ALGORİTMA

ve

PROGRAMLAMAYA GİRİŞ

AFC Uygulaması ve Örnek Problemler

Hazırlayan

Öğr. Gör. Umut ORHAN

TOKAT - 2010

İÇİNDEKİLER

1. Algoritmaya Giriş

1.1. Tarihçe

Bir kaynağa göre *algoritma (algorithm)* sözcüğü 9. yüzyılda cebir alanındaki çalışmalarını kitaba dökerek matematiğe büyük bir katkı sağlayan el Harezmi isimli alimden kaynaklanır. Dünyanın ilk cebir ve algoritma koleksiyonunu oluşturan eserlerinin Latince çevirisi Avrupa'da çok ilgi görür. Fakat söylenmesi zor geldiği için alimin ismi *algorizm* olarak telaffuz edilir. O dönemde sayısal işlemlerde roma rakamlarını kullanan Avrupalı araştırmacılar aritmetik problemleri Arap sayıları kullanarak çözmeye *algorizm* demişlerdir. Daha sonraki zamanlarda da kelime değişerek *algorithm* şekline dönüşmüştür. Başka bir kaynakta ise Arapçada *el-cebir* diye okunan kitabın telaffuzu yüzünden Avrupa'da önce *al-gebra* sonra da *algorithm* şeklindeki yazılışın kullanıldığı söylenir.

1.2. Tanım

Kelime anlamı olarak algoritma, bir amaç için oluşturulan yöntemin sonuç elde edilinceye kadar sürdürülmesidir. Başka bir deyişle algoritma, belirli bir kurala bağlı olarak yapılan her türlü hesaplamaya verilen isimdir. Matematikte ise algoritma bir sorunun yanıtını ya da bir problemin çözümünü sonlu sayıda aşmada veren sistematik yöntemdir.

Algoritma, sezgisel çözümlemeye ve deneme-yanılmaya karşıt olan bir yöntemdir. Algoritmalar özel durumlara çözüm sunmazlar, genel çözümlerin işlem adımlarını içerirler. Bir algoritma çalışmasında belirsizliklerin giderilmesi gerekmektedir.

1.3. Algoritma ve Bilgisayar

Hayatı kolaylaştırmak için geliştirilen bilgisayarların temel amacı doğadaki problemleri çözebilmektir. Problemlerin bilgisayarlara aktarılmasında algoritmalar bir köprü konumundadır. Bir problemin bilgisayar modelini oluşturabilmek için önce problemi anlayarak analiz etmek ve çözüm yollarını ortaya koymak gerekir. Diğer bir deyişle problemin ve çözümün algoritması geliştirilir.

Bir bilgisayar, problemin nasıl çözüleceği konusunda insanlara yardımcı olamaz. Sadece insanların çözüm için gösterdiği yollardan giderek istenenleri hatasızca uygular. Bilgisayarın doğru sonuca ulaşabilmesi için kendisine gösterilen çözüm yolunda hiçbir belirsizlikle karşılaşmaması gerekir. Bu nedenle hazırlanacak algoritmada her türlü detayın önceden düşünülmeli ve karşılaşılabilecek değişik durumlarda bilgisayarın çözüme nasıl devam edeceği bildirilmiş olmalıdır. Bir algoritma sonluluk, kesinlik ve genellik özelliklerine sahip olmalıdır.

1.4. Problem Çözme

Bir problemin çözümünün hazırlanabilmesi için aşağıdaki adımlara dikkat edilmelidir.

a. Problemin tanımlanması

Her şeyden önce çözülecek problem tam olarak anlaşılmalıdır. Yanlış anlaşılmış bir problemin çözümü yanlış olacak ve istenileni vermeyecektir. Bu adımda yapılacak en ufak bir hata daha sonraki adımların yeni baştan yapılmasını gerektirebilir. Problemin tanımı yapılırken verilen bilgiler, anlamları ve birbirleri ile ilişkileri iyi düşünülmelidir. Daha sonra istenenler

belirlenmeli ve bunların verilen bilgiler ile ilişkileri keşfedilmelidir. Son olarak sonuca ulaşmak için yapılacak ara işlemler belirlenmelidir.

b. Algoritma geliştirme

Algoritma bir problemin çözümü için izlenecek yolun tanımıdır. Problem tanımını tam olarak yaptıktan sonra çözüm için yol aramak gerekir. Genellikle bir problemin birden fazla çözüm yolu olabilir. Bunlardan en uygunu seçilmeye çalışılır. Problem ne kadar karışık olursa olsun alt parçalara ayrılabilir. Problem alt parçalara ayrılarak her bir alt parçanın çözümü ayrı yapılır. Dikkat edilmesi gereken şey parçalar arasındaki ilişkinin korunmasıdır.

c. Girdi ve çıktı biçimi belirleme

Sonuçların dış ortama (kullanıcıya) aktarımı en uygun biçimde yapılmalıdır. Program çıktısı olarak alınması istenilen raporun biçimi belirlenmelidir. Bir rapor biçimi tasarlanırken anlaşılabilir ve kullanılabilir olmasına özen gösterilmelidir.

d. Akış şemasını çizme

Akış şeması, kısaca algoritmanın şemalarla gösterilmesidir. Algoritma geliştirildikten sonra daha iyi anlaşılabilir olması ve programlama dillerine aktarımı daha kolay olması nedeniyle akış şeması haline getirilir. Böylece problemin çözüm basamakları, birbirleri ile ilişkileri ve bilgi akışı daha kolay görülebilir ve yanlışlıklar düzeltilebilir.

e. Kodlama

Problem çözmenin en önemli adımı algoritma geliştirmedir. Algoritma doğru hazırlandıktan sonra akış şemasının çizilmesi daha basittir. Akış şemasından sonra kodlama için kullanılacak programlama dilinin hangisi olduğu çoğunlukla önemli değildir. Problemde yeni teknolojiye bağımlı bir gereksinim yoksa yeni veya eski bir programlama dili kullanmak bile önemli olmayabilir. Yine de problemin yapısına uygun bir programlama dili seçilmesi gerekebilir. Akış şemasındaki her şematik gösterimin seçilen programlama dilinin bir veya birkaç komutuna karşılık geldiği söylenebilir. Bu yüzden eğer problemin akış şeması doğru çizildiyse problemin programlanması çok kolay olacaktır.

f. Programı sınama

Program kodlandıktan sonra sonuçları daha önceden bilinen veriler girilerek eldeki sonuçlarla çıkan sonuçlar karşılaştırılır. Programın doğru çalışıp çalışmadığı sınanır. Bu kontrol aşaması çok uç örnekler için tekrar edilmelidir. Çünkü bir algoritma, problemin geneline çözüm üretmelidir.

1.5. Değişken Tanımlama

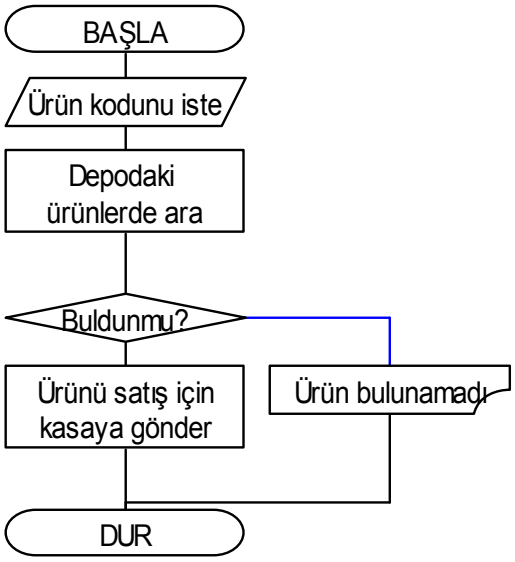
Belirli bir tanım aralığında farklı değerler alabilen sembollere değişken denmektedir. Bilgisayar, işlem yaparken RAM belleği kullanır. Program çalıştırılırken problemin çözümü için gerekli olan bilgilerin RAM bellekte tutulması değişkenler sayesinde mümkündür. Değişkenlerin RAM bellekte bulunuşları aynen bir sinema salonunda koltuklara film bitene kadar oturmuş seyirciler gibi düşünülebilir. Aynen sinemada bilet alındığında koltuk numarası ile ilgili yere ulaşılabildiği gibi bir değişken

tanımlandığında da RAM bellekte bir yer ayrılır ve bu yere değişkenin ismiyle ulaşılır. Program içinde kullanılacak olan değişkenler problemin yapısına göre algoritma tasarlama aşamasında belirlenmelidir. Değişken isimlerini tanımlarken (veya ilk kullanımında) Türkçe karakter (ç, Ç, ğ, Ğ, ı, İ, ö, Ö, ş, Ş, ü, Ü) kullanmamaya dikkat edilmelidir. İngilizcede küçük ı harfinin ve büyük İ harfinin olmadığını unutmamalıyız. Değişken isminin ilk harfi İngilizce bir karakter olmalıdır. Diğer harfleri ise rakam, harf veya özel bazı karakterler olabilir (x35_c7, abc9x12,…).

2. Akış Şeması

2.1. Akış Şeması Mantığı

Akış şeması bir işin başlangıcından sonlandırılmasına kadar uygulanması gereken işlemlerin şematik bir biçimde gösterilmesidir. İş akışının grafiksel gösterimi ilk kez Frank Gilbreth tarafından 1921'de kullanılmıştır. Daha sonra endüstri mühendisliğinde Gilbreth'in araçları oldukça benimsenmiştir. Temel olarak başlangıç, bitiş, yapılması zorunlu olan işlemler, bazı durumlarda karşılaşılan ikilemler ve işin akış yönünü gösteren oklardan ibarettir. Akış şemaları günümüzde çeşitli alanlardaki işlem ve programların yönetilmesi, raporlanması, tasarlanması ve analiz edilmesinde yaygın bir şekilde kullanılmaktadır.



Yanda ürün koduna göre telefonla sipariş alan bir mağazanın işleyişi şematik olarak gösterilmiştir.

2.2. Matematiksel Akış Şeması

Diğer alanlarda kullanılan akış şemalarından farklı olarak bilgisayar ortamında matematiksel bir problemin çözümünü tanımlayan akış diyagramlarında basit cümleler yerine matematiksel ifadeler, denklemler ve işlemler kullanılmalıdır. Bir bilgisayar programının oluşturulmasında akış diyagramlarının hazırlanması algoritma oluşturma aşamasından sonra gelmektedir. Bilgisayar programının oluşturulması sırasında algoritma aşaması atlanarak doğrudan akış diyagramlarının hazırlanmasına başlanabilir. Programlama tekniğinde önemli ölçüde yol almış kişiler bu aşamayı da atlayarak direkt olarak programın yazımına geçebilirler. Akış şemalarının algoritmadan farkı adımların simgeler şeklinde olması ve adımlar arasındaki ilişkilerin oklar ile gösterilmesidir. Daha iyi anlayabilmek için aşağıda basit matematiksel işlemleri kullanarak oluşturulan örnek problemleri inceleyelim.

