

Flowgorithm Uygulaması ile Algoritma Geliştirme

DOÇ. DR. ÖMER UYSAL & UZMAN GÜRKAN GÜREŞCİ

EĞİTİM
yayınevi

FLOWGORITHM UYGULAMASI İLE ALGORİTMA GELİŞTİRME

Doç. Dr. Ömer Uysal, Uzman Gürkan Güreşçi

Yayınevi Grubu Genel Başkanı: Yusuf Ziya Aydoğan (yza@egitimyayinevi.com)

Genel Yayın Yönetmeni: Yusuf Yavuz (yusufyavuz@egitimyayinevi.com)

Sayfa Tasarımı: Kübra Konca Nam

Kapak Tasarımı: Eğitim Yayınevi Tasarım Birimi

T.C. Kültür ve Turizm Bakanlığı

Yayıncı Sertifika No: 76780

E-ISBN: 978-625-385-621-2

1. Baskı, Aralık 2025

Kütüphane Kimlik Kartı

FLOWGORITHM UYGULAMASI İLE ALGORİTMA GELİŞTİRME

Doç. Dr. Ömer Uysal, Uzman Gürkan Güreşçi

184 s., 210x297 mm

Kaynakça var, izin yok.

E-ISBN: 978-625-385-621-2

Copyright © Bu kitabın Türkiye'deki her türlü yayın hakkı Eğitim Yayınevi'ne aittir. Bütün hakları saklıdır. Kitabın tamamı veya bir kısmı 5846 sayılı yasanın hükümlerine göre kitabı yayımlayan firmanın ve yazarlarının önceden izni olmadan elektronik/mechanik yolla, fotokopi yoluyla ya da herhangi bir kayıt sistemi ile çoğaltılamaz, yayımlanamaz.



Yayınevi Türkiye Ofis: İstanbul: Eğitim Yayınevi Tic. Ltd. Şti., Atakent mah. Yasemen sok. No: 4/B, Ümraniye, İstanbul, Türkiye

Konya: Eğitim Yayınevi Tic. Ltd. Şti., Fevzi Çakmak Mah. 10721 Sok. B Blok, No: 16/B, Safakent, Karatay, Konya, Türkiye
+90 332 351 92 85, +90 533 151 50 42
bilgi@egitimyayinevi.com

Yayınevi Amerika Ofis: New York: Egitim Publishing Group, Inc.
P.O. Box 768/Armonk, New York, 10504-0768, United States of America
americaoffice@egitimyayinevi.com

Lojistik ve Sevkiyat Merkezi: Kitapmatik Lojistik ve Sevkiyat Merkezi, Fevzi Çakmak Mah. 10721 Sok. B Blok, No: 16/B, Safakent, Karatay, Konya, Türkiye
sevkiyat@egitimyayinevi.com

Kitabevi Şubesi: Eğitim Kitabevi, Şükran mah. Rampalı 121, Meram, Konya, Türkiye
+90 332 499 90 00
bilgi@egitimkitabevi.com

İnternet Satış: www.kitapmatik.com.tr
bilgi@kitapmatik.com.tr

EĞİTİM YAYINEVİ
GRUBU

EĞİTİM
yayınevi

SALON
yayınları

Kitapmatik
yayıncılık

kitapmatik
İnternet kitabevi

EĞİTİM
kitabevi

Teşekkür Notu

Gürkan Güreşci: Bu kitabın ortaya çıkma sürecinde bana yol gösteren, sabrımı ve motivasyonumu her daim tazeleyen hocam Doç. Dr. Ömer Uysal'a; bu çalışma için zamanlarından fedakârlık ettiğim eşim Gizem'e ve oğullarım Selim ile Alper'e teşekkür ederim.

Ömer Uysal: Bu kitabın ortaya çıkma sürecinde gösterdiği gayret ve azim için Gürkan Güreşci Hocama; yoğun akademik süreçlerde zaman zaman kendilerini ihmal ettiğim Rüstem Babama, Kadriye Anneme, eşim Zeliha'ya, kızım Zehra'ya ve oğlum Enes'e teşekkür ederim.

Geliştirdikleri algoritmalar ile insanlık adına katma değer üretmeye çalışan herkese teşekkür ederiz.

Önsöz

Flowgorithm Uygulaması ile Algoritma Geliştirme isimli bu kitap, programlama bariyerlerini aşmak ve algoritma mantığını görselleştirerek öğrenmeyi sağlamak amacıyla kaleme alınmıştır. Kitabımız, El-Harezmi'den günümüze uzanan algoritma tarihini modern eğitim araçlarıyla buluşturmaktadır. Flowgorithm'in sunduğu görsel arayüz sayesinde, karmaşık kod satırları arasında kaybolmadan; akış şemaları ile problemleri parçalara ayırmayı, algoritma geliştirmeyi ve sistematik düşünmeyi öğreneceksiniz.

