BÖLÜM 3

ALGORİTMA VE AKIŞ DİYAGRAMLARI

ANAHTAR KAVRAMLAR

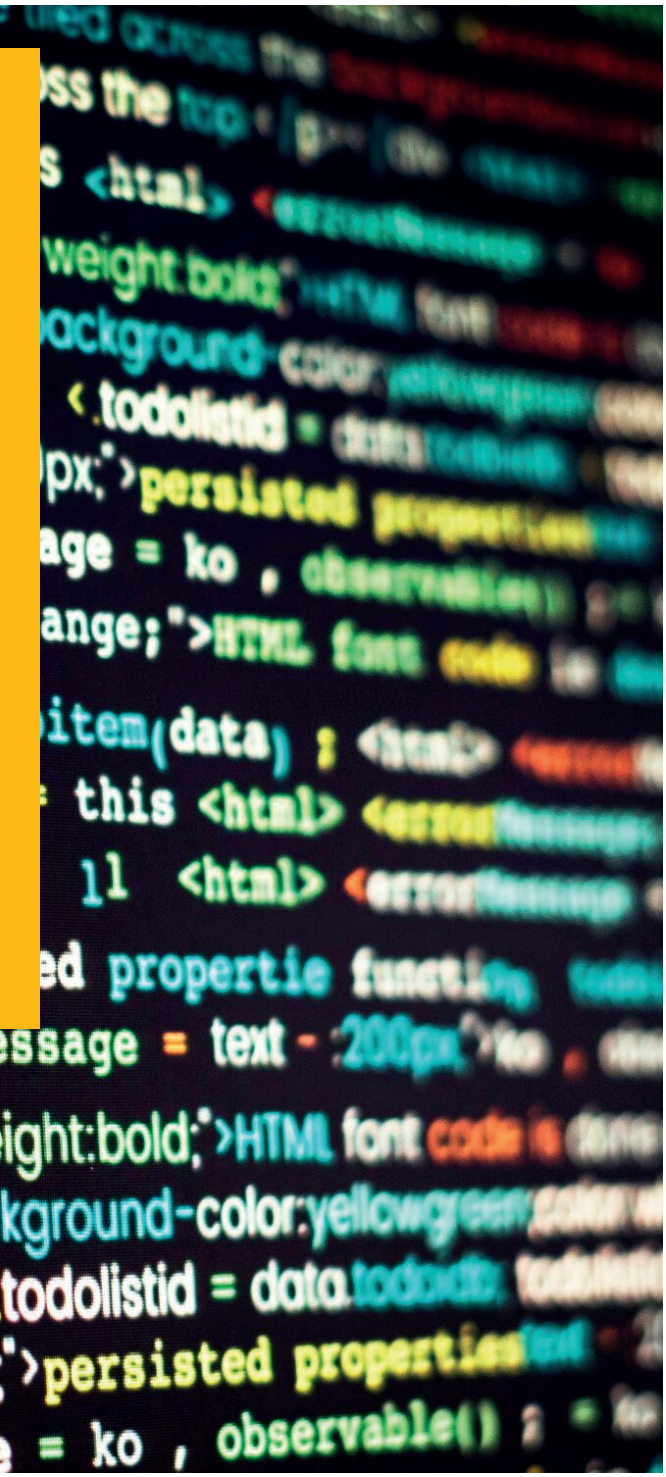
Algoritma, Sözde Kod, Akış Diyagramı, Yazılım Geliştirme Süreci, Problem Çözme Yapıları

ÖĞRENME HEDEFLERİ

1. Öğrenci bölümlere ayırmaya ilişkin temel kavramları bilir
2. Öğrenci bölümlere ayırma ilkelerini, sistemlerini ve önemini bilir
3. Öğrenci bölümlere ayırma sistemlerinin üstünlük ve sakıncalarını bilir
4. Öğrenci işletmelerde bölümlere ayırma sürecinde karşılaşılan temel sorunları algılar ve çözüm üretir
5. Öğrenci işletmenin örgüt yapısını analiz eder

ÜÇÜNCÜ BÖLÜM

GİRİŞ



Bölüm 3 programlamaya yeni başlayanlar için problem çözme yapılarının bahsedildiği bir kısımdır. Bu bağlamda algoritma, akış diyagramı hazırlama ve sözde kod adı verilen algoritmanın yapısal bir programlama dili ile gerçekleştirilmesi işlemleri çeşitli örneklerle anlatılmaktadır. Amaç, programlama ile çözülecek bir problem ile karşılaşıldığında temel yapıları kullanarak algoritma tasarlama becerisi ve algoritmaların sözde kodlar ile gerçekleştirilmesidir. Bölüm 2'de verilen programlama ile ilgili teknik kavramlar kısmı bu bölüm için temel teşkil etmektedir. Varsa ilgili bölümdeki eksikliklerin giderilmesi gerekir.

1. TEMEL KAVRAMLAR

Bilgisayarlı yapılarda sistemin kendisi ve üzerinde koşturulan işlemler yazılımlar ile gerçekleştirilir. Bu bağlamda, sistemin çalışması ve diğer birimlerle haberleşmede ikilik elemanların birleşiminden oluşan komutlar kullanılmaktadır. Kullanılan komutlar her işlemcide kendine özgü bir yapıya sahiptir. Bahsi geçen komut kümeleri kullanılarak çeşitli yazılımlar elde edilmektedir. Söz konusu yazılımlar çok karmaşık bir yapıya sahip olabileceği gibi basit bir görevi yerine getirmek için de kullanılabilir. Dolayısıyla bu yazılımları elde etmede kullanılan programlar amaca göre makine dili, çevirici dil veya yüksek seviyeli bir dil ailesinden seçilmektedir. Aşağıda farklı gruplara dâhil edilebilecek yazılımlara örnek verilmiştir.

- Sistem Yazılımları (Microsoft Windows, Google ChromeOS, Linux vb.)
- Programlama Yazılımları (C++, C#, Python, JavaScript, R, Swift vb.)
- Uygulama Yazılımları (Open Office (Kelime İşlemci/Elektronik Tablo/Sunu ofis uygulamaları), Adobe Creative Cloud, Autodesk CAD/CAM/CAE ve PCB (Çizim/Tasarım Uygulamaları), Medya (Ses/Video) Yürütücüler vb.)

Bilgisayar/bilgisayarlı yapılar/mobil aygıtlar (telefon/tablet vb.) sayılarla çalışırlar. Diğer semboller/harfler ise bu sayıların belirli standartlarla (ASCII/UNICODE) ifade edilmesi ile oluşmaktadır. Söz konusu verilerin işlemcinin belirli ünitelerinde işlemlere tabi tutularak belirli görevleri yerine getirmesi sağlanmaktadır. Örneğin suyun kaynamasını kontrol etmek istediğimizde mantıksal (lojik); en yüksek maaşı alan personelin tespit edilmesinde karşılaştırma (karar) ve barajlardaki suların doluluk oranlarını göstermede matematiksel işlemler yapılmaktadır.

EK BİLGİ

Program, bir işlevi yerine getiren kodlar/komutlar bütünüdür. Program yazılırken öncelikle algoritma ve akış diyagramları oluşturulmaktadır. Yazılım ise donanımı kontrol eden bütün programları ve yardımcı araçları kapsamaktadır.

Matematiksel İşlemler

Toplama, çıkarma, çarpma, bölme, modüler aritmetik ve üs alma gibi hesaplamalar matematiksel ifadeler ile yapılmaktadır. Tablo 3.1'de matematiksel operatörlerin (+, -, :, x, AB, mod) programlamadaki karşılıkları gösterilmektedir. Ayrıca hesaplamalarda işlemler genelde soldaki ifadeden başlayarak operatörlerin öncelik sırasına göre yürütülmektedir (Tablo 3.2). Ancak üs alma, önek, koşul ifadeleri ve birleşik atama işlemleri sağdan sola doğrudur. Operatörler dışında kalan kısımlar yani A, B, C... gibi isimler ya da sayılar işlenen olarak ifade edilir.

ARAŞTIRALIM

Derleyiciler kaynak koddaki matematiksel işlemleri önek (prefix) ya da postfix (sonek) notasyonlarına çevirmektedir. Alışa geldiğimiz ifadeler ise içek (infix) yapıdadır.
İçek (infix): 2+4*6
Önek (prefix): *+246
Sonek (postfix): 2 4 6 * +

Tablo 3.1. Matematiksel İfadelerin Programlamadaki Gösterimleri.

İşlem	Matematiksel İfade	Kod Gösterimi
Toplama	A+B	A+B
Çıkarma	A-B	A-B
Çarpma	AxB	A*B
Bölme	A:B	A/B
Üs Alma	AB	A^B veya A**B veya pow(A,B)
Modüler İşlem	A mod B	A%B

Kaynak kodda yazacağımız matematiksel ifadelerde içek yazım türünün derleyici tarafından çözümlenmesi zordur. Yapılacak hesaplamalarda matematiksel işlemlerin operatör önceliklerine göre ayrıştırılması, ardından bunların sıralanması ve ilgili dizilime göre işlem yapılması gerekmektedir. Derleyiciler işlemleri kolaylaştırmak için önek ya da sonek notasyonunu kullanır.

Tablo 3.2. Matematiksel İşlemlerde Öncelik.