Örnek 1: Kullanıcı tarafından verilen iki sayının toplamının ekrana yazdırılması istenirse;

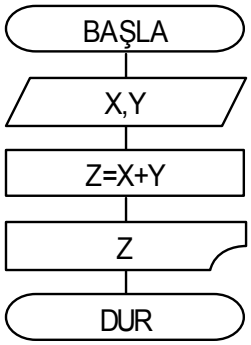
Çözüm: Öncelikle aklımızda belirdiği şekliyle problemin adımlarını yazalım.

- İki değişken kullanarak bu iki değişkenin içeriklerini dışarıdan istemeliyiz. Örneğin bu iki değişken X ve Y olabilir.
- X ve Y değişkenlerini toplayıp Z gibi üçüncü bir değişkene aktarmalıyız.
- Z değişkeninin değerini ekrana yazdırmalıyız.

Şimdi açık şekilde yazdığımız bu adımları biraz daha basit ve terimsel olarak ifade edelim.

- X ve Y değişkenlerini iste
- Toplamını al. Z = X + Y
- Z değişkenini yaz

Şimdi de şematik olarak göstermek için akış şemasını çizelim.



Örnek 2: Dışarıdan girilen üç sayının ortalamasını ekrana yazdırılması istenirse;

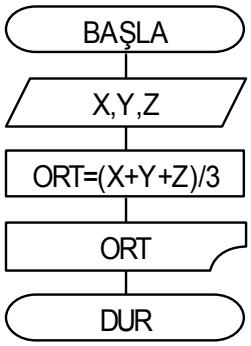
Çözüm: Problemin adımlarını anlaşılabilir bir dille yazalım.

- Üç değişken kullanmalıyız ve bu değişkenlerin içeriklerini dışarıdan istemeliyiz. Örneğin bu değişkenler X, Y ve Z olabilir.
- Bu üç değişkenin ortalamasını bulmalıyız. X, Y ve Z değişkenlerinin toplamını bulup 3'e bölersek ortalamasını bulmuş oluruz. Bulduğumuz ortalama için de bir değişken kullanmalıyız (ORT olabilir).
- ORT değişkenini ekrana yazdırmalıyız.

Şimdi açık şekilde yazdığımız bu adımları biraz daha basit ve terimsel olarak ifade edelim.

- X, Y ve Z değişkenlerini iste
- Ortalama bul. ORT = (X + Y + Z) / 3
- ORT değişkenini yaz

Şimdi de şematik olarak göstermek için akış şemasını çizelim.



Örnek 3: Dışarıdan girilen sayının mutlak değerinin ekrana yazdırılması istenirse;

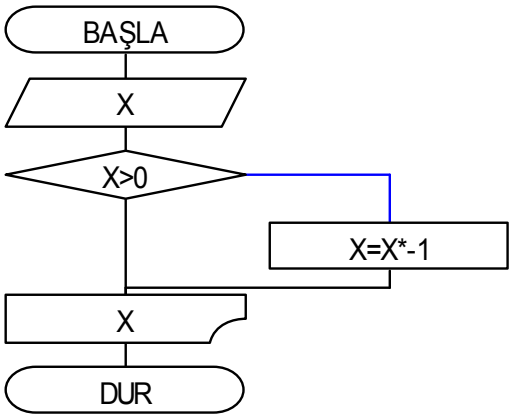
Çözüm: Öncelikle aklımızda belirdiği şekliyle problemin adımlarını yazalım.

- Tek değişken kullanmalıyız. Bu değişkeni dışarıdan istemeliyiz. Örneğin bu değişken X olabilir.
- Eğer X değişkeninin değeri sıfırdan büyükse bir sonraki adıma geçebiliriz, ama değilse değişkeni -1 ile çarpıp pozitif sayı haline getirmeli ve bu çarpım değerini yine X değişkenine aktarmalıyız.
- X değişkeninin değerini ekrana yazdırmalıyız.

Şimdi açık şekilde yazdığımız bu adımları biraz daha basit ve terimsel olarak ifade edelim.

- X değişkenini iste
- Eğer X > 0 ise en son adıma git, değilse bir sonraki adıma git
- Değişkeni pozitif yap. X = X * -1
- X değişkenini yaz

Algoritmanın akış şemasını çizelim.



3.AFC Programı

3.1. Araç Menüsü

AFC programının üstündeki butonlardan oluşan araç menüsü ile aşağıdaki işler yapılabilir;

- Dosya Aç : kaydedilmiş afc dosyaları ekrana yüklenir.
- Dosya Kaydet : mevcut çalışma, afc formatında kaydedilir.
- Resim Olarak Kaydet : çalışmanın ekrandaki görüntüsü emf grafiği olarak kaydedilir.
- Ekranı Temizle : mevcut çalışma, bellekten ve ekrandan temizlenebilir.
- Çalıştır : üzerinde çalışılan algoritma çalıştırılmaya başlanabilir.

- Adımlı İlerle : akış şeması üzerinde adımlı ilerleme sağlanabilir.
- Durdur : çalışmakta iken program akışı durdurulur.
- Sürekli-Adımlı Mod : programın çalışma akışının yavaş veya hızlı olmasını belirler.
- Kaydırmalar : ekrandaki çalışmanın tamamını hareket ettirir.
- İfadeleri Gizleme : çalışmanın nesne içi ifadelerini gösterir veya gizler.
- AFC Nesneleri : algoritmik akış şeması ortamında herhangi bir problemi çözmek için gerekli sembolleri içerir.

3.2. AFC Nesneleri

Araç menüsündeki akış şeması nesneleri sırasıyla şu nesnelerden oluşmaktadır;

- Seçme nesnesi,
- Başla-dur nesnesi,
- Veri giriş nesnesi,
- İşlem nesnesi,
- Döngü nesnesi,
- Karşılaştırma nesnesi,
- Yazdırma nesnesi

Bu menüden akış şeması nesneleri seçilerek çalışma alanına istenilen programın algoritması tasarlanıp adımlı çalışması test edilebilir.

3.3. Pop-Up Menüsü

Çalışma alanındaki nesnelerden birisine fareyle sağ tıklanınca karşımıza çıkan menüdür. Nesne Bağla, Yanlışsa Bağla, Döngü İçi Bağla, Nesneyi Sil, Tüm Bağlarını Temizle seçeneklerinden oluşmaktadır.

Nesne Taşıma İşlemi: Herhangi bir nesne, farenin sol tuşuyla seçilip sürüklenerek yeri değiştirilebilir.

Nesneleri Bağlama İşlemi: Çalışmaya başlarken ilk nesne olarak her zaman "Başla-Dur" nesnesi eklenmelidir. İkinci ve daha sonraki her nesneye bağlantı yapılabilmesi için nesne seçilerek sağ tıklanır ve pop-up menüsünden "Nesneyi Bağla" seçeneği seçilir. Daha sonra başka bir yere tıklanmadan bağlantının yapılacağı nesneye tıklanınca nesneler ardışık çalışmak üzere birbirine bağlanmış olur.

Karşılaştırma Yanlışsa Bağlama İşlemi: Eğer seçilen nesne "Karşılaştırma" nenesi ise pop-up menüsünden "Yanlışsa Bağla" seçeneği yardımıyla karşılaştırma işleminin olumsuz olduğu zaman program akışının hangi nesne ile devam edilebileceği seçilip bağlanır.

Döngü İçini Bağlama İşlemi: Eğer seçilen nesne "Döngü" nenesi ise pop-up menüsünden "Döngü İçini Bağla" seçeneği yardımıyla döngü işleminin içerisinde program akışının hangi nesne ile devam edilebileceği seçilip bağlanır.

Nesne Silme İşlemi: Seçilen nesne silinecekse pop-up menüsünden "nesneyi sil" seçeneği seçilip sorulan soru onaylanır. Böylece mevcut nesne silinerek onunla ilgili tüm bağlantılar da silinmiş olur.

Nesne Bağlantılarının Silinmesi İşlemi: Eğer nesne değil de üzerinden çıkan tüm bağlantılar silinmek isteniyorsa o zaman pop-up menüsünden "Tüm Bağlarını Temizle" seçeneği seçilir.

3.4. Uygulama İşleyişi

Nesne İçi İfadelerinin Yazılması İşlemi: Nesnenin yapacağı işi temsil eden ifadeyi yazmak için nesneye çift tıklanınca aktif olan metin kutusu kullanılır. İfade yazılıktan sonra hemen altındaki buton tıklanır. Her nesnenin içine yazılan ifade için o nesneye ait olan yazım kuralı kullanılmalıdır. Aksi halde çalıştırma esnasında program akışı hata ile durdurulur.

Program Çıktıları, Uyarılar ve Direktifler: Sağ üstte duran formun kara ekranlı panosunda yapılması gereken işlemler, düzeltilmesi gereken hatalar ve akış şeması çıktıları görüntülenir.

Veri Girişi: Sağ üstteki formun en altındaki metin kutusundan istenen bilgilerin girilmesi sağlanır. Bu kutu sadece "Veri Giriş" nesnesi çalıştırılırken aktif olur ve veri girilip sağındaki buton tıklanınca yeniden pasif hale geçer.

3.5. Çalıştırma ve Adımlı İzleme

Çalıştırma İşlemi: Araç kutusundaki çalıştırma butonuna tıklanınca akış şeması adımlı çalışma için başlatılır. Bu işlem ile birlikte çalıştır butonu ile birlikte tüm araç kutusu butonları görünmez olur. Sadece adımlı ilerleme ve yardım butonları aktif halde kalır.

Adımlı İlerleme İşlemi: Çalışma anında ilerleme butonuna her tıklamada duruma göre işlenmesi gereken nesne seçilir ve kırmızı renkle belirginleştirilir. Aynı zamanda sağ alttaki formda o anda bellekte kullanıma ayrılmış değişkenlerin isimleri ve anlık değerleri görüntülenir. Böylece her nesnenin, kendi sırası geçinceye kadar bellekte ne gibi değişiklikler yarattığı görüntülenmiş olur.

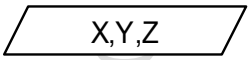
3.6. Yazım Kuralları

Nesnelerin içerisinde yazılı olan ve görevini temsil eden ifadeleri bazı kurallar doğrultusunda yazmamız gerekmektedir. Bazı örnekler aşağıda verilmiştir.

