop döngüsünde kontrol hangi aşamada yapılır?

(Çoktan Seçmeli)

- (•) - İlgili kod bloğunun başında
- (•) - İlgili kod bloğunun sonunda
- (•) - Program duruktan sonra
- (•) - Başla komutundan önce
- (•) - Hiçbiri

Cevap-1 :

İlgili kod bloğunun başında

Soru-2 :

do loop...while döngüsünde kontrol hangi aşamada yapılır?

(Çoktan Seçmeli)

- (•) - İlgili kod bloğunun başında
- (•) - İlgili kod bloğunun sonunda
- (•) - Program duruktan sonra
- (•) - Başla komutundan sonra
- (•) - Hiçbiri

Cevap-2 :

İlgili kod bloğunun sonunda

Soru-3 :

do while...loop ve do loop...while döngü yapıları arasındaki temel fark nedir?

(Çoktan Seçmeli)

- (•) - Sadece yazım şekillerinin farklı olması
- (•) - Farklı dillerde yazılmış olmaları
- (•) - Tekrarlanan işlemin uzun olması
- (•) - Birinde tekrarlanmak istenen işlemin en az bir kez yürütülmesi
- (•) - Hiçbiri

Cevap-3 :

Birinde tekrarlanmak istenen işlemin en az bir kez yürütülmesi

Soru-4 :

Sayı 100 iken aşağıdaki kod çalışmaktadır. Bu kodda yapılan hata nedir?

DO WHILE (sayı >= 0)

YAZ (sayı)

LOOP

(Çoktan Seçmeli)

- (•) - Yazım hatası yapılmıştır.
- (•) - Döngü sonlanmamaktadır.
- (•) - Tekrarlanan işlem sonuç vermez.
- (•) - Geçersiz bir sayı girilmiştir.
- (•) - Hiçbiri

Cevap-4 :

Döngü sonlanmamaktadır.

Soru-5 :

0 - 10 arasındaki sayıların toplamını do while...loop döngüsü kullanarak hesaplamak için hangi kod doğru yazılmıştır? (sayı = 0, toplam = 0 başlangıçta tanımlanmış varsayılr.)

(Çoktan Seçmeli)

- (•) - DO LOOP (sayı < 10)

toplam = toplam + sayı

sayı = sayı ++

WHILE

- (•) - DO WHILE (sayı < 10)

toplam = toplam + sayı

sayı = sayı ++

LOOP

- (•) - DO WHILE (sayı < 10)

toplam = toplam + sayı

sayı = sayı +

LOOP

- (•) - DO WHILE (sayı < 10)

toplam = toplam + sayı

sayı = sayı + sayaç

LOOP

(•) - DO LOOP ($\text{sayı} < 10$)

toplam = toplam + sayı

sayı + 1 = sayı

WHILE

Cevap-5 :

DO WHILE ($\text{sayı} < 10$)

toplam = toplam + sayı

sayı = sayı ++

LOOP

Soru-6 :

Aşağıdaki kontrollerden hangisi döngülerin bir türünü ifadede kullanılır?

(Çoktan Seçmeli)



Cevap-6 :



Soru-7 :

Bir for döngüsünde sayaç değişkeninin artması gereğini ifade etmek için hangi eleman kullanılır?

(Çoktan Seçmeli)

**Cevap-7 :**

Soru-8 :

10-0 arasındaki sayıları for döngüsü kullanarak azalarak yazdırılmak istendiğinde aşağıdakilerden hangisi bu yapı ile ilgili doğru bir ifade olur?

(Çoktan Seçmeli)

- (•) - Başlangıç değeri 0 olmalıdır.

- (•) - Bitiş değeri 10 olmalıdır.
- (•) - Sayaç birer birer arttırılmalıdır.
- (•) - Sayaç birer birer azaltılmalıdır.
- (•) - Sayaç olmasına gerek yoktur.

Cevap-8 :

Sayaç birer birer azaltılmalıdır.

Soru-9 :

for döngüsü ile ilgili aşağıdakilerden hangisi yanlıştır?

(Çoktan Seçmeli)

- (•) - Başlangıç değeri belirlenmelidir.
- (•) - Bitiş değeri belirlenmelidir.
- (•) - Akış diyagramında sayaç ayrı bir işlem elemanı olarak gösterilebilir.
- (•) - Akış diyagramında altigen eleman ile gösterilebilir.
- (•) - Artış miktarı mutlaka belirtilmelidir.

Cevap-9 :

Artış miktarı mutlaka belirtilmelidir.

5. ALT PROGRAMLAR, FONKSİYONLAR

Birlikte Düşünelim

1. Alt program ve fonksiyonlar programlama açısından ne gibi bir avantaj sağlar?
2. Alt program ve fonksiyonlar hangi şekillerde yazılabilir?
3. Farklı alt program tipleri nelerdir?

Başlamadan Önce

Bu bölümde programcılık açısından fayda sağlayan ve kodlamayı kolaylaştıran alt programlar ve fonksiyonlar aktarılmıştır. Bu yapılar kullanılarak önceki bölümlerde aktarılan farklı yapılardan da yararlanılarak, ana programa ait alt program parçalarının nasıl oluşturulabileceği gösterilmiştir.

5.1. Giriş

Bilgisayar programlarında birçok işlem bir arada yürütülerek bir girdiden belli bir çıktı elde edilmeye çalışılmaktadır. Bu işleyiş esnasında bazı işlemler veya rutinler birden fazla sefer yapılabilir. Örneğin finansal bir hesaplama sırasında değişik miktarlar üzerinden birçok kez vergi alınabilir ya da program akışı sırasında para birimleri arasında defalarca dönüşüm yapılabilir.

Tekrar eden kodları programın değişik yerlerine ayrı ayrı yerleştirmek yerine bu kodları program içinde bir yere toplayıp ihtiyaç oldukça çağrırmak, sayısız avantajı beraberinde getirir. Bu avantajların ilki kodun kısa olmasıdır. Kolay okunurluk açısından kodun kısa olması her zaman avantajdır ve aynı kod defalarca yazılabileceğine bir kere yazılıp ihtiyaç olduğunda çağrıldığında, program da haliyle kısalmış olur. Bir başka avantaj da tekrar eden kodda değişiklik yapılacağı zaman ortaya çıkar. Kodun içinde herhangi bir değişiklik yapılacağı zaman bunu defalarca yapmak yerine bir kere yapmak, hem zamandan tasarruf ettirir hem de olası hataların önüne geçer. Olası hatalardan kasıt, aynı değişikliği defalarca yaparken yapılabilecek dikkatsizlik hatalarıdır.

Program içerisinde bir yerde toplanan ve programındaki değişik yerlerden çağrılabilen program parçalarına alt program veya alt yordam denir. Bilgisayar programlamasında alt program, birim olarak paketlenmiş belirli bir görevi gerçekleştiriren bir dizi program talimatıdır. Bu birim daha sonra, söz konusu görevin gerçekleştirilmesi gerekiğinde, programındaki herhangi bir yerden uygun şekilde çağrılarak kullanılabilir. Alt programlar, programlar içinde veya birçok program tarafından kullanılabilen kitaplıklarda tanımlanabilirler. Farklı programlama dillerinde, bir alt program, rutin, işlev, yöntem veya prosedür olarak adlandırılabilir.

Alt programların amacı, programın yapısal olmasını sağlamak ve birbiriyle ilgili komutları veya programın bir bölümünü istenen isim altında toplamaktır. Bu şekilde programın okunması kolaylaşmakta ve program yapısal bir görünüm kazanmaktadır. Bir alt program, bir veya daha fazla ifade içerebilir. İyi yazılmış bir programda, her alt program yalnızca tek bir görev yürütür. Alt programlar tek başına çalışabilen yapılar değildir, sadece ana program içerisinde çağrılarak çalıştırılırlar. Bir alt program fikri ilk olarak John Mauchly tarafından ENIAC üzerine yaptığı çalışma sırasında ortaya çıkmıştır yani alt programların, ilk bilgisayarlara kadar dayanan bir tarihi vardır. Aşağıdaki örnekte alt program “Merhaba Dünya” yazmaktadır.

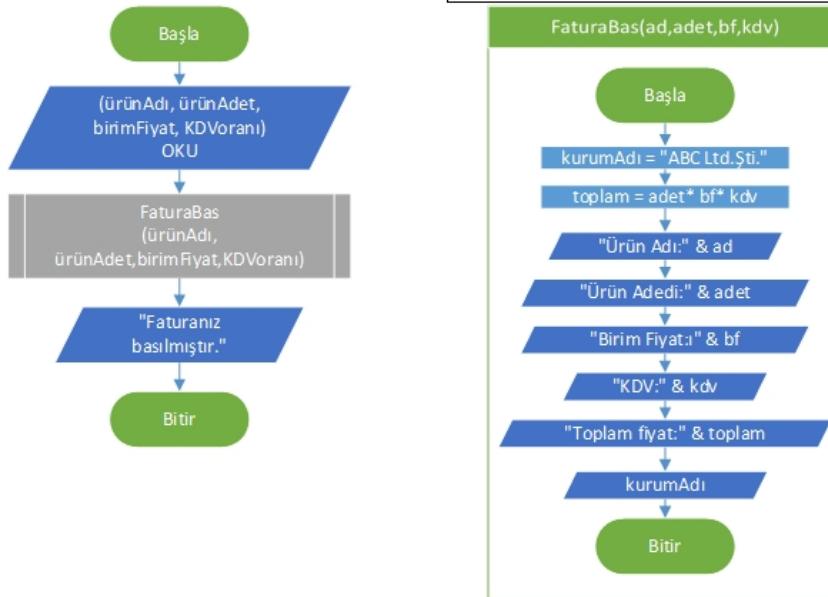
Ana Program	Alt Program
Başla	AltProgram MDYazdır()
Call MDYazdır()	Başla
Bitir	Yaz ("Merhaba Dünya!")
	Bitir

Alt programın programlama dillerinden bağımsız özellikleri vardır:

- Her alt programın bir ismi vardır. Bu isim verilirken, değişken isimleri tanımlarken kullanılan aynı kuralları göz önünde bulundurmak gereklidir. Ana program içerisindeki alt program çağrılarında bu isim kullanılır.
- Bir alt programın iş yapabilmesi için kendini çağırılan modülden aldığı bilgilere *parametre*, kendisini çağırılan programa döndürdüğü değere de *geri dönüş değeri (return value)* denir.
- İsteğe bağlı olarak programın diğer bölümlerinden alt programa nasıl erişilebileceği, erişim nitelikleri yardımıyla belirlenebilir. Eğer nitelik kullanılmazsa *private (özel)* olarak belirlenir ki bunun anlamı sadece yazıldığı modülde ulaşılabilir olması demektir. Eğer alt programa her yerden erişilsin isteniyorsa alt program *public (genel)* olarak tanımlanmalıdır.
- Alt programlar geri değer döndürenler ve döndürmeyenler olarak iki gruba ayrılabilirler. Geri değer döndürmeyenler programlama diline bağlı olarak *void*, *procedure*, *proc*, *sub* gibi isimler alırken geri değer döndürenler *function* gibi bir ismin yanı sıra sadece fonksiyonun geri döndürdüğü değişkenin tipiyle de tanımlanabilirler (*int*, *single* vb.).
- Alt programlar kullanım yerlerine göre parametre de alabilirler. Bu parametreler alt programı çağrılarında yollanır ve alt programda farklı isimlerle tanımlanarak kullanılır. Bir alt program çağrılarında programlama diline göre farklı sayıda parametre yollanabilir. Yollanan parametreler bazı dillerde virgül ile bazlarında ise noktalı virgülle ayrırlar. Eğer metod hiç parametre kullanmayacaksa parametre listesi de boş olur.

Aşağıdaki örnekte, sol taraftaki modül içerisinde, kullanıcıdan gerekli bilgileri alıp "FaturaBas" isimli alt programı kullanarak fatura bastıran bir program algoritması görülmektedir. Alt programda kurum adı doğrudan kodlanmıştır yani alt programın içinde değişken değeri atanmaktadır. Kalan değişkenler alt programın çağrıdığı yerden gönderilmiştir.

Ana Program	Alt Program
BAŞLA	AltProgram FaturaBas (ad, adet, bf, kdv)
OKU (ürünAdı, ürünAdet, birimFiyat, KDVOranı)	BAŞLA
Call FaturaBas (ürünAdı, ürünAdet, birimFiyat, KDVOranı)	kurumAdı = "ABC Ltd.Şti."
YAZ ("Faturanız basılmıştır.")	toplam = adet * bf * kdv
BİTİR	YAZ ("Ürün Adı:" & ad)
	YAZ ("Ürün Adedi:" & adet)
	YAZ ("Birim Fiyatı:" & bf)
	YAZ ("KDV:" & kdv)
	YAZ ("Toplam Fiyat:" & toplam)
	YAZ (kurumAdı)
	BİTİR



Alt yordamlar güçlü bir programlama aracıdır ve birçok programlama dilinin sözdizimi bunları yazmak ve kullanmak için destek içerir. Alt programların makul kullanımı (örneğin, yapılandırılmış programlama yaklaşımı aracılığıyla), genellikle büyük bir programın geliştirilmesi ve sürdürülmesinin maliyetini önemli ölçüde azaltırken, kalitesini ve güvenilirliğini arttırmır. Genellikle kütüphanelerde toplanan alt rutinler, yazılım paylaşımı ve ticareti için önemli bir mekanizmadır. Nesne yönelimli programlama disiplini, nesnelere ve metodlara (bu nesnelere veya nesne sınıflarına bağlı alt programlar) dayanır.

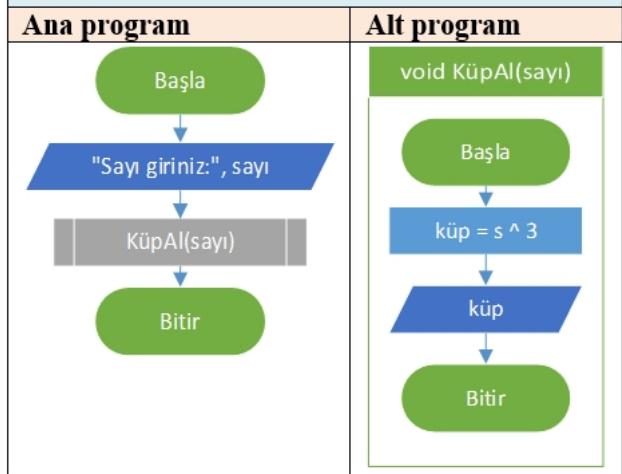
5.2. Geri Değer Döndürmeyen Alt Programlar (void)

Bu tip alt programlar kodlarının gerektirdiği işlemi yapar bitirir, ancak geriye herhangi bir değer döndürmezler, yani daha önce geri dönüş değeri (*return value*) olarak bahsedilen durum bu tip alt programlarda yoktur. Bu tarz rutinleri tanımlarken geri dönüş tipi yerine programlama diline bağlı olarak *void* gibi anahtar kelimeler yazılır. Bu alt programlar çağrılrken *call* ifadesi kullanılır. Aşağıda bazı örnekler görülmektedir.

- a) Aşağıdaki programda klavyeden girilen bir tam sayının küpünü bulan alt program ve bu alt programın çağrılmama kodu gösterilmektedir. Bu alt program çıktısını verdikten sonra ana programa dönülecek ve başka bir şey yapılmayacaktır. Bu tarz geri değer döndürmeyen programlardan *return* satırı yardımıyla veri döndürülmeye kalkılırsa hata alınır.

```
void KüpAl (int s)
BAŞLA
int küp = s^3
YAZ (küp)
BİTİR
```

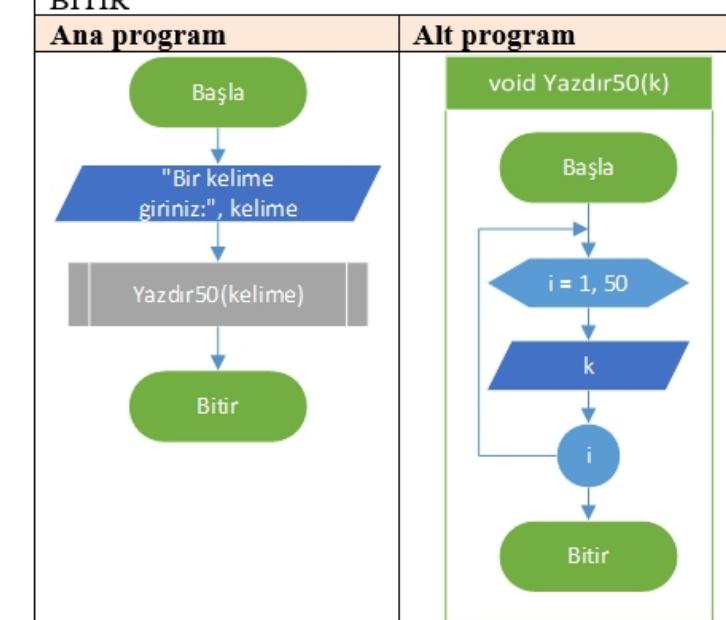
```
Ana Program
BAŞLA
OKU ("Sayınızı giriniz", sayı)
Call KüpAl(sayı)
BİTİR
```



- b) Aşağıdaki alt programda klavyeden girilen bir kelime ekrana 50 defa yazdırılmaktadır.

```
void Yazdır50 (string k)
BAŞLA
FOR i=1 TO 50
    YAZ (k)
NEXT i
BİTİR
```

```
Ana Program
BAŞLA
OKU ("Bir kelime giriniz:", kelime)
Call Yazdır50 (kelime)
BİTİR
```



- c) Aşağıdaki programda parametre olarak gönderilen kullanıcı adı ve parola kontrol edilerek, önceden belirlenmiş olan bir kullanıcı adı ve şifreyle karşılaştırılmaktadır.

```
void GirişYap (string k_adi, string k_parola)
```

```
BAŞLA
```

```
IF (k_adi=="yönetici" AND k_parola=="deneme") THEN
```

```
    YAZ ("Kullanıcı ve parola doğru.")
```

```
ELSE
```

```
    YAZ ("Kullanıcı ve parola yanlış.")
```

```
END IF
```

```
BİTİR
```

```
Ana Program
```

```
BAŞLA
```

```
OKU ("Kullanıcı adı:", kullanıcıAdı)
```

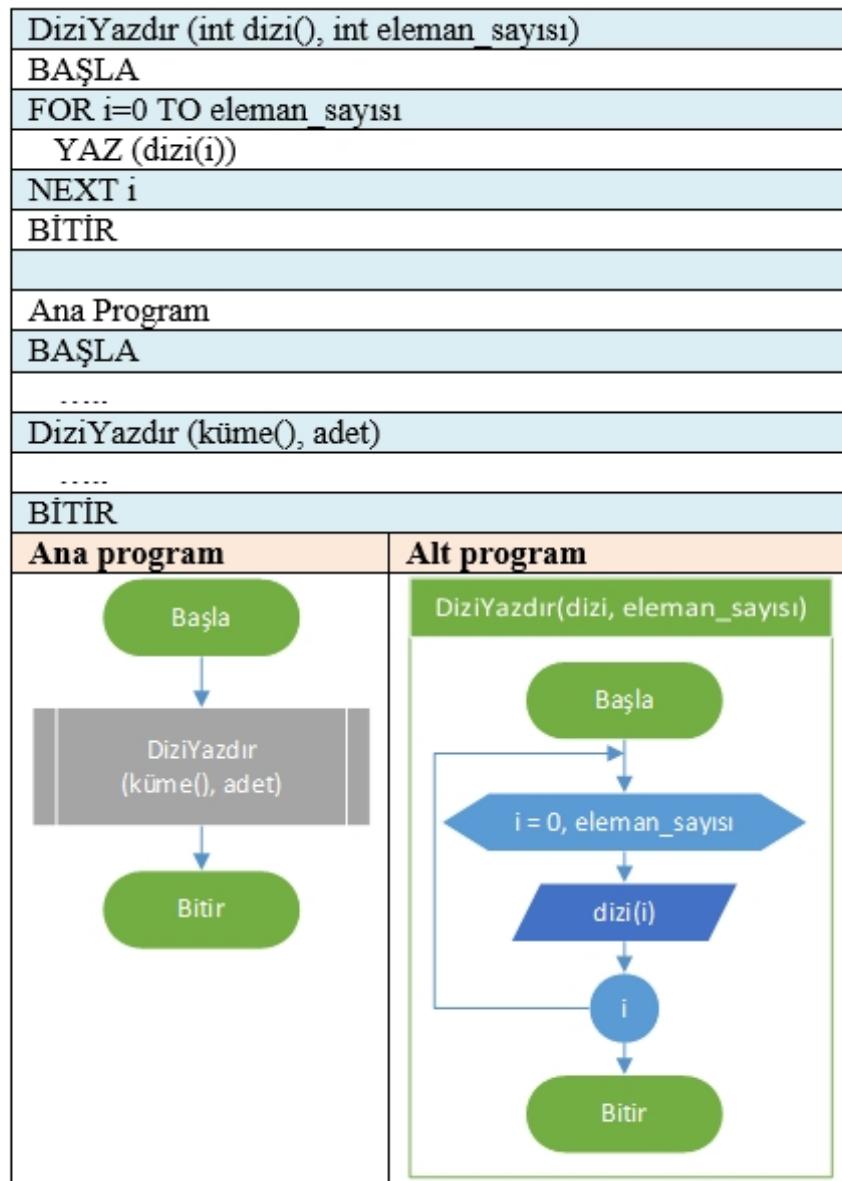
```
OKU ("Parola:", parola)
```

```
Call GirişYap (kullanıcıAdı, parola)
```

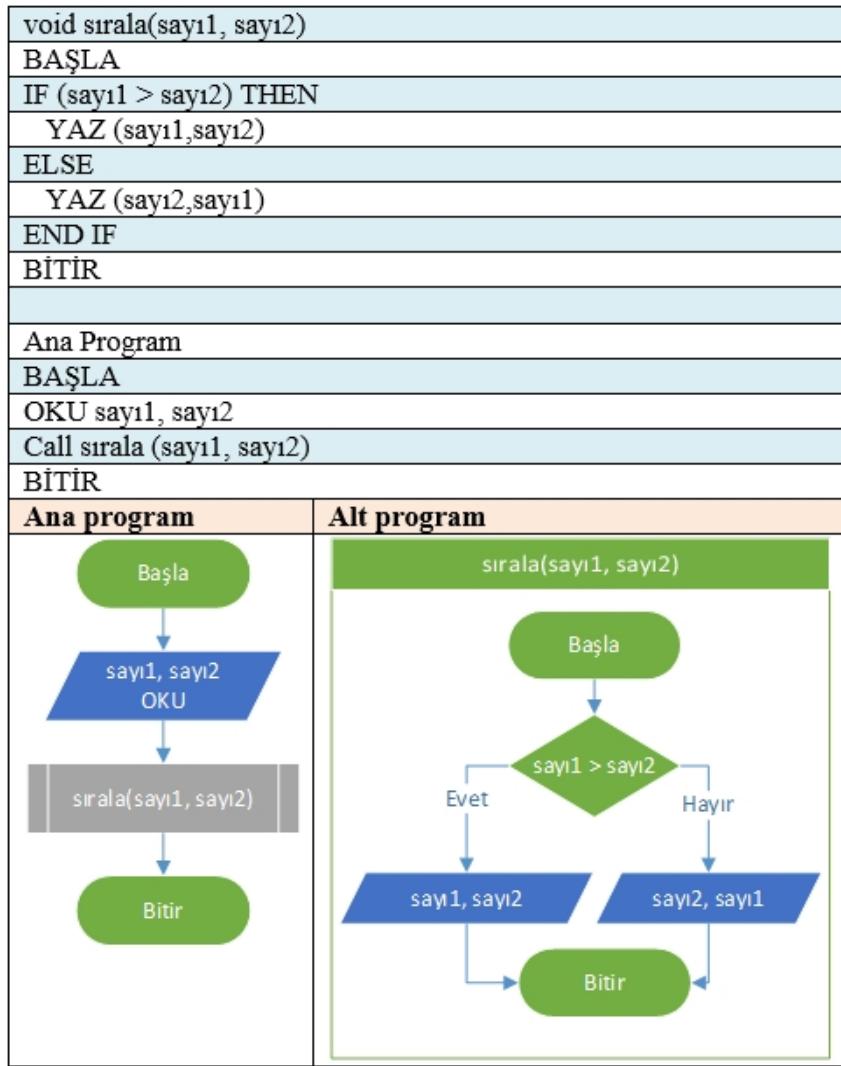
```
BİTİR
```

Ana program	Alt program
<pre> Başla ↓ "Kullanıcı adı:", kullanıcıAdı ↓ "Parola:", parola ↓ GirişYap (kullanıcıAdı,parola) ↓ Bitir </pre>	<pre> GirişYap(k_adi, k_parola) Başla ↓ k_adi = "yönetici" AND k_parola = "deneme" ↓ Evet --> "Kullanıcı adı ve parola doğru." Hayır --> "Kullanıcı adı ve parola yanlış." ↓ Bitir </pre>

- d) Aşağıdaki fonksiyon bir dizideki sayıları sırayla yazdırmaktadır.