İşlem Sırası	İşlem	Kod Gösterimi
1.	Parantez içi	(…)
2.	Üs alma	A^B veya A**B veya pow(A,B)
3.	Çarpma/Bölme	A*B ve A/B
4.	Toplama/Çıkarma	A+B ve A-B

Bölme ve mod alma işlemlerinde mantık hatasına düşmemek programcının dikkat etmesi gereken bir unsurdur. Çünkü kullanılan sistemler tam sayılı işlemlerde yuvarlama yapmaktadır. Mod alma işlemlerinde ise sayının pozitif veya negatif olmasına göre sonuç değişmektedir. Aşağıda duruma uygun örnekler verilmiştir.

DİKKAT EDELİM

Bölme işlemlerinde işlenen değerinin tam sayı ya da ondalıklı sayı olması önemlidir. Tam sayılı işlemlerde yuvarlama yapılmaktadır. Hassas işlemlerde değerleri ondalıklı tanımlamak gerekir.

Mod alma örnekleri:

16 % 3 = 1

–16 % 3 = –1

16 % –3 = 1

–16 % –3 = –1

Bölme örnekleri:

9/4 = 2

–9/4 = –3

Bazı hesaplamalar karmaşık yapıya sahip olabilir. Bu tarz ifadelerin kaynak koda yazılabilir hale dönüştürülmesi gerekir. Aşağıda duruma uygun bir örnek gösterilmektedir:

a x=(-b±((b^2-(4*a*c)))^(1/2))/2*a

Karşılaştırma (Karar) İşlemleri

İki nümerik değerden hangisinin büyük/küçük/büyük eşit/küçük eşit/eşit/eşit değil/denk olması durumları karşılaştırma (karar) işlemleri olarak ifade edilmektedir. Tablo 3.3 bu işlemlerin program yazımındaki karşılıklarını vermektedir.

Tablo 3.3. Karşılaştırma (Karar) İşlemleri.

İşlem	Kaynak Kod Gösterimi
Eşit	= =
Eşit Değil	!= veya < >
Büyük	>
Büyük Eşit	> =
Küçük	<
Küçük Eşit	< =
Denk	= = =

Tablo 3.3'de verilen gösterimlerin kullanılan programlama dilinin söz dizimine (syntax) göre değişiklik gösterebileceği unutulmamalıdır. Dersin ilerleyen bölümlerinde C++ söz dizimine göre operatörler kullanılacaktır. İlaveten, zincirleme karşılaştırma operatörleri ise her programlama dilinde kullanılmaz. Örneğin Python'da "c > b > a" gibi ifadeler "c > b ve b > a" olarak değerlendirilir, ancak bu tür zincirleme C++'da olmaz.

Mantıksal ve Bitsel İşlemler

Mantıksal ve Bitsel (Bitwise) işlemler problem içerisinde koşul gerektiren durumlarda karşımıza çıkmaktadır. Söz konusu durum bir veya birden fazla olabilir. Bir koşulun olmamasını istediğimizde değilse anlamında "Mantıksal DEĞİL" işlemi; iki koşulun birden sağlanmasını istediğimizde "Mantıksal VE" işlemi; iki koşuldan en az birisinin doğru olmasını beklediğimizde "Mantıksal VEYA" işlemi; iki koşuldan sadece birinin doğru olmasını beklediğimiz durumlarda da "Bitsel XOR" işlemi yaparız. Tablo 3.4 "Mantıksal DEĞİL", Tablo 3.5 "Mantıksal VE", Tablo 3.6 "Mantıksal VEYA" ve Tablo 3.7 "Bitsel XOR" doğruluk tablosunu göstermektedir. Sırası ile bu işlemler "!", "&&", "||", "^" sembolleri ile kullanılmaktadır.

Tablo 3.4. Mantıksal DEĞİL Doğruluk Tablosu.

A	!A
0	1
1	0

Tablo 3.5. Mantıksal VE Doğruluk Tablosu.

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

Tablo 3.5. Mantıksal VEYA Doğruluk Tablosu.

A	B	A || B
0	0	0
0	1	1
1	0	1
1	1	1

Tablo 3.5. Bitsel XOR Doğruluk Tablosu.

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Tablo 3.5'de verilen Bitsel XOR'un yanı sıra Bitsel VE (&), Bitsel VEYA (|) ve Bitsel DEĞİL (~) işlemleri de vardır. Ayrıca Bitsel XOR operatörünün sembolü, üs alma işleminin operatörü ile benzemektedir. Bu durumda bazı programlama dilleri operatör yerine fonksiyonlar kullanarak işlem yaptırmaktadır (pow gibi).

Bitsel operatörlerde baytlar yerine bit seviyesinde işlem yapılır. Bu durumda ilk ve ikinci işlenenin karşılık gelen bitleri aradaki operatöre göre işletilir.

Bitsel VE işlemi,

10101010 & 01010101 = 00000000

Bitsel VEYA işlemi,

10101010 | 01010101 = 11111111

Bitsel XOR işlemi,

10101010 ^ 01010101 = 11111111

Bitsel DEĞİL işlemi,

10101011 = 01010100

Mantıksal VE işlemi,

(Yaş>18) && (Ehliyet == "Var") durumunda her iki koşul da doğru ise işlem yürütülecektir.

2. ALGORİTMA

Algoritma adımlarla tanımladığımız bir çözüm yoludur. Burada adım sayısı sonludur ve sözlü olarak ifade edilir. İlk algoritma, Ebu Abdullah Muhammed İbn Musa el-Hârizmî tarafından 9. yüzyılda cebirde kullanılmıştır. el-Hârizmî'nin bu alanda yazdığı kitap tüm dünyada olumlu bir yankı uyandırmıştır. Matematik alanında sağlanan bu başarı bilgisayarların gelişme sürecindeki problemlerin çözümünde de kullanılarak devam etmiştir. Bu bağlamda, problemlerin çözülmesinde algoritmik ve sezgisel yaklaşımlar önerilmektedir. Algoritmik olan yaklaşımlar çözüm için olası en uygun yönteme odaklanırken, sezgisel yaklaşımlar hıza önem vermektedir. Sezgisel algoritmalar en iyiye yakın çözümü hızlı ve kolay yapmak içindir.

Algoritmalar bir programlama diline dönüştürülebilir. Öncelikle programcıların algoritmayı herkesin anlayabileceği bir standarda yani akış diyagramına çevirmeleri gerekir. Ardından bir programlama dili kullanarak bilgisayarın anlayabileceği bir kod dizisine dönüştürürler. İşte bu kod dizilerine program adı verilir. Son olarak programlar çeşitli işleri yerine getiren yazılım haline getirilmektedir. Temel kavramlar ünitesinde bu konuya detaylı olarak yer verilmiştir.

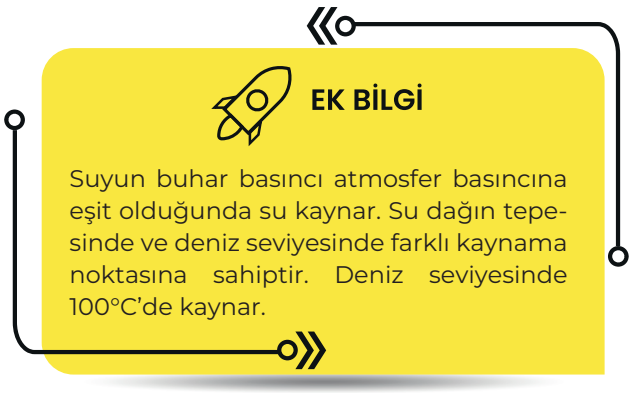
Akış diyagramları sözlü olarak ifade edilen adımların standart şekillerle ifade edilmesidir. Bütün programcılar geliştirme sürecinde önce algoritma sonrasında akış diyagramı oluşturarak süreci belgelemek zorundadırlar. Algoritma oluşturulurken herkesin anlayabileceği bir kesinlikte, sıralı ve sonlu bir süreçte tanımlanması gerekir.

Örneğin bir suyu kaynatmak istiyoruz. Bunu problem cümlesi olarak ele alalım. Çözüm için ise örnek bir algoritma tasarlayalım. Kabaca önce suyun varlığı kontrol edilmelidir. Ardından "su varsa suyu kaynat" şeklinde bir adım izlememiz gerekir. Algoritma tasarlama süreci kesinlik istemektedir. Su varsa suyu kaynat doğru bir adım olmakla birlikte çok genel bir ifadedir. Suyun niteliği, kaynatma aracı, atmosferik durum, kaynama derecesi gibi koşullar belirlenmediği için önerilen adım ilk kez suyu kaynatacak birisi için ucu açık kalacaktır. Algoritma bahsi geçen koşullar dikkate alınarak tekrardan tasarlandığında;

• Su saf mı, yoksa bir karışım mı? Evet, su saf. O zaman kaynatacak kabı seçelim.

• Seçilen kaynatma kabı var mı? Evet. O zaman suyu kaba koyalım.

• Bulunduğumuz yer deniz seviyesinde mi? Evet. O halde kaynama noktasını 100°C yapalım.