Başla nesnesi

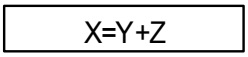
Dur nesnesi

Veri giriş nesnesi için;



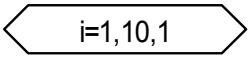
- X, Y, Z
- degiskenA

İşlem nesnesi için;



- z= x%2 ('%' işareti kalanlı bölme işlemindeki kalan değerini bulmak için kullanılıyor)
- a=b\10 ('\' işareti kalanlı bölme işlemindeki bölüm değerini bulmak için kullanılıyor)

- degisken=a*b+(c^3*d)-x/y%10\4

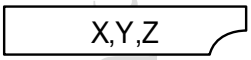


Döngü nesnesi için;

- i=1,10,1
- y=x+1,z*b,1-c

Karşılaştırma nesnesi için;

- x+1=a*b-3 (eşit ise)
- a^2>=5 (büyük eşit ise)
- 89<z+9*x (küçük ise)
- Z<>5 (eşit değil ise)



Yazdırma nesnesi için;

- a, b, c
- degiskenX
- S, ‘sayısı asaldır’
- ‘Sonuç:’, Z+Y

3.7. Kısayol Tuşları

Ctrl + 1	Başla/Dur nesnesini seç
Ctrl + 2	Veri giriş nesnesini seç
Ctrl + 3	İşlem nesnesini seç
Ctrl + 4	Döngü nesnesini seç
Ctrl + 5	Karşılaştırma nesnesini seç
Ctrl + 6	Yazdırma nesnesini seç
Ctrl + (nokta)	Çalışmayı, nesnelerin içinde ifadeleri yazarak göster

Ctrl + O	AFC dosyası aç
Ctrl + S	Çalışmayı AFC formatında kaydet
Ctrl + X	Çalışmanın görüntüsünü EMF grafiği olarak kaydet
Ctrl + F2	Çalışmayı durdur
Ctrl + Del	Çalışma ekranını ve belleği temizle
Ctrl + Shift + Del	Ekranı en son temizleme işleminden önceki çalışmayı yükle
Alt + Enter	Ana formu tam ekran yap
Del	Seçili nesneyi sil
F1	Program hakkında
F5	Çalışmayı çalıştır
F8	Çalışma çalışmaya başladıktan sonra adım adım ilerlet
F9	Çalışmayı sürekli veya adımlı yap
F12	Tam ekran çalışma, İzleme ve Ortam pencereleri görünmez
Ctrl + Sol Tıklama	Çalışma alanını büyütür – Zoom In
Ctrl + Sağ Tıklama	Çalışma alanını küçültür – Zoom Out

3.8. OCX Kaydı

Programda kullanılan bazı VB bileşenleri için kurulum klasöründe verilen "comdlg32.ocx", "msflxgrd.ocx" ve "mscomctl.ocx" dosyalarını işletim sistemine tanıtmak gerekmektedir. Bunun için "register.bat" dosyasını çalıştırmanız yeterlidir.

Windows Vista ve 7 için "register.bat" dosyası doğrudan çalıştırılamayabilir. Bunun yerine bu dosyaya sağ tıklayıp çıkan menüden "Yönetici olarak çalıştır" veya "Run as Administrator" başlığı seçilerek çalıştırılmalıdır.

Çalıştırma sonrası çıkacak onay pencerelerini takip ediniz. Olumsuz uyarı veren mesajlar olursa programın çalışmasında sorunlar çıkabilir. Sorularınız için irtibat umutorhan@hotmail.com adresine e-posta yazabilirsiniz.

4. Temel Problemler

4.1. Gauss Toplamı

Problem adını 1777–1855 yıllarında yaşamış olan ünlü Alman kökenli matematikçi ve bilim adamı olan Carl Friedrich Gauss'tan almıştır. Henüz üç yaşındayken babasının kağıt üzerinde yaptığı hesapları Gauss'un kafasından kontrol edip düzelttiği söylenir. Bahsedeceğimiz problem, Gauss'un ilkokul öğretmeni tarafından öğrencilerini oyalamak için sorduğu 1'den 100'e kadar olan sayıların toplamını bulma sorusuyla ortaya çıkar. Gauss cevabı birkaç saniye içinde bularak herkesi şaşırtır. Sayı listesinin iki zıt ucundan birer sayı alıp topladığında hep aynı sonucun çıktığını fark etmişti. (1 + 100 = 101, 2 + 99 = 101, 3 + 98 = 101, ...) Böylece 1'den 100'e kadar olan sayıların toplamı 50 × 101 = 5050 oluyordu. Bunu formüle dökerek 1'den istenen bir N sayısına kadar olan sayıların toplamını $N\times(N+1)/2$ ile temsil etmiştir.

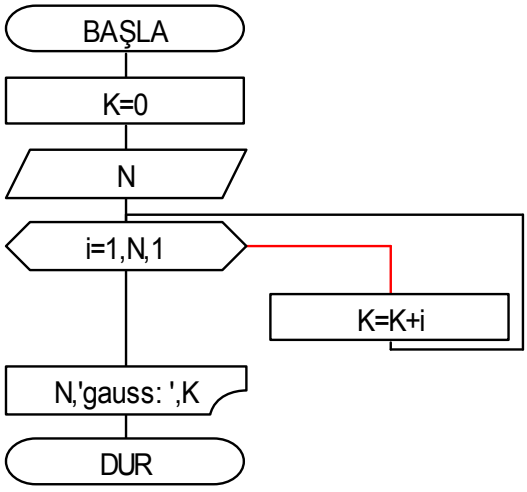
Uzun sürebilecek bir işlemi insan beyni hemen başka bir kısa yoldan yapmayı ister. İşte bu yüzden bir işi formüle etmek hem zaman hem de güç kazancı sağlar. Fakat biz problemleri bilgisayar gibi çok üst düzeyde matematiksel işlem kapasitesine sahip bir cihazla uygulayacağımız için problemleri formüle etmekten ziyade problemi bilgisayara aktarmakla uğraşacağız.

Tekrarlı Toplama

İnsan beyninin Gauss formülüyle basitçe hesaplayabileceği 1'den N'e kadar olan sayıların toplamı problemini bilgisayarda tekrarlı toplama dediğimiz bir yöntemle hesaplayacağız. Tekrarlı toplamayı örnekle anlatmak gerekirse

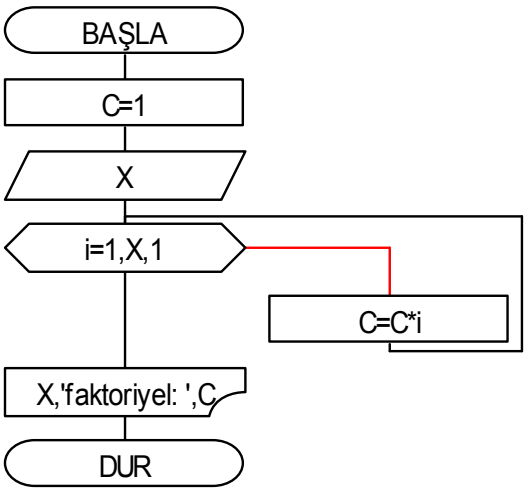
bir kumbarada para biriktirmeye benzetebiliriz. Eğer kumbaranın içinde ne kadar para biriktirdiğinizi bilmek istiyorsanız kesinlikle ilk başta kumbaranın boş olduğundan emin olmalısınız. Her sefer kumbaraya attığınız para, kumbaranın o an içerdiği para miktarına eklenerek değerini arttıracaktır.

Örnekten esinlenerek kumbarayı temsilen *K* değişkenini kullanacağız ve işlemin hemen başında bu değişkene kumbaranın içini boşaltmayı temsilen 0 değerini vereceğiz. Burada yapacağımız işlem toplamayla ilgili olduğu için *K* değişkeni toplama işlemine göre etkisiz eleman olan değerle başlamalıdır. Daha sonra bir döngü yardımıyla bu *K* değişkenine sırasıyla 1,2,3,…,*N* değerlerini ekleyeceğiz. Döngünün içerisindeki *i* değişkeni her seferinde *K* değişkenine eklenecek değeri içerdiği için tekrarlı toplamada kullanacağımız arttırma miktarını *i* temsil edecektir. Bu işlem şekildeki akış şemasıyla gösterilmiştir.



4.2. Faktöriyel İşlemi

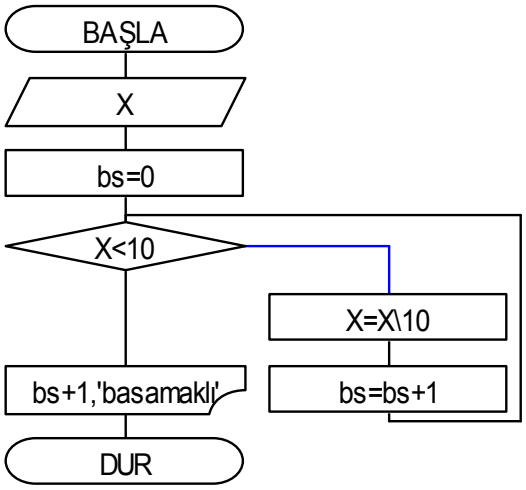
Bir pozitif tamsayının faktöriyel değeri, 1 ile o sayı arasındaki bütün tam sayıların çarpımını temsil eder. Daha önce bahsettiğimiz tekrarlı toplama işlemine çok benzer bir akış şemasına sahiptir. Tek farkı toplama işlemi yerine çarpma işleminin kullanılmasıdır. Tekrarlı çarpma işleminde ana değişkenin ilk değeri 0 yerine çarpma işleminin etkisiz elemanı olan 1 olmalıdır. Burada tekrarlı çarpma değişkeni olarak *C* kullanılmıştır. Şekilde akış şeması verilmiştir.



4.3. Basamak Sayısını Bulma

Dışarıdan girilen bir sayının kaç basamaklı olduğunu bulmak için 10 ile kalanlı bölme işleminden yararlanabiliriz. Herhangi bir sayı 10 ile kalanlı bölündüğünde bölünen sayının birler basamağı kaybolur. Örneğin 89 sayısı 10 ile kalanlı bölünürse (89 \ 10) bölüm 8 olarak bulunur. Kalanlı bölme işlemi, klasik bölme ile karıştırılmamalıdır. Klasik bölmede 89 / 10 sonucu 8.9 rasyonel olarak bulunur. Eğer sayı bölüm sonucu 10 sayısından küçük

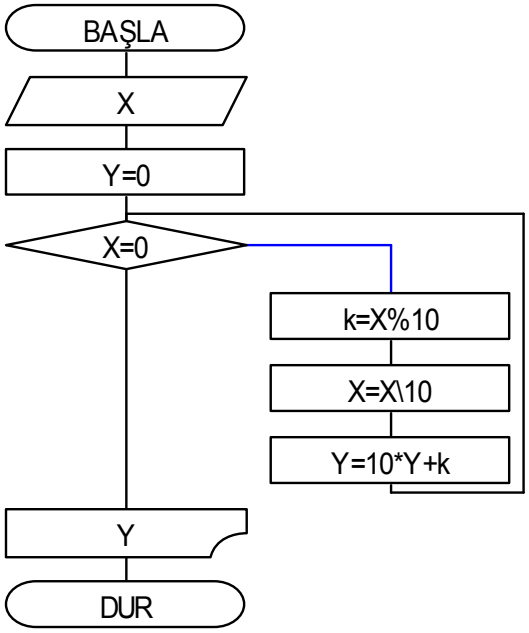
olana kadar sürekli olarak 10 ile kalanlı bölünürse ve basamak sayısını tutan bir tekrarlı toplama değişkeni (bs) her bölme yapıldığında 1 arttırılırsa problem çözülmüş olur. Daha önce 10 ile kalanlı bölme yapıldığında sayının birler basamağı silinir demiştik. Her bölme yapıldığında silinen birler basamağı için bs değişkenine 1 eklersek kaybolan basamak sayısını saklamış oluruz. Bir döngü içinde bu işlemi sürekli yapmamız gerekir. Döngünün bitiş şartı da bölümün 10 sayısından küçük olması durumudur. Daha sonra ekrana bs değişkeni yazdırılır ve program sonlandırılır.