- e) Verilen iki sayıyı eşitliklerine bakmadan büyükten küçüğe sıralayarak ekrana yazdırın algoritma aşağıda verilmiştir.

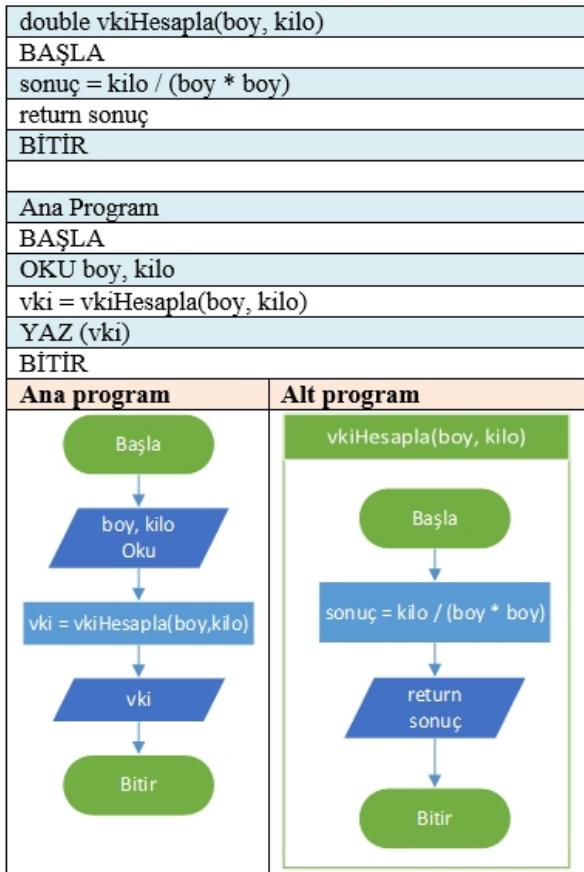


5.3. Geri Değer Döndüren Alt Programlar (fonksiyonlar)

Bu tür alt programlarda işlemler yapıldıktan sonra hesaplanan bir değer çağrııldığı yere geri döndürülmektedir. Bu yüzden bu tür alt programlar *call* gibi bir komutla çağrılmazlar, aksine geri dönen verinin değerine uygun bir değişkene eşitlenerek çağrırlılar. Bu tarz alt programlar genellikle tanımlanırken başına *void* gibi bir kelime yerine programlama diline bağlı olarak fonksiyonun geri döndürdüğü verinin tipi yazılır.

Parametrenin tanımını ve kullanımını daha önce alt programların kendine has özelliklerinden bahsederken aktarmıştık. Parametre listeleri, tek bir türde verileri içeren bir liste olabileceği gibi, farklı türlerde de veriler içerebilen listelerdir. Parametreler herhangi bir veri türünde olabileceği gibi nesneler de parametre olarak bir metoda gönderilebilirler. Parametreler aralarına virgül kullanılarak birbirinden ayrırlılar. Parametrelerin her biri için değişken isimlerinden önce ayrı ayrı veri türleri de yazılmak zorundadır. Çeşitli veri türlerinin parametre olarak fonksiyonlara nasıl gönderildikleri aşağıdaki örneklerde görülebilir.

- f) Girilen boy ve kiloya göre vücut kitle indeksini hesaplayan algoritma aşağıda verilmiştir.



- g) Aşağıdaki programda klavyeden girilerek parametre olarak gönderilen bir sayı alt program tarafından sayının asal sayı olup olmadığını kontrol edilerek, eğer sayı asal ise *true*, değilse *false* değeri döndürülmektedir.

```

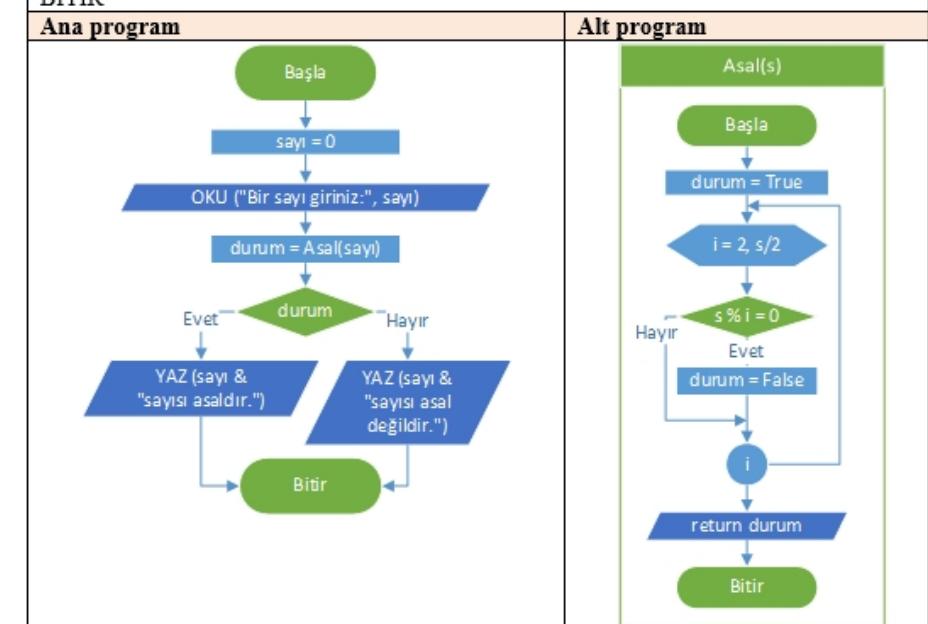
bool Asal(int s)
BAŞLA
bool durum=true;
FOR i = 2 TO i< s/2
    IF (s%i=0) THEN
        durum=false
    END IF
NEXT i
return durum
BITİR

```

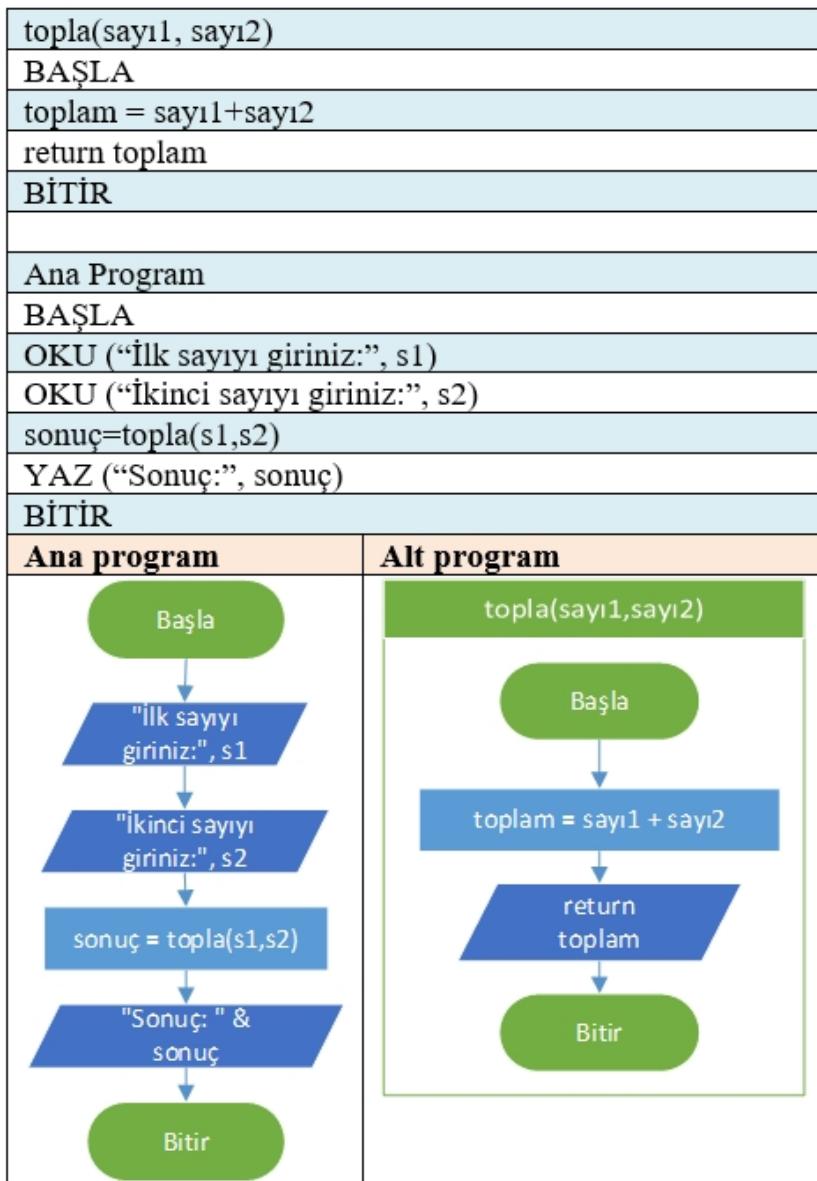
```

Ana Program
BAŞLA
int sayı=0
OKU ("Bir sayı giriniz:", sayı)
durum=Asal(sayı)
IF durum THEN
    YAZ (sayı & "sayısı asaldır.")
ELSE
    YAZ (sayı & "sayısı asal değildir.")
END IF
BITİR

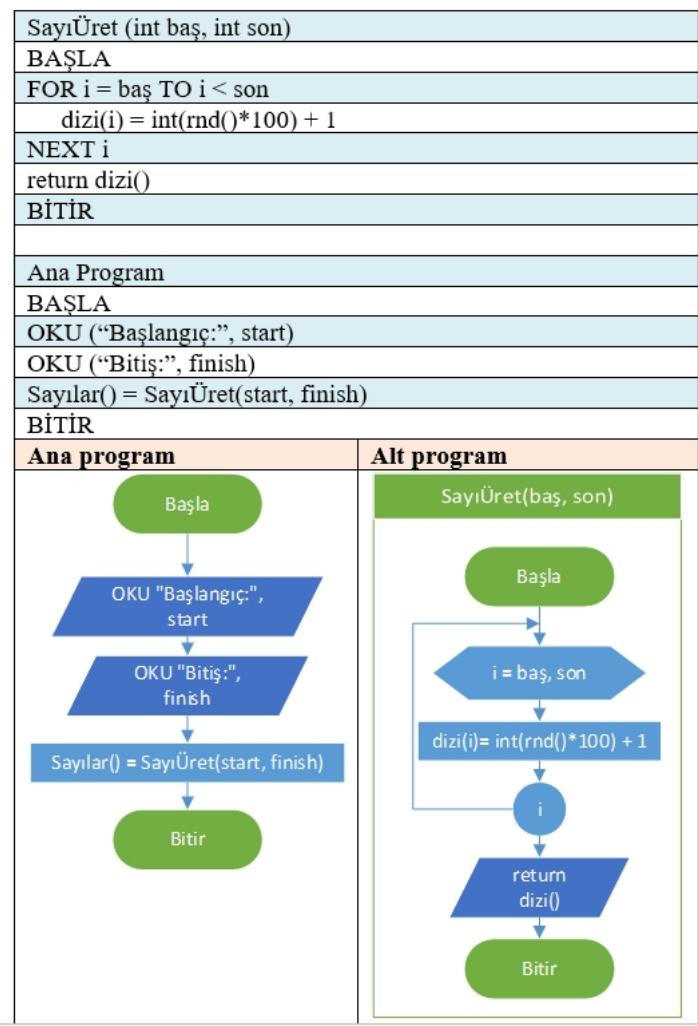
```



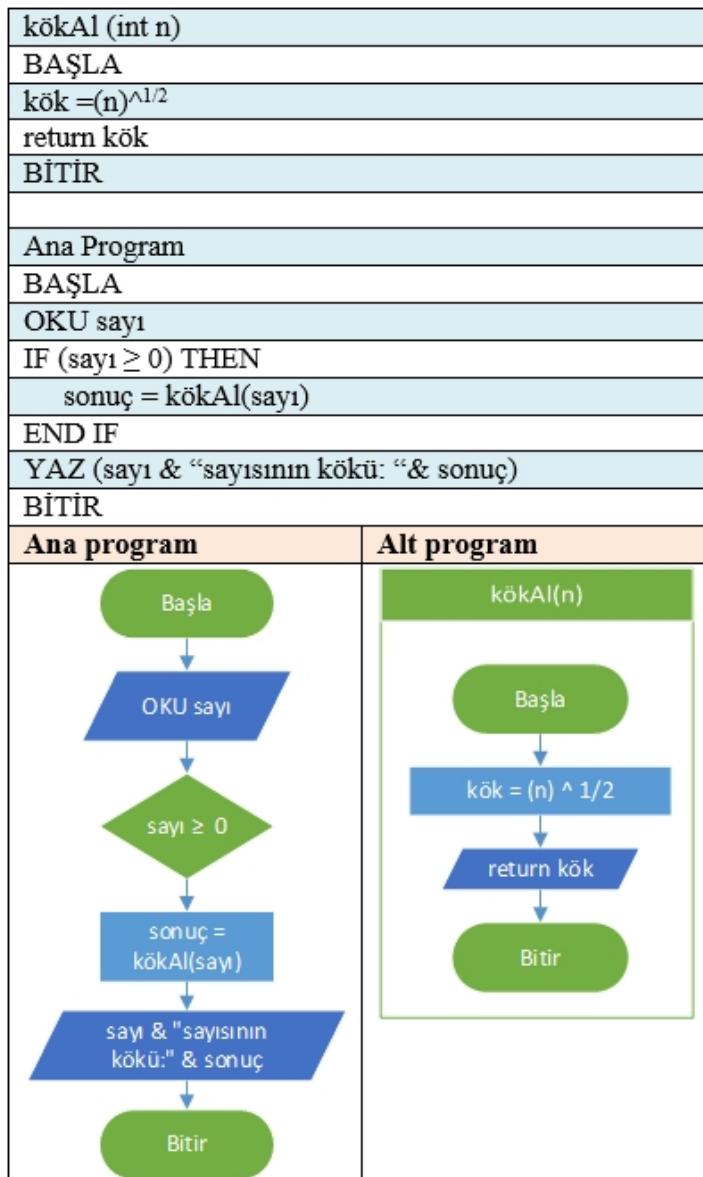
- h) Aşağıdaki örnekte topla isimli alt program vasıtasyyla iki sayı toplanıp sonuç ana programda ekrana yazdırılmaktadır.



- i) Aşağıdaki fonksiyon başlangıç ve bitiş sayıları kullanıcı tarafından belirtilen rasgele sayılar üretmektedir. Burada kullanılan rnd fonksiyonu birçok programlama dilinde 0 – 1 arası sayı üretmeye kullanılır. Üretilen sayı 100 ile çarpılıp 1 eklendiğinde sayı 1 – 100 aralığında olmuş olur.



- j) Verilen sayının karekökünü alan program aşağıda verilmiştir.



- k) Aşağıdaki fonksiyon bir dizideki en büyük sayıyı bulup yazdırmaktadır.

```
EnBüyük (int dizi(), int eleman sayısı)
```

```
BAŞLA
```

```
en büyük=dizi(0)
```

```
FOR i = 1 TO i < eleman sayısı
```

```
IF (dizi(i) > en_büyük) THEN
```

```
en_büyük = dizi(i)
```

```
END IF
```

```
NEXT i
```

```
return en_büyük
```

```
BİTİR
```

```
Ana Program
```

```
BAŞLA
```

```
.....
```

```
büyük = EnBüyük (küme(), adet)
```

```
.....
```

```
BİTİR
```

```
Ana program
```

```
Alt program
```

```
büyük = EnBüyük(küme(), adet)
```

```
EnBüyük(dizi(), eleman sayısı)
```

```
en_büyük = dizi(0)
```

```
i = 1,eleman sayısı
```

```
dizi[i] > en_büyük
```

Evet

Hayır

```
EnBüyük = dizi(i)
```

```
return
```

```
en_büyük
```

5.4. Özyinelemeli (Recursive) Metodlar

Bir alt programın kendi kendini çağrımasına yinelenme (*recursion*), kendi kendini çağrıran fonksiyonlara da yinelenen, özyineli veya özyinelemeli (*recursive*) fonksiyonlar denir. Program yazarken zaman zaman ihtiyaç duyulan bu durum, sadece geri değer döndüren alt programlara yani fonksiyonlara uygulanabilir. Bu tür fonksiyonlar tasarlanırken yapılacak bir hata fonksiyonun durmaksızın kendisini çağrımasına yani sonsuz bir döngünün oluşmasına sebep olabilir. Bu yüzden döngünün bitme şartının kontrol edilerek alt programın bittiğinden emin olunmalıdır. Bu tür fonksiyonlar giriş seviyesinde programcılara zor gelse de kısa ve öz kodlar oluşturduklarından zarif bir kod olarak nitelendirilirler. Aşağıda bu fonksiyonlara örnekler verilmiştir.

- I) 1'den başlayarak klavyeden girilen sayıya (n) kadar olan sayıların toplamını hesaplayan programın kodunu yazınız.

```
int Topla(int n)
BAŞLA
IF (n = 0) THEN
    return 0
ELSE
    return (n + Topla(n - 1))
END IF
BITİR
```

Ana Program

BAŞLA

OKU ("Bir sayı giriniz:", sayı)

int sonuç = Topla(sayı)

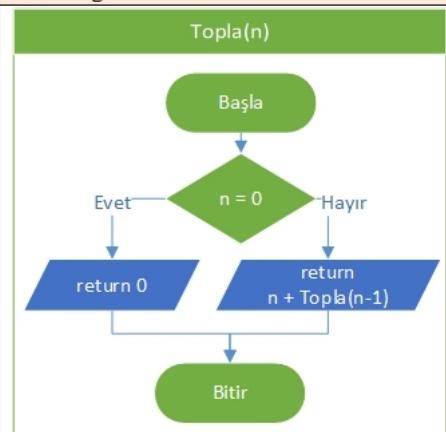
YAZ ("Birden " & "ya kadar olan sayıların toplamı:", sonuç)

BITİR

Ana Program



Alt Program



m) Klavyeden girilen sayının faktöriyelini hesaplayan programın kodunu yazınız.

```
double Faktoriyel(int n)
BAŞLA
IF (n = 0) THEN
    return 1
ELSE
    return (n * Faktoriyel (n - 1))
END IF
BITİR
```

Ana Program

BAŞLA

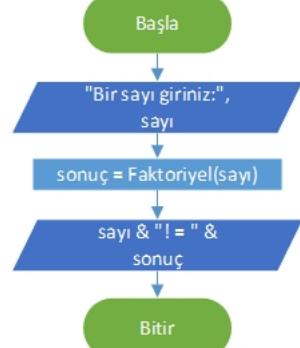
OKU ("Bir sayı giriniz:", sayı)

double sonuç = Faktoriyel(sayı)

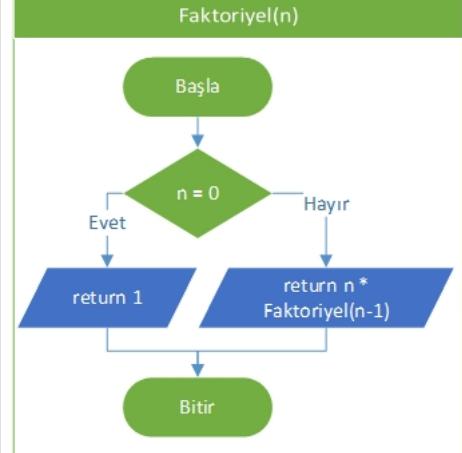
YAZ (sayı & "! = ", sonuç)

BITİR

Ana Program



Alt Program



5.5. Main Metodu

Program ve alt programlar anlatılırken hep bir ana programdan bahsedilmiştir. Aslında bu ana program geriye değer döndürmeyen programlardan farklı bir program değildir. Diğer prosedürlerden tek farkı, bulunduğu program dosyası içinde o dosyanın başlamasını sağlayan (bir nevi başlangıç noktası) program olmasıdır. Bu ana program, programlama dillerine bağlı olarak *void main*, *sub main* gibi isimler alabilir. Aslında bazı durumlarda *int* türünde bir geri dönüş değerine sahip olsa da bu dönen değer, programcılar işine pek yaramadığından algoritma anlatılırken bundan bahsedilmez. Bu değer, genellikle (bütün programların üzerinde çalıştığı) işletim sisteminin, yazılan programın nasıl sonlandırıldığıyla ilgili bilgi almasını sağlayacak bir değerdir.

Bahsedilenlerin dışında bazı durumlarda bu ana program dışarıdan parametre alabilir. Normalde modül derlendikten sonra komut satırından çalıştırılabilir olmaktadır. Bu gibi durumlarda aşağıdaki tarzda bir satırla komut satırından program, parametreli olarak çalıştırılabilir.

```
C:\.....\bin\Debug\deneme.exe test
```

Burada yolu verilen (derlenmiş ve çalıştırılmaya hazır) deneme.exe isimli dosya test değeri aktarılırak komut satırından çalıştırılmaktadır. Eğer program hatasız şekilde çalışırsa çıktıları görüntülenecektir.

5.6. Alt Programlarla İlgili Uyarılar

1. Geriye değer döndürmeyen alt programlarda *return* komutu kullanılamaz.
2. Alt program çağrılrken gönderilen parametre sayısı, sırası ve türü alt programdaki deklarasyonla uyumlu olmalıdır.
3. Parametre listesinde tanımlanan değişkenlerin isimlerinin, alt program içerisinde tanımlanacak başka bir değişkende tekrar kullanılması önerilmemektedir.
4. Alt program ismi ne başka bir alt program veya ana programda ne de kendi kodunda bir değişken için kullanılabilir.
5. Geri değer döndüren alt programdan dönen değerin cinsi ana programda o fonksiyonu çağrılrken kullanılan parametre türüyle uyuşmazsa hata verecektir. Örneğin, *int* türünde tanımlanmış olan sonuç değişkeni içerisinde *single* türünde tanımlanmış bir fonksiyonun geri dönüşü atanmaya çalışılırsa hata mesajı alınır.
6. Bir fonksiyon içinde başka fonksiyon tanımlanamaz ancak çağrılabılır.
7. Alt programların içerisinde tanımlanan değişkenlerin hiçbir alt program dışında kullanılamazlar, geçersiz olurlar. Örneğin, topla isimli bir fonksiyon içerisinde tanımlanan *int* türündeki *c* isimli değişkene metod dışında tekrar erişilmek istenirse hata mesajı alınır.