Bu şekilde bir algoritma tasarlandığında suyun kaynatılması problemi için daha kesin ifadeler belirlemiş oluruz. Aynı zamanda buradaki adımlar belli bir sırayı takip etmektedir. Yani suyu ve atmosferik durumu belirlemeden su derecesini ayarlamak bizi yanlış bir çözüme götürecektir. İlaveten suyun kaynama noktasını belirledikten sonra kaynatmayı durdurmamız gerekir. Yoksa su kaynaya kaynaya bitecektir ya da yanlış bir kaynama noktası belirlenirse ya çok geç kaynayacak ya da hiç kaynamayacaktır. O halde adımlarımızı sonlu yapabilmek adına en sona aşağıdaki gibi bir çözüm eklememiz gerekir:

• Kaynama gerçekleşti mi? Evet. O zaman kaynatmayı durduralım.

Problemin belirlenmesi çözüme giden yoldaki en önemli adımlardan birisidir. Problem belirlendikten sonra deneysel yöntemler ya da yukarıda anlatıldığı gibi algoritmik metot tercih edilir. Bu bağlamda sıra olarak problemin tanımlanması, probleme etken olan girdi ve çıktıların belirlenmesi, çözüm yollarının aranması, çözümün test edilmesi ve test sonuçlarına göre iyileştirilmesi süreçleri başlar. Bu süreç istenilen şekilde tamamlandıktan sonra algoritma tasarımına geçilip, hazırlanan algoritmanın test ve geri dönütler ile iyileştirilmesi gerekmektedir.

Problemin tanımlanmasının çözümün istenilen görevi yeri getirmesi adına önemli olduğunu daha önce belirtmiştik. Bu sebeple problemin yanlış anlaşılmaya maruz vermeyecek şekilde tekrar tekrar gözden geçirilmesi gerekir. Problemin doğru analiz edilmesi çözüm/çözümler elde etmenin bir parçasıdır. Girdi ve çıktı konusu ise çözüme etki eden bütün parametrelerin dâhil edilmesi anlamına gelir. Sonradan dâhil olabilecek bir girdi baştan belirlenmezse çözüm eksik kalacaktır. Çıktı konusu ise sistemden beklentilerimiz doğrultusunda tüm girdileri dikkate alarak farklı bakış açıları geliştirdiğimiz değerlerdir. Metinsel ve grafiksel raporlar örnek çıktı türleridir. Çıktılar da çoğunlukla matematik ya da istatistik bilimlerinden faydalanılmaktadır. Girdilerin cebirsel işlemlerle matematik olarak ya da standart sapması, ortalaması gibi istatiksel olarak yorumlanması bu duruma örnek verilebilir. Günümüzde pek çok matematik ya da istatistik teoremler bulunmaktadır. Yapay zekâ, derin öğrenme ise popüler değerlendirme yaklaşımlarıdır.

Algoritma basamağına geçilmeden problemin kâğıt üstünde çalışılması en çok tercih edilen bir metottur. Bu süreçte çözüme gidecek yollar yazılır. Arasından belirlenen en uygun yöntem, yazılacak algoritma için kullanılacaktır. Problemin çözümü bilgisayarlı bir yapıda giderilecek

ise kullanılacak donanım, işletim sistemi, ihtiyaç duyulan bellek ya da belleğin sistemi yormayacak şekilde optimize edilmesi algoritmaya etki edecek bileşenler olarak karşımıza çıkar. Tüm bu durumlar analiz edildikten sonra algoritma yazma aşamasına geçilir. Burada da algoritmanın hangi programlama dili ile gerçekleştirileceği, algoritmanın sınanıp sınanmadığı, hataların giderilip giderilmediği ve performans durumuna bakmak gerekir. Algoritmanın yazılıma dönüşmeden önceki geçirdiği süreç bu şekildedir. Ancak algoritmanın yazılıma dönüştürülmesinde her program geliştiren ekip farklı yöntemler izlemektedir. Bu yöntemler ve geliştirme süreci yazılım geliştirme süreci kısmında detaylı olarak verilmektedir.

3. YAZILIM GELİŞTİRME SÜRECİ

Bir problemin çözülmesi için geliştirilen algoritmalar ve akış diyagramları bir yazılıma dönüştürüldüğünde de belli bir süreci takip etmektedir. Bu sürece yazılım geliştirme yaşam döngüsü denmektedir. Döngü; planlama, analiz, tasarım, gerçekleştirme ve son olarak teslim ve bakım olarak gerçekleşir (Şekil 3.1).

Şekil 3.1. Yazılım Geliştirme Yaşam Döngüsü.

Planlama; proje ihtiyaç analizinin yapıldığı kısımdır. Donanımsal ve yazılımsal maliyet, çalışacak personel sayısı ve rolleri, kimin projenin hangi kısmında yer alacağı ve süreç gibi durumlar belirlenir. Ayrıca yazılım ne yapacağı yani hangi problemi çözeceği yine burada işlenmektedir.

Analiz; programın işlevlerini ayrıntılı bir şekilde belirlediğimiz kısımdır. Belirlenen işlevler bir modüle dönüştürülerek her bir kısım için net bir şekilde ne yapılacağına karar verilmeli ve belgelenmelidir.

Tasarım; mantıksal ve fiziksel olarak gerçekleşir. Programdan beklenen istemler, mantıksal kısımda sistemin yapısını oluşturmada ve fiziksel kısımda ise yazılımın bileşenlerini detaylı bir şekilde göstermede kullanılır. Ayrıca hangi programlama dilinin kullanıldığı, teknoloji, mimari tasarım bu süreçte ele alınmaktadır.

Gerçekleştirme; tasarım aşamasında belirlenen programlama dili ile belirlenen ihtiyaçlar doğrultusunda kodlamaya ve gerekli dokümantasyonların yapılmaya başlandığı kısımdır. Kodlama kısmında herkesin okuduğunda anlayabileceği bir teknik oluşturulur. Kodlar mümkün mertebe basit olmalıdır. Kodlamada değişkenler, fonksiyonlar ve sınıflar için isimlendirme kuralı belirlenir. Ayrıca yazılan her parça için açıklama/yorum satırları eklenmeye özen gösterilmektedir. Son olarak, programın farklı girdilerle iste-

nen sonucu üretip üretmediği ile ilgili testler de burada gerçekleştirilir.

Teslim ve bakım; adından da anlaşılacağı üzere programın diğer basamaklardaki süreçlerinin tamamlanıp kuruluma hazır hale getirildiği kısımdır. Bu kısımda yazılımın kısıtları son kullanıcıya gösterilmelidir. Kısıtlardan kasıt, yazılımın hangi işletim sistemlerinde kullanılabileceği, ihtiyaç duyduğu diğer yazılım ve donanım gereksinimleridir. Bakım olarak da son kullanıcıdan gelen geri dönüşlere göre ilave modüller yazılabilir ya da programın varsa hataları giderilir.

Yazılımın geliştirme süreci yukarıda tanımlanan basamaklardan geçirilirken farklı stratejiler uygulanmaktadır. Her stratejinin kendine göre avantaj ve dezavantajları bulunmaktadır. Üniversitelerde bu stratejiler yazılım geliştirme mimarileri şeklinde ayrı bir ders olarak verilmektedir. Biz burada sadece isimlerine yer vereceğiz. Daha detaylı bilgi için bölümün faydalı kaynaklar ve internet kaynakları kısmına bakılabilir.

Yazılım süreç modelleri,

• Kodla ve Düzelt Model Yaşam Döngü Modelleri

• Çağlayan Yaşam Döngü Modeli

• V Süreç Modeli

• Evrimsel Geliştirme Süreç Modeli

• Prototipleme Yaşam Döngü Modeli

• Spiral Model

• Formal Sistem Geliştirme Modeli

• Yeniden Kullanıma Yönelik Geliştirme Modeli

• Artırımsal Geliştirme Süreç Modeli

• Çevik Yazılım Geliştirme Metodolojileri (En popüler yaklaşımdır)

o Aşırı Programlama (Extreme Programming - XP)

o Birleşik Rasyonal İşlem (Rational Unified Process)

o Özellik Odaklı Geliştirme (Feature-Driven Development - FDD)

o Test Odaklı Geliştirme (Test-Driven Development - TDD)

o LEAN Geliştirme (LEAN Development)

o Dinamik Sistem Geliştirme Metodolojisi (Dynamic System Development Methodology - DSDM)

o Microsoft Çözüm Çerçevesi (Microsoft Solution Framework - MSF)

o SCRUM

3.1. Algoritmanın Yazılım Geliştirme Sürecindeki Yeri

Algoritma yazımı yazılım geliştirirken ilk olarak oluşturulan ve tüm programlama dilleri kullanımında da temel teşkil eden en önemli bölümdür. Algoritma oluşturulmadan başlanan programlarda sonradan bir şey eklemek veya hatayı tespit etmek daha güç olmaktadır. Kodlamaya geçmeden önce bu kısmın oluşturulması profesyonel yazılımlar için vazgeçilmezdir. Ancak küçük çaplı ofis yazılımlarında, örneğin ek bütçe hesabı, personel taşınır bilgisi veya küçük ölçekli veri analizleri, ağ cihazları haberleştirilirken yazılan betik (script) programlar gibi durumlarda algoritma yapmadan da programlamaya geçilebilmektedir.