4.4. Sayının Basamaklarını Ters Çevirme

Dışarıdan girilen bir sayının rakamlarını tersine çevirmeye çalışalım. Örneğin 123 sayısının 321 sayısına dönüştürülmesi gibi. Öncelikle dışarıdan girilen *X* sayısını basamaklarına ayırmamız gerekir. *X* sayısının 10 ile bölümünden kalanı, *X* sayısının birler basamağıdır. Benzer şekilde bölümü ise *X* sayısının birler basamağı silinmiş halidir. Bir döngü içinde bu iki işlem *X* sayısı sıfır olana kadar yapılırsa tüm basamaklar bulunmuş olur. Basamak değerleri elde ediş sırasına göre *Y* sayısına eklenir ve her eklenişte *Y*

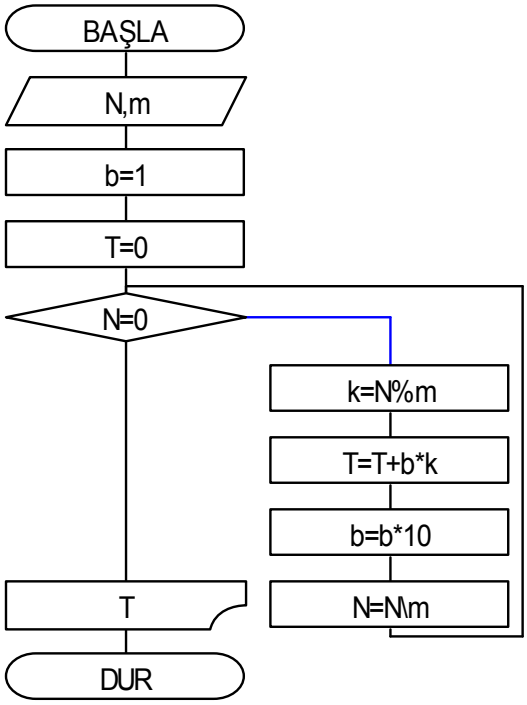
değişkeni 10 ile çarpılır. Bu şekilde döngü bitiminde *Y* değeri *X* sayısının basamakları ters dönmüş şekli olacaktır.



4.5. Sayı Sistemi Dönüşümü

Matematiksel işlemlerde özel bazı hesaplamalar ve fonksiyonlar hariç genelde onluk sayma sistemi kullanılır. Ama onluk sayma sistemini kullanan insan beyni ile başka bir sayma sistemini kullanan bilgisayar gibi araçlar arasında bazen sayı sistemi dönüşümü yapılarak iletişim kurulması gerekir. Örneğin bilgisayar tüm hesaplamaları ikilik sistemde yapmasına rağmen arka planda gizlice yaptığı sayı dönüşümü sayesinde ekrana sayı değerlerini onluk sayma sisteminde yazar. Bu problemin çözümünde dönüştürülmesi istenen onluk sistemdeki sayıyı *N* değişkeniyle ve dönüştürülecek sayı sistemini *m* değişkeniyle gösterelim. *N* sayısını sürekli olarak *m* değerine bölüp, bölme işlemlerindeki kalan değerleri yan yana

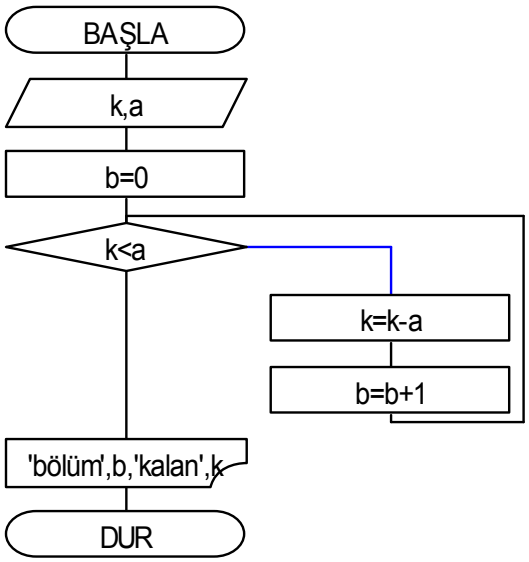
yazarsak çözümü üretmiş oluruz. Burada en önemli nokta yan yana yazma işleminde yine matematiği kullanmak zorunda oluşumuzdur. Bunun içinde bulduğumuz kalan değerlerini sanki onluk sistemdeki rakamlarmış gibi 10 ile çarpıp bir tekrarlı toplama değişkeni yardımıyla biriktirmektir.



4.6. Çıkartma İşlemiyle Bölme Benzetimi

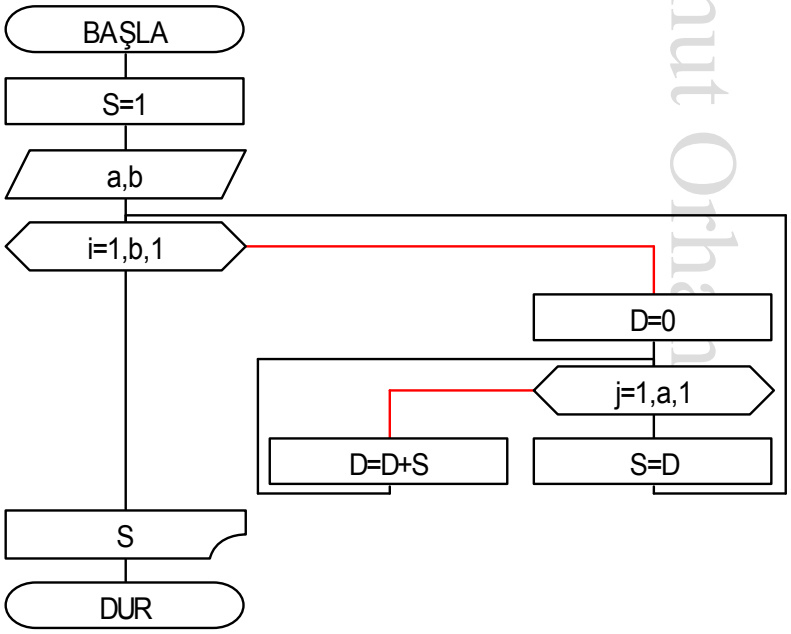
Bir işlemin başka fonksiyonlar yardımıyla gerçekleştirilmesine benzetim (simülasyon) diyoruz. Bu konuda çarpmanın toplamayla ve benzer şekilde bölmenin çıkartmayla benzetimi verilebilecek en temel örneklerdir. Temsili olarak bölme işleminin çıkartma işlemiyle benzetimini inceleyelim. Bölme işlemi, bölünen sayı içerisinde bölen sayıdan kaç adet olduğunu öğrenmemizi sağlar. Çıkartma işlemiyle bu benzetimi nasıl yapabileceğimizi düşünmeden önce çarpma işlemini hatırlayalım. Basit bir

çarpma işlemi örneği olarak 3x4 işlemini toplamayla benzetirsek 4+4+4+4 şeklinde yazabiliriz. Bir döngünün 3 kez dönmesini sağlayıp tekrarlı toplamayla 4 rakamlarını toplarsak problemi çözmüş oluruz. Bölme işlemi de çarpmanın tersi demiştik. O halde 12 / 4 işlemi için 12–4=8, 8–4=4, 4–4=0 benzetimini yazabiliriz. Burada yaptığımız çıkartma işlemi sayısı bölüm olmaktadır. Bu problemin çarpma benzetiminden farkı çıkartma işleminin kaç kez yapılacağı bilinmiyor olmasıdır. Bu yüzden klasik bir döngü nesnesi kullanamayız. Bu problemin çözümü için bir döngü içinde sürekli çıkartma işlemi yapmamız, bu döngüyü çıkartma işleminin sonucunu saklayan değişkenin değeri bölen sayıdan küçük olana kadar devam ettirmemiz ve bu döngünün kaç kez döndüğünü tutan bir sayaç değişkeni kullanmamız gereklidir.



4.7. Toplama İşlemiyle Üs Alma Benzetimi

Bu problem içerisinde saklı iki ayrı alt problem bulunmaktadır. Öncelikle problemimizi iki alt parçaya bölmemiz gereklidir. Benzer şekilde karşılaştığımız diğer problemlerde de sorunu küçük parçalara bölebilirsek çözmemizi kolaylaştıracaktır. Hatta bazen ayırdığımız küçük parçalardan birisi tanıdığımız bir çözüme sahip olabilir. Bu problemde üs alma işleminin toplama ile benzetimi istenmektedir. Problemi üs alma işleminin çarpma ile benzetimi ve çarpmanın toplamayla benzetimi olarak iki alt başlık şekliden düşünelim. Çarpmanın toplamayla benzetiminden yukarıda kısaca bahsetmiştik. Döngünün kaç kez döneceğini çarpanlardan birisi ile temsil edebileceğimiz için klasik döngü nesnesini kullanabiliriz. Döngü içindeki tekrarlı toplama işleminde artış değerinin de diğer çarpan ile temsil edebiliriz.



Üs alma işleminin çarpma benzetiminde ise benzer şekilde tekrarlı çarpma yerine tekrarlı çarpma işlemi kullanılacaktır. Döngünün tekrarlanma sayısını üs değerini, tekrarlı çarpmadaki etkin çarpan içinse taban değerini kullanabiliriz. Öyleyse iki problemi iç içe düşünerek bu problemi çözebiliriz. Burada dikkat edilecek en önemli nokta tekrarlı toplama sonucunun tekrarlı çarpma işlemindeki etkin çarpanı belirlemesidir. Dolayısıyla her tekrarlı toplama döngüsünün bitişinde tekrarlı toplama değişkeni değerini artış değerini gösteren değişkene aktarmamız gerekmektedir.

4.8. Asal Sayı Bulma

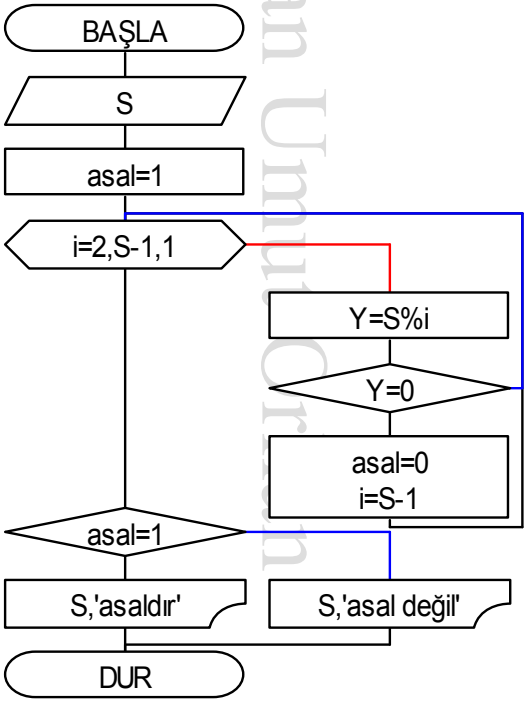
Matematiğin en güzel ve önemli alanlarından biri de sayıları inceleyen sayılar teorisidir. Her ne kadar matematik çok eski bir bilim dalı olsa da günümüzde bile çözülememiş ilginç problemleriyle sayılar teorisi alanı halen en ilgi çeken konulardan birisidir. İlgi çeken konulardan birisi de asal sayılardır.