Bu eserde sadece şekillerin nasıl çizileceğini değil; bir problemin nasıl algoritmaya dönüştüğünü adım adım deneyimleyeceksiniz. Üstelik kurduğunuz mantığın C#, Python, Java gibi popüler dillerdeki karşılıklarını görerek, teorik bilginizi uygulamaya dönüştürme fırsatı bulacaksınız. Kitapta, Flowgorithm yazılımı en ince ayrıntısına kadar ele alınmış; temel kavramlardan başlayarak koşullu ifadeler, döngüler, diziler, fonksiyonlar ve daha ileri algoritma yapılarına kadar sistematik bir yapı izlenmiştir.

Kitap ile uyumlu olarak Flowgorithm ile hazırlanan örnek algoritmalar da açık erişiminize sunulmaktadır. Çünkü örnek algoritmalar incelenerek yeni algoritmalar daha kolay geliştirilebilmektedir. Özellikle Flowgorithm ile işlemlerin adım adım takibi, algoritma geliştirme ve hata ayıklama süreçlerini ciddi anlamda görselleştirmekte ve somutlaştırmaktadır. **Flowgorithm uygulama örnekleri şu adres üzerinden siz değerli okurlarımız ile paylaşılmaktadır:** https://drive.google.com/file/d/1ZYg9mJL4yRGCPBen36HBO TPzVaJi7pKT/view?usp=drive_link

Bu kitapla hangi fırsatlar sunulmaktadır?

Bilgi İşlemsel Düşünme (BİD): Problemleri parçalara ayırma, örüntüleri tanıma, soyutlama, algoritmik düşünme gibi 21. yüzyıl temel becerilerini güçlendireceksiniz.

Görsel Öğrenme: Flowgorithm'in dokunmatik arayüzü ile algoritmaları sadece okuyarak değil, akış şemaları çizerek ve uygulayarak öğreneceksiniz.

Hata Ayıklama ve Analiz: Kurduğunuz algoritmayı adım adım çalıştırarak değişken değerlerini görmeyi, mantık hatalarını tespit etmeyi ve bir uzman gibi analiz yapmayı deneyimleyeceksiniz.

Çoklu Dil Desteği: Geliştirdiğiniz algoritmanın **Python, C#, Java, C++** gibi dillerdeki kod karşılığını tek tıkla göreceksiniz, algoritmadan gerçek kodlamaya zahmetsizce geçiş yapacaksınız.

Geleceğin dünyasını kodlayacak olan öğrencilere, onlara rehberlik eden değerli eğitimcilere ve algoritma geliştirme becerisini kazanmak isteyen tüm okurlarımıza, bu yolculukta sağlam bir temel ve güvenilir bir yol arkadaşı olması dileğiyle.

Ömer UYSAL ve Gürkan GÜREŞCİ

Aralık 2025, Bursa

İçindekiler Tablosu

Teşekkür Notu	2
Önsöz	3
Tablolar Listesi	8
Şekiller Listesi	8
1 Bölüm.....	13
1.1 BİLGİ İŞLEMSEL DÜŞÜNCE	13
1.1.1 Eğitim Sistemindeki Önemi ve Uygulama Alanları	14
1.1.2 BİD ve Bilgi ve İletişim Teknolojileri (BİT) Yeterlikleri.....	15
1.2 ALGORİTMA.....	16
1.2.1 Algoritma Nedir?.....	16
1.2.2 Algoritmanın Tarihi	17
1.3 Algoritmadan Programlamaya Aşamalar	17
1.4 Algoritmalarda Kullanılan Terimler	23
1.4.1 Tanımlayıcı	23
1.4.2 Değişken.....	23
1.4.3 Atama.....	23
1.4.4 Sayaç	23
1.4.5 Döngü.....	24
1.5 Algoritmada Kullanılan Operatörler	24
1.5.1 Matematiksel İşlem Operatörleri.....	25
1.5.2 Karşılaştırma Operatörleri.....	25
1.5.3 Mantık Operatörleri	26
1.5.4 Metin Operatörü.....	26
1.6 Algoritma Örnekleri.....	26
2 Bölüm.....	29
2.1 Flowgorithm İndirme Linki	29
2.2 Program Hakkında	29
2.3 Programın Özellikleri ve İşlevleri	30
2.3.1 Arayüz	30
2.3.2 Çoklu Dil Desteği	30
2.3.3 Araç Çubuğu.....	31
2.3.4 Menü Çubuğu.....	33
2.3.5 Dosya menüsü.....	33
2.3.6 Düzenle Menüsü	34