Algoritma yazımı bizim programlama dillerinden hangisini seçeceğimiz noktasında bir yol haritası niteliğindedir. Bölümün başında verilen su kaynatma problemi için su ısıtıcılara yazılan algoritma, işletim sisteminden bağımsız bir programlama dilini kullanmamız gerektiğine bir işarettir. Genelde bu durumlarda Java programlama dili kullanılır. Ayrıca algoritmalar birden

fazla çözüm yolunu kodlamadan sözlü olarak geliştirebileceğimiz en rahat ortamlardır. Kodlama yapılarak başlanan projelerde birden fazla çözüm belirlemenin zor olacağı aşikârdır.

Şekil 3.2'de örnek bir algoritma yazımı verilmiştir:

Adım 1: Örnek görselleri sisteme aktar
Adım 2: Görsellerden renk, şekil ve desen özellikleri çıkar
Adım 3: Özellikleri veritabanına kaydet
Adım 4: Sorgu görseli yükle
Adım 5: Sorgu görselin renk, desen ve şekil özelliklerini çıkar
Adım 6: Sorgu görselin özelliklerini benzerlik ölçüm algoritmaları ile veritabanındaki diğer özelliklerle karşılaştır
Adım 7: Karşılaştırma sonucuna göre en benzeyen ilk 3 görseli yüzdeleri ile ekrana getir
Adım 8: Dur

Şekil 3.2. Örnek Algoritma: Sorgu Görsel ile Benzer Görselleri Arama Algoritması.

3.2. Algoritmaların Sınıflandırılması

Algoritmalar problemin türlerine göre basit (lineer), mantıksal ve döngüsel olarak ele alınmaktadır. Basit algoritmalar, bölümün temel kavramlar kısmında belirtilen mantıksal ve bitsel işlemlerin yapılmadığı ve adımların düz bir hatta sahip olduğu yani dallanmaların yapılmadığı türlerdir. Mantıksal işlemler yer almadığı için de herhangi bir koşul/karar yapısı bulunmamaktadır. Şekil 3.3'de örnek bir basit algoritma yazılmıştır:

Adım 1: Dairenin yarıçapını giriniz (cm); **r**
Adım 2: Sabit Pi değerini giriniz; **pi**
Adım 3: Dairenin yarıçapı ile pi değerini çarpınız; **Alan=pi*r**
Adım 4: Hesaplanan değeri, "Dairenin alanı " ile " cm^2 dir." karakter katarları arasında ekrana yazınız; **Alan <Alan> cm2 dir.**
Adım 5: Dur

Şekil 3.3. Dairenin Alanını Hesaplayan Basit Algoritma.

Mantıksal algoritmalar, adından da anlaşılacağı üzere karar/koşul yapısını barındıran algoritmalardır. Şekil 3.4'de örnek bir mantıksal algoritma sunulmaktadır.

Adım 1: Kitabı al
Adım 2: Çalışma süresini ayarla
Adım 3: Eğer kitap ayracı yoksa ayraç al
Adım 4: Konulara çalış
Adım 5: Örnek soruları çöz
Adım 6: Eğer bulunduğun ortamda bir kişi daha varsa öğrendiklerini anlat
Adım 7: Eğer çalışma süresi tamamlandıysa kaldığın yere ayraç koy
Adım 8: Eğer ayraç yoksa sayfaya elle işaret koy
Adım 9: Dur

Şekil 3.4. Ders Çalışma için Mantıksal Algoritma.

Döngüsel algoritmalar ise hem karar/koşul yapısını barındıran hem de bir adımın birden fazla tekrar ettiği algoritmalardır. Örnek bir algoritma Şekil 3.5'de gösterilmektedir.

Adım 1: Şampuanım bitti mi?
Adım 2: Evet ise Adım 5'e git. Değilse Adım 3'e git.
Adım 3: "Markete gitme"
Adım 4: Adım 8'e git.
Adım 5: "Yağmur yağıyor mu?"
Adım 6: Evet ise Adım 3'e git. Değilse Adım 7'ye git.
Adım 7: "Markete git"
Adım 8: Dur

Şekil 3.5. Kendi Kendine Öğrenme (Self-learning) ile Mantıksal Algoritma.

Şekil 3.5'de verilen algoritma kendi kendine öğrenme, derin öğrenme ve makine öğrenmesi gibi algoritmalarda sıklıkla kullanılmaktadır. Faydalı kaynaklar ve internet kaynakları kısmında daha fazla örnek öğrenme algoritmaları yer almaktadır.

Algoritmalarda üç tür kontrol ifadesi vardır: karar kontrolü (dallanma), yinelemeli (döngü) ve atlama ifadeleri. Dallanma, hangi eylemlerin yapılması gerektiğine karar vermek anlamına gelirken, döngü ise eylemin kaç kez yapılması gerektiğine karar verir. Atlama ifadeleri ise kontrolü bir noktadan başka bir noktaya aktarır.

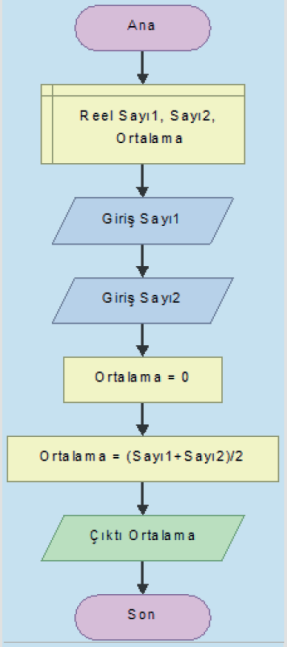
ARAŞTIRALIM

Şekil 3.2'de verilen algoritma hangi algoritma sınıflandırmasına girmektedir?

3.3. Algoritma Geliştirilmesi

Algoritma geliştirilirken tespit edilen problemin bulduğumuz çözüm adımları metin halinde cümlelere dönüştürülmelidir. Her bir cümlede adım sırası önemlidir. Ardından akış diyagramları (flow-chart) ile yazılan adımlar standart şekillere çevrilir. Son olarak, belirlenen bu adımlar kod benzeri metinlerle veya kısaltmalarla sözde koda (pseudo-code) dönüştürülür. Tüm bu aşamalar algoritma geliştirilmesi için gereklidir. Tablo 3.6'da örnek bir algoritma geliştirme süreci gösterilmektedir. Burada verilen sözde kod ve akış diyagramı işlemi için ilerleyen bölümde daha detaylı bilgi verilecektir.

Tablo 3.6. Algoritma Geliştirme Süreci: Satır Algoritma, Sözde Kod, Akış Diyagramı.

Satır Algoritma	Sözde Kod	Akış Diyagramı
Adım 1: Başla Adım 2: 1. sayıyı oku Adım 3: 2. sayıyı oku Adım 4: Ortalamayı al Adım 5: Sonucu ekrana yaz Adım 6: Dur	1. Başla/Ana 2. Reel olarak Sayı1, Sayı2, Ortalama değerlerini tanımla 3. Oku Sayı1 4. Oku Sayı2 5. Ortalama=0 6. Ortalama=(Sayı1+Sayı2)/2 7. Yaz Ortalama 8. Dur/Son	

Algoritma tasarımında nümerik ve alfanümerik olmak üzere çeşitli veriler kullanılmaktadır. Söz konusu veriler için "değişken" adı verilen ve bellek üzerinde bir adrese etiketlenen alanlar kullanılmaktadır. Tablo 3.6'da verilen örnekte Sayı1, Sayı2 ve Ortalama bir değişkendir ve içerisinde reel sayılar tutulmaktadır. Değişkenlerin değeri program içerisinde değişebilir. Pi sayısı, ışık hızı ve bunun gibi değeri değişmeyen alanlar için ise "sabit" tanımı yapılmaktadır. Değişken ve sabitlerin yanı sıra alt yordam da tanımlayıcı olarak algoritmada kullanılan diğer bir kavramdır. Alt yordam, algoritma içerisinde tekrar eden işleri topladığımız kısımdır. Örneğin bir görselin birden fazla görsel ile karşılaştırıldığı bir durum düşünelim. Karşılaştırma işlemi her bir görsel için tekrar edeceği için her seferinde karşılaştırma işlemi için kod yazmak yerine bir tane karşılaştırma alt yordamı yazılır. Karşılaştırılan değerler ise alt yordama parametre olarak verilir. Böylece karşılaştırma için gerekli algoritma tekrar tekrar yazılmamaktadır.

Algoritma geliştirilirken "sayaç" adını verdiğimiz ve yaptığımız işleri belli bir sayıda tekrar ettirdiğimiz durumlar olmaktadır. Örneğin bir alış veriş sitesindeki belirlediğimiz markadan kaç tane ürün varsa bunların ekrana listelendiğini düşünelim. Bu durumda o markadan olan ürün kadar bir sayaç kullanmamız gerekir. Örneğin bir mağazada 100 tane telefon ürünü olduğunu ve bunların ekrana listelendiğini varsayalım (Şekil 3.6):

Adım 1: Ürünü seç
Adım 2: Sayaç değerine 1 ata
Adım 3: Sayaç'ın değeri 100'den küçük ise Adım 4'e git değilse Adım 6'ya git.
Adım 4: Ürüne ait Sayaç'ın değerindeki markayı ekrana getir
Adım 5: Sayaç'ın değerini 1 artır Adım 3'e git
Adım 6: Dur

Şekil 3.6. Algoritma ile Sayaç Kullanımı.