Anlatılan uyarılarla ilgili örneklerle aşağıdaki tabloda (Tablo 9) aynı sıra numarası üzerinden ulaşılarak bahsi geçen problem incelenebilir.

Tablo 9: Potansiyel Hatalar

1) Sağ taraftaki void tipi alt programdan ana programa değer döndürilmeye çalışılmaktadır (kırmızı kod), bu durum hataya sebep olacaktır.	void Topla BAŞLA sonuç = a+b return sonuç BİTİR
2) Sağdaki alt program kırmızı ile işaretli kodlarda gösterildiği gibi eksik veya fazla parametreli olarak çağrılamaz.	int Topla (int a, int b) BAŞLA t = a + b return t BİTİR Sonuç = Topla (sayi1, sayi2, sayi3) Sonuç = Topla (sayi1)
3) Sağdaki alt programda kırmızı işaretli satır hata verecektir.	int Topla (int a, int b) BAŞLA string a="Ders" BİTİR
4) Sağdaki kırmızı işaretli satır hata verecektir.	int Topla (int a, int b) BAŞLA BİTİR Ana Program string Topla = "Toplam"
5) Sağdaki fonksiyonun döndürdüğü değerin tipiyle alttaki kırmızı satırındaki değerin tipi uymadığından hata verecektir.	single Topla (int a, int b) BAŞLA t = a + b return t BİTİR int sonuç = Topla (sayi1, sayi2)
6) Sağdaki alt programın içinde kırmızıyla gösterilen kod sorun yaratacaktır ancak yeşil olan problemsiz çalışır.	int Topla (int a, int b) BAŞLA int YAZ (int c, int d) bool kontrol = KontrolPozitif (a) BİTİR bool KontrolPozitif(int e) BAŞLA if (a>0) return True else return False end if BİTİR
7) Sağdaki ana programın içinde kırmızı ile gösterilen satır, değişken alt programda tanımlandığı için doğru çalışmayaçaktır ve hata verecektir.	int Topla (int a, int b) BAŞLA c = a + b BİTİR Ana Program BAŞLA int sonuç = Topla(x, y) YAZ (c) BİTİR

Bölüm Özeti

Bu bölümde alt programlar ve fonksiyonlardan bahsedilmiştir ve bunların farklı türlerine yer verilmiştir.

İlk bölümde alt program ve fonksiyonların programcılık açısından sağladığı faydalardan söz edilmiştir. Bunların kullanımı tekrarlanacak işlemler için programın farklı yerlerinde tekrar tekrar aynı kodun yazılmasını engeller. Böylelikle tüm program daha kısa bir şekilde ifade edilebilir. Yazılan bir alt program veya fonksiyon, sadece ihtiyaç duyulduğu zamanlarda ve yerlerde çağırılacağı için kod yazımı kısalır ve program yapısal bir görünüm kazanır.

İkinci bölümde geri değer döndürmeyen alt programlar ele alınmıştır. Bu alt programlar adlarından da anlaşılabilceği gibi geriye herhangi bir değer döndürmezler yani bir geri dönüş değeri (*return value*) bu alt program türünde mevcut değildir. *void* anahtar kelimesi ile bu programlar tanımlanabilir. *call* anahtar kelimesi ile çağrırlılar, *return* anahtar kelimesi bu programlarda kullanılamaz.

Üçüncü bölümde geri değer döndüren alt programlardan bahsedilmiştir. Bu alt programlara fonksiyon adı da verilir. Bu alt program türünde ise hesaplanan bir değer çağrıldığı yere geri döndürülür. *call* anahtar kelimesi ile çağrılamaz, geri dönen verinin değerine uygun bir değişkene eşitlenerek çağrılmaları mümkündür. Bu bölümde parametre listeleri ile ilgili bilgilere de yer verilmiştir.

Dördüncü bölümde özyineli fonksiyonlardan yani kendisini çağrıran fonksiyonlardan bahsedilmiştir. Bu fonksiyonların kullanımı başlangıçta zor gibi görünse de bunların kullanımı programcılık açısından kısa ve öz kodlar oluşturulmasını sağlar.

Beşinci bölümde ise diğer bölgümlerde sıkılıkla sözü geçen *main* metodu ile ifade edilen ana program kısaca ele alınmıştır. Altıncı ve son bölüm ise alt program ve fonksiyonların kullanımında yapılabilecek olan potansiyel hatalar listelenmiş ve her bir duruma uygun örnekler sunulmuştur.

Kaynakça

Microsoft. (2022). Visual Basic Belgeleri. <https://docs.microsoft.com/tr-tr/dotnet/visual-basic/>

Ünite Soruları

Soru-1 :

Bir programdaki ana program nasıl adlandırılır?

(Çoktan Seçmeli)

- (•) - void
- (•) - main
- (•) - proc
- (•) - mainprog
- (•) - static void

Cevap-1 :

main

Soru-2 :

Geri değer döndürmeyen alt programlar aşağıdakilerden hangisi gibi ifade edilemez?

(Çoktan Seçmeli)

- (•) - void
- (•) - procedure
- (•) - proc
- (•) - func
- (•) - void Programısmi

Cevap-2 :

proc

Soru-3 :**Geriye değer döndüren alt programlar aşağıdakilerden hangisi gibi adlandırılır?**

(Çoktan Seçmeli)

- (•) - void
- (•) - procedure
- (•) - proc
- (•) - function
- (•) - return

Cevap-3 :

function

Soru-4 :**Geriye değer döndüren alt programlar aşağıdakilerden hangisi ile çağrırlır?**

(Çoktan Seçmeli)

- (•) - call
- (•) - Uygun değişkene eşitlenerek
- (•) - main
- (•) - void metodu ile
- (•) - Hiçbiri

Cevap-4 :

Uygun değişkene eşitlenerek

Soru-5 :**Kendini çağıran programlara ne ad verilir?**

(Çoktan Seçmeli)

- (•) - void
- (•) - Geri döndürmeyen alt program
- (•) - Tekrarlı ana program
- (•) - Geri değer döndüren alt program
- (•) - Özyineli fonksiyon

Cevap-5 :

Özyineli fonksiyon

Soru-6 :**Değişkenlerin ismi parametre listesinde tanımlandıktan sonra alt program içerisinde tekrar kullanılabilir mi?**

(Çoktan Seçmeli)

- (•) - Evet
- (•) - Hayır
- (•) - Koşula göre değişir.
- (•) - Döngü varsa kullanılır.
- (•) - Hiçbiri

Cevap-6 :

Hayır

Soru-7 :**Geri değer döndüren alt programdan dönen parametrenin cinsi ana programda o fonksiyonu çağırırken kullanılan parametre tipiyle uyusmazsa ne olur?**

(Çoktan Seçmeli)

- (•) - Program sorunsuz çalışır.
- (•) - Sadece alt program çalışır.
- (•) - Program sonsuz döngüye girer.
- (•) - Hata alınır.

- (•) - Hiçbir şey olmaz.

Cevap-7 :

Hata alınır.

Soru-8 :

Metotların içerisinde tanımlanan değişkenler metot dışında kullanılabılır mı?

(Çoktan Seçmeli)

- (•) - Kullanılabilirler.
- (•) - Kullanılamazlar.
- (•) - Duruma göre kullanılabilir.
- (•) - Özyine fonksiyonsa kullanılabilir.
- (•) - Hiçbiri

Cevap-8 :

Kullanılamazlar.

Soru-9 :

Özyineli fonksiyonlar nasıl nitelendirilebilir?

(Çoktan Seçmeli)

- (•) - Kaba
- (•) - Verimsiz
- (•) - Çok uzun
- (•) - Öz
- (•) - Hiçbiri

Cevap-9 :

Öz

6. DİZİLER

Giriş

Dizi, aynı türden verilerin oluşturduğu veri yapısı olarak tanımlanabilir (Deitel et al., 2014). Diğer yandan dizi birden fazla elemanı içinde bulunduran temel veri yapısı olarak da adlandırılabilir (Skiena, 2020). Dizi içerisinde her bir eleman bir indeks/indis veya başka bir deyişle adresi ile birlikte ifade edilir (Skiena, 2020). Yani farklı değişkenleri başka bir deyişle elemanları barındıran bir veri yapısıdır (Deitel et al., 2014). Dizide her bir verinin belli bir adresi vardır ve bu adresten değer çağrırlabilir. Bir dizideki elemanlar parantez veya köşeli parantez kullanılarak ile ifade edilebilir. Bu durum programlama dilinden diline farklılık gösterir. Aynı şekilde indislerin başlangıç değerleri de dilden dile farklılaşmaktadır. Bazı dillerde indis değerleri 0'dan başlarken bazlarında 1'den başlar. C# ve VB dillerinde indisler 0'dan başlar. Örneklerde de bu sebeple ilk indis numarası 0 kabul edilmiştir.

Burada dizilere örnek olarak bir sınıfındaki öğrenciler verilebilir. Öğrencilerin bir veri kümesi oluşturduğu düşünüldüğünde, bu veri kümesindeki her öğrencinin adresi öğrenci numarası olacaktır. Yani burada indis yerine geçen değer öğrenci numarasıdır. Bu durumda her öğrenci numarası bir öğrenciyi işaret eder.

6.1. Tek Boyutlu Diziler

Birden fazla elemanı her bir indise bir tanesi denk düşecek şekilde barındıran değişken tipidir. Aşağıdaki dizinin yapısına bakıldığında, dizinin aynı türde yani sayısal değer içeren 7 elemandanoluştugu görülmektedir. Dizinin elemanlarını çağırmak için girişte de bahsedildiği gibi indis değerleri kullanılabilir. Örneğin, ikinci indiste ($i(2)$) 15 elemanı bulunmaktadır. İkinciörnekte ise *string* değerler barındıran 4 elemanlı başka bir dizi mevcuttur. Dizinin ilk elemanı ($i(0)$)'da tutulan "Matematik" değeridir. Aşağıdaki verilen basit örnekler tek boyutlu dizilere örnek oluşturur. Tek boyutlu dizilerde zaman karmaşıklığı $O(n)$ ile ifade edilir. Çünkü diziyi oluşturmak için tek bir *for* döngüsü kullanmamız yeterli olur.

$i(0)$	$i(1)$	$i(2)$	$i(3)$	$i(4)$	$i(5)$	$i(6)$
9	20	15	38	25	30	4

$i(0)$	$i(1)$	$i(2)$	$i(3)$
Matematik	Programlama	Algoritma	Veritabanı

Tek boyutlu dizi yapıları VB dilinin yapısına aşağıdaki gibi uyarlanabilir:

- Dim [dizi ismi] (4) As Integer → Dizi oluşturma

[dizi ismi](0) = değer → Dizi elemanlarını atama
 [dizi ismi](1) = değer
 [dizi ismi](2) = değer
 [dizi ismi](3) = değer

- Dim [dizi ismi] (4) {değer, değer, değer, değer} → Dizi oluşturma

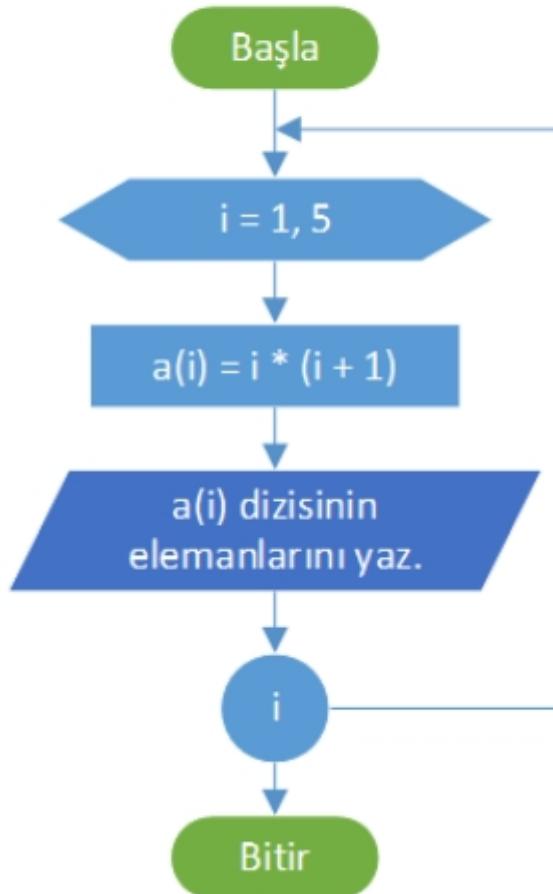
Öğrenci notlarından oluşan bir dizi oluşturmak için aşağıdaki yapı kullanılabılır.	Öğrenci notlarından oluşan diziden eleman çağrılmak için aşağıdaki işlem gerçekleştirilebilir.
Dim öğrenciler = New Integer (4) {40, 78, 34, 91}	öğrenciler(1)

Tek boyutlu dizilerle yapılan işlemlere farklı örnekler verilebilir:

a) Örnek olarak verilen dizinin oluşturulması için gereken program aşağıdaki gibidir.

indis	0	1	2	3	4	5
a	0	2	6	12	20	30

BAŞLA
FOR i = 0 TO 5
a(i) = i * (i+1)
YAZ (a(i))
NEXT i
BİTİR



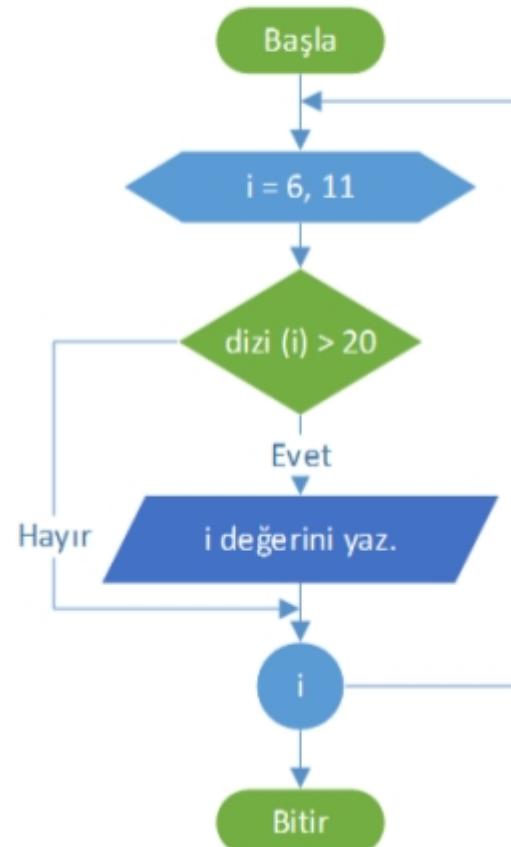
b) Şubeler bazında ziyaretçi sayısını tutan diziye göre 20'den fazla ziyaretçi olan şubelerin indis değerlerini yazdırın program örneği aşağıda verilmiştir.

i	6	7	8	9	10	11
Ziyaretçi sayısı	30	18	22	40	21	18

```

BAŞLA
FOR i = 6 TO 11
    IF (dizi(i) > 20) THEN
        YAZ (i)
    END IF
NEXT i
BİTİR
Çıktı:
6
8
9
10

```



- c) Şubelere göre ziyaretçi sayısını tutan diziye göre 20'den fazla ziyaretçi olan şubelerdeki toplam ziyaretçi sayısını yazdırın algoritma aşağıda verilmiştir

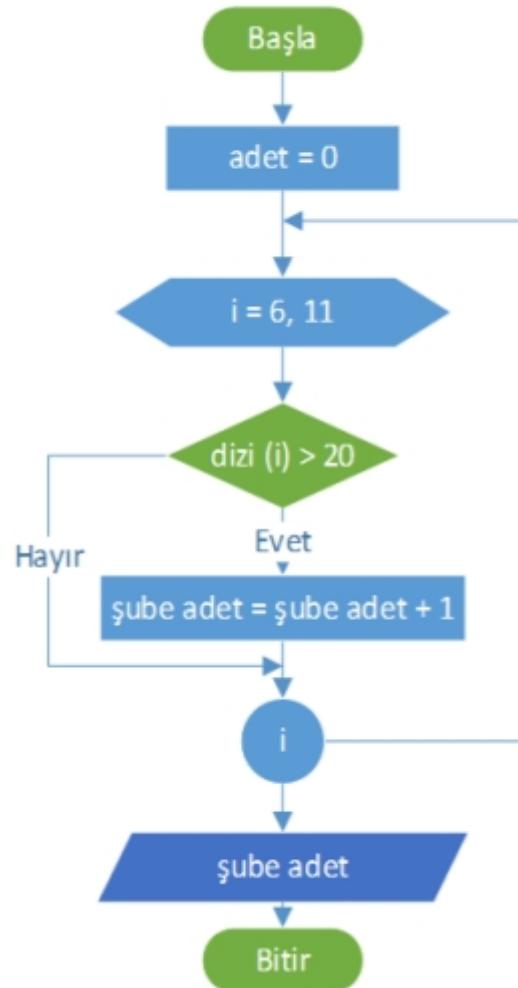
BAŞLA
toplam = 0
FOR i = 6 TO 11
IF (dizi(i) > 20) THEN
toplam = toplam + dizi(i)
END IF
NEXT i
YAZ (toplam)
BİTİR
Çıktı:
113
<hr/> $30+22+40+21 = 113$

- d) Yukarıdaki örnekteki aynı diziye göre 20'den fazla ziyaretçi olan kaç şube olduğunu yazdırın program aşağıda verilmiştir.

```

BAŞLA
şube adet = 0
FOR i = 6 TO 11
    IF (dizi(i) > 20) THEN
        şube adet = şube adet + 1
    END IF
NEXT i
YAZ (şube adet)
BİTİR
Çıktı:
3

```

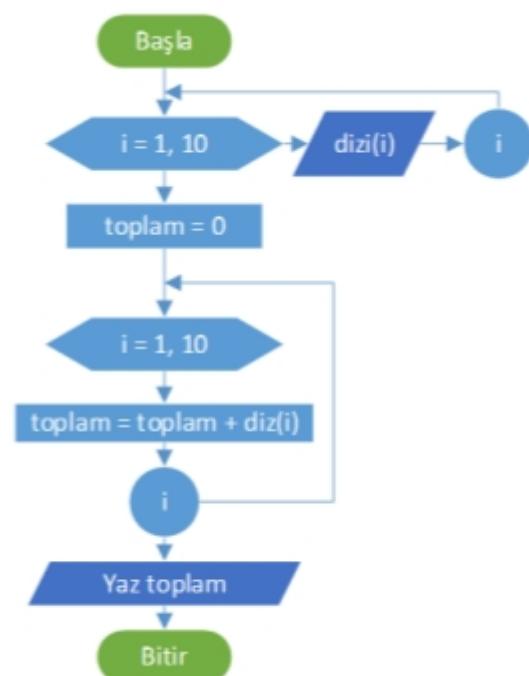


e) 10 tane eleman bulunduran bir dizinin tüm elemanlarını toplayan program aşağıda verilmiştir.

```

BAŞLA
FOR i = 1 TO 10
    YAZ (dizi(i))
NEXT i
toplam = 0
FOR i = 1 TO 10
    toplam = toplam + dizi(i)
NEXT i
YAZ (toplam)
BİTİR
Çıktı:
indis | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
değer | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
55

```



f) Bir mağazadaki farklı ürünlerden fiyatı 15 TL ve altında olanların fiyatı 10 TL olarak güncellenmek istenmektedir. Bu işlemi gerçekleştiren program aşağıda verilmiştir.

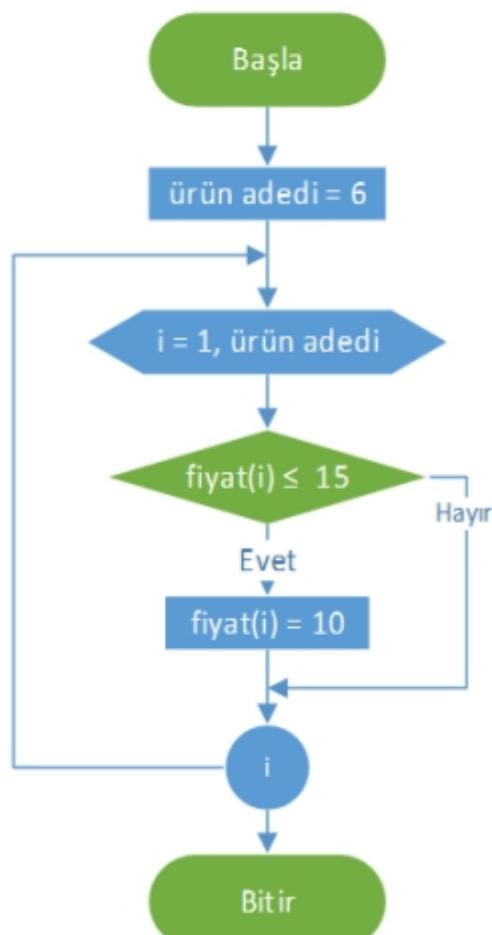
indis	1	2	3	4	5	6
fiyat	30	5	10	45	7	15

```

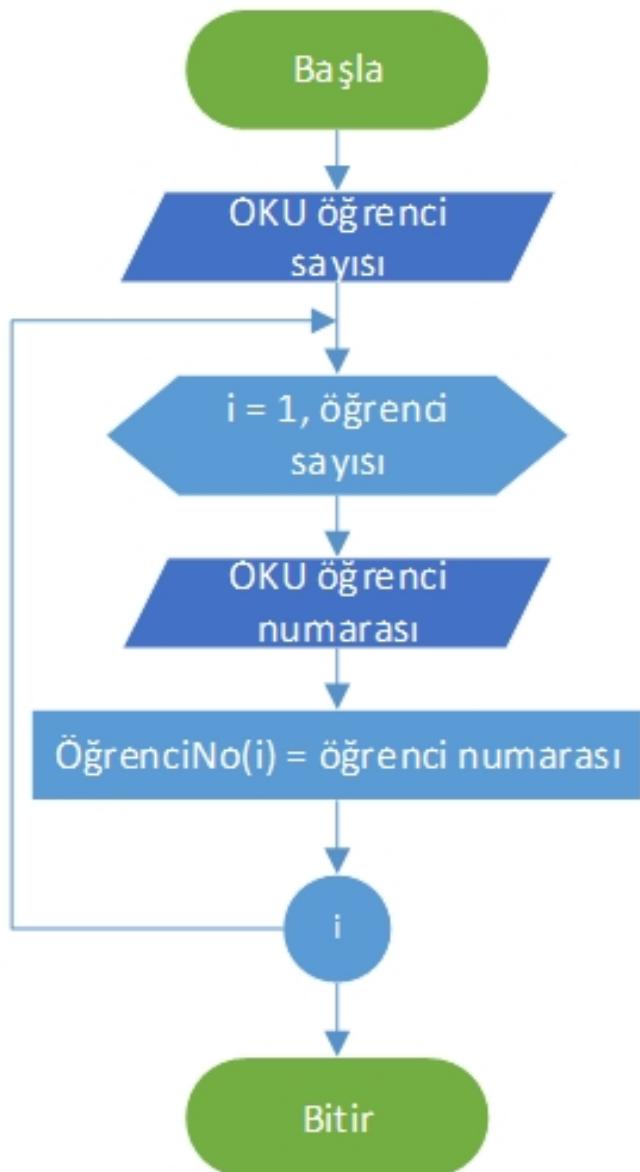
BAŞLA
ürün adedi = 6
FOR i = 1 TO ürün adedi
    IF fiyat(i) ≤ 15
        fiyat(i) = 10
    END IF
NEXT i
BITİR
Güncel fiyat listesi:

indis 1 2 3 4 5 6
fiyat 30 10 10 45 10 10

```



- g) Bir okuldaki öğrencilerin numaraları kullanıcıdan alınarak bir dizi şeklinde saklanmak istenmektedir. Bu uygulamayı yapan algoritma aşağıda verilmiştir.