Asal Sayı Tanımı

Pozitif bir tamsayı kendinden ve 1'den başka bir tamsayıya bölünemiyorsa bu sayı asaldır denir. 1'den büyük asal olmayan sayılara bileşik sayılar denir. Örneğin, 2, 3 ve 5 sayıları asal iken 6 bileşik sayıdır. Tüm pozitif tamsayılar en az bir tane asal bölene sahiptir. Sayı asal ise asal böleni kendisidir, sayı bileşik ise asal çarpanlarına ayırarak bulunur.

Kendinden ve 1'den başka hiçbir sayıya bölünemeyen sayılar asaldır. Bu cümleyi örnekli olarak biraz irdeleyelim. Asal olup olmadığını araştırdığımız sayı *S* olsun. Her tamsayı zaten 1'e ve kendine bölünür. Bunu

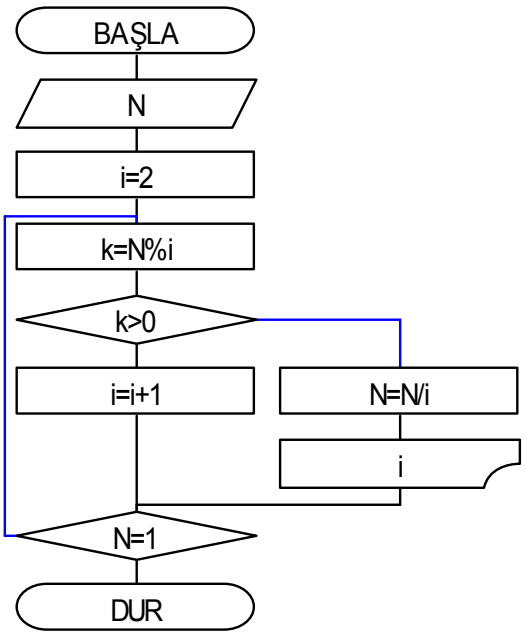
test etmeye gerek yoktur. Öyleyse *S* sayısına asal diyebilmemiz için ne yapmamız gerekiyor? Sayımız *S*=7 olsun, bu durumda kural gereği 1 ve 7 sayıları hariç diğer sayıların hepsini *S*=7 sayısına bölen olarak kullanmalıyız. Asal sayı kavramının pozitif sayma sayıları için geçerli bir kavram olduğunu hatırlarsak araştırmamızı 1 ile +∞ arasındaki tamsayılar için yapmamız gerekir. Dolayısıyla 1'den başlayıp sonsuza kadar giden tamsayılar içersinden kural gereği 1 ve 7 sayılarını hariç tutup diğer sayıların hepsini 7'yi bölmek için denemeliyiz. Herhangi bir sayının kendinden büyük başka bir sayıya tam olarak bölünemeyeceğini bilmemiz araştırma yapacağımız sayılar kümesini biraz daha küçültecektir. Öyleyse elimizde kalan 1 ve 7 arasındaki sayıları incelememiz yeterli olacaktır. Problemi genelleştirme yaparak ifade edersek bir *S* sayısının asal olup olmadığını bulabilmek için [2 *S*-1] sayı aralığındaki her sayıyı *S* sayısına bölen olarak seçmeliyiz. Herhangi bir bölen sayı tam bölme işlemi yapabilirse diğer bir deyişle kalan 0 olursa, o halde bu bölen yüzünden *S* sayısı asal değildir deyip işlemi bitirmeliyiz. Aksi takdirde tüm sayılar için denemelere devam etmeliyiz. Eğer bölen sayıların hiçbirisi tam bölme işlemini



sağlayamazsa *S* sayısı asaldır diyebileceğiz. Tam bölmeyi temsilen matematikte kalanlı bölme işlemindeki kalan değerini bulmaya yarayan *mod* (modülasyon) işlemini kullanabiliriz. AFC programında *mod* işlemini temsilen % operatörü kullanılmaktadır.

4.9. Sayıyı Asal Çarpanlarına Ayırma

Yukarıda asal sayı probleminde bahsettiğimiz bileşik sayılar aslında bazı asal sayıların birbirleriyle çarpımları sonucu bulunurlar. Dışarıdan verilen bir *N* sayısı eğer asal değilse bileşik sayıdır. Bir sayının bileşik olduğunu bulmak asal olduğunu bulmaktan daha kolaydır. Çünkü bir sayının asal olduğundan emin olmak için tüm ihtimaller sonuna kadar denenmek zorundadır. Bu problemde ilk değeri 2 olan bölen sayı *i*, eğer *N* sayısını tam bölemiyorsa diğer bir deyişle mod (%) işlemi sonucu (*k*) sıfırdan büyükse bölen sayı *i* bir arttırılıp döngüye devam edilir.

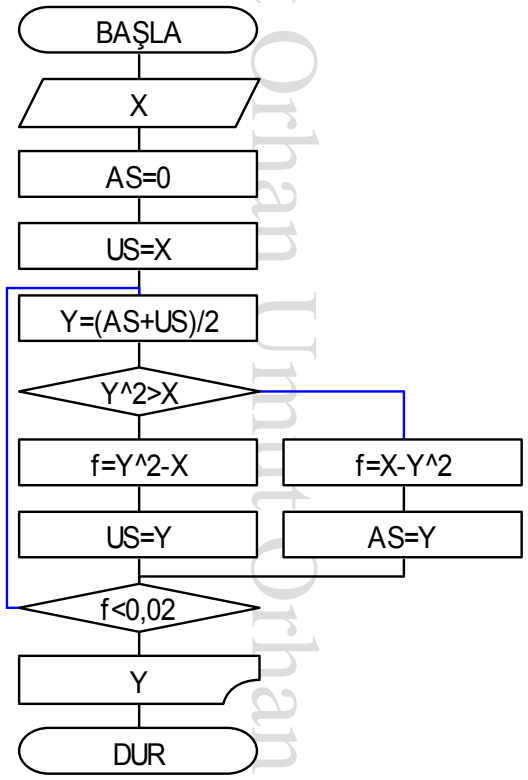


N sayısı 1 olduğunda artık başka bir bölen bulunamayacağı için döngü sonlandırılır. Bu problemde dikkat edilecek en önemli nokta, tam bölünebilme sağlandığında bölen sayı durumdaki asal çarpan *i* değeri arttırılmaz ve *N* sayısını tekrar bölüp bölemediği kontrol edilir.

4.10. İkiye Bölerek Karekök Bulma

Matematikteki en önemli işlemlerden birisi de karekök hesaplamadır. Köklerini bulmak için yapılan hesaplama işlemi $\sqrt{9}$ ve $\sqrt{16}$ gibi tam kare sayılar için oldukça kolay iken $\sqrt{3}$ ve $\sqrt{7}$ gibi tam kare olmayan sayılar için oldukça zordur. Bir hesap makinesine $\sqrt{2}$ sayısının değerini hesaplattığımızda hızlı bir şekilde 1,414214 yanıtını vereceğini görürsünüz. Ancak hesap makinesi bu işlemi bu kadar hızlı ve doğru şeklide nasıl yapmaktadır? Hesap makineleri bu hesaplamayı yapabilmek için hızlı bir algoritma kullanırlar. Ne zaman hesap makinesinin karekök tuşuna dokunsak makine sayısal bir tekrarlama işlemini devreye sokmakta ve sonucu bulmaktadır. Hesap makinesi olmayan yıllarda insanlar sayıların kareköklerini nasıl hesaplıyorlardı? Karekök hesaplama işleminde basit sayısal algoritmalar bulma işinin ilk olarak yaklaşık 4000 yıl önce Babilliler tarafından bulunduğu bilinmektedir. Köklü sayıları hesaplamada Babil metodu bir tekrarlama sürecine dayanmaktadır. Her tekrarda hassas sonuca adımlı olarak yaklaşan bu algoritmaya benzer başka bir basit algoritmadan bahsedeceğiz. İkiye bölme yöntemi başka problemlerde de kullanılan basit bir yaklaşımdır. Temel mantık aranan hedefin bilinen alt ve üst sınırlarının ortasında olduğunu düşünmeye dayanır. Eğer hesaplanan orta değer aranan hedeften küçük ise yeni bulunan bu orta değer alt sınır değişkenine, büyük ise üst sınır değişkenine aktarılarak tekrar edilir. Örneğin $\sqrt{25}$ değeri

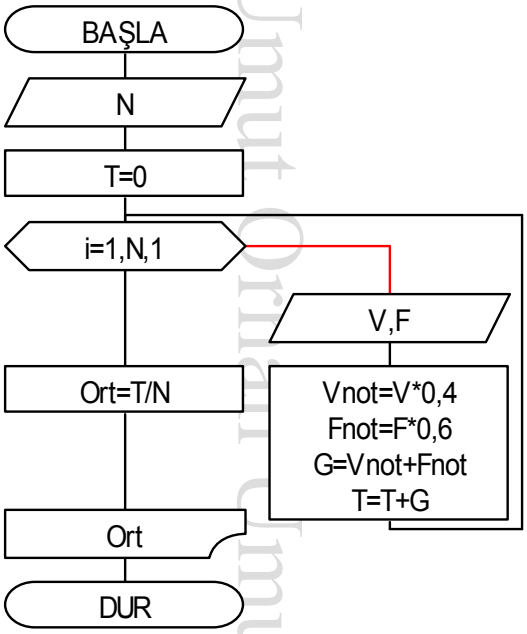
hesaplanmak istensin. Bu durumda başlangıç değerleri olarak alt sınır 0, üst sınır da 25 olur. Aranan hedef (0+25)/2 = 12,5 olarak hesaplanır. 12,5 orta değeri 25 sayısının karekökü müdür? Tam tersine düşünerek soruyu tekrar düşünelim. 12,5 sayısının karesi 25 sayısı mıdır? $12,5^2$=156,25 dolayısıyla $12,5^2 > 25$ olacaktır. Bu durumda ilk değeri 25 olan üst sınır değişkeninin yeni değeri olarak 12,5 atanır. Bir sonraki hedef değer (0+12,5)/2=6,25 bulunur. $6,25^2$=39,0625 değeri 25 değerinden yine büyük olduğu için üst sınır 12i5 iken yeni değeri 6,25 yapılır. Yeni hedef (alt sınır + üst sınır)/2 formülünden 3,125 bulunur. $3,125^2$=9,765625 bulunur. Bu değer de 25 değerinden küçük olduğu için bu kez eski değeri 0 olan alt sınır 3,125 olarak güncellenir ve bu işlem belirlenen hassasiyet sağlanana kadar devam ettirilir. Burada hazırlanan algoritmik akış şemasında hassasiyet değeri 0,02 olarak düşünülmüştür.