Şekil 3.6'da verilen sayacın değeri bizim belirlediğimiz adım kadar artar ya da azalabilir.

Sayac=Sayac±Adım. Sayaç kullanımında sayacın başlangıç değerini, sayacın koşul durumunu ve adım değerini belirlememiz gerekmektedir.

Algoritmalarda bir de döngü adını verdiğimiz bir kullanım bulunmaktadır. Algoritmanın belli adımlarının bir koşula bağlı olarak yürütülmesi durumu buna örnek olarak verilebilir. Burada sayaç kullanımına benzer bir durum vardır. Ancak bir başlangıç ve adım değeri başlangıçta belirlenmez. Örneğin bir sisteme yaşı 18'den büyük olanların kayıt yaptırıldığı bir durum düşünelim. Şekil 3.7 bu duruma örnek olarak verilmektedir.

Adım 1: Yaşınızı girin
Adım 2: Yaş değeri 18'den büyükse Adım 3'e git değilse Adım 5'e git
Adım 3: Kullanıcı ismini girin
Adım 4: Diğer kullanıcı bilgilerini girin
Adım 5: Dur

Şekil 3.7. Algoritma ile Döngü Kullanımı.

3.4. Algoritma Yazma Kuralları

Algoritma yazımında satırlar 1'den başlayarak numaralandırılır. Adımlar sonlu sayıda ve özellikle sıralı olmalıdır. Tekrar eden işlemlerden kaçınılmalı ya da bu kısım bir alt yordam algoritması olarak tasarlanmalıdır. Gereksiz ve etkisiz kullanım bellekte boş yere işgal edeceğinden ilgili adımlar tespit edilip algoritmadan çıkarılmalıdır. Algoritma içerisindeki dallanmalarda adım numarası belirtilmelidir. Adımların herkes tarafından anlaşılabilir bir keskinlikte olmasına dikkat edilmelidir. Her algoritmada bir girdi ve çıktı bulunmalı ve algoritma Dur/Son ile bitirilmelidir.

4. AKIŞ DİYAGRAMI

Akış diyagramı, algoritmada yazılan adımların belirli bir standarda göre sembolize edilmesidir. Akış diyagramları için IBM ve Spesifikasyon ve Açıklama Dili (Specification and Description Language – SDL) gibi pek çok farklı standart bulunmaktadır. Akış diyagramı sembolleri ve açıklamaları Tablo 3.7'de verilmektedir.

Tablo 3.7. Akış Diyagramı Sembol ve Açıklamaları.

Sembol	Açıklama
Input	Girdi (Input) sembolü değerlerin klavyeden okutulması için kullanılmaktadır.
Output	Çıktı (Output) sembolü değerlerin ekrana yazdırılması için kullanılmaktadır.
Declare	Girdi değerlerin tam sayı, reel, alfanümerik ya da doğru/yanlış gibi mantıksal değerler ile tanımlamada (Declare) kullanılan semboldür.
Assign	Algoritmadaki işlem basamaklarının yapıldığı atama (Assign) işlem diyagramıdır.
If	Mantıksal, bitsel ya da karşılaştırma işlemleri için gerekli koşullar (Eğer - If) bu sembol ile ifade edilmektedir.
Call	Algoritmada tekrar eden işlemler için yazılan alt yordam çağrı (Call) diyagramıdır.
While	Algoritmada döngü kullanımı bu diyagramla ifade edilir. Koşul sağlandığında ilgili adımlar çalıştırılır (While).
For	Sayaç kullanımı için kullanılmaktadır. Bunun için bir başlangıç bir artış ve koşul belirlenmelidir (For).
Do	Algoritmada döngü kullanımı için tercih edilen ikinci döngü yapısıdır. Diğer döngü diyagramında farklı olarak çalıştırılması gereken adımlar en az bir kez çalıştırılır (Do/Do-While).

Tablo 3.7'de gösterilen semboller farklı standartlarda farklı şekillerle gösterilebilir. Bu bölümde akış diyagramları çizilirken Flowgorithm uygulaması kullanıldığından ilgili uygulamanın sembolleri dikkate alınmaktadır. Flowgorithm, programlamaya başlayanlar için basit grafik akış şemalarına dayanan ücretsiz bir yazılımdır.

Flowgorithm ile akış diyagramlarını kullanarak, tipik bir programlama dilinin tüm nüansları yerine programlama kavramlarına odaklanabilirsiniz. Programlarınız doğrudan Flowgorithm'de de çalıştırılabilmektedir. Ayrıca Flowgorithm ile hazırlanan akış diyagramlarınızı tek tıklama ile sözde koda dönüştürebilirsiniz. Programlama mantığını akış diyagramları ile anladıktan sonra, ana dillerden birini öğrenmeniz çok kolay olmaktadır. Flowgorithm ile akış çizelgenizi etkileşimli olarak 18'den fazla programlama diline dönüştürme imkânı da bulunmaktadır. Bunlardan bir kısmı C#, C++, Java, JavaScript, Lua, Perl, Python, Ruby, Swift, Visual Basic .NET ve VBA'dir. İlaveten, programın pek çok dil desteği olup, Türkçe bunlardan birisidir. Dersteki uygulamaları ve sizlerin geliştirmek istediğiniz

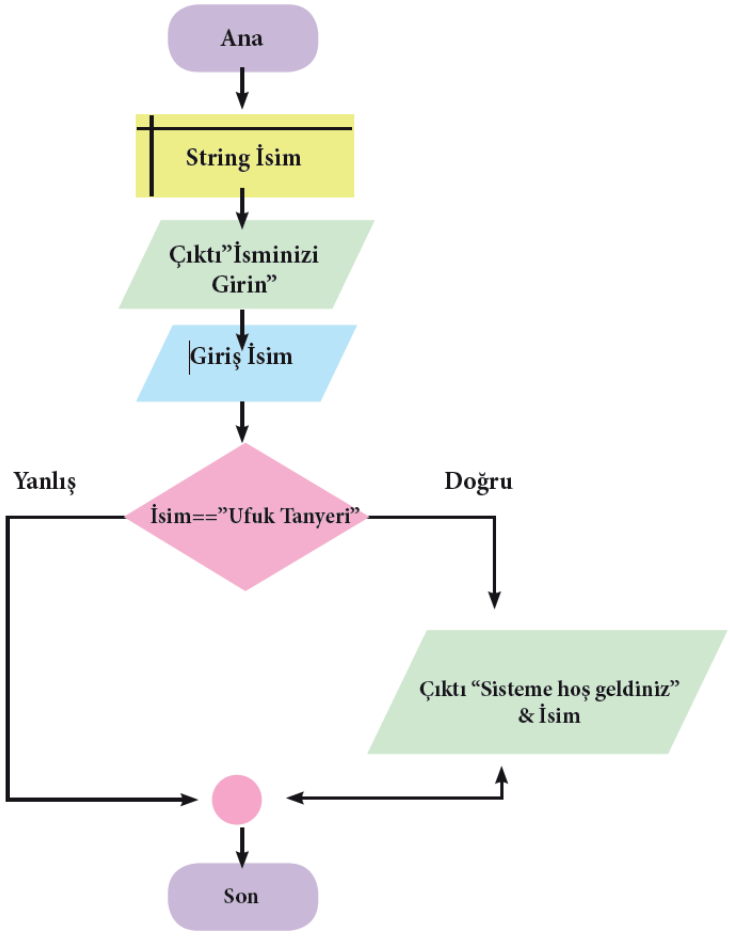
diğer algoritmalar için uygulamayı http://www.flowgorithm.org/download/index.html adresinden indirebilirsiniz. Ancak kullandığınız Windows işletim sisteminin 32/64 bit sürümleri için uygun olanını kullanmanız gerekmektedir.

EK BİLGİ

Akış şemalarını kullanarak programlar oluşturmak için kullanabileceğiniz tek uygulama Flowgorithm değildir. Tercih edilebilecek diğer uygulamalar:

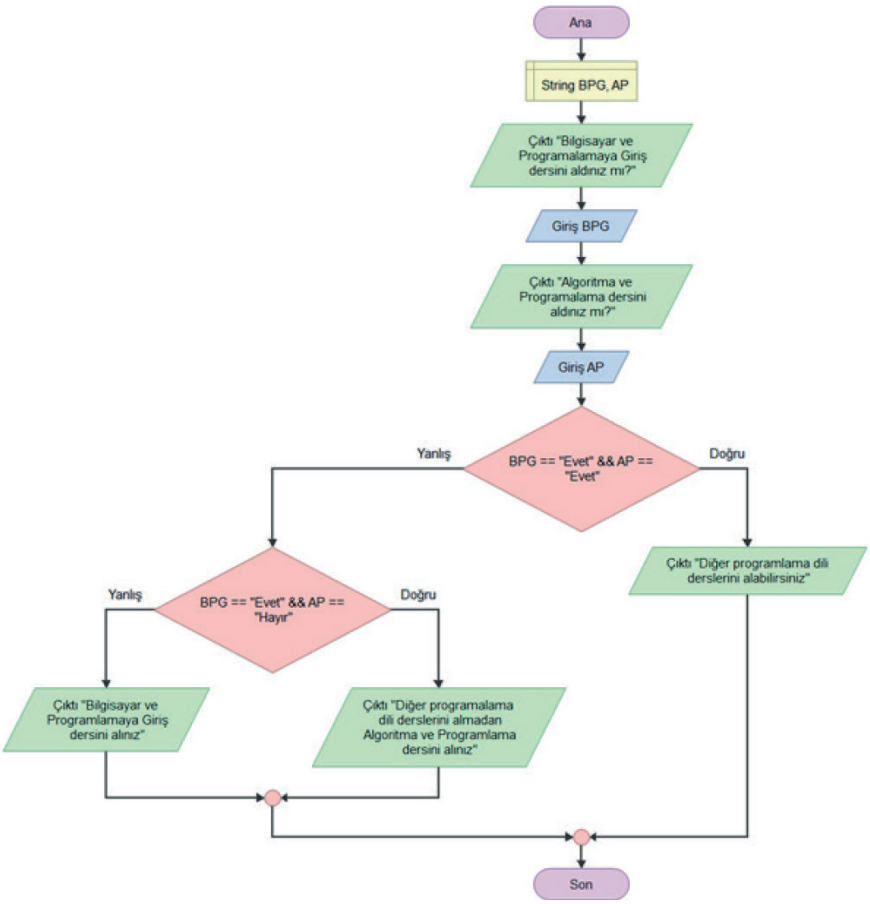
- Algobuild
- LARP
- PSeInt (İspanyolca)
- Raptor
- Visual Logic