BAŞLA								
OKU öğrenci sayısı								
FOR i = 1 TO öğrenci sayısı								
OKU öğrenci numarası								
ÖğrenciNo(i) = öğrenci numarası								
NEXT i								
BİTİR								
Güncel dizi değerleri:								
<table border="1"><tr><td>indis</td><td>1</td><td>2</td><td>3</td></tr><tr><td>ÖğrenciNo</td><td>135809017</td><td>135809015</td><td>135809019</td></tr></table>	indis	1	2	3	ÖğrenciNo	135809017	135809015	135809019
indis	1	2	3					
ÖğrenciNo	135809017	135809015	135809019					
Öğrenci numarası: 135809017 Öğrenci numarası: 135809015 Öğrenci numarası: 135809019 Yukarıdaki numaralar girilmiştir.								

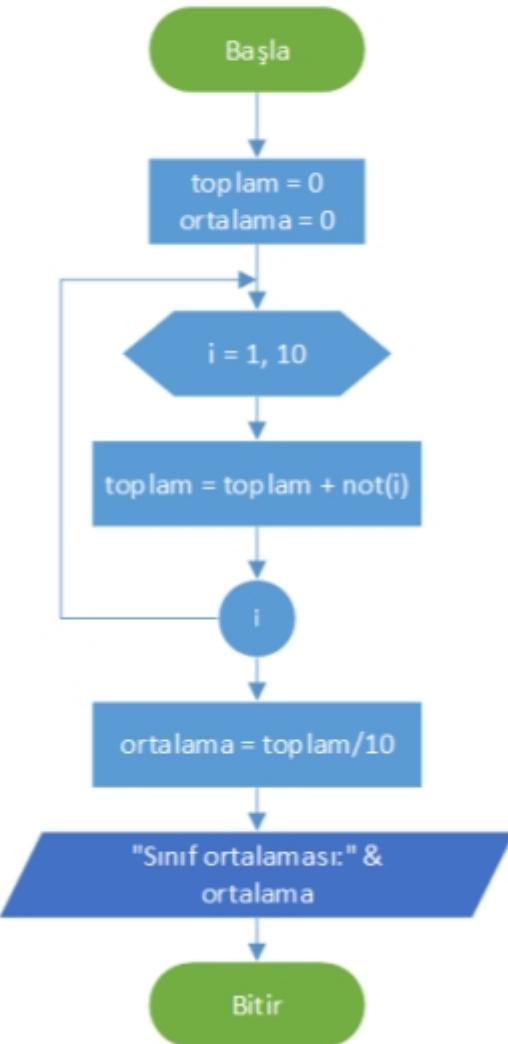
h) Notları bir dizi şeklinde verilen 10 kişilik bir sınıfın not ortalamasını hesaplayan algoritma aşağıda verilmiştir.

öğrenci	1	2	3	4	5	6	7	8	9	10
not	30	100	90	45	70	80	65	55	40	20

```

BAŞLA
toplam = 0
ortalama = 0
FOR i = 1 TO 10
    toplam = toplam + not(i)
NEXT i
ortalama = toplam / 10
YAZ ("Sınıf ortalaması:" & ortalama)
BİTİR
Çıktı:
Sınıf ortalaması: 59.5

```



6.2. İki Boyutlu Diziler

İki boyutlu diziler ise tek boyutlu olanlara ek olarak ikinci bir boyutu daha olan dizilerdir. Sütun ve satıldan oluşan diziler iki boyutlu dizi olarak adlandırılır (Çobanoğlu, 2014). Bu dizilere aynı zamanda matris de denilmektedir (Vatansever, 2011). Başka bir deyişle birden fazla tek boyutlu dizinin bir araya gelmesi ile matris olur (Cormen et al., 2022). Bu bir tablo yapısı gibi düşünülebilir. Düzenli olarak tuttuğumuz bir öğrenci tablosunda, her bir öğrenci için öğrenci numarası, ad, soyad ve sınav notu sütunlarında ilgili verileri sakladığımızı düşünelim. Burada satırlar öğrencileri temsil eder, sütunlar ise bu öğrencilere ait farklı değişkenleri. Yani buradaki yapı bir matristir. Yine başka örnek üzerinden gidecek olursak aşağıda verilen bir A matrisi, 2 satır ve 4 sütundan oluşmaktadır. Sıklıkla, satırları adreslemek için i , sütunları adreslemek için ise j harfi kullanılır. İki boyutlu dizilerde zaman karmaşıklığı $O(n^2)$ ile ifade edilir. Satır ve sütunları oluşturmak için iki tane *for* döngüsü kullanıldığından zaman karmaşıklığı tek boyutlu dizilere göre daha büyütür.

A	j(0)	j(1)	j(2)	j(3)
i(0)	9	20	15	38
i(1)	7	25	30	4

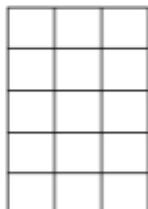
İki boyutlu dizi yapıları VB dilinin yapısına aşağıdaki gibi uyarlanabilir:

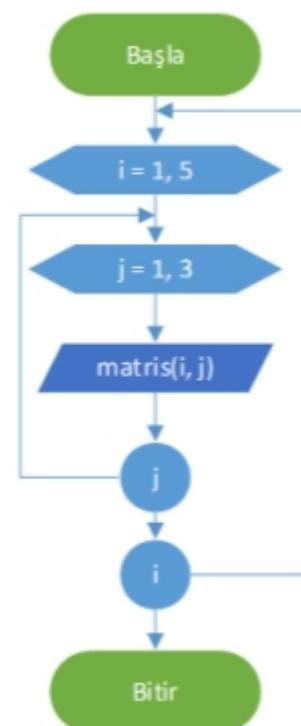
- Dim [matris ismi] (satır sayısı, sütun sayısı) As Integer → Dizi oluşturma
- Dim [matris ismi] = New Integer (satır sayısı, sütun sayısı) {{değer, değer, değer}, {değer, değer, değer}} → Dizi oluşturma

Öğrenci notlarından oluşan bir dizi oluşturmak için aşağıdaki yapı kullanılır.	Öğrenci notlarından oluşan diziden eleman çağrılmak aşağıdaki yapı kullanılır.
Dim (notlar) (4, 4) As Integer	notlar(1,3)
Dim (notlar) = New Integer (2,3) {{25, 50, 60}, {70, 80, 100}}	

İki boyutlu dizilerle yapılan işlemlere farklı örnekler verilebilir:

- i) Aşağıda 5×3 boyutunda bir matrisi yazdıran algoritma örneği aşağıda verilmiştir. Burada satırları yazdırmak için i, sütunları yazdırmak için ise j sayacı kullanılır.

BAŞLA
FOR i = 1 TO 5
FOR j = 1 TO 3
YAZ (matris(i, j))
NEXT j
NEXT i
BİTİR
Çıktı:

Tabloya matris değişkeninin içindeki değerler gelecektir.

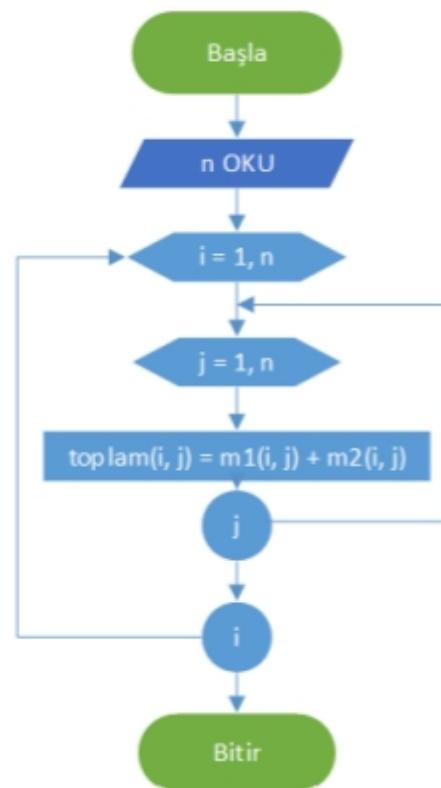


- j) İki kare matrisi ($n \times n$) toplayıp yeni bir matris oluşturan program aşağıdadır.

```

BAŞLA
OKU n
FOR i = 1 TO n
    FOR j = 1 TO n
        toplam(i, j) = m1(i, j) + m2(i, j)
    NEXT j
NEXT i
BITİR

```



k) Bir kare matrisin ($n \times n$) boyutları kullanıcı tarafından belirlenir ve matrisin tüm elemanları 1 olarak atanır. Bu işlemi gerçekleştiren program aşağıda verilmiştir.

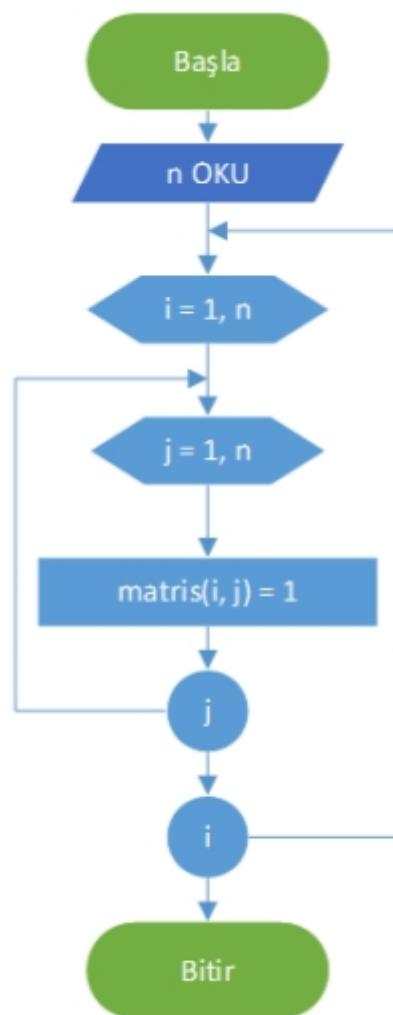
```

BAŞLA
OKU n
FOR i = 1 TO n
    FOR j = 1 TO n
        matris(i, j) = 1
    NEXT j
NEXT i
BITİR
Çıktı:

```

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

n = 5 seçilmiştir.



Bölüm Özeti

Bu bölümde tek boyutlu ve iki boyutlu dizilerden bahsedilmiş ve bunlarla ilgili uygulamalı örnekler paylaşılmıştır. Aynı tüden verilerin oluşturduğu yapılar dizi olarak adlandırılır. Diziler tek boyutlu ve iki boyutlu olabilmektedir. Dizilerde her bir verinin tutulduğu belli bir adres bulunmaktadır. Bu adres indis veya indeks olarak adlandırılır. İndis değerleri, 0'dan veya 1'den başlayabilir. Ayrıca dizi elemanlarının ifade edilmesi için parantez veya köşeli parantez kullanılabılır. Bahsettiğimiz bu iki durum, programlama dillerine göre farklılaşmaktadır.

İlk bölümde tek boyutlu dizilere yer verilmiştir. Bu dizilerde indis değerleri i değeri ile ifade edilir. Bu dizilerin zaman karmaşıklığı, dizi oluşturulurken tek bir *for* döngüsüne ihtiyaç duyulduğundan $O(n)$ olarak ifade edilir.

İkinci bölümde ise iki boyutlu dizilerden kabaca bahsedilmiştir. İki boyutlu dizilerde adından da anlaşılabilcegi gibi iki boyut vardır. Bu boyutlar i ve j harfleri ile gösterilir. Satırları ifade etmek için i , sütunları ifade etmek için ise j harfi kullanılır. Bu dizilerin zaman karmaşıklığı ise $O(n^2)$ ile gösterilir. Çünkü bu dizilerin oluşturulması için iki adet *for* döngüsü kullanmamız gereklidir.

Kaynakça

Çobanoğlu, B. (2014). Algoritma Geliştirme ve Veri Yapıları (5th ed.). Pusula Yayıncılık.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms (4th ed.). The MIT Press.

Deitel, P., Deitel, A., & Deitel, H. (2014). Visual Basic 2012 How to Program (6th ed.). Pearson.
<https://deitel.com/visual-basic-2012-how-to-program/>

Skiena, S. S. (2020). The Algorithm Design Manual (3rd ed.). Springer International Publishing.
<https://doi.org/10.1007/978-3-030-54256-6>

Vatansever, F. (2011). Algoritma Geliştirme ve Programlamaya Giriş (8th ed.). Seçkin Yayıncılık.

Ünite Soruları

Soru-1 :

Tek boyutlu dizilerde indis değerini ifade etmek için sıkılıkla hangi harf kullanılır?

(Çoktan Seçmeli)

(•) - a

(•) - i

(•) - l

(•) - z

(•) - x

Cevap-1 :

i

Soru-2 :

İki boyutlu dizilerde ikinci indis değerini ifade etmek için sıkılıkla hangi harf kullanılır?

(Çoktan Seçmeli)

(•) - a

(•) - p

(•) - l

(•) - j

(•) - x

Cevap-2 :

j

Soru-3 :

Aşağıdakilerden hangisi dizilerin indislerini ifade etmek için kullanılabilir?

(Çoktan Seçmeli)

(•) - Sadece parantez

(•) - Sadece köşeli parantez

(•) - Sadece tırnak işaretti

(•) - Parantez veya köşeli parantez

(•) - Süslü parantez

Cevap-3 :

Parantez veya köşeli parantez

Soru-4 :

Dizilerin indis numarası ile ilgili aşağıdakilerden hangisi doğrudur?

(Çoktan Seçmeli)

(•) - İndis numarası mutlaka 0'dan başlar.

(•) - İndis numarası mutlaka 1'den başlar.

(•) - İndis başlangıç numarası dilden dile farklılık gösterir.

(•) - İndis numarası x'ten başlar.

(•) - İndis numarası string tipinde bir değerdir.

Cevap-4 :

İndis başlangıç numarası dilden dile farklılık gösterir.

Soru-5 :

Dizilerin zaman karmaşıklığı ile ilgili olarak aşağıdakilerden hangisi doğrudur?

(Çoktan Seçmeli)

- (•) - Dizilerin zaman karmaşıklığı çok azdır.
- (•) - Dizilerin zaman karmaşıklığı ölçülemez.
- (•) - Tek boyutlu dizilerin zaman karmaşıklığı iki boyutlu dizilere göre daha fazladır.
- (•) - İki boyutlu dizilerin zaman karmaşıklığı tek boyutlu dizilere göre daha fazladır.
- (•) - Tek ve iki boyutlu dizilerin karmaşıklığı her zaman aynıdır.

Cevap-5 :

İki boyutlu dizilerin zaman karmaşıklığı tek boyutlu dizilere göre daha fazladır.

Soru-6 :

Diziler ile ilgili olarak aşağıdakilerden hangisi yanlıştır?

(Çoktan Seçmeli)

- (•) - Dizilerde string tipinde değerler tutulabilir.
- (•) - Dizilerde char tipinde değerler tutulabilir.
- (•) - Dizilerde sayısal değerler tutulabilir.
- (•) - Bir dizinin elemanları aynı veri tipinde olmak zorunda değildir.
- (•) - Bir dizinin eleman sayısı programcı tarafından belirlenir.

Cevap-6 :

Bir dizinin elemanları aynı veri tipinde olmak zorunda değildir.

Soru-7 :

Dizilerin indis numarası ile ilgili aşağıdakilerden hangisi doğrudur?

(Çoktan Seçmeli)

- (•) - İndis numarası mutlaka 0'dan başlar.
- (•) - İndis numarası mutlaka 1'den başlar.
- (•) - İndis başlangıç numarası dilden dile farklılık gösterir.

- (•) - İndis numarası x'ten başlar.
- (•) - İndis numarası string tipinde bir değerdir.

Cevap-7 :

İndis başlangıç numarası dilden dile farklılık gösterir.

Soru-8 :

Aşağıdakilerden hangisi iki boyutlu diziler ile ilgili yanlıştır?

(Çoktan Seçmeli)

- (•) - Birden fazla tek boyutlu dizinin bir araya gelmesi ile oluşur.
- (•) - Satır ve sütunlardan oluşur.
- (•) - İki boyut iki ayrı indis ile ifade edilir.
- (•) - Elemanları sadece sayısal tipte olabilir.
- (•) - Bir dizinin elemanları aynı veri tipinde olmak zorunda değildir.

Cevap-8 :

Elemanları sadece sayısal tipte olabilir.

Soru-9 :

İki boyutlu dizinin oluşturulması ile ilgili olarak hangisi doğrudur?

(Çoktan Seçmeli)

- (•) - İki boyut i indis ile ifade edilir.
- (•) - Seçimli yapı kullanılarak oluşturulması gereklidir.
- (•) - İki boyutlu dizide sadece sayısal değerler tutulabilir.
- (•) - Bir for döngüsü kullanılarak oluşturulması gereklidir.
- (•) - İki for döngüsü kullanılarak oluşturulabilir.

Cevap-9 :

İki for döngüsü kullanılarak oluşturulabilir.

7. ARAMA ve SIRALAMA ALGORİTMALARI

7.1. Arama Algoritmaları

Arama algoritmaları, bir listedeki veya dizideki bir elemanın yerinin bulunmasını sağlar (Çobanoğlu, 2014). Bir değerin, elimizdeki bir değer listesinde olup olmadığını belirlemek istiyorsak değeri listede aramamız gereklidir (Mailund, 2021). Bu bölümde bu soruna çözüm sağlayan doğrusal arama ve ikili arama algoritmalarından bahsedilmiştir.

7.1.1. Doğrusal Arama (Linear Search)

Doğrusal arama algoritması basit bir mantığa dayanır. Bu mantıkta aranan eleman veya hedef değer ne ise onu bulmak için dizideki tüm elemanlar döngüye alınır ve her biri sırayla aranan değer ile karşılaştırılır; istenen eleman bulunursa, hedef eleman aranan listenin içinde mevcuttur, bulunamazsa tam tersi geçerlidir yani eleman bu listede yoktur (Althoff, 2022; Mailund, 2021). Doğrusal arama, dizideki değerler hakkında hiçbir şey varsaymaz (Mailund, 2021). Aranan eleman bulunursa, o elemana ait olan indis değerini döndürür.

Arama yapılan dizi tek boyutlu olduğundan doğrusal aramanın zaman karmaşıklığı $O(n)$ ile ifade edilir (Althoff, 2022; Mailund, 2021).

Doğrusal arama için kaba kod şu şekilde ifade edilir (Çölkesen, 2017):

```
for (İlk elemandan başlayıp son elemana kadar)
```

```
if (Sıradaki eleman = aranan eleman)
```

```
return indis;
```

```
return -1;
```

Sıralanmamış liste:

i(0) i(1) i(2) i(3) i(4) i(5) i(6)
9 20 15 38 25 30 4

38 elemanı bu liste içerisinde aranıyor. Öncelikle listenin ilk elemanından başlanır.

Adım 1: $9 = 38$? Hayır, aramaya devam et.

i(0) i(1) i(2) i(3) i(4) i(5) i(6)
9 20 15 38 25 30 4

Adım 2: $20 = 38$? Hayır, aramaya devam et.

i(0) i(1) i(2) i(3) i(4) i(5) i(6)
9 20 15 38 25 30 4

Adım 3: $15 = 38$? Hayır, aramaya devam et.

i(0) i(1) i(2) i(3) i(4) i(5) i(6)
9 20 15 38 25 30 4

Adım 4: $38 = 38$? Evet, aranan eleman listede bulundu, indisini geri döndür.

i(0) i(1) i(2) i(3) i(4) i(5) i(6)

9 20 15 **38** 25 30 4

Eğer 38 değeri dizinin devamında yer almıyordu aramaya 38'i bulana kadar devam edilecekti. Eğer 38 listede yer almıyor olsaydı, listenin son elemanı da karşılaşıldığtan sonra “Bulunamadı” mesajı döndürülecekti.

7.1.2. İkili Arama (Binary Search)

İkili arama algoritması, doğrusal aramaya benzer şekilde, bir elemanın bir dizide aranmasını sağlar ancak diğerinden farklı olarak, bu algoritmada taranan dizinin sıralanmış olması gereklidir (Althoff, 2022). Dizide istenen değerin aranabilmesi için dizinin sıralı olması bir ön şarttır diyebiliriz. Bu algoritmada; taranan dizi ikiye ayrılır, ortadaki değer kontrol edilir, bu değerin aranan değerden küçük ya da büyük olması durumunda dizinin ilgili tarafında yeni bir ikili arama yapılır (Althoff, 2022). Dizi zaten sıralı olduğu için, eğer aranan değer ortaki değerden küçükse, ortadaki değerin sağında kalan elemanlar arasında arama yapmaya gerek kalmaz. Aynı şey tersi için de geçerlidir. Aradan değer ortadaki değerden büyükse bu sefer de ortadaki değerden önce gelen yani dizinin sol tarafındaki elemanlar arasında arama yapmaya gerek kalmaz. Dizideki tüm elemanların kontrol edilmesi gerekmeli için doğrusal aramadan daha verimli çalışan bir algoritmadır (Althoff, 2022). Yani ortadaki değer referans alınarak, arama işlemi daha ufak bir dizide yapılmış olur, böylelikle tüm elemanlar taranmaz. İkili aramanın zaman karmaşıklığı $O(\log n)$ ile ifade edilir (Althoff, 2022; Mailund, 2021).

İkili arama için kaba kod şu şekilde ifade edilir (Çölkesen, 2017):

sol = dizinin ilk elemanı, sağ = dizinin son elemanı

while (sol <= sağ olduğu sürece)

if(aranan eleman dizinin ortadaki elemanı ise)

bulundu

else if (aranan eleman dizinin ortadaki elemanından büyük ise)

aranan eleman dizinin sağ tarafında; sol = orta + 1 olmalı

else

aranan eleman dizinin sol tarafında; sağ = orta - 1 olmalı

Aşağıdaki listeden bir eleman ikili arama algoritması ile belirtilen adımlar takip edilerek aranmıştır:

Sıralanmış liste:

i(0) i(1) i(2) i(3) i(4) i(5) i(6)

4 9 15 20 25 30 38

Adım 1: 38 elemanı bu liste içerisinde aranıyor. Öncelikle listenin ortasındaki eleman belirlenir. Mevcut listede tek sayıda eleman var, ortadaki elemanı bulmak için dizinin eleman sayısı ikiye bölünüür ($7/2=3,5$) veya en küçük ve en büyük indis değerleri toplanır ikiye bölünüür ($((0+6)/2=3)$, elde edilen değer ondalıklı sayı ise değer aşağı yuvarlanır. Bu bize hangi indisteki elemanın ortanca değer olduğunu verir. Bu örnekte ortadaki eleman i(3) pozisyonunda bulunan 20 değeridir. Ortadaki değere göre listenin iki parçadanoluştugu varsayılabılır.

i(0) i(1) i(2) **i(3)** i(4) i(5) i(6)

4 9 15 **20** 25 30 38

Adım 2: $20 = 38$? Hayır, yani 20 aranan değer olan 38'e eşit değil, aramaya devam edilir.