5. Dizi Değişkenli Problemler

5.1. Ortalama Bulma

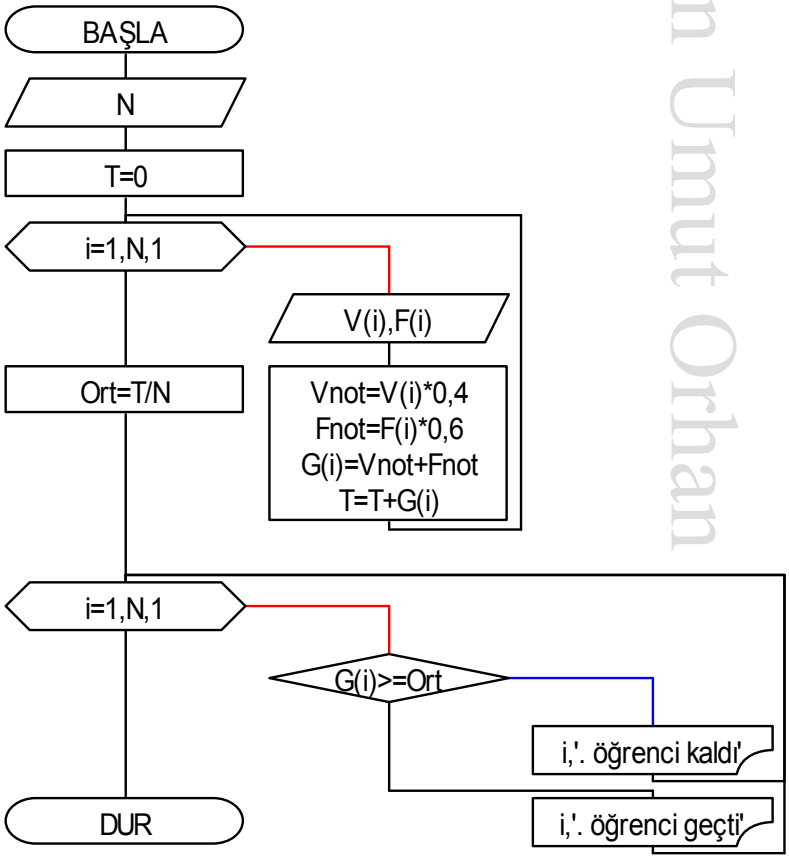
Ortalama bulma, dışarıdan girilen birçok sayının aritmetik ortalamasını bulmayı gerektiren bir problemdir. En sık karşılaşıldığı şekliyle, bir sınıftaki öğrencilerin bir sınavdan aldıkları notların ortalamasını bulmak istediğimizde iki durum söz konusudur. İlk durumda dizi değişkeni tanımlamaya gerek kalmayabilir. Bu durum, problemde istenen sadece sınıfın not ortalaması ise oluşur.



Sağdaki akış şemasına dikkat edilirse öğrenci notları ilk veri girişinde V ve F değişkenlerine saklanıyor, daha sonra bu değerler hesaplanıp T değişkenine ekleniyor. Bir sonraki öğrencinin notları için veri giriş sırası geldiğinde ise yine V ve F değişkenleri kullanılıyor. Dolayısıyla bir önceki öğrencinin notlarına bir daha ulaşmamız mümkün olamıyor. Ama problemde böyle bir ihtiyaç olmadığı için bu bir sorun olmayacaktır.

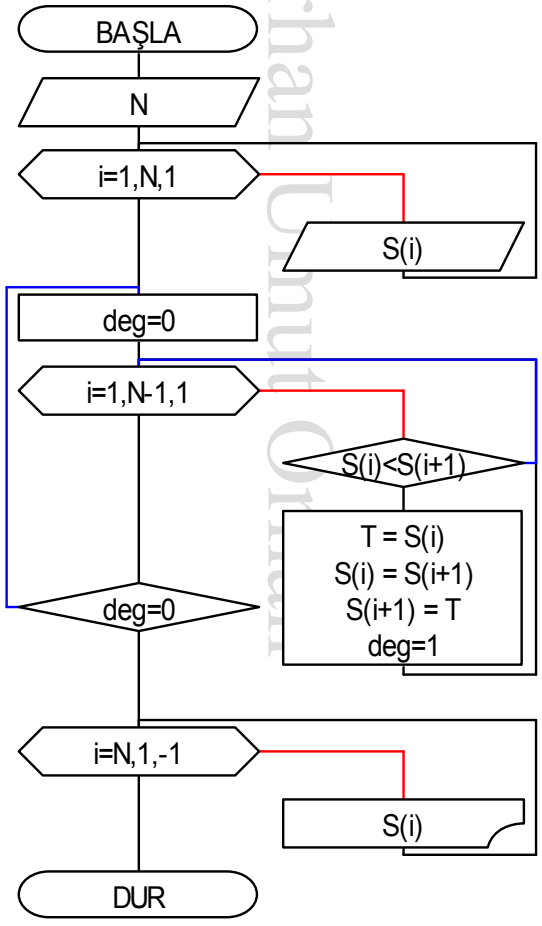
İkinci durumda ise dizi değişkeni tanımlamak kesinlikle gereklidir. Problemde istenen, öğrencilerin notlarının sınıf ortalamasının üzerinde olup

olmadığını kontrol etmek ise öğrenci notlarını iki defa incelememiz ve dolayısıyla notları ayrı değişkenlerde saklamamız gerekmektedir. Dizi değişkeni kullanarak notlar program bitene kadar incelenip kullanılabilir. Tabii öncelikle öğrenci notlarının tek-tek dizi değişkenine girilmesi gerekmektedir. Girilen her değer, dizi değişkeninin ayrı bir indeks değeriyle belirtilen hücresine kaydedilir. Eğer dizi tanımlaması olmasaydı 100 kişilik sınıf için 100 ayrı değişken tanımlamamız gerekecekti. 100 ayrı değişken tanımlayıp kullanmanın zorluğu haricinde farklı sınıf mevcutları için hazırlayacağımız algoritmayı genelleştirmemiz imkansız olabilirdi. Dizi değişkeni kullanılarak hazırlanan akış şeması aşağıda gösterilmiştir.



5.2. Kabarcık Sıralaması

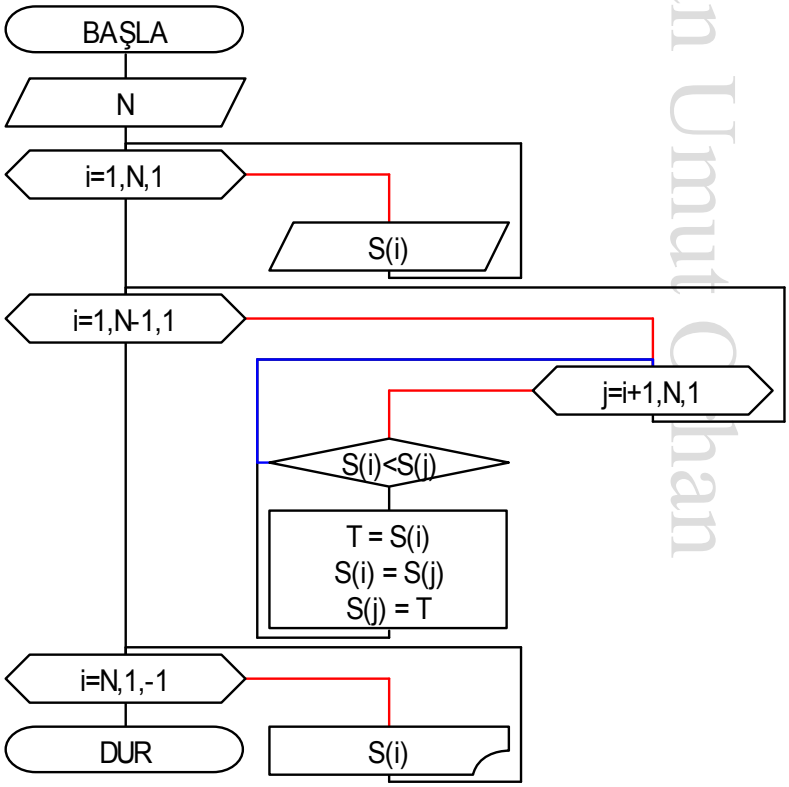
Kabarcık sıralaması, bilgisayar bilimlerinde kullanılan yalın bir sıralama algoritmasıdır. İlk kullanılışı 1950lere dayanmaktadır. Sıralanacak dizinin üzerinde sürekli ilerlerken her defasında iki öğeyi birbiriyle karşılaştırarak öğelerin yanlış sırada olmaları durumunda yerlerinin değiştirilmesi mantığına dayanır. Algoritma, herhangi bir değişiklik yapılmayıncaya kadar dizinin başına dönerek kendisini yineler. Adına *Kabarcık* denmesinin nedeni büyük olan sayıların aynı suyun altındaki bir kabarcık gibi dizinin üstüne doğru ilerlemesidir. Bu ilerleme, seçmeli sıralama algoritmasındaki dizideki değeri küçük olan elemanların dizinin başında kümelenmesi yöntemine benzer şekilde gerçekleşir. Kabarcık sıralaması dizinin başından başlar ve dizi elemanlarını sırayla seçer. Seçilen dizi elemanı kendinden sonra gelen elemandan büyükse bu iki elemanın yerleri değiştirilir. Bu işlem sonucunda dizinin en büyük elemanı dizi sonuna yerleştirildiğinden bir sonraki adımda arama sınırı bir



eleman geri çekilir. Bu işlem, dizinin sonundaki elemanın karşılaştırılmasına kadar yinelenerek sürdürülür.