Algoritmada adımların akışını değiştiren ifadeler karar kontrol ifadeleridir. Şekil 3.8'de bu duruma örnek bir akış diyagramı verilmektedir.



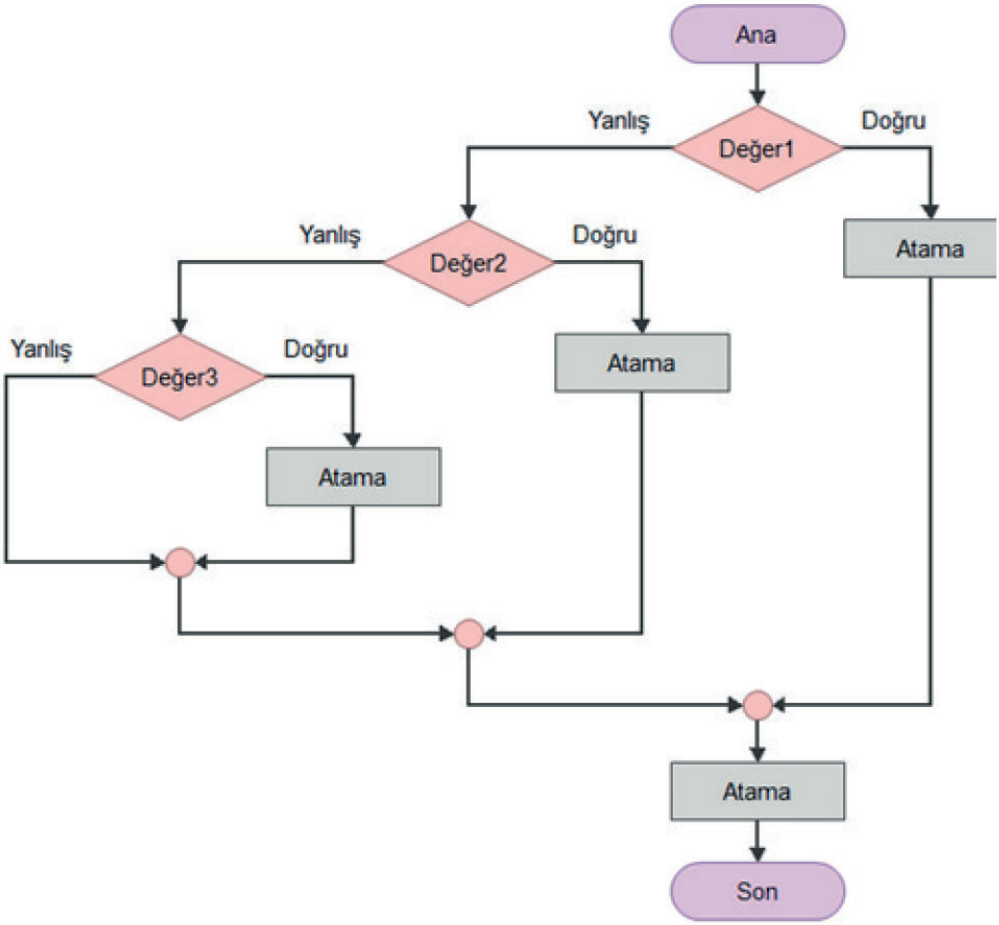
Şekil 3.8. Eğer İfadesi Akış Diyagramı.

Şekil 3.8'de klavyeden okutulan değer "İsim" adlı bir değişkene aktarılmaktadır. Eğer değer "Ufuk Tanyeri" ise "Sisteme hoş geldiniz" yazısı ekrana yazılıp program sonlanacaktır. Değilse sistem sonlanacaktır. Daha karmaşık bir eğer ifadesi Şekil 3.9'da verilmektedir.



Şekil 3.9. Eğer- Değilse Eğer- Değilse İfadesi Akış Diyagramı.

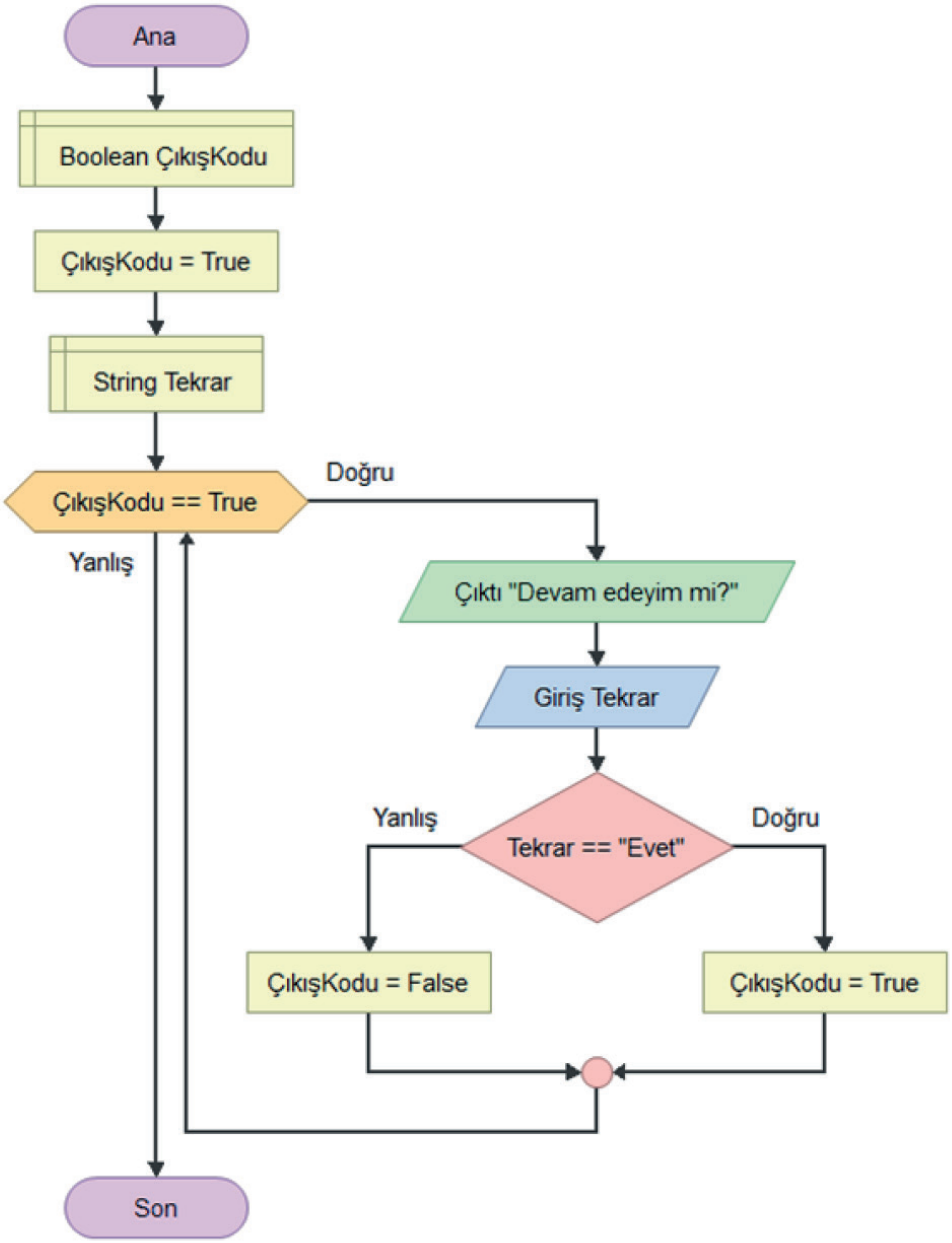
Şekil 3.9'daki yapılar daha karmaşık eğer yapısına örnektir. Burada iç içe eğer ifadesi kullanılmaktadır. Programda Bilgisayar ve Programlamaya Giriş dersi BPG, Algoritma ve Programlama dersi AP isimli karakter katarı (string) değişkenler olarak tanımlanmıştır. Algoritma kişinin bahsi geçen dersleri alma durumuna verdiği "Evet" veya "Hayır" cevabına göre öneride bulunmaktadır. Çok-yönlü karar yapıları için de Şekil 3.10 örnek olarak verilmektedir.