Adım 3: Ortadaki değere göre aranan değer olan 38'in listenin hangi tarafında olabileceği incelenir. 38 değeri 20'den büyük olduğu için listenin 20'nin sağındaki kısmı ile arama işlemeye devam edilir. Yani dizinin 20'den küçük olan kısmına bundan sonraki adımlarda ihtiyaç kalmaz.

i(4) i(5) i(6)

25 30 38

Adım 4: Dizinin geri kalan tarafında yine ortadaki değer belirlenir. $30 = 38$? Hayır, yani ortadaki değer yine 38'e eşit değil, aramaya devam edilir.

i(4) i(5) i(6)

25 30 38

Adım 5: Adım 3'teki işlem tekrarlanır. Yani aranan değer ile ortadaki değer karşılaştırılır. 38 değeri 30'dan büyük olduğu için dizinin ortadaki değerin sağındaki kısmı ile devam edilir. Geriye kalan tek elemanlı alt dizide aranan değer olan 38 bulunmuş olur.

i(6)

38

Aranan sayı eğer listede değilse bu adımdan sonra listenin sol değeri sağ değerinden büyük olacağından arama sonuçlanır ve aranan değerin bulunamadığı geri döndürülür.

7.2. Sıralama Algoritmaları

Sıralama algoritmaları, bir grup verinin artan veya azalan bir şekilde sıralanmasını sağlayan algoritmalarıdır (Çobanoğlu, 2014). Bu bölümde sıralama algoritmalarından bazıları detaylı olarak ele alınmıştır.

7.2.1. Karşılaştırma Temelli Algoritmalar

Sıralama yaparken bazı algoritmalar karşılaştırma temelli olarak çalışırlar. Burada kastedilen, elemanların tek tek birbiriyle karşılaştırılması ve buna göre yerlerinin değiştirilerek istenen dizinin sıralanmasıdır. Bu, bazı algoritmaların çalışma mantığını oluşturmaktadır. Bu bölümde, bu mantığı temel alarak çalışan, kabarcık sıralama ve yerleştirmeli sıralama algoritmalarından bahsedilecektir.

7.2.1.1 Kabarcık Sıralaması (Bubble Sort)

Kabarcık sıralama ya da İngilizce adıyla “bubble sort” algoritması, bir sayı listesinden ilerleyerek, her seferinde bir sayının kendisinden sonra gelen diğer sayıyla karşılaştırıldığı ve eğer doğru sırada değilse bu sayıların yerlerinin değiştirildiği bir sıralama algoritmasıdır (Althoff, 2022). Bilgisayar bilimlerinde bu algoritmanın ismi, listedeki en büyük sayıların listenin en sonunda ortaya çıkması veya belirmesini ifade etmesinden gelmektedir (*bubble up*) (Althoff, 2022).

Özetlemek gerekirse, dizinin her bir elemanı birbiriyle tek tek karşılaştırılır ve sıraya dizilir. Bunu yaparken listenin birinci elemanından $a(0)$ başlanır ve bu eleman hemen yanındaki yani ikinci sıradaki dizi elemanı $a(1)$ ile karşılaştırılır. Eğer ilk elemanın değeri ikinci elemandan küçükse ikinci $a(1)$ ve üçüncü $a(2)$ elemanların karşılaştırılmasına geçilir. Eğer ilk eleman ikinci elemandan büyükse, birinci ve ikinci elemanlar yer değiştirir. Ardından ikinci ve üçüncü eleman karşılaştırılır. Bu işlemler listenin sonuna kadar devam ettirilir. En büyük eleman listenin en sonuna yerleştikten sonra geri kalan elemanlar için bu işlemler tekrarlanır. Kabarcık sıralamasının zaman karmaşıklığı $O(n^2)$ ile ifade edilir (Althoff, 2022).

Kabarcık sıralama için kaba kod şu şekilde ifade edilir (Çölkesen, 2017):

for (İlk elemandan başlayıp sondan bir önceki elemana kadar)

for (İlk elemandan o andaki elemana kadar)

if (O andaki eleman bir sonraki elemandan büyükse)

O andaki ve bir sonraki elemanları yer değiştir.

Aşağıdaki listede bulunan elemanlar küçükten büyüğe doğru kabarcık algoritması ile belirtilen adımlar takip edilerek sıralanmıştır:

Sıralanmamış liste:

i(0) i(1) i(2) i(3)

25 4 15 9

Adım 1: $25 > 4$? Evet, yerlerini değiştir

i(0) i(1) i(2) i(3)

4 25 15 9

Adım 2: $25 > 15$? Evet, yerlerini değiştir.

i(0) i(1) i(2) i(3)

4 15 25 9

Adım 3: $25 > 9$? Evet, yerlerini değiştir.

i(0) i(1) i(2) i(3)

4 15 9 25

Böylelikle, listenin en büyük değere sahip elemanı olan 25 listenin en sonuna yerleşmiş oldu. Ancak listenin tamamı hala doğru şekilde sıralanmıyor. Bunun sağlanması için en başa geri dönüp karşılaştırma ve sıralama işlemlerinin tekrarlanması gerekiyor.

i(0) i(1) i(2) i(3)

4 15 9 25

Adım 4: $4 > 15$? Hayır, bir şey yapma. İkinci ve üçüncü elemaları karşılaştırmaya geç.

i(0) i(1) i(2) i(3)

4 15 9 25

Adım 5: $15 > 9$? Evet, yerlerini değiştir.

i(0) i(1) i(2) i(3)

4 9 15 25

Her ne kadar liste doğru sıralanmış olsa da bilgisayar için ilk iki sayının doğru sıralanıp sıralanmadığı henüz belli değildir.

Adım 6: $4 > 9$? Hayır, bir şey yapma.

i(0) i(1) i(2) i(3)

4 9 15 25

Bu adımla beraber listedeki elemanlar küçükten büyüğe sıralanmış oldu. Listedeki en büyük elemanın 25 olduğu 4. adımda belli olmuþtu. Bu adımdan sonra diğer elemanlar kendi aralarında aynı işlem adımları takip edilerek tekrardan karşılaştırıldı ve küçükten büyüğe dizilmiş oldu.

Sıralanmış liste:

i(0) i(1) i(2) i(3)
4 9 15 25

7.2.2.2. Yerleştirmeli Sıralama (Insertion Sort)

Yerleştirmeli sıralama algoritması, sıralanmamış bir listede, elemanların seçilerek sıralanmış listede doğru yere eklenmesini sağlar, bunu yaparken de aslında sıralanacak listenin veya dizinin, o anki mevcut işlem adımına kadar sıralanmış ve sıralanmamış olmak üzere iki parçadanoluþtuðu düşünür (Çobanoðlu, 2014; Erwig, 2017). Burada şekilde görüldüğü gibi oyun kartları üzerinden algoritmanın çalışma mantığı sembolize edilebilir (Cormen et al., 2022). Sağ elde tutulan kartlar sırayla sol eldeki uygun yere geçirilmektedir. Sağ elde sırası gelen kartın değeri sol eldeki en saðdaki kartla karşılaştırılır, sağ taraftaki elemanın değeri sol eldekinden büyük ise, sol elde karşılaştırılan elemanın yanına yerleştirilir. Eğer küçükse sol eldeki karşılaştırılan elemanın solundakiyle karşılaştırma işlemi tekrarlanır. Bu işlem bu şekilde devam eder ve en sonunda sağ elde kağıt kalmaz, tüm kağıtlar sol ele geçer. Yerleştirmeli sıralamanın zaman karmaþıklığı, eldeki dizi sıralıya yakın veya sıralı bir yapıdaysa $O(n)$, tamamen sırasız bir yapıdaysa $O(n^2)$ olur (Althoff, 2022).



Figure 2.1 Sorting a hand of cards using insertion sort.

(Cormen et al., 2022)

Yerleştirmeli sıralama için kaba kod şu şekilde ifade edilir (Çölkesen, 2017):

for (İkinci elemandan başlayıp son elemana kadar)

O andaki elemani *arayaEkle* adlı değişkene al.

for (Bir önceki elemandan ilk elemana kadar, *arayaEkle* daha küçükse)

Dizi elemanını yer açmak için sağa kaydır.

Açılan yere *arayaEkle* 'yi yerleştir.

Aşağıdaki listede bulunan elemanlar küçükten büyüğe doğru, araya yerleştirme algoritması ile belirtilen adımlar takip edilerek sıralanmıştır:

Sıralanmamış liste:

i(0) i(1) i(2) i(3)
15 9 25 4

Burada karşılaştırmaya dizinin ikinci elemanından yani $i(1)$ başlanacaktır. Birinci eleman ve onun solundaki değerlerin oluşturduğu dizinin sıralanması yapılacaktır. Ardından dizinin sağındaki elemanların sırasıyla soldaki sıralanmış bölümdeki değerlerle karşılaştırılarak uygun pozisyonlara getirilmesi sağlanacaktır.

Adım 1: $9 > 15$? Hayır, yerlerini değiştir.

$i(0) i(1) i(2) i(3)$

9 15 25 4

Listenin sol tarafı yani (9 ve 15) sıralandı ancak sağ tarafı sıralanmamış durumda. Üçüncü elemana geçildi.

Adım 2: $25 > 15$? Evet, bir şey yapma.

$i(0) i(1) i(2) i(3)$

9 15 25 4

Listenin sol tarafı yani $i(0)$ ile $i(2)$ arasındaki elemanlar küçükten büyüğe doğru sıralanmış durumda. Her biri tek tek karşılaştırıldı ve her adımda sıralı bir alt dizi üzerinden gidildiği için en son yerleşen 25 değeri ile 9 değerlerini karşılaştırmaya gerek yok. Bu aşamaya kadar zaten dizide tek tek tüm elemanların karşılaştırılarak sıralanması üzerine kuruldu.

Adım 3: $4 > 25$? Hayır, yerlerini değiştir. Bu aşamada $i(3)$ 'te bulunan 4 elemanı, kendi solundaki sıralanmış olan listede nereye yerleşmesi gerekiyorsa, sola doğru tek tek tüm elemanlar ile bu değerin karşılaştırılması yapılır. Bu karşılaştırma 4 doğru sıraya yerlesene kadar devam eder.

$i(0) i(1) i(2) i(3)$

9 15 4 25

Adım 4: $4 > 15$? Hayır, yerlerini değiştir.

$i(0) i(1) i(2) i(3)$

9 4 15 25

Adım 4: $4 > 9$? Hayır, yerlerini değiştir.

$i(0) i(1) i(2) i(3)$

4 9 15 25

Sıralanmış liste:

$i(0) i(1) i(2) i(3)$

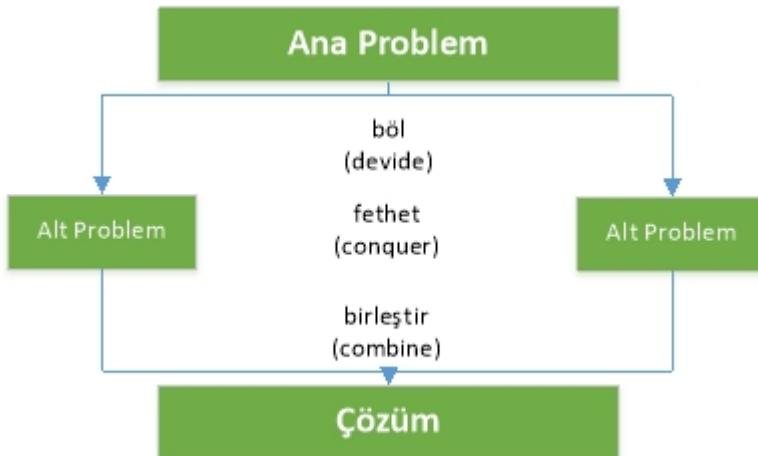
4 9 15 25

7.2.3. Böl ve Fethet Temelli Algoritmalar

İngilizce “divide and conquer” kavramının karşılığı olan “böl ve fethet” veya “böl ve yönet” yöntemi, bazı algoritmaların çalışma mantığını oluşturmaktadır. Bu bölümde, bu mantığı temel alarak çalışan, hızlı sıralama ve birleştirici sıralama algoritmalarından bahsedilecektir. Böl ve fethet mantığında, mevcut sorun, kendisine benzeyen ancak boyut olarak daha küçük olan birkaç alt soruna bölünür. Alt sorunlar (5.4. Özyineli (Recursive) Metodlar bölümünde bahsedilen) özyineli (*recursive*) olarak çözülür ve ardından bu çözümler esas büyük probleme bir çözüm oluşturmak için birleştirilir (Cormen et al., 2022). Yani özetlemek gerekirse (Cormen et al., 2022; Mailund, 2021):

- Bir algoritma ile çözülmek istenen problem, aynı türden alt problemlere ayrılır. (böl - *divide*)
- Alt problemler özyineli olarak çözülür. (fethet - *conquer*)

- Özyineli olarak çözülen problemlerin sonuçları bir araya getirilir ve böylece sonuç olarak esas problem çözülmüş olur. (bir araya getir - *combine*)



7.2.3.1 Hızlı Sıralama (Quick Sort)

Hızlı sıralama algoritması, özyineli algoritmaların biri olup böl ve fethet mantığı olan bir çalışma prensibine dayanır (Mailund, 2021). Bu algoritmada, mevcut dizi, bir pivot eleman seçildikten sonra, pivottan küçük olan elemanlar ve pivota eşit veya pivottan büyük onlar olmak üzere iki alt kümeye yani iki alt diziyeye ayrılır (Skiena, 2020). Dizi, pivottan büyük veya eşit, pivottan küçük veya eşit olmak üzere ikiye ayrılır ve sıralama bu diziler içerisinde yapılır (Cormen et al., 2022). Her iki taraf da zaten kendi içinde sıralanmış olduğundan en son durumda iki grubu birleştirmeye gerek kalmaz, tüm dizi sıralanmış olur (Cormen et al., 2022; Skiena, 2020). Hızlı sıralamanın zaman karmaşıklığı $O(n \log n)$ ile ifade edilir (Çölkesen, 2017).

Hızlı sıralama için kaba kod şu şekilde ifade edilir (Çölkesen, 2017):

A sıralanacak dizi olmak üzere:

HızlıSıralama (A, sol, sağ) {

if (sol < sağ) {

A'yı sınır değerine göre ikiye böl ve sınır değeri indirini q'da tut;

HızlıSıralama (A, sol, q);

HızlıSıralama (A, q+1, sağ);

}

}

Aşağıdaki listede bulunan elemanlar küçükten büyüğe doğru hızlı sıralama algoritması ile belirtilen adımlar takip edilerek sıralanmıştır:

Sıralanmamış liste:

i(0) i(1) i(2) i(3)
9 15 4 25

Adım 1: Listedeki pivot olarak ilk eleman seçildi. Bu elemanı listenin sağdan son elemanı ile karşılaştır.

P < sağ ?: 9 < 25 ? Evet, bir şey yapma.

i(0) i(1) i(2) i(3)
 9 25
 15 4
pivot **sağ**

Adım 2: Pivot elemanla karşılaştırmak için sağdaki indis bir azaltıldı ve o indiste bulunan eleman ile pivotun değeri karşılaştırıldı.

P < sağ ?: 9 < 4 ? Hayır, yer değişti.

i(0) i(1) i(2) i(3)
 9 4
 15 25
pivot **sağ**

Adım 3: Pivot elemanla karşılaştırmak için soldaki indis bir artırıldı ve o indiste bulunan eleman ile pivotun değeri karşılaştırıldı.

P < sol ?: 9 < 15 ? Evet, yer değişti.

i(0) i(1) i(2) i(3)
 4 9
 15 25
sol **pivot**

Adım 4: Pivot elemanla karşılaştırmak için soldaki indis bir arttırıldı ve o indiste bulunan eleman ile pivotun değeri karşılaştırıldı.

P < sol ?: 9 < 15 ? Evet, yer değişti.

i(0) i(1) i(2) i(3)
 9
 4 15 25
pivot

Adım 5: Listenin sol tarafında sıralayacak eleman kalmadığı için eski pivotun sol tarafındaki kısım sıralanacak. Sol tarafta yeni pivot eleman olarak sondaki eleman seçildi, listenin solundaki eleman ile karşılaştırıldı.

P < sol ?: 15 < 25 ? Evet, yer değişti.

i(0) i(1) i(2) i(3)
 15 25
 4 9
sol pivot

Sıralanmış liste:

i(0) i(1) i(2) i(3)
 4 9 15 25

7.2.3.2. Birleşirmeli Sıralama (Merge Sort)

Birleşirmeli sıralama algoritması, özyineli algoritmaların biri olup büyük problemleri daha küçük problemlere indirger (Skiena, 2020). Böl ve fethet mantığına dayanan bir çalışma prensibi vardır (Althoff, 2022). Sıralanmak istenen dizideki elemanları ikiye bölgerek, sıralama problemi alt problemlere yani alt

dizilere bölünmüş olur. Bunu yaparken algoritma, her bir alt dizide sadece tek bir eleman kalıncaya kadar bölmeye işlemi devam eder sonrasında ise her bir alt dizideki elemanlar karşılaştırılarak kombinlenir (Cajic, 2019). Tüm birleştirme işlemlerinin sonunda sıralanmış bir dizi elde edilir. Yani sıralama probleminin çözülmesi için, her bir eleman tek başına bir alt dizi oluşturma kadar bölmeye yapılır. Sonrasında tek tek elemanlar karşılaştırılır ardından bir araya gelen ikili/üçlü gruplar arasında karşılaştırma, sıralama ve birleştirme işlemi yapılır. Temel çalışma prensibi aşağıdaki görselde aktarılmıştır (Cajic, 2019). Birleştirilmeli sıralamanın zaman karmaşıklığı $O(n \log n)$ ile ifade edilir (Althoff, 2022).

Birleştirilmeli sıralama için kaba kod şu şekilde ifade edilir (Çölkesen, 2017):

```
birleştirilmeliSıralama (A[], sol, sağ) {
```

```
    if (sol < sağ) {
```

```
        orta = (sol + sağ) / 2
```

```
        birleştirilmeliSıralama (A, sol, sağ);
```

```
        birleştirilmeliSıralama (A, orta+1, sağ);
```

```
        birleştir (A, sol, sağ);
```

```
}
```

```
}
```

Aşağıdaki listede bulunan elemanlar küçükten büyüğe doğru birleştirilmeli sıralama algoritması ile belirtilen adımlar takip edilerek sıralanmıştır:

Sıralanmamış liste:

i(0) i(1) i(2) i(3)
9 15 4 25

Adım 1: Listeyi her birinde tek bir eleman kalan kadar alt kümelere böl.

i(0) i(1) i(2) i(3)
9 15 4 25
i(0) i(1) i(2) i(3)
9 15 4 25
i(0) i(1) i(2) i(3)
9 15 4 25

Adım 2: Öncelikle elemanlar ikili olarak karşılaştırıp gruplar oluşturulur. İkili grupların oluşturulması aynı zamanda bu alt kümelerin aşama aşama kendi içlerinde sıralanmasını sağlar.

i(0) i(1) i(2) i(3)
9 15 - 4 25

$9 > 15$? Hayır. $4 > 25$? Hayır.

i(0) i(1) i(2) i(3)
9 15 4 25

Adım 3: Bu aşamada elde edilen iki alt kümedeki birinci elemanlar karşılaştırılır ve değeri küçük olan eleman alttaki listeye alınır.

i(0) i(1) i(2) i(3)
9 15 4 25

$9 > 4$? Evet, 4'ü listeye ekle.

i(0) i(1) i(2) i(3)
4 - - -

Adım 4: Bu aşamada ilk eleman ile geri diğer alt kümedeki eleman karşılaştırılır. Değeri küçük olan listeye alınır.

i(0) i(1) i(2) i(3)
9 15 4 **25**

$9 > 25$? Hayır, 9'u listeye ekle.

i(0) i(1) i(2) i(3)
4 **9** - -

Adım 5: Bu aşamada elde edilen iki alt kümedeki birinci elemanlar karşılaştırılır ve eğer gerekiyorsa elemanların yerleri değiştirilir.

i(0) i(1) i(2) i(3)
9 **15** 4 25

$15 > 25$? Hayır, 15'i listeye ekle.

i(0) i(1) i(2) i(3)
4 9 **15** -

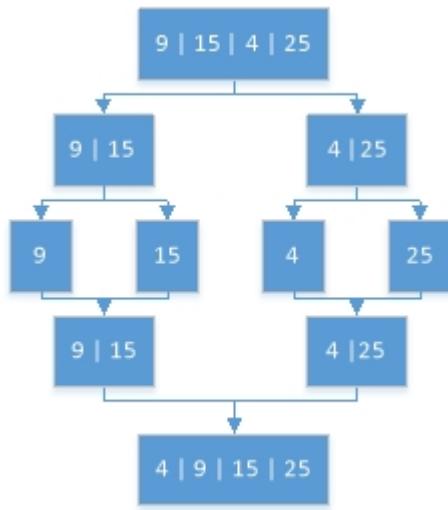
Adım 6: Son adımda da en sona kalan (en büyük değere sahip eleman) listeye alınır ve liste böylece sıralanmış olur.

i(0) i(1) i(2) i(3)
9 15 4 **25**

Sıralanmış liste:

i(0) i(1) i(2) i(3)
4 9 15 25

Görsel olarak ifade edilirse öncelikle dizinin bölündüğü sonra da bölünen parçaların bir araya getirildiği rahatlıkla görülecektir:



7.3. Algoritmaların Karşılaştırılması

Arama ve sıralama algoritmalarının seçiminde performanslarının karşılaştırılması mantıklı olacaktır. Ufak problem çözümlerinde algoritmanın performansının önemi göz ardı edilebilir ancak veri boyutunun büyümesi ile birlikte algoritmanın performansı da büyük ölçüde etkilenecektir. Bu noktada karşımıza yine büyük O notasyonu çıkar. Bu değer ile ifade edilen büyüklüklerin karşılaştırılması algoritma seçiminde programcıya yardımcı olacaktır. Problemin büyüklüğüne bağlı olarak algoritmanın ne kadar karmaşıklaşacağı programcı tarafından bilindiğinde, tercih de buna bağlı olarak değişecektir. Kitapta yer verilen arama ve sıralama algoritmaları ile beraber kitapta bahsedilmeyen birkaç algoritmanın karşılaştırılması aşağıdaki tabloda (Tablo 10) verilmiştir. Bunlar en hızından en yavaşa doğru sıralanmıştır.

Tablo 10: Algoritmaların Karşılaştırılması

En hızlı Büyük O Gösterimi Algoritmalar

$O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$ En yavaş	İkili arama Doğrusal arama Kova sıralaması (<i>bucket sort</i>) Basamağa göre sıralama (<i>radix sort</i>) Hızlı sıralama Birleştirimeli sıralama Yığın sıralaması (<i>heap sort</i>) Kabarcık sıralama Yerleşitirmeli sıralama Seçmeli sıralama (<i>selection sort</i>) Kabuk sıralaması (<i>shell sort</i>)
---	---

Bölüm Özeti

Bu bölümde arama ve sıralama algoritmaları ele alınmıştır. Bu algoritma aileleri ve bunların altında yer alan farklı algoritmalar aktarılmıştır.

İlk bölümde yer alan arama algoritmaları aktarılmış ve arama algoritmalarından doğrusal arama ve ikili arama algoritmaları detaylandırılmıştır. Doğrusal arama, aranan elemanın istenen dizide bulunması için tüm

elemanların döngüye alındığı ve her bir elemanın tek tek aranan eleman ile karşılaştırıldığı bir algoritmadır. Bu algoritmanın zaman karmaşıklığı $O(n)$ ile ifade edilir. İkili aramada ise öncelikle dizinin sıralanmış olması gereklidir. Dizi ikiye ayrılır ve ortadaki değer kontrol edilir. Aranan değerinin bu değerden küçük veya büyük olmasına göre, ortadaki değere göre dizinin sağında veya solunda arama yapmaya devam edilir. Dizi sıralanmış olduğu için tüm elemanları tek tek aramaya gerek kalmaz. İkili aramanın zaman karmaşıklığı $O(\log n)$ ile ifade edilir.

İkinci bölümde sıralama algoritmaları ele alınmıştır. Bu algoritmalar ise karşılaştırma temelli ve böl ve fethet temelli algoritmalar olmak üzere iki alt başlıkta incelenmiştir.