5.3. Seçmeli Sıralama

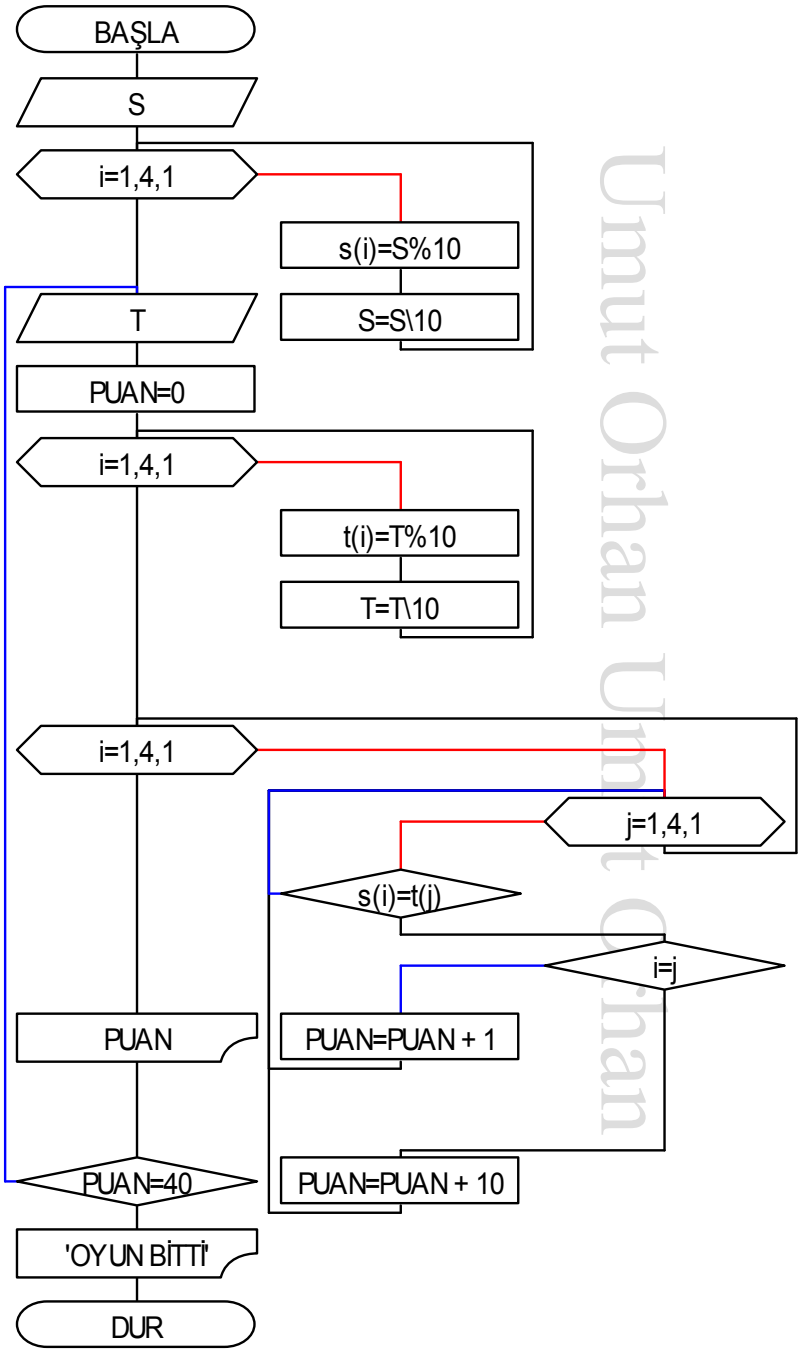
Seçmeli sıralamanın en basit sıralama algoritması olduğu söylenebilir. Elimizdeki dizide sıralanması gereken N tane sayı olsun. Dizinin ilk elemanından başlayarak tüm elemanları kontrol edilerek en küçük sayı bulunur. Bu sayı dizinin ilk sayısıyla yer değiştirir. Daha sonra geriye kalan sayılar için her defasında en küçük sayı bulup sıradaki yerine yerleştirilerek ilerlenir.



Elimizde bulunan her türlü bilgiye kolay ulaşmak için o bilgilerin belirli bir düzen ve sıra içinde olması, aradığımız bilgiye kolay ulaşmak için gerekli olan temel ihtiyaçlardan biridir. Örneğin 50 gözlü bir kitaplık ve 50 kitabımız olsun. Kitaplar raflara rastgele yerleştirildiğinde istediğimiz kitabı bulmamız uzun sürebilir. Ama kitapları raflara sırayla yerleştirirsek istediğimiz numaralı kitaba tek bir rafa bakarak ulaşabiliriz. Benzer şekilde elimizdeki bilgilerin sıralı olması da arama işlemleri için işimizi oldukça kolaylaştıracaktır.

5.4. Sayı Tahmin Oyunu

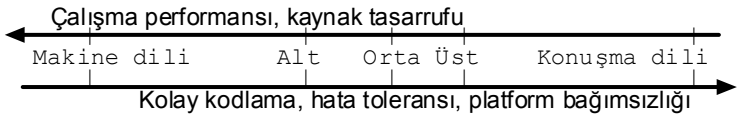
Bir çoğumuzun kağıt ve kalemle oynadığı ilginç bir tahmin oyunudur. Karşılıklı iki oyuncunun ürettiği rasgele sayıları rakip oyuncunun tahmin etmesine ve her seferinde cevap olarak bir ipucu almalarına dayanır. Oyunun tüm detaylarını kağıt-kalemle inceledikten sonra bilgisayarlarla oynanacak şekle getirmek zor olmayacaktır. Birinci oyuncunun bütün rakamları birbirinden farklı bir sayıyı bilgisayara girmesi oyunun ilk adımıdır. Temel amaç ikinci oyuncunun gizli sayılarının her rakamını olduğu basamağı da dikkate alarak tahmin ederek bulmaya çalışmasına dayanır. Eğer tahmindeki sayının rakamları haneleri ile doğru tahmin edilmişse her hane için 10 puan, rakam tahmini doğru ama hanesi tutmamışsa da 1 puan verilir. Tüm rakamlar için alınan puanların toplamı ekrana yazılır. Oyun 4 basamaklı oynanır. Dolayısıyla 40 puan için oyun bitirilir. Bu şekilde oluşturmuş olduğumuz algoritma adımlarını aşağıdaki akış diyagramına dönüştürmemiz çokta zor olmayacaktır.



6. Visual Basic Programlama Diline Giriş

6.1. Genel Tanıtım

Programlama dilleri, bilgisayarın anladığı dil olarak bilinen makine diline yakınlığına göre aşağıdaki şekilde gösterildiği gibi seviyelere ayrılmıştır.



Programlama dili, makine diline yaklaştıkça ürettiği programlarda çalışma performansı ve kaynak tasarrufu yüksek seviyede olur. Diğer bir deyişle program hızlı çalışır, belleklerde az yer işgal eder ve bilgisayar yan donanımlarını daha etkin kullanır. Eğer kullandığımız programlama dili konuşma diline yakın olursa kolay kodlama yaparız çünkü dilin söz dizim kuralları konuşma diline benzer, hata toleransı yüksektir dolayısıyla yaptığımız bazı hataları kendisi giderir, platform bağımsızdır bu yüzden de farklı işletim sistemlerinde çalışabilir. Örneğin C++ dili orta seviyeli bir dildir ve çoğu işletim sistemleri bu dil ile kodlanmaktadır. Assembly dili alt seviyedir ve uzun kodlar yazılması çok zordur. Basic dili ise üst seviyedir.

Basic dili ilk başta basit program parçaları yazmak ve derin programlama bilgisine ihtiyaç duymayan programcılar için hazırlanmış olsa da daha sonraki geliştirmelerle oldukça gelişmiş bir programlama dili haline gelmiştir. Görsel bir programlama dili olarak sunulan Visual Basic dilinde eski Basic dilindeki söz dizim kuralları ve neredeyse tüm komutlar aynen kullanılmıştır. Buna rağmen basitliğini koruyarak programcı kitlesini arttırmıştır.

Görsel programlamada temel mantık, sınıf ailesine sahip nesne tanımına ve olay tabanlı alt program parçalarını kodlamaya dayanmaktadır. Programlama sırasında önceden tanımlanmış sınıflar ve bu sınıflara ait olaylar kullanılmalıdır. Örneğin buton yaratmak için buton sınıfına ait bir nesne tanımlamalı ve özelliklerini belirlemeniz yeterlidir. Görsel programlama tasarım birimleri de oldukça gelişmiş özelliklere sahip olduğu için tanımlamayla uğraşmasına gerek kalmayan programcıya sadece hazır nesneleri sürükleyip bırakmak kalıyor. Kodlama yapmak için ise bir olayın ne zaman gerçeklenmesini istediğinizi belirlemeniz yeterli. Örneğin bir butona tıklandığı zaman veya bir program açıldığı anda vb.

6.2. Kodlama için Ön Hazırlık

Biz sadece matematiksel problem çözme ile ilgilendiğimiz için Visual Basic dilinin sadece kısıtlı bir kısmını kullanacağız. Bundan sonraki kısımlarda Visual Basic dili için VB kısaltmasını kullanacağız.

VB tasarım arayüzünü açtığınızda diğer seçeneklerle ilgilenmeden *Standart EXE* seçeneğine çift tıklayacağız. Karşımıza *Project1* isimli bir pencere ve onun içinde de *Form1* isimli bir form tasarımı açılacaktır. Problem çözümleri için hazırladığımız VB kodlarını yazmak için *Form_Load* olayını kullanacağız. Genel olarak *Load* olayı bağlı olduğu nesne yüklendiği anda tetiklenen ve içindeki kodları çalıştıran bir olaydır. Olaylar bağlı oldukları nesneler ile beraber yazılınca olay alt programlarını temsil ederler. *Form_Load* olayı da programın temel nesnesi olan *Form* nesnesine bağlı bir olay olduğundan bir bütün halinde program ilk çalıştığı anda tetiklenip içinde yazılmış kodlar yürütülecektir. Kodları yazacağınız alt programı (prosedür) görmek için *Form1* nesnesinin ortasında bir yere çift

tıklayabilirsiniz. Bu durumda ekrana hazır yazılmış olarak gelen *Private Sub Form_Load* cümlesi Başla nesnesini, *End Sub* cümlesi ise Dur nesnesini temsil etmektedir. Bu iki cümle arasına her satırı bir algoritma adımını gösterecek şekilde kodları yazacağız. Yazdığımız kodları denemek amacıyla yürütmek (çalıştırmak) istediğimizde F5 tuşuna basmamız yeterli olacaktır. Kodların yürütülmesi bittiğinde ise *Form1* nesnesi ekranda görünecektir. Bu pencereyi sağ üstünde bulunan X tuşu yardımıyla kapatıp tekrar kodlama ve tasarım ekranına dönebilirsiniz. Form tasarımı ekranını kullanmayacak olsak da Ctrl + TAB tuşları yardımıyla kodlama sayfası ve form tasarımı sayfaları arasında geçiş yapabilirsiniz.

Her ne kadar bu çalışmalarımızda görsel özelliklerini kullanmasak da sonraki dönemki derslerimizde tüm özellikleriyle VB dilini öğreneceğimiz için bu dil üzerinde uygulama yapmayı seçtik. Bu sayede VB diliyle tanışmış olacağız.

7. Akış Şemasını VB Diline Dönüştürme

7.1. Şematik Nesnelerin Kodlanması

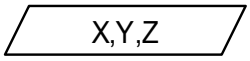
Bu bölüme kadar problem çözümünün algoritma ve akış şeması oluşturma adımlarını inceledik. Akış şeması düzgün şekilde hazırlanan bir problemin herhangi bir dile aktarımının basit olduğunu söylemiştik. Şimdi bu aşamayı göreceğiz. Her şematik nesnenin seçilen programlama dili olan VB 6.0 kodlaması için birebir dönüşüm kuralları yazacağız ve bu kurallar ile kodlama yapacağız.