Şekil 3.10. Çok-yönlü Karar İfadesi Akış Diyagramı.

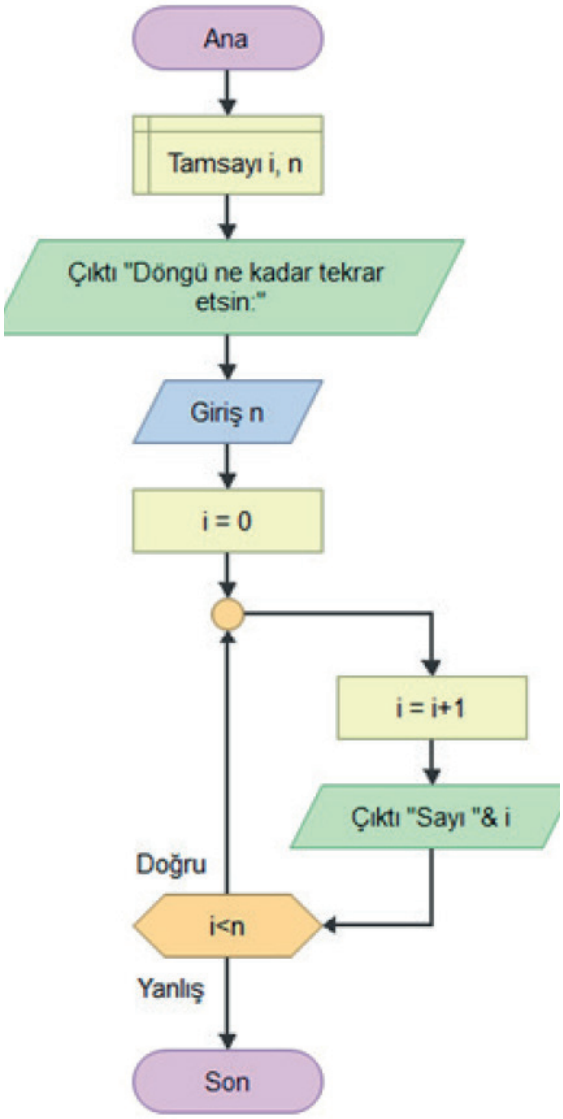
Algoritmada yinelemeli ifadeler belirtilen koşul yanlış olana kadar bir dizi adımın yürütülmesini tekrarlamak için kullanılmaktadır. Bilinen üç tür yinelemeli ifade bulunmaktadır:

- While döngüsü
- Do/Do-While döngüsü
- For döngüsü



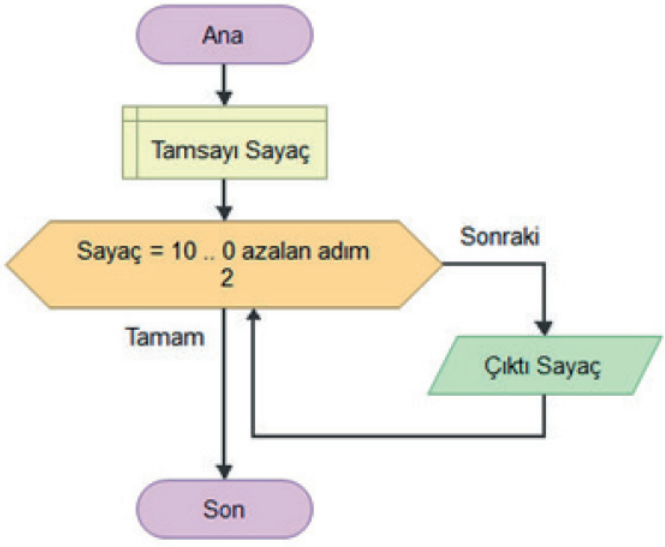
Şekil 3.11. While Döngüsü Akış Diyagramı.

Şekil 3.11'de verilen ÇıkışKodu, Doğru/Yanlış (True/False) değerini tutan mantıksal bir değişkendir ve başlangıçta değeri True'dur. ÇıkışKodu değeri True olduğu sürece program "Devam edeyim mi?" sorusunu soracaktır. Cevap evet olduğu sürece de döngü kendini tekrar edecektir. Başlangıçta ÇıkışKodu değişkeni False olursa döngüye hiç girilmeyecektir. Şekil 3.12, Do While döngüsü için bir örnektir.



Şekil 3.12. Do While Döngüsü Akış Diyagramı.

Şekil 3.12'de verilen Do While döngüsü, Şekil 3.11'de verilen While döngüsünden farklı olarak döngü içerisindeki adımları en az bir kez çalıştıracaktır. Algoritmada i ve n isimli iki tane tam sayı değişken bulunmaktadır. i'nin değeri n'den küçük olduğu sürece Ekrana "Sayı " kelimesi ile birlikte i'nin değeri yazılacaktır. Burada n'nin değeri klavye aracılığıyla kullanıcıdan alınmaktadır. Döngülerin sonuncusu olan for döngüsü örneği de Şekil 3.13'dedir.



Şekil 3.13. For Döngüsü Akış Diyagramı.

Şekil 3.13'de verilen For döngüsü ile While ve Do-While döngüsünde olduğu gibi belirlenen koşul sağlanana kadar döngü içinde verilen adımlar tekrarlanacaktır. Şekil 3.13'deki örnekte tam sayı olarak tanımlanmış Sayaç değişkenine başlangıçta 10 değeri verilmiş ve değeri 2'şer azaltılmıştır. Döngüye koşul olarak da Sayaç değişkeninin 0'dan büyük eşit olması verilmiştir. Program çalıştığında ekrana 10'dan 0'a kadar 0 dahil çift sayılar yazılacaktır.

5. SÖZDE KOD

Buraya kadar algoritmanın yazılmasını ve akış diyagramlarının algoritmadaki adımlara göre gösterilmesini gördük. Sözde kod ise hazırladığımız algoritmanın belli bir programlama dili ile kodlamasına geçilmeden önce koda yakın bir dille kabaca yazılması işlemidir. Bir nevi ara kod olarak söylenebilir. Kendine has bir dili vardır. Gaddis ve IBO sözde kod için kullanılan standart gösterimlerdir. Akış diyagramında verilen örneklerin sözde kod halleri Gaddis kullanılarak aşağıda verilmiştir.

Şekil 3.8'de verilen Basit bir karar yapısı örneği olan "Eğer İfadesi" nin sözde kod hali:

```
Declare String isim
Display "İsminizi girin"
Input isim
If isim == "Ufuk Tanyeri" Then
    Display "Sisteme hoş geldiniz ", isim
End If
```

Burada Declare, "isim" adında bir değişken tanımlamak için kullanılmaktadır. Declare ile tam sayı (Integer), ondalıklı sayı (Real), karakter katarı (String) ve doğru/yanlış (True/False) gibi mantıksal değerler (Boolean) tanımlanabilir. Dersimizin ilerleyen bölümlerinde C++ söz dizimine uygun tanımlamalar anlatılacaktır. Display ile yanında tırnak içerisinde verilen bilgi ya da değişken ekrana yazılır. Input, klavyeden bilgi girişi yapmak için kullanılmaktadır. Değer Input yanında yazılan değişkene aktarılır. If Then End If kısmı isi Eğer ifadesinin gerçekleştiği yerdir. If ile koşul atanır. Then koşul sağlandığında işletilecek adımları vermektedir. End If ile de koşul sonlandırılır.

Şekil 3.9'da verilen "Eğer- Değilse Eğer- Değilse İfadesi" nin sözde kod hali:

```
Declare String bPG, aP

Display "Bilgisayar ve Programalamaya Giriş dersini aldınız mı?"
Input bPG
Display "Algoritma ve Programalama dersini aldınız mı?"
Input aP
If bPG == "Evet" AND aP == "Evet" Then
    Display "Diğer programlama dili derslerini alabilirsiniz"
Else
If bPG == "Evet" AND aP == "Hayır" Then
    Display "Diğer programalama dili derslerini almadan Algoritma ve Programlama dersini alınız"
Else
    Display "Bilgisayar ve Programlamaya Giriş dersini alınız"
   End If
End If
```

Şekil 3.10'da verilen "Çok-yönlü Karar İfadesi" nin sözde kod hali:

```
If Değer1 Then
   // Set Ad = Açıklama
Else
   If Değer2 Then
      // Set Ad = Açıklama
Else
   If Değer3 Then
      // Set Ad = Açıklama
   End If
 End If
End If
// Set Ad = Açıklama
```

İlgili işaretten sonra yazılan kod işletilmez. Set ise işlem basamaklarını ifade eder. İlgili örnekte rastgele bir işlem tanımlaması yapılmıştır.

Şekil 3.11'de verilen "While Döngüsü" nün sözde kod hali:

```
Declare Boolean çıkışKodu

Set çıkışKodu = True
Declare String tekrar

While çıkışKodu == True
    Display "Devam edeyim mi?"
    Input tekrar
    If tekrar == "Evet" Then
      Set çıkışKodu = True
    Else
      Set çıkışKodu = False
    End If
End While
```

Bu kısımda diğer sözde kod örneklerinden farklı olarak While döngüsünü ifade eden While End While yapısı yer almaktadır. While'dan sonra yazılan kısım koşuldur. While çıkışKodu = = True ile çıkış-Kodu değişkeninin değerinin True olma şartı aranmaktadır. Değer True olduğu sürece End While yazan satıra kadar ki bütün satırlar işletilecektir.