Karşılaştırma temelli algoritmaların ilki kabarcık sıralama algoritmasıdır. Kabarcık sıralaması, her bir değer kendinden sonra gelen değer ile karşılaştırılarak yer değişikliğinin yapıldığı bir algoritmadır. Kabarcık sıralamasının zaman karmaşıklığı $O(n^2)$ ile ifade edilir. Karşılaştırma mantığına dayanan ikinci algoritma ise yerleşirmeli sıralama algoritmasıdır. Bu algoritmada sıralanmamış bir listede elemanların seçilmesi ve listede doğru yere eklenmesi sağlanır. Sıralanmak istenen dizi, sıralıya yakın veya sıralı bir yapıdaysa algoritmanın zaman karmaşıklığı $O(n)$, tamamen sırasız bir yapıdaysa $O(n^2)$ ile ifade edilir.

Böl ve fethet algoritmaları ise dizinin ikiye bölünmesi, küçük parçalara ayrılarak sıralanması ve tekrardan birleştirilmesi mantığına dayanan algoritmalarıdır. Bunlardan hızlı sıralama algoritmasında bir pivot eleman seçilir ve dizideki elemanlar pivot ile kıyaslanarak alt dizilere bölünür ve sıralanır. Hızlı sıralamanın zaman karmaşıklığı $O(n \log n)$ olur. Bu aileden ele alınan ikinci algoritma ise birleştirici algoritmadır. Bu algoritmada, her bir alt dizide sadece tek bir eleman kalıncaya kadar dizi alt kümelere bölünür ve her biri karşılaştırılarak bir araya getirilir ve sıralanmış dizi elde edilir. Birleştirici sıralamanın zaman karmaşıklığı $O(n \log n)$ ile ifade edilir.

Kaynakça

- Althoff, C. (2022). *The self-taught computer scientist : the beginner's guide to data structures & algorithms*. John Wiley & Sons, Inc.
- Cajic, D. (2019). *An Illustrative Introduction to Algorithms*. Independently published.
- Çobanoğlu, B. (2014). *Algoritma Geliştirme ve Veri Yapıları* (5th ed.). Pusula Yayıncılık.
- Çölkesen, R. (2017). Algoritmalar. In T. R. Çölkesen & O. Aliefendioğlu (Eds.), *Bilgisayar Bilimine Giriş* (pp. 201–247). Papatya Yayıncılık Eğitim.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms* (4th ed.). The MIT Press.
- Erwig, M. (2017). *Once upon an algorithm : how stories explain computing*. MIT Press.
- Mailund, T. (2021). *Introduction to Computational Thinking: Problem Solving, Algorithms, Data Structures, and More*. Apress. <https://doi.org/10.1007/978-1-4842-7077-6>
- Skiena, S. S. (2020). *The Algorithm Design Manual* (3rd ed.). Springer International Publishing. <https://doi.org/10.1007/978-3-030-54256-6>

Ünite Soruları

Soru-1 :

Aşağıdakilerden hangisi bilinen bir arama algoritmasıdır?

(Çoktan Seçmeli)

- (•) - Çapraz arama
- (•) - İkili arama
- (•) - Arama algoritması
- (•) - Eleman arama
- (•) - Hiçbiri

Cevap-1 :

İkili arama

Soru-2 :

Aşağıdakilerden hangisi bilinen bir arama algoritmasıdır?

(Çoktan Seçmeli)

- (•) - Çapraz arama
- (•) - Tek tek arama
- (•) - Arama algoritması
- (•) - Doğrusal arama
- (•) - Hiçbiri

Cevap-2 :

Doğrusal arama

Soru-3 :

Aşağıdakilerden hangisi karşılaştırma temelli olarak çalışan bir sıralama algoritmasıdır?

(Çoktan Seçmeli)

- (•) - Hızlı sıralama
- (•) - Birleştirmeli sıralama
- (•) - İkili sıralama
- (•) - Kabarcık sıralama
- (•) - Hiçbiri

Cevap-3 :

Kabarcık sıralama

Soru-4 :

Aşağıdakilerden hangisi karşılaştırma temelli olarak çalışan bir sıralama algoritmasıdır?

(Çoktan Seçmeli)

- (•) - Hızlı sıralama
- (•) - Yerleşirmeli sıralama
- (•) - İkili sıralama
- (•) - Doğrusal sıralama
- (•) - Hiçbiri

Cevap-4 :

Yerleşirmeli sıralama

Soru-5 :

Aşağıdakilerden hangisi böl ve fethet mantığına göre çalışan bir sıralama algoritmasıdır?

(Çoktan Seçmeli)

- (•) - Hızlı sıralama
- (•) - Yerleşirmeli sıralama
- (•) - Birleşik sıralama
- (•) - Doğrusal sıralama
- (•) - Hiçbiri

Cevap-5 :

Hızlı sıralama

Soru-6 :

Aşağıdakilerden hangisi böl ve fethet mantığına göre çalışan bir sıralama algoritmasıdır?

(Çoktan Seçmeli)

- (•) - Lineer sıralama
- (•) - Yerleşirmeli sıralama
- (•) - Birleşik sıralama
- (•) - Birleştirilmiş sıralama
- (•) - Hiçbiri

Cevap-6 :

Birleşirmeli sıralama

Soru-7 :

İkili arama algoritmasının ön şartı nedir?

(Çoktan Seçmeli)

- (•) - Lineer sıralamanın uygulanmış olması gereklidir.
- (•) - Dizide aranan elemanın olması gereklidir.
- (•) - Herhangi bir ön şartı yoktur.
- (•) - Dizinin küçükten büyüğe sıralanmış olması gereklidir.
- (•) - Hiçbiri

Cevap-7 :

Dizinin küçükten büyüğe sıralanmış olması gereklidir.

Soru-8 :

Böl ve fethet algoritmaları ile ilgili hangisi yanlışdır?

(Çoktan Seçmeli)

- (•) - Problem küçük alt problemlere ayrılarak çözülür.
- (•) - Özyineli olarak çalışan algoritmalarıdır.
- (•) - Sıralama tek seferde yapılır.
- (•) - Birden fazla algoritma bu aileye aittir.
- (•) - Sıralanan dizi alt dizilere ayrılır.

Cevap-8 :

Sıralama tek seferde yapılır.

Soru-9 :

1) Aşağıdakilerden hangisi ile ifade edilen zaman karmaşıklığı en büyüktür?

(Çoktan Seçmeli)

$O(\log n)$

$O(n)$

$O(n \log n)$

$O(n^2)$

$O(n^3)$

Cevap-9 :

$O(n^3)$

8. GENEL UYGULAMALAR

8.1. Örnekler

- a) Haftalık normal çalışma süresi 40 saat ve aldığı ücret 3000 lira olan bir çalışanın mesai yaptığı her saat başına ekstra 50 lira kazandığı biliniyor. Kullanıcı tarafından girilen mesai saatine göre çalışanın alması gereken haftalık toplam ücreti hesaplayıp ekrana yazdırın algoritmayı yazın ve akış diyagramını çizin.

BAŞLA

OKU mesai saatı

IF mesai saatı > 40 THEN

fazla mesai = 40 – mesai saatı

ELSE

fazla mesai = 0

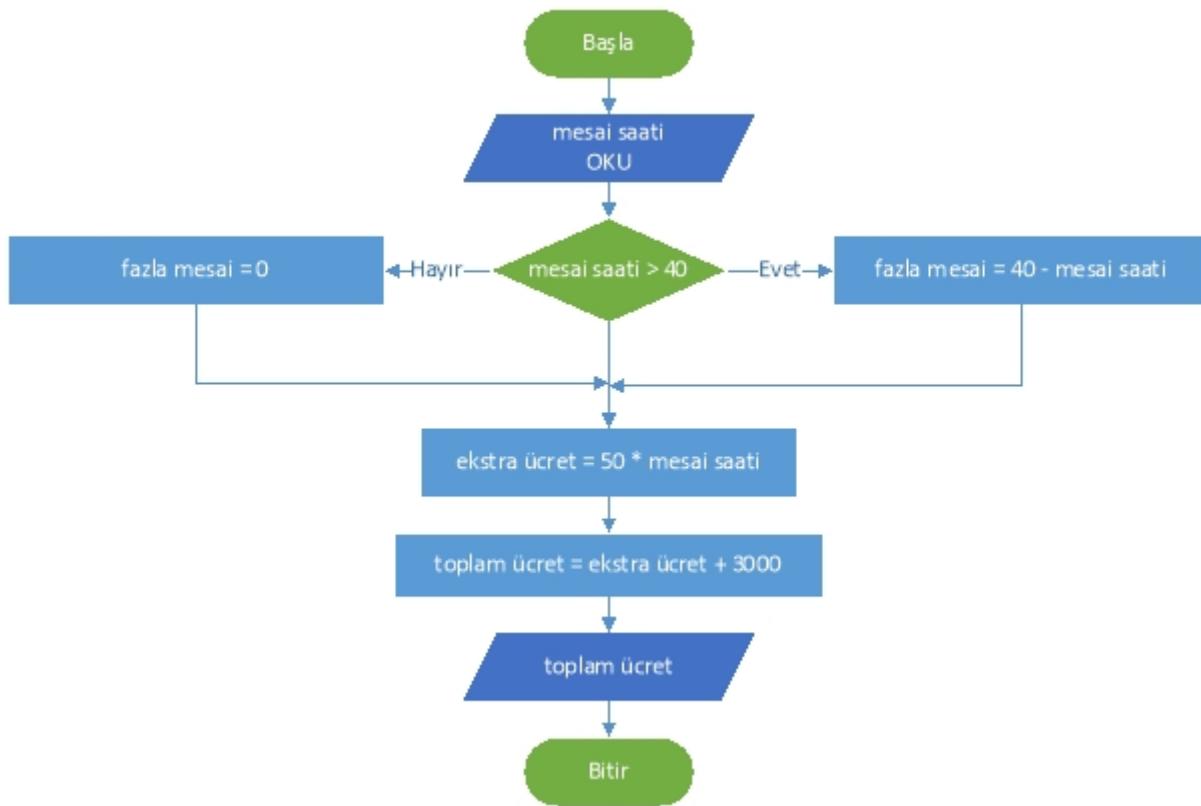
END IF

ekstra ücret = 50 * fazla mesai

toplam ücret = ekstra ücret + 3000

YAZ toplam ücret

BİTİR



- b) Fiyatı 100.000 TL ve yukarı evler için ödenecek olan vergi oranı %20, fiyatı 50.000 ile 100.000 TL arasındaki evler için vergi oranı %15, fiyatı 50.000 TL'den az olanlar içinse vergi oranı %10 olarak belirlenmiştir. Buna göre kullanıcı tarafından girilen fiyata göre ödenmesi gereken vergi miktarını hesaplayan algoritmayı yazınız ve akış diyagramını çiziniz.

BAŞLA

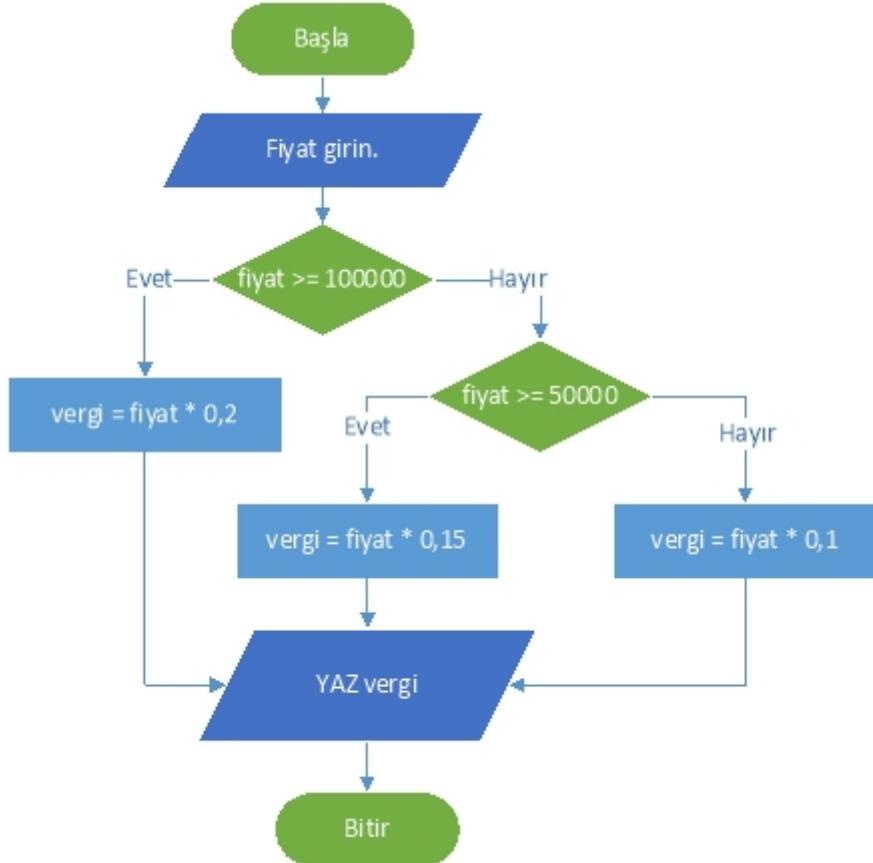
OKU fiyat

IF fiyat >= 100000 THEN

```

vergi = fiyat * 0,2
ELSE IF fiyat >= 50000
vergi = fiyat * 0,15
ELSE
vergi = fiyat * 0,1
END IF
YAZ (vergi)
BITİR

```



- c) Kenar ölçüleri verilen bir dörtgenin eşkenar dörtgen olup olmadığını hesaplayan ve çevre uzunluğunu bulan algoritmayı yazınız ve akış diyagramını çiziniz.

BAŞLA

OKU kenar1, kenar2, kenar3, kenar4

çevre = kenar1 + kenar2 + kenar3 +kenar 4

IF (kenar1=kenar2) AND (kenar2=kenar3) AND (kenar3=kenar4) THEN

YAZ ("Eşkenar dörtgen")

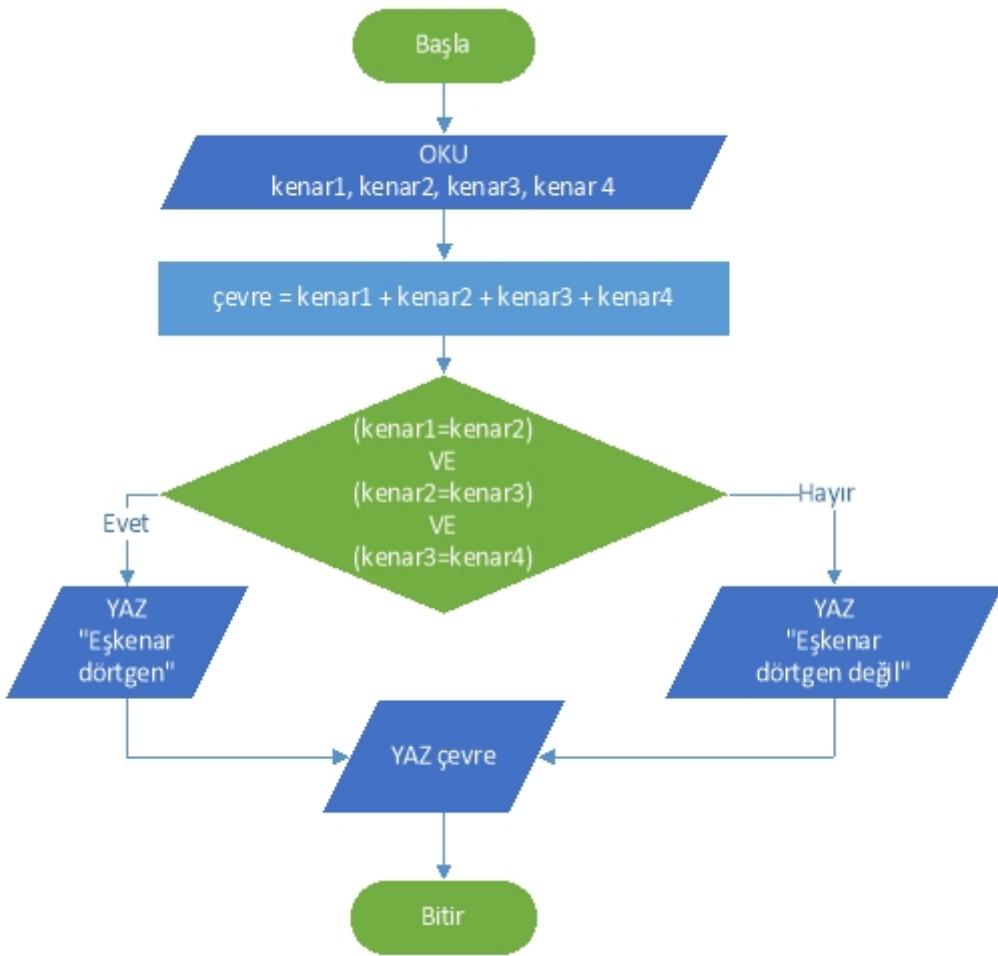
ELSE

YAZ ("Eşkenar dörtgen değil")

END IF

YAZ (çevre)

BITİR



d) Kullanıcı “q” değerini girine kadar sayı alıp “q” değerini girince bu sayıların ortalamasını alan algoritmayı yazın ve akış diyagramını çizin.

BAŞLA

toplam = 0, sayı adedi = 0, değer = 0

OKU değer

DO WHILE değer <> “q”

toplam = toplam + değer

sayı adedi = sayı adedi + 1

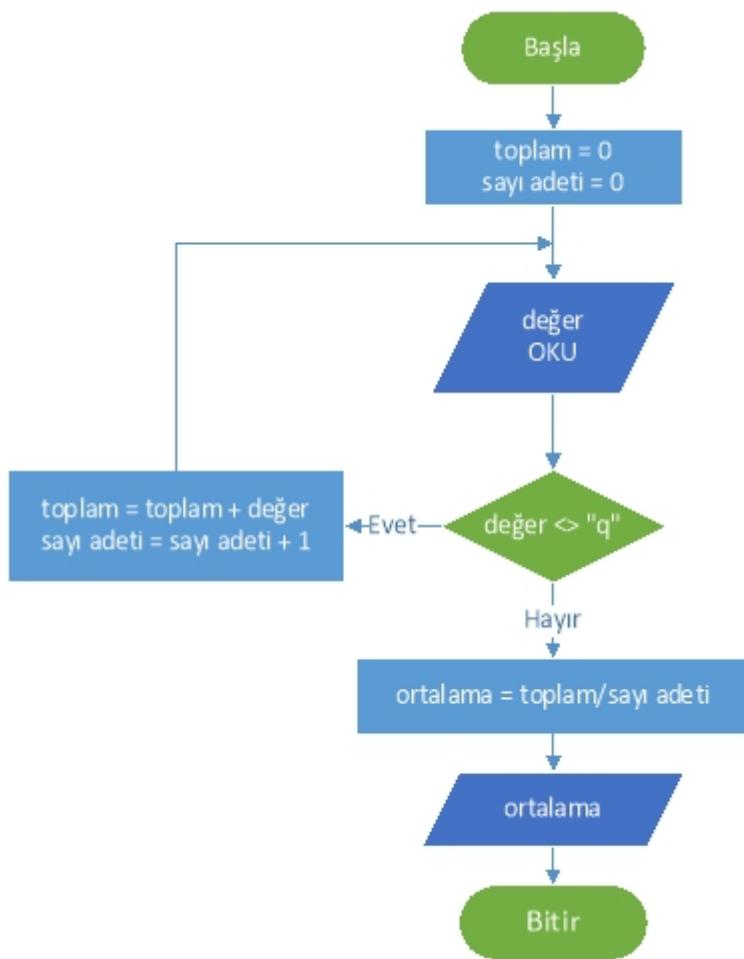
OKU değer

LOOP

ortalama = toplam/sayı adedi

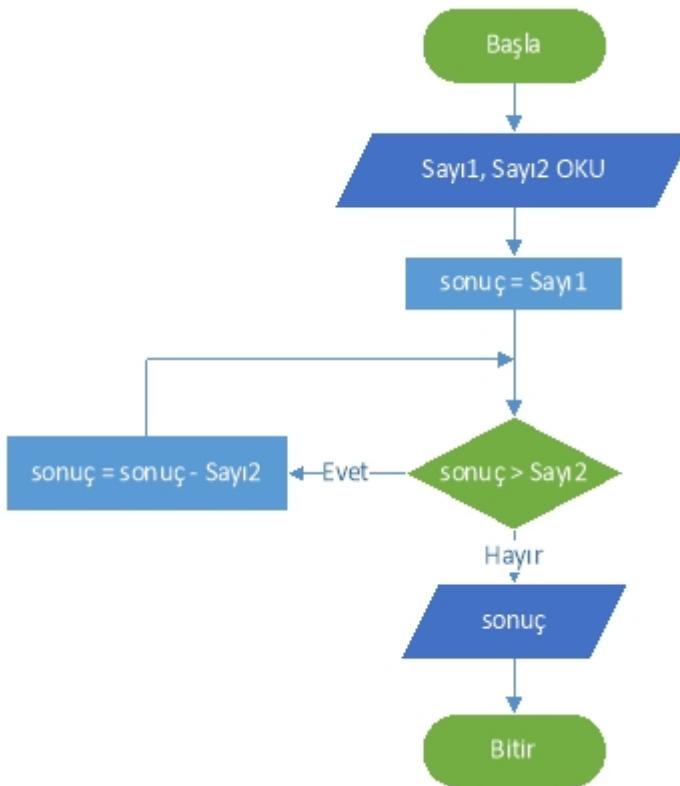
YAZ (ortalama)

BİTİR



- e) '%' operatörünü kullanmadan verilen 2 sayının modunu alan uygulamanın algoritmasını yazın ve akış diyagramını çizin.

BAŞLA
 OKU sayı1,sayı2
 sonuç=sayı1
 DO WHILE sonuç > sayı2
 sonuç = sonuç - sayı2
 LOOP
 YAZ sonuç
 BİTİR



- f) Girilen harf "q" değilse kullanıcı tarafından girilen harfleri ekranda gösteren, girilen harf "q" ise programdan çıkış yapan algoritmayı yazınız ve akış diyagramını çiziniz.

BAŞLA

OKU harf

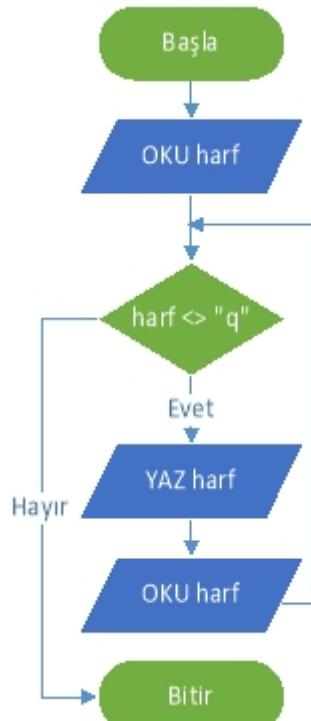
DO WHILE harf <> "q"

YAZ (harf)

OKU harf

LOOP

BİTİR



g) 1 ile 50 arasındaki tam sayılardan tek sayıların toplamı ile çift sayıların toplamının farkının negatif mi, pozitif mi olduğunu bulan programın algoritma ve akış diyagramını çiziniz.