Başla nesnesi program akışını başlatan nesnedir. Aslında birebir VB dili karşılığı yoktur. Ama temsili olarak biz bu nesneyi olay tabanlı prosedür başlangıcına karşılık olarak düşüneceğiz. VB karşılığı aşağıda gösterildiği şekliyle form nesnesinin yüklenmesi olayının başlangıcıdır.

```
Private Sub Form_Load ()
```

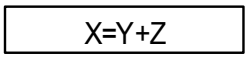
Dur nesnesi program akışının bittiği yeri gösteren nesnedir. Aynen Başla nesnesi gibi VB dilinde tam bir karşılığı yoktur. Temsili olarak bu nesneyi olay tabanlı prosedür bitişi olarak düşüneceğiz. VB karşılığı aşağıda gösterildiği şekliyle form nesnesinin yüklenmesi olayının bitişidir.

```
End Sub
```



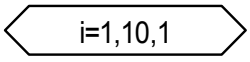
Veri giriş nesnesi kullanıcı tarafından bilgi girileceği zaman kullanılır. VB dilinde birden fazla karşılığı vardır. VB karşılığı aşağıda gösterildiği şekliyle bir iletişim kutusu içerisinde değişken değeri isteyen bir komut olarak temsil edilecektir. Virgül ile ayrılarak istenen her değişken VB dilinde ayrı bir satır olarak yazılmalıdır.

```
X = InputBox("X değişkeni giriniz")
Y = InputBox("Y değişkeni giriniz")
Z = InputBox("Z değişkeni giriniz")
```



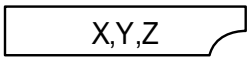
İşlem nesnesiyle matematiksel işlemler yapılır ve VB dilinde karşılığı yine kendisidir. Nesne içeriği aynen VB karşılığı olarak yazılır.

```
X = Y + Z
```



Döngü nesnesi sabit bir sayı veya belli bir değişken içeriği kadar tekrar edilmesi istenen komutlar için kullanılır. VB dilinde karşılığı `For-Next` komutudur. Aşağıda kodlama gösterimi verilmiştir.

```
For i=1 To 10 Step 1
    ... Tekrar edecek komutlar
Next
```



Yazdırma nesnesi kullanıcıya bilgi verileceği zaman kullanılır. VB dilinde birden fazla karşılığı olsa da aşağıda gösterildiği şekliyle bir iletişim kutusu içerisinde yazdırılacak değişken değerlerinin yan yana birleştirildiği bir komut olarak temsil edilecektir.

```
MsgBox X & ",” & Y & ",” & Z
```

Karşılaştırma nesnesi matematiksel şartların kontrol edildiği durumlarda kullanılır. VB dilinde iki farklı karşılığı olabilir. Bunlardan hangisinin seçileceği programın gidişatı için çok önemlidir. Birinci seçenek şartın sağlanması durumunda tek sefer yapılacak işleri temsil eden `if-then-else` yapısıdır. VB karşılığı aşağıda gösterildiği şekildedir.

```
If X > Y + Z Then
```

... Şartın sağlandığı durumda çalışacak komutlar buraya yazılır

```
Else
```

... Şartın sağlanmadığı durumda çalışacak komutlar buraya yazılır

```
End If
```

İkinci seçenek ise şarta bağlı olarak sürekli tekrar edecek işleri temsil eden `do-loop` yapısıdır. Şartları yazarken `while` veya `until` yapıları ile beraber kullanılmalıdır. VB karşılığı aşağıda gösterildiği şekildedir.

```
Do While X > Y + Z
```

... Tekrar edecek komutlar

```
Loop
```

`While` yerine karşılaştırma ifadesinin tersini alarak `Until` kullanılabilir. Duruma göre bazen şartların `Loop` komutu yanına yazılması da gerebilir.

```
Do
```

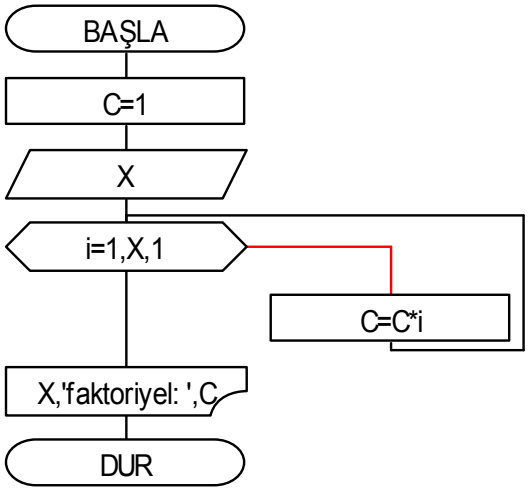
... Tekrar edecek komutlar

```
Loop Until X > Y + Z
```

7.2. Akış Şemasından Kodlama Örnekleri

Önceki konularda incelediğimiz problemlerden bazılarının akış şemalarını kodlamaya çalışalım.

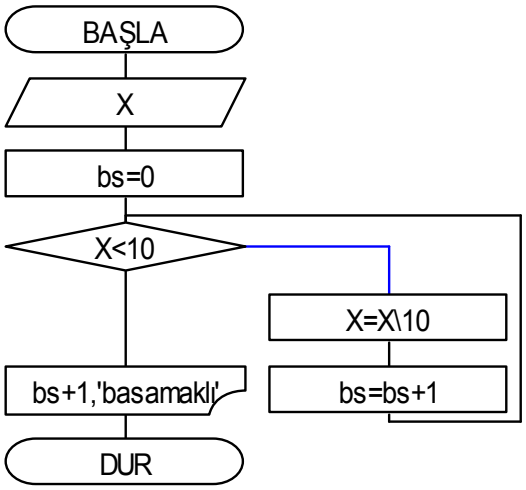
Örnek 1: Faktöriyel probleminin akış şemasını hatırlayalım.



Problemin VB dilinde birebir kodlama yaparsak;

```
Private Sub Form_Load ()
     C = 1
     X = InputBox("X değişkenini giriniz")
     For i = 1 To X Step 1
          C = C * i
     Next
     MsgBox X & "faktoriyel: " & C
```

Örnek 2: Basamak sayısı bulma problemi için oluşturduğumuz algoritmanın akış şemasını hatırlayalım.



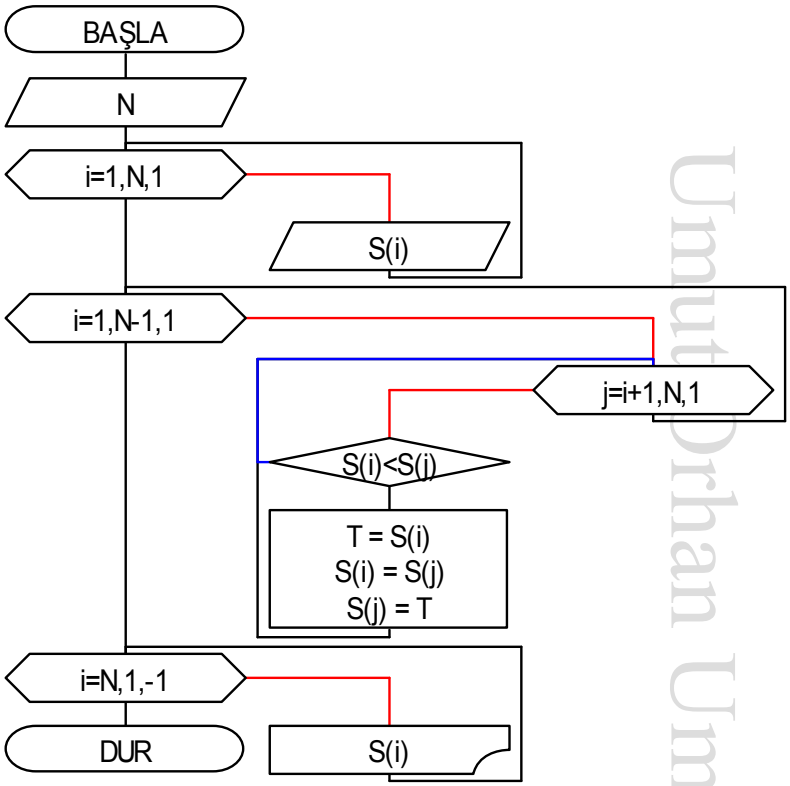
Bu şemada dikkat edilecek en önemli nokta karşılaştırma nesnesidir. Eğer şart sağlanmazsa sağda bulunan iki işlem nesnesinin sürekli tekrar etmesi gerekmektedir. O halde karşılaştırma nesnesi burada döngü görevi yapmaktadır. Bu yüzden `if-then-else` yapısını değil `do-loop`

yapısını kullanmalıyız. Diğer bir husus döngünün, karşılaştırma nesnesinin şartın sağlanamamasını gösteren kolundan kurulmasıdır. Eğer şartlar yanlış olduklarında bir döngü oluşturuyorlarsa `while` değil `until` komutu şartı vurgular. Buradaki örnekten farklı şekilde döngü komutlarının bittiği yerde şart olsaydı ve döngü bu şekilde oluşsaydı o zaman şart vurgusunun `Loop` komutunun yanına yazılması gerekecekti.

VB dilinde kodlama yaparsak;

```
Private Sub Form_Load ()
     X = InputBox("X değişkenini giriniz")
     bs = 0
     Do Until X < 10
          X = X \10
     Loop
     MsgBox bs+1 & "basamaklı"
End Sub
```

Örnek 3: Seçmeli sıralama problemi için oluşturduğumuz algoritmanın akış şemasını hatırlayalım. Bu örnekte dışarıdan girilen N adet sayısının sıralanması için seçmeli sıralama yöntemi kullanılmıştır.



Bu şemadaki karşılaştırma nesnesinin bağlantı kollarının herhangi bir tekrar yaratmadığına dikkat edelim. Bu problemde tekrara sebep olan nesneler döngü nesneleridir. Eğer döngünün kaynağı karşılaştırma nesnesi olursa bir önceki örnekte olduğu gibi `Do-Loop` yapısı kullanılmalıdır. Fakat burada karşılaştırma nesnesi döngü yaratmadığı için `if-then-else` yapısını temsil etmektedir.

Üstteki iki örnekte sadece standart değişkenler kullanılmıştır. Bu problemde kullanılan dizi değişkeni standart değişkenlerden birkaçının taşıyabileceği değerleri tek başına saklayabilmektedir. Dizi kullanımı yüzünden kullanılan indeksli değişkenler de VB dilinde aynen yazılmaktadır. VB dilinde kodlama yaparsak;

```
Private Sub Form_Load ()
   N = InputBox("N değişkenini giriniz")
   For i = 1 To N
      S(i) = InputBox("S(i) değişkenini giriniz")
   Next
   For i = 1 To N-1
      For j = i+1 To N
         If S(i) < S(j) Then
            T = S(i)
            S(i) = S(j)
            S(j) = T
         End If
      Next
   Next
   For i = N To 1 Step -1
      MsgBox S(i)
   Next
End Sub
```


Tekrar etmek gerekirse bir problemin çözümündeki en önemli adım algoritmanın doğru oluşturulmasıdır. Algoritmanın doğru hazırlanması sağlanabilirse akış şeması da kodlama da kolay olacaktır. Uzman programcılar genellikle algoritma ve akış şemasını zihinde canlandırıp (kağıt üzerinde hazırlamadan) kodlamaya geçerler. Bu durumda algoritma bilgisi gereksizdir diye düşünmek yanlış olacaktır. Çünkü problem çözmede hedef, yeterli seviyeye gelen kişilerin algoritma ve akış şemasını zaten akıldan yapabilmelerini sağlamaktır.