Şekil 3.12'de verilen "Do While Döngüsü" nin sözde kod hali:

```
Declare Integer i, n

Display "Döngü ne kadar tekrar etsin:"
Input n
Set i = 0
Do
   Set i = i + 1
  Display "Sayı ", i
While i < n
```

Bu kısımda Do ile While arasındaki satır en az bir kez çalıştırılacaktır. Daha sonra While'ın yanındaki koşul sağlandığı sürece işletilecektir.

Şekil 3.13'de verilen "For Döngüsü" nün sözde kod hali:

```
Declare Integer sayaç

For sayaç = 10 To 0 Step -2
Display sayaç
End For
```

For ve End For, for döngüsünün işletildiği kısmı vermektedir. For'un yanında verilen sayaç değişkenine bir başlangıç değeri ve Step ile de adım değeri tanımlanır. Koşulun tamamlanması ise To'dan sonra gelen değer ile belirlenir.

Gaddis sözde kodda kullanılan matematiksel operatörler:

- + (Toplama)
- - (Çıkarma)
- * (Çarpma)
- / (Bölme)
- MOD (Mod İşlemi)
- ^ (Üs operatörü)

İlişkisel operatörler:

- > (Büyüktür)
- < (Küçüktür)
- >= (Büyük eşit)
- <= (Küçük eşit)
- == (Eşit)
- != (Eşit değil)

Mantıksal operatörler:

- AND (Ve)
- OR (Veya)
- NOT (Değil)

Sözde kod, kodlamaya geçmeden pek çok programcının kullandığı bir yazımdır. Bu şekilde yazılan bir program, sonradan istenilen programlama diline çevrilmede kolaylık sağlayacaktır. Bir programlama diline göre başlanılan kodlar ise başka bir programlama diline çevrilirken daha zor olacaktır.

BÖLÜM ÖZETİ

Bu bölüm programlama dillini öğrenmek isteyen tüm programcılar için ortak bir kısımdır. Çünkü hangi programlama diline başlanırsa başlansın ilk yapılan işlemler algoritma oluşturma, akış diyagramı çizme ve son olarak sözde koda dönüştürme olarak ilerlemektedir. Ayrıca bu adımlar program geliştiren tüm yazılım firmalarında zorunlu olarak yapılmaktadır. Günümüzde eğitim amaçlı kullanılan programlama dilleri de bulunmaktadır. Bu dillerin özellikle programlamaya yeni başlayanlar ya da küçük yaştaki programcılar için temel seviyede bir yapısı bulunmaktadır. Scracth, Alice ve Microsoft Small Basic bu tarz programlama dilleridir. Scracth, Massachusetts Teknoloji Enstitüsünün (MIT) hikâyeler, oyunlar ve animasyonlar yaparak kodlamayı öğretme hedefi vardır. Alice, animasyonlar oluşturmayı, etkileşimli anlatılar oluşturmayı veya basit oyunları 3D olarak programlamayı kolaylaştıran yenilikçi bir blok tabanlı programlama ortamıdır. Small Basic, öğrencilerin blok tabanlı kodlamadan metin tabanlı kodlamaya geçişlerine yardımcı olmak için özel olarak oluşturulmuş tek programlama dilidir. Small Basic, sözdizimi tabanlı dillerin temel öğelerini ulaşılabilir bir şekilde öğreterek öğrencilere Java ve C# gibi daha karmaşık programlama dillerinin üstesinden gelme becerisi ve güveni verir. Ayrıca Small Basic kullanarak Kinect, Lego Mindstorm, Raspberry Pi, Arduino, Oculus Rift ve daha fazlası için uygulamalar oluşturabiliriz.

GÖZDEN GEÇİRELİM

1. **Aşağıda verilen yazılım türlerinden hangisi bir sistem yazılımıdır?**
   a. ChromeOS
   b. Python
   c. Open Office
   d. JavaScript
   e. Adobe Creative Cloud

2. **A'nın B'ye göre modunu al matematiksel işleminin kod gösterimi aşağıdakilerden hangisidir?**
   A) A*B
   B) A%B
   C) A+B
   D) A/B
   E) A**B

3. **Baytlar ile işlem yapan ve iki koşulun birden sağlanması istediğimiz mantıksal işlem aşağıdakilerden hangisidir?**
   işlem aşağıdakilerden hangisidir?
   A) Mantıksal DEĞİL
   B) Bitsel XOR
   C) Mantıksal VEYA
   D) Mantıksal VE
   E) Bitsel VE

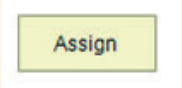
4. **Bir problemin çözümünü sonlu ve sözel adımlarla anlatma işi nedir?**
   a. Program
   B) Yazılım
   C) Sözde Kod
   D) Akış Diyagramı
   E) Algoritma

5. **Aşağıdakilerden hangisi akış diyagramı için kullanılan araçlardan biri değildir?**
   a. Algobuild
   B) Gaddis
   C) PSeInt (İspanyolca)
   D) Raptor
   E) LARP

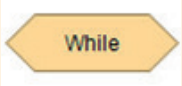
6. Belli bir koşul sağlandığı sürece algoritmanın ilgili satırlarını çalıştıran ifade için gerekli akış diyagramı aşağıdakilerden hangisidir?

a.

b.

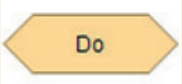


c.



d.

e.



7. Display "Bilgisayar ve Programalamaya Giriş dersini aldınız mı?" ile verilen sözde kod komut satırı ne işe yaramaktadır?

a. Ekrana tırnak içerisinde verilen bilgiyi yazar

b. Klavyeden veri girişi yapar

c. Tırnak içerisinde verilen koşulu arar

d. Yorum satırıdır

e. Bir değişkene tırnak içerisindeki bilgiyi aktarır

8. Gaddis standardına göre mod alma işlemi nasıl yapılır?

A) /

B) ^

C) **

D) %

E) MOD

9. Aşağıdakilerden hangisi eğitsel amaçlı kullanılan bir programlama dili değildir?

A) Scracth

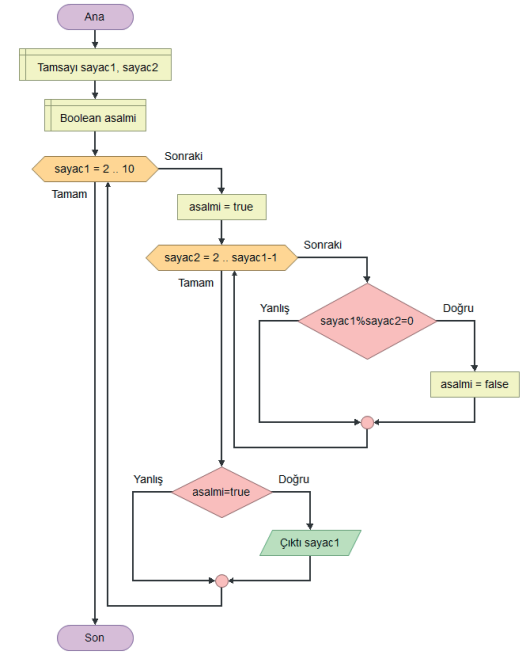
B) Alice

C) Gaddis

D) Microsoft Small Basic

E) Hiçbiri

10. Aşağıdaki akış diyagramının ekran çıktısı aşağıdakilerden hangisidir?



a. 2 4 6

b. 1 3 5 7

c. 3 5 7 9

d. 2 3 5 7

e. 2 4 6 8

Yanıt Anahtarı: 1-A, 2-B, 3-D, 4-E, 5-B, 6-C, 7-A, 8-E, 9-C, 10-D

Kaynakça

Algoritma: Uygulamalı Algoritma Kılavuzu, 5. Baskı, Kadir Çamoğlu, KODLAB, 2011

Algoritma Geliştirme ve Programlamaya Giriş, 13. Baskı, Fahri Vatansever, Seçkin Yayıncılık, 2017

Algoritma ve Programlamaya Giriş, 6. Baskı, Ebubekir Yaşar, Ekin Basım Yayın, 2016

Gaddis, T. (2016). Starting Out with Programming Logic and Design, 4th Edition. Pearson. ISBN 9780133998160

Faydalı İnternet Kaynakları

Kaçar, B. (2021, Aralık 8). YAZILIM GELİŞTİRME VE SÜREÇ MODELLERİ - Berfin Kaçar. Medium. https://medium.com/@brfn.kcr26/yazilim-geli%CC%87%C5%9Fti%CC%87rme-ve-s%C3%BCre%-C3%A7-modelleri%CC%87-2131ea5f09b2

Self learning algorithm example. (n.d.). 11 Şubat 2022'de erişildi. https://www.jolinaforuno.top/products.aspx?cname=self%2Blearning%2Balgorithm%2Bexample&cid=7

Flowgorithm - Flowchart Programming Language. (2014, Ağustos 6). Flowgorithm. 11 Şubat 2022'de erişildi. http://www.flowgorithm.org/