BAŞLA

tek = 0, çift = 0

FOR i = 1 TO 50 Step 2

tek = tek + i

çift = çift + (i + 1)

NEXT i

IF ((tek - çift) < 0) THEN

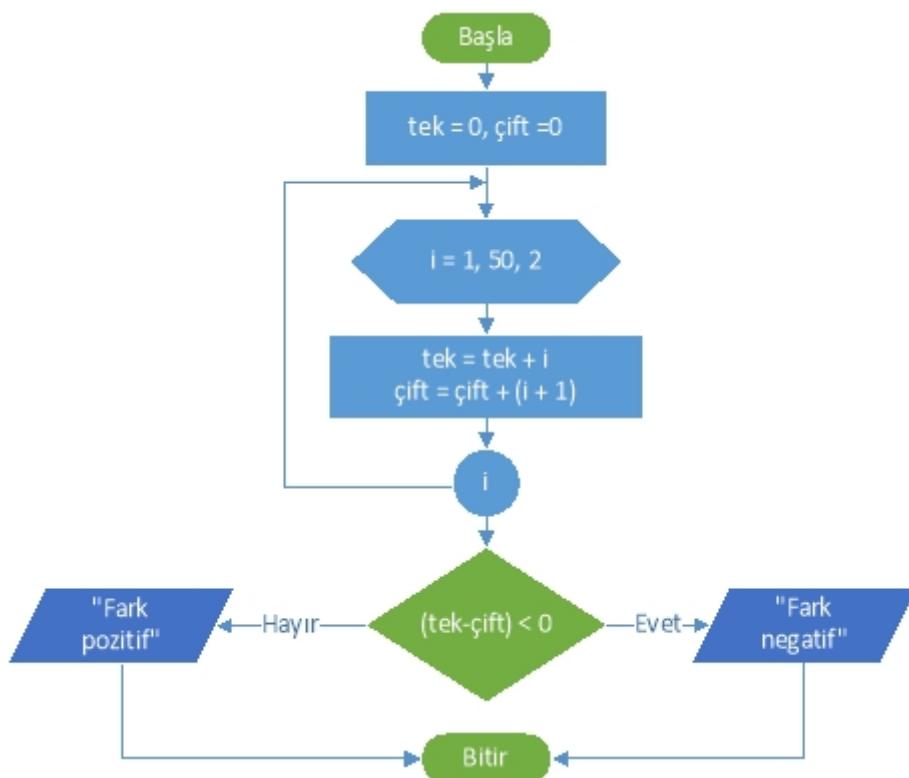
YAZ ("Fark negatif")

ELSE

YAZ ("Fark pozitif")

END IF

BİTİR



h) Verilen iki sayının en büyük ortak bölenini bulan algoritmayı yazıp akış diyagramını çiziniz.

BAŞLA

OKU say1,sayı2

döngü = True

DO WHILE döngü = True

IF sayı1 > sayı2 THEN

temp = sayı1

sayı1 = sayı2

sayı2 = temp

ELSE IF sayı1 ≤ 0 THEN

sonuç = sayı2

döngü = False

ELSE

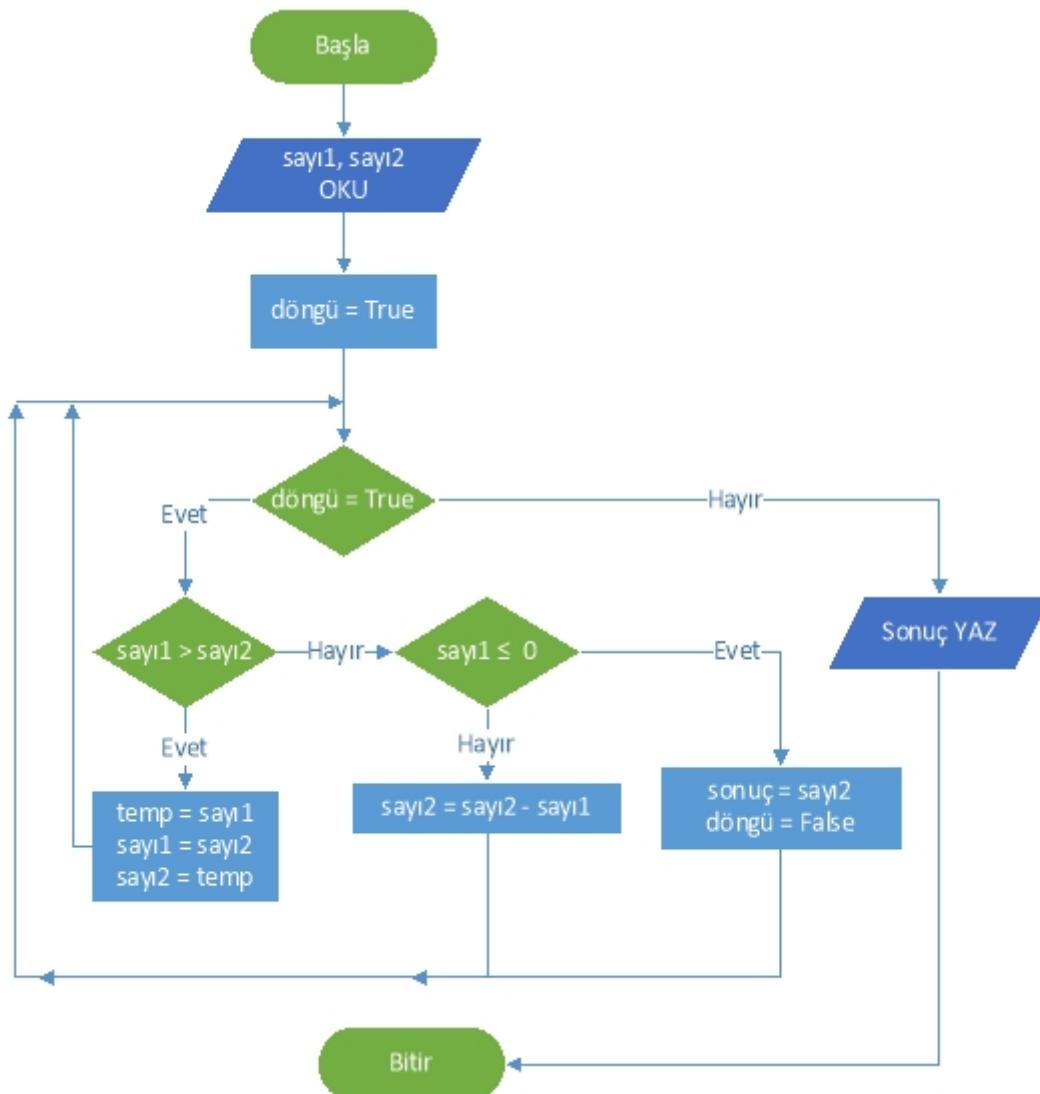
```
sayı2 = sayı2 - sayı1
```

```
END IF
```

```
LOOP
```

```
YAZ sonuç
```

```
BİTİR
```



- i) 50 kişilik bir sınıfta yıl sonu notlarını kullanıcidan alarak, sınıf ortalamasının altında kalanlar için “Yetersiz”, üstünde veya eşit olanlar içinse “İyi” yazan algoritmayı ve akış diyagramını çiziniz.

```
BAŞLA
```

```
toplam not = 0
```

```
FOR i = 1 TO 50
```

```
OKU not(i)
```

```
toplam not += not(i)
```

```
NEXT i
```

```
ortalama = toplam_not/50
```

```
FOR i = 1 TO 50
```

```
IF not(i)<ortalama THEN
```

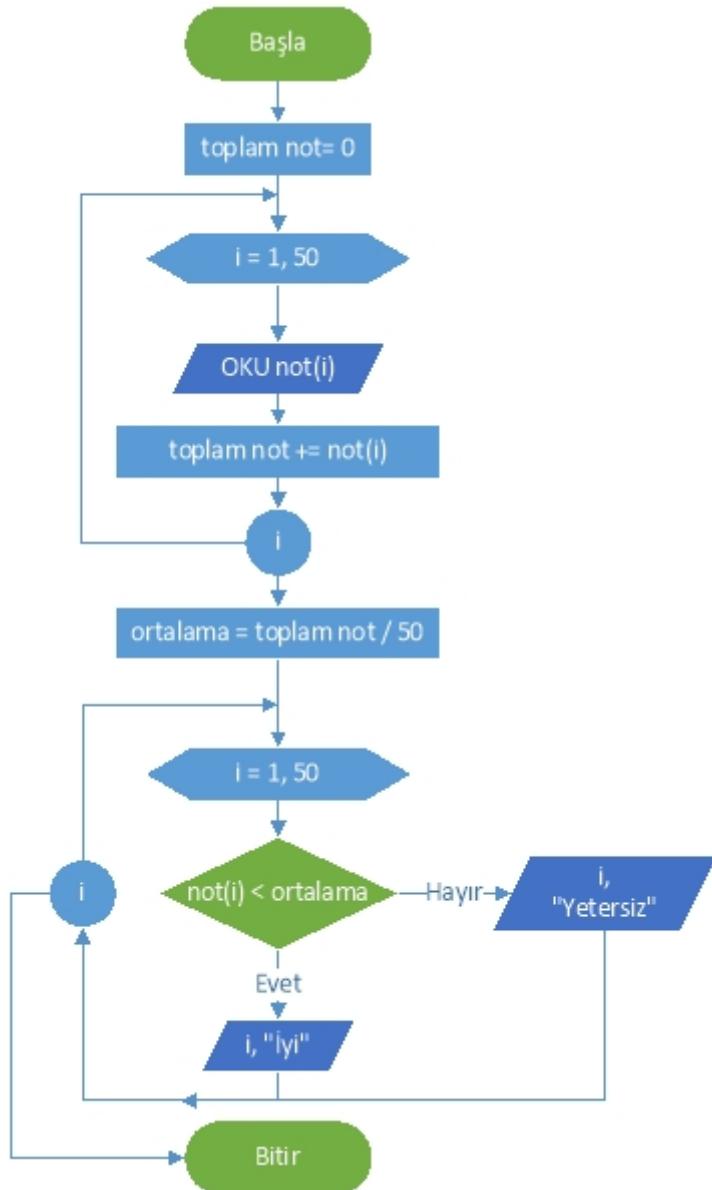
```
YAZ (i, "Yetersiz")
```

```
ELSE
```

```
YAZ (i, "İyi")
```

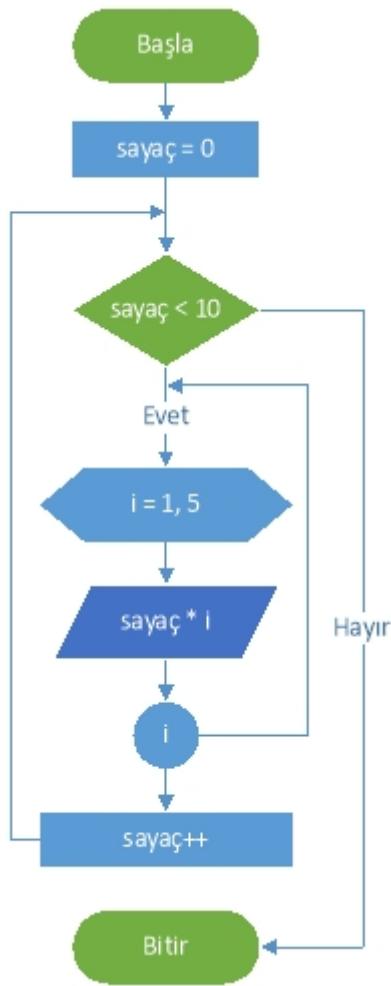
```
END IF
```

NEXT i
BİTİR



- j) 0'dan 10'a kadar olan her sayıyla 1'den 5'e kadar olan her sayının çarpımını ekrana yazdırın algoritma ve akış şemasını çiziniz.

BAŞLA
 sayaç = 0
 DO WHILE sayaç < 10
 FOR i = 1 TO 5
 YAZ (sayaç * i)
 NEXT i
 sayaç = sayaç + 1
 LOOP
 BİTİR



k) Toplam n tane puan olan bir puan dizisinin ortalamasını bulan ve ekrana yazdırın algoritma ve akış şemasını çiziniz.

BAŞLA

toplam = 0

OKU n

FOR i = 1 TO n

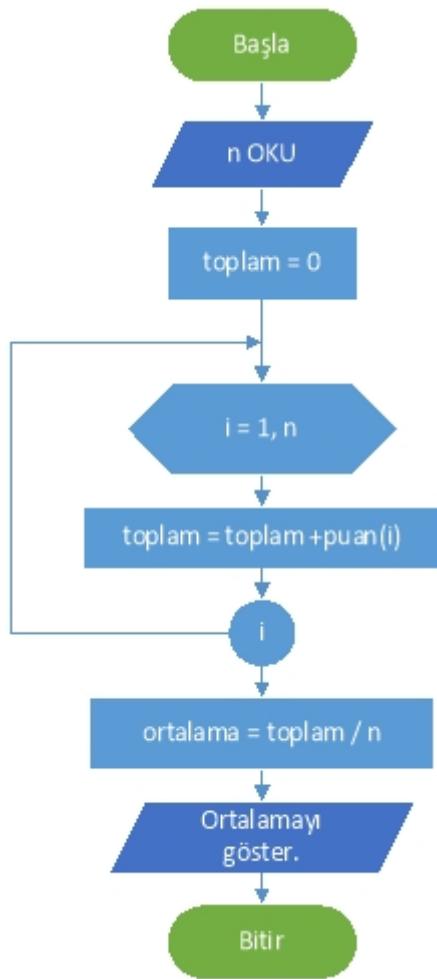
toplam = toplam + puan(i)

NEXT i

ortalama = toplam/n

YAZ (ortalama)

BİTİR



- I) Kullanıcı tarafından girilen sayıların toplamını ve ortalamasını hesaplayan bir program aşağıda verilmiştir. Kullanıcı -1 değerini girmediği müddetçe, kullanıcıdan sayı girmesi istenir. -1 girildiğinde toplama işlemi sonlanır. Sayacın 0 olduğu durumda ise sayı girilmediği yazdırılır.

BAŞLA

Sayaç = 0

Toplam = 0

Sayı OKU

DO WHILE (Sayı <> -1)

 Toplam = Toplam + Sayı

 Sayaç = Sayaç + 1

 Sayı OKU

LOOP

IF (Sayaç <> 0) THEN

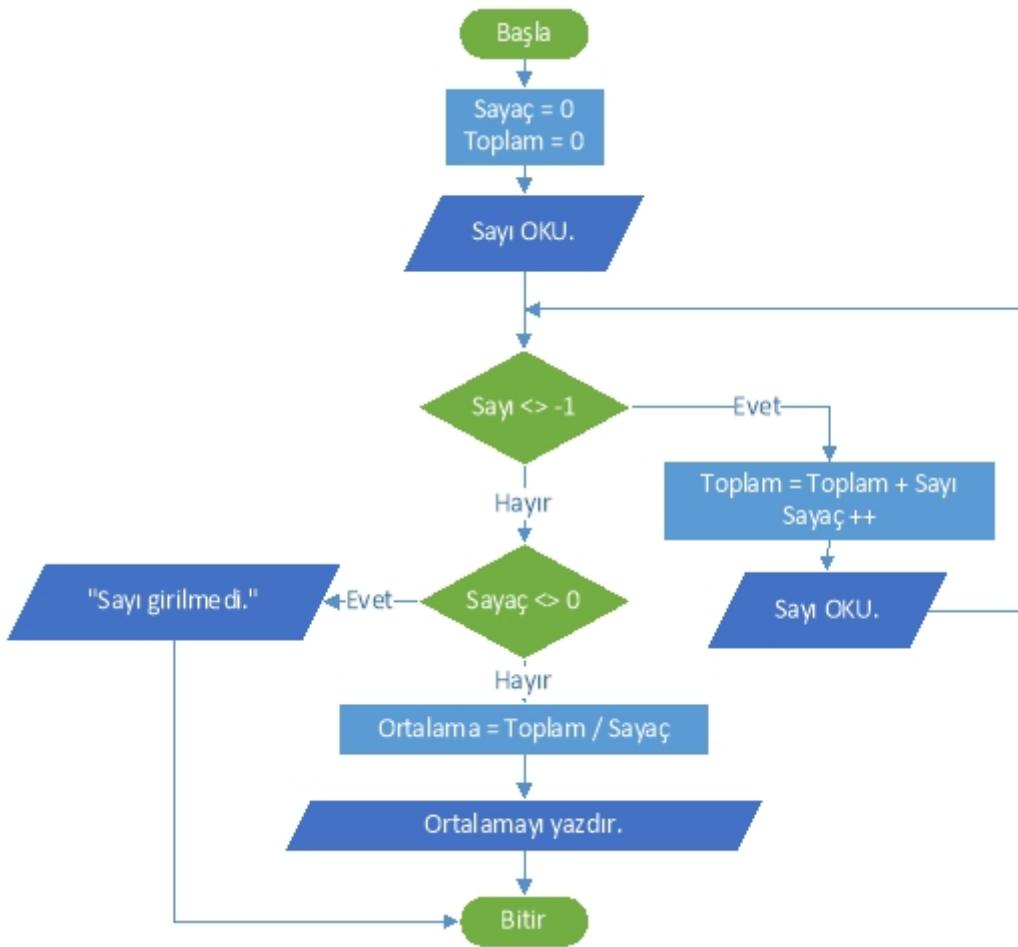
 Ortalama = Toplam / Sayaç

 YAZ (Ortalama)

ELSE

 YAZ ("Sayı girilmedi.")

BİTİR



m) 10 güne ait hava sıcaklığı değerleri kullanıcı tarafından girilecektir. Bu değerlere bağlı olarak serin ve sıcak günlerin sayısını tutan programın kaba kodu ve akış diyagramı aşağıdadır.

BAŞLA

sıcak = 0, serin = 0, sıcaklık = 0

FOR i = 1 TO 10

OKU (sıcaklık)

IF (sıcaklık < 20) THEN

serin ++

ELSE

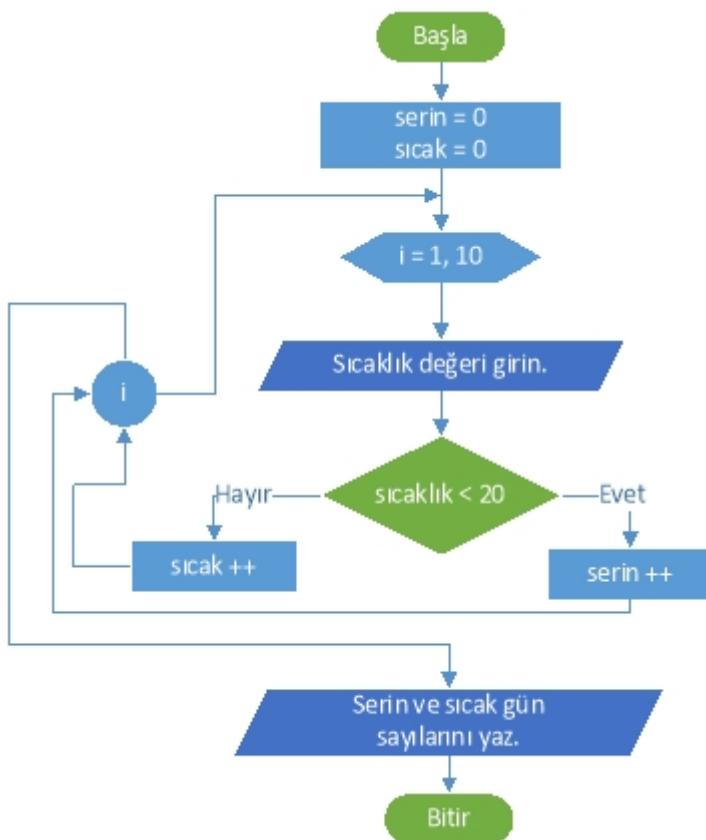
sıcak ++

END IF

NEXT i

YAZ (sıcak, serin)

BİTİR



- n) Bir şirketteki 1000 çalışanın maaşları bir dizi şeklinde verilmiştir. Çalışanlardan maaşları 15000 liraya eşit ve 15000 liradan yüksek olanlara 5000 lira, 30000 lira ve 30000 liradan yüksek olanlara 10000 lira zam yapılmış maaşların olduğu dizi güncellenecek ve en son tüm maaşlar yazdırılacaktır. Bunu yapan algoritma aşağıda verilmiştir.

BAŞLA

çalışan sayısı = 1000

FOR $i = 1$ TO çalışan sayısı

IF ($maaş(i) \geq 15000$) THEN

$maaş(i) = maaş(i) + 5000$

ELSE IF ($maaş(i) \geq 30000$)

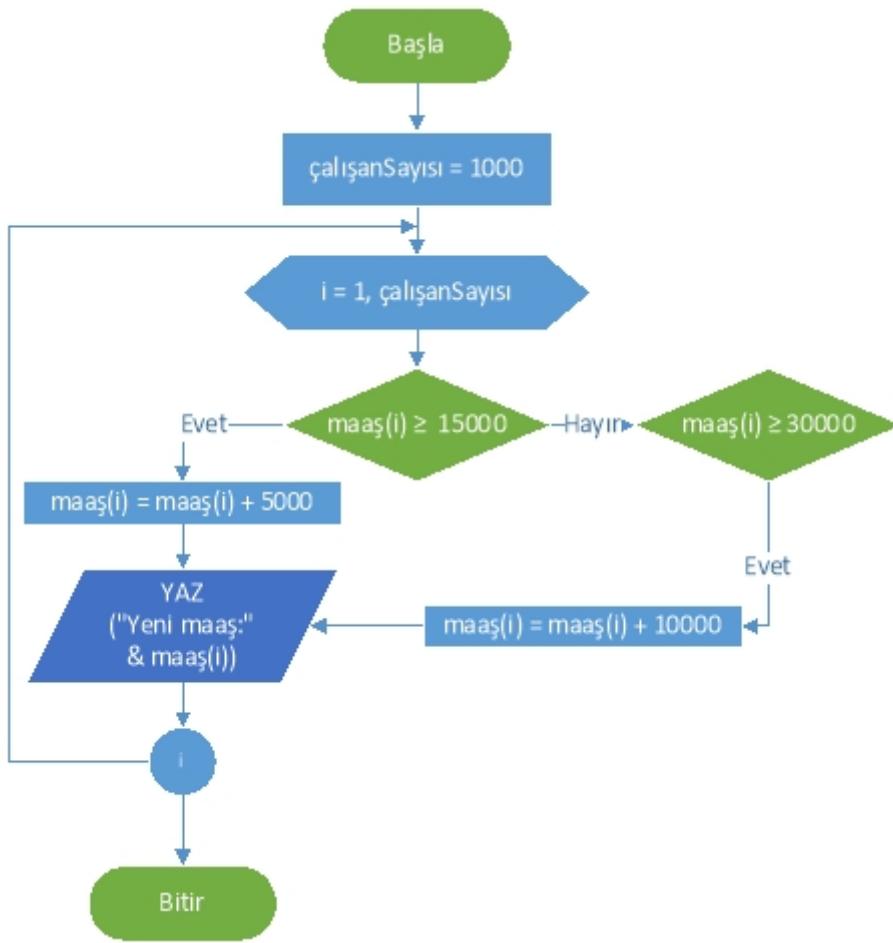
$maaş(i) = maaş(i) + 10000$

END IF

YAZ ("Yeni maaş:" $maaş(i)$)

NEXT i

BİTİR



Ay Ocak Şubat Mart Nisan Mayıs Haziran

Buzdolabı adedi 350 560 360 450 400 900

- o) Yukarıda bir firmanın ilk 6 ayındaki buzdolabı satışları buzdolabı isimli dizide verilmiştir. Bu dizideki verilere göre üç farklı algoritma sorusu aşağıda verilmiştir.

- 400 adetten fazla buzdolabı satılan ayların toplam sayısı:

BAŞLA

toplam ay = 6, say1 = 0

FOR i = 1 TO toplam ay

IF (buzdolabı(i) > 400) THEN

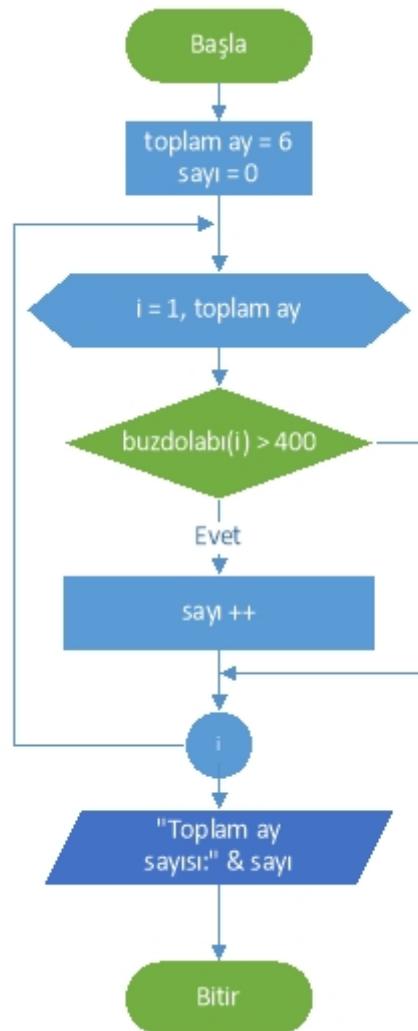
say1 ++

END IF

NEXT i

YAZ ("Toplam ay sayısı:" & say1)

BİTİR



- İlk 3 aydaki toplam buz dolabı satış adedi:

BAŞLA

toplasm ay = 6, adet = 0

FOR i = 1 TO toplam ay

IF (i < 4)) THEN

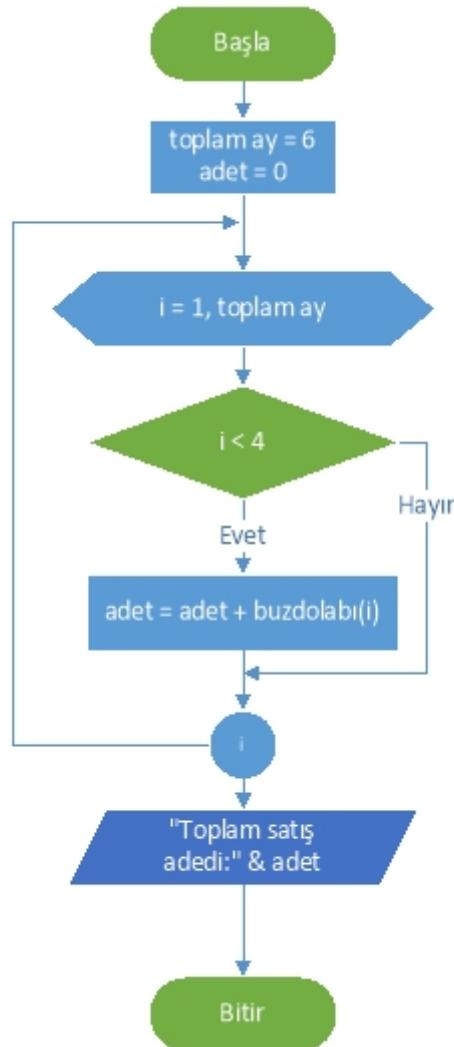
adet = adet + buz dolabı(i)

END IF

NEXT i

YAZ ("Toplam satış adedi:" & adet)

BİTİR



- 300'den fazla 450'den az satış yapılan ayların sayısı:

BAŞLA

toplaml_ay = 6, say1 = 0

FOR i = 1 TO toplaml_ay

IF (buzdolabi(i)>300 AND buzdolabi(i)<450) THEN

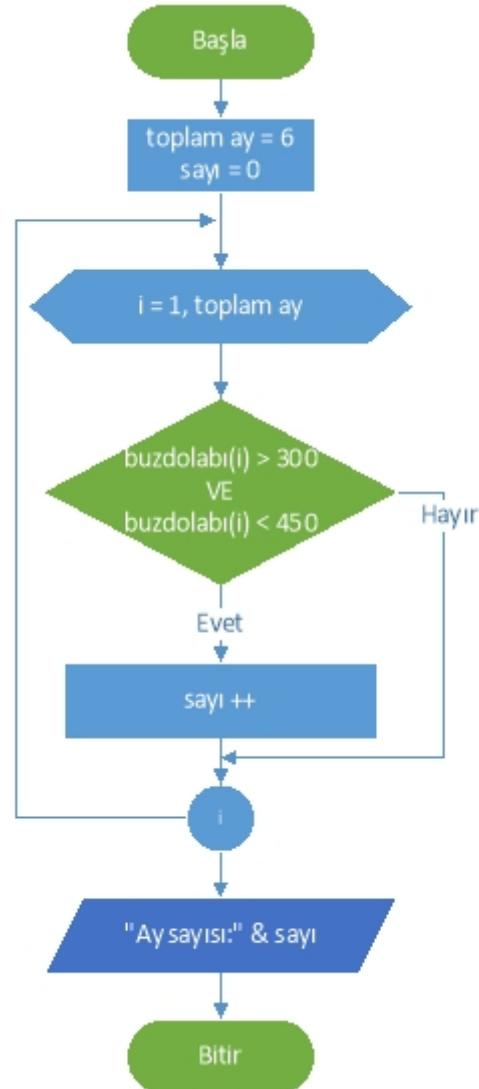
say1 ++

END IF

NEXT i

YAZ ("Ay sayısı:" & say1)

BİTİR



p) Fabrikadaki tüm ürünlerin maliyeti bir dizide verilmiştir. Bu fabrikada yeni üretime olan bir ürünün maliyetinin eski ürünlerin kaç tanesinin maliyetinden daha fazla olduğunu bulan algoritma ve akış diyagramı aşağıda verilmiştir.

BAŞLA

OKU maliyet, ürün sayısı

toplamsayı = 0

FOR i = 1 TO ürün sayısı

IF (maliyet > dizi(i)) THEN

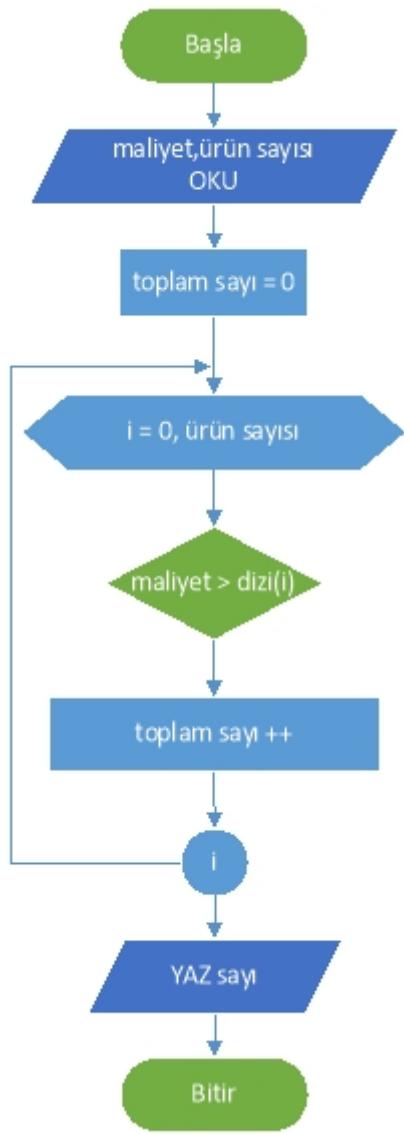
toplamsayı ++

END IF

NEXT i

YAZ (toplamsayı)

BİTİR



q) Girilen doğum yılına göre bir kişinin ehliyet alıp alamayacağını hesaplayan algoritma ve akış diyagramı aşağıda verilmiştir.

```

int yaşHesapla(yıl)
BAŞLA
    şimdikiYıl = now()
    sonuç = şimdikiYıl - yıl
    return sonuç
BİTİR
    
```

```

void ehliyet(girdi)
BAŞLA
    IF girdi ≥ 18 THEN
        YAZ ("Ehliyet alabilir.")
    ELSE
        YAZ ("Ehliyet alamaz.")
    END IF
BİTİR
    
```

Ana Program

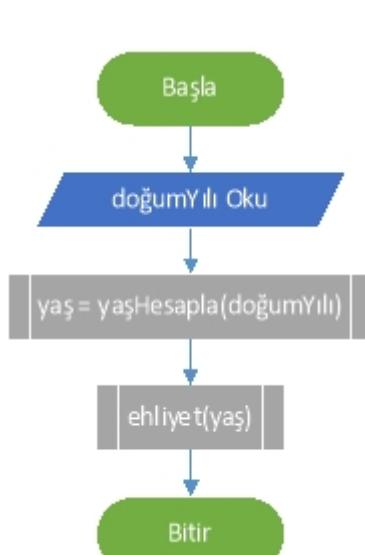
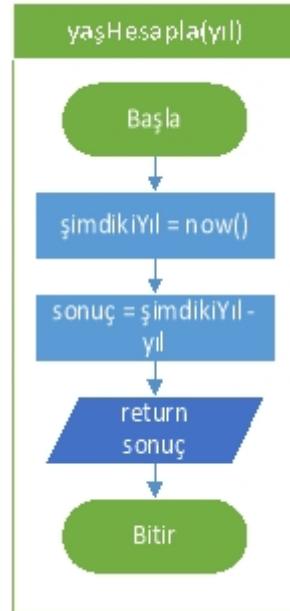
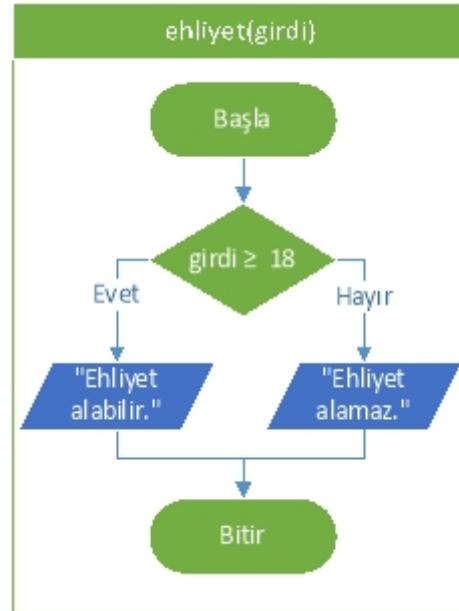
BAŞLA

OKU doğumYılı

yaş = Call yaşHesapla(doğumYılı)

Call ehliyet(yaş)

BİTİR

Ana Program**Alt Program****Alt Program**

- r) Girilen sayının girilen dereceden kuvvetini çarpması işlemiyle alan algoritma ve akış diyagramı aşağıda verilmiştir.

```

int üsAl(sayı, üs)
BAŞLA
toplam = 1
DO WHILE (üs <> 0)
toplam = toplam * sayı
üs = üs - 1
LOOP
return toplam
BİTİR
  
```

Ana Program

BAŞLA

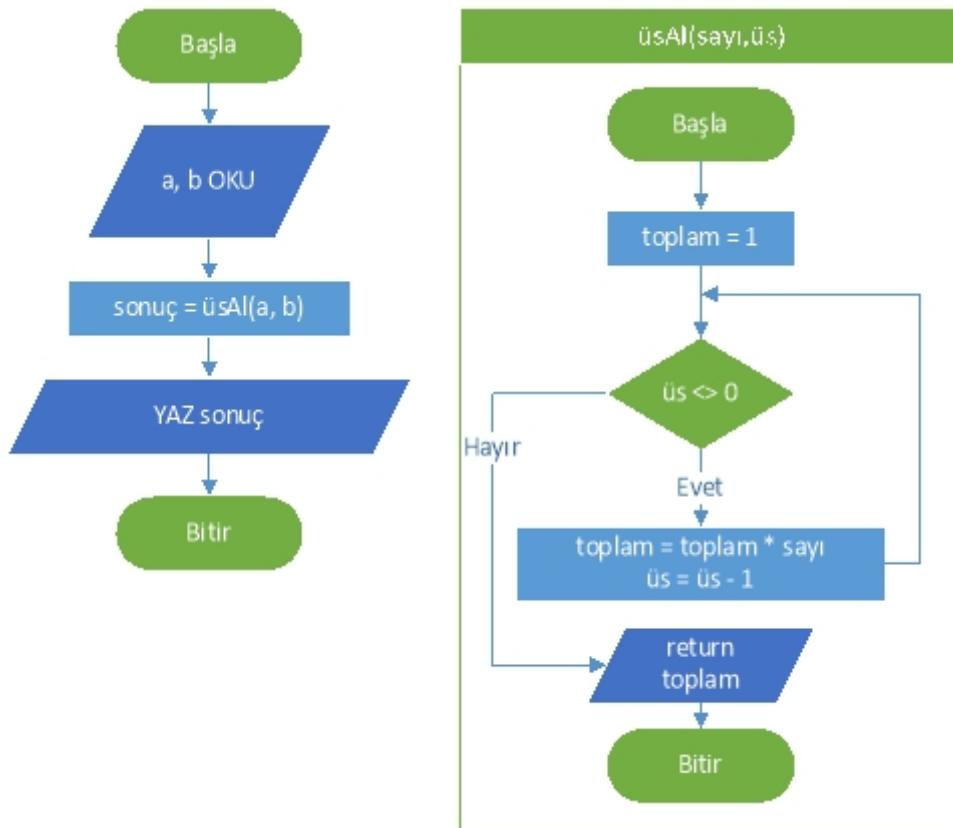
OKU a, b

sonuç = üsAl(a, b)

YAZ (sonuç)

BİTİR

Ana Program**Alt Program**



s) Yarıçapı girilen kürenin yüzey alanını ve hacmini hesaplayan algoritma ve akış diyagramı aşağıda verilmiştir.

```
int alanHesapla(yarıçap, p)
```

BAŞLA

```
sonuç = 4*p*yarıçap*yarıçap
```

return sonuç

BİTİR

```
int hacimHesapla(yarıçap, p)
```

BAŞLA

```
sonuç = (4*p*yarıçap*yarıçap*yarıçap)/3
```

return sonuç

BİTİR

Ana Program

BAŞLA

OKU r

pi = 3,14

alan = alanHesapla(r, pi)

hacim = hacimHesapla(r, pi)

YAZ ("Kürenin yüzey alanı: " & alan)

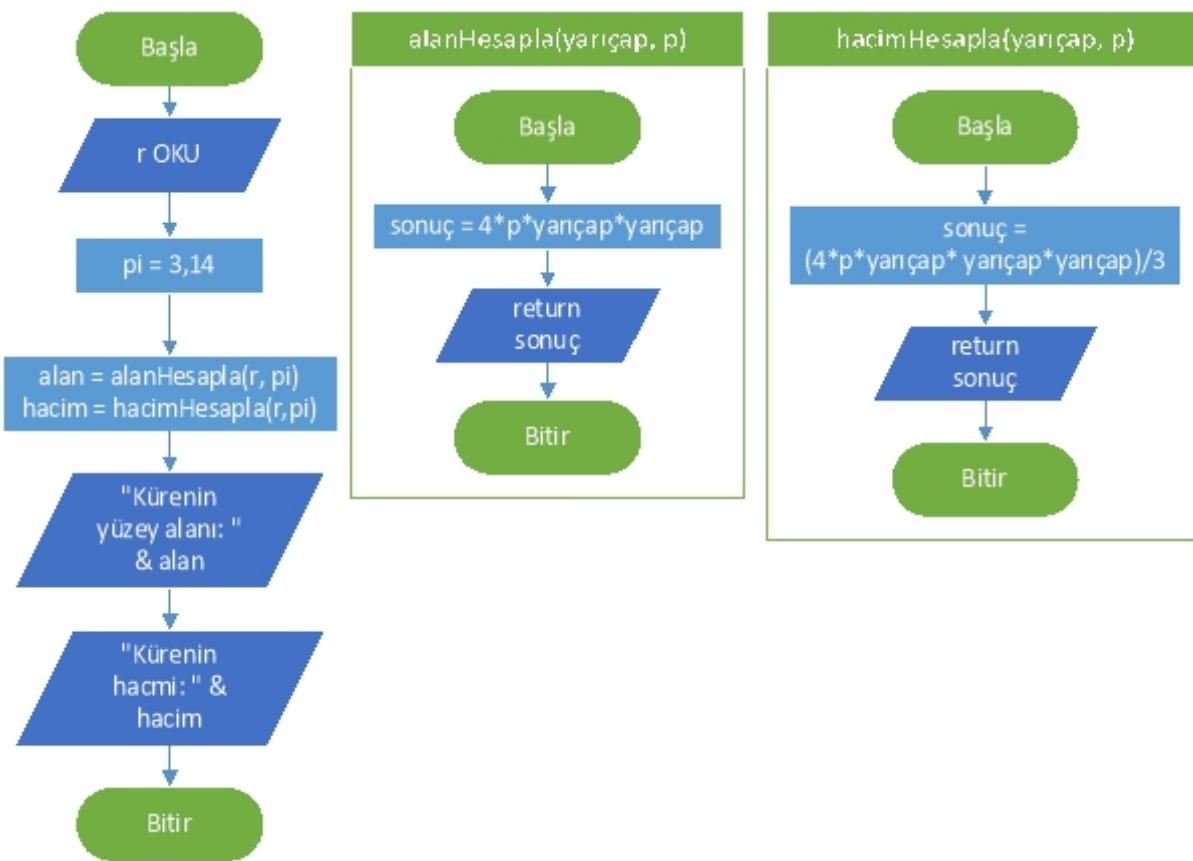
YAZ ("Kürenin hacmi: " & hacim)

BİTİR

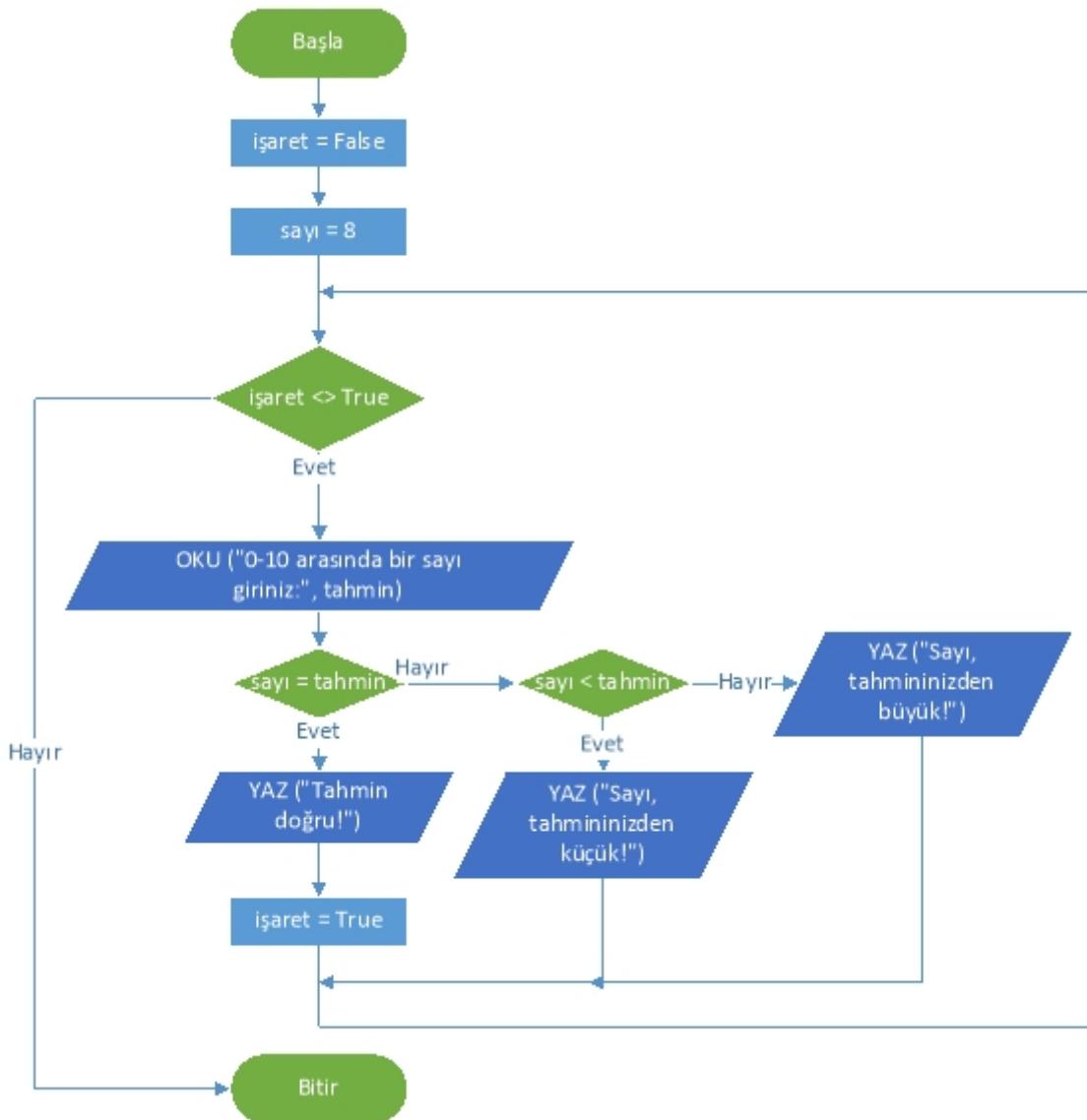
Ana Program

Alt Program

Alt Program



- t) Kullanıcı tarafından tahmin edilen bir değerin, sayının tanımlanmış değeri ile karşılaştırıldığı algoritma örneği ve akış diyagramı aşağıdadır.



Bölüm Özeti

Bu bölümde bundan önceki bölümlerde öğrenilen tüm konu başlıklarına yönelik olarak örnekler çözülmüştür. Bu örneklerin konuları pekiştirmesi ve uygulamaya dönük olarak farklı problemlerin algoritma ile nasıl göstereceğinin pratığının yapılması sağlanmıştır.

Kaynakça

- Althoff, C. (2022). *The self-taught computer scientist : the beginner's guide to data structures & algorithms*. John Wiley & Sons, Inc.
- Cajic, D. (2019). *An Illustrative Introduction to Algorithms*. Independently published.
- Çobanoğlu, B. (2014). *Algoritma Geliştirme ve Veri Yapıları* (5th ed.). Pusula Yayıncılık.
- Çobanoğlu, B. (2019). *Herkes İçin Python* (2nd ed.). Pusula Yayıncılık. <http://files/37010/478372.html>
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms* (4th ed.). The MIT Press.
- Deitel, P., Deitel, A., & Deitel, H. (2014). *Visual Basic 2012 How to Program* (6th ed.). Pearson.
- Erwig, M. (2017). *Once upon an algorithm : how stories explain computing*. MIT Press.

Gezgin, D. M. (2015). Açıklamalı Algoritma Soruları ve Çözümleri. Kriter Yayınları.
https://www.researchgate.net/publication/347508715_Aciklamali_Algoritma_Ornek_Sorulari_ve_Cozumleri

Mailund, T. (2021). Introduction to Computational Thinking: Problem Solving, Algorithms, Data Structures, and More. Apress. <https://doi.org/10.1007/978-1-4842-7077-6>

Skiena, S. S. (2020). The Algorithm Design Manual (3rd ed.). Springer International Publishing.
<https://doi.org/10.1007/978-3-030-54256-6>

Vatansever, F. (2011). Algoritma Geliştirme ve Programlamaya Giriş (8th ed.). Seçkin Yayıncılık.

Walia, R. K. (2022). Algorithm & Flowchart Manual for Students.
<https://courses.minia.edu.eg/Attach/16036flowchart-algorithm-manual.pdf>