2.3.7	Görünüm Menüsü	35
2.3.8	Program Menüsü	36
2.3.9	Araçlar Menüsü	37
2.3.10	Yardım Menüsü	38
2.4	Programlamada Değişken Kavramı	39
2.4.1	Değişken Türleri	40
2.4.2	Değişkenlere İsim Verme Kuralları	40
2.4.3	Değişken Atama ve Değer Verme	42
2.5	Değişken Değer Bilgisini Kullanıcıdan Alma(Input)	47
2.6	Operatörler	51
2.7	İçsel Fonksiyonlar	52
2.8	Yerleşik Sabitler	54
2.9	Değişken Uygulaması Örneği (Dairenin Çevre ve Alanını Hesaplama)	55
3	Bölüm	56
3.1	Kontrol Yapıları (If)	56
3.1.1	If Uygulaması Örneği (Öğrencinin karne notuna göre başarı durumu)	58
3.1.2	Vücut Kitle Indexi Hesaplayan Program	59
3.2	For Döngüsü	59
3.2.1	Örnek for döngüsü uygulaması (1'den n'ye kadar olan sayıları toplayan program)	62
3.2.2	Örnek For Uygulaması 2(Mükemmel Sayıyı Bulan Program)	63
3.3	While Döngüsü	64
3.3.1	While Döngüsü Örneği(Asal Çarpanları Bulma Uygulaması)	67
3.3.2	While Döngüsü Örneği (Ebob bulma)	70
3.4	DO Döngüsü	71
4	Bölüm	74
4.1	DİZİLER	74
4.1.1	Dizi Örneği 1 (Rakamları dizinin elemanı yapan program)	76
4.1.2	Dizi Örneği 2 (Kullanıcıdan Alınan 5 farkı sayıyı sıralayan program)	77
4.1.3	Dizi Örneği 3 (Kullanıcının Girdiği En Büyük Sayıyı Bulan Program)	80
5	Bölüm	81
5.1	Fonksiyonlar	81
5.1.1	Fonksiyon kullanımı örneği (Bankamatik Uygulaması)	87
6	Bölüm	94
6.1	Grafik Şekilleri	94
6.1.1	Şekli Temizle	94

6.1.2	İleri	94
6.1.3	Ev	94
6.1.4	Dönüş	95
6.1.5	Çokgen Çizme Uygulaması	95
6.1.6	Merdiven çizme uygulaması	99
6.1.7	Fraktal Çizdirme Uygulaması.....	101
7	Bölüm.....	104
7.1	Dosya I/O Şekilleri	104
7.1.1	Açık.....	104
7.1.2	Kapalı	104
7.1.3	Okuma.....	104
7.1.4	Yazma.....	105
7.1.5	Dosya Okuma Örneği	105
7.1.6	1'den 10'a kadar olan sayıların karelerini alıp txt dosyasına yazdıran program	108
8	Bölüm.....	111
8.1	Farklı Yazılım Dillerinde Çıktı Alma	111
8.2	Genel Örnek Uygulamalar	112
8.2.1	ATM Simülasyonu	112
8.2.2	Verilen şemanın Python çıktısı	120
8.2.3	XOX Oyununun Akış Şeması	122
8.2.4	Oyunun Python Çıktısı	140
8.2.5	Ocak Saati Uygulaması	144
8.2.6	Adam Asmaca Oyunu	155
8.2.7	Programın Python Çıktısı.....	165
8.2.8	Bir Belgede Kelime Arama Uygulaması	169
8.2.9	Verilen programın Python Çıktısı	180
9	Kaynakça	182
	Kısa Özgeçmiş.....	183

Tablolar Listesi

Tablo 1-1 Bilgi işlemsel düşünme tanımları.....	16
Tablo 1-2 Akış şemalarında kullanılan semboller (Vatansever, 2017)	20
Tablo 1-3 Matematiksel işlem operatörleri (Aydoğdu, 2020)	25
Tablo 1-4 Karşılaştırma operatörleri (Aydoğdu, 2020)	25
Tablo 1-5 Mantık operatörleri (Şefik & Urhan, 2023)	26
Tablo 2-1 Programın Üst Sekmeleri	32
Tablo 2-2 Tablo sekmelerinin açıklamaları (Atzori, 2022)	34
Tablo 2-3 Düzenle menüsü sekmelerin açıklamaları (Atzori, 2022).....	34
Tablo 2-4 Görünüm menüsü sekmelerin açıklamaları (Atzori, 2022).....	35
Tablo 2-5 Program menüsü sekmelerin açıklamaları (Atzori, 2022)	36
Tablo 2-6 Araçlar menüsü sekmelerin açıklamaları (Atzori, 2022).....	38
Tablo 2-7 Yardım menüsü sekmelerin açıklamaları (Atzori, 2022).....	39
Tablo 2-8 Değişkenin yapısını veren tablo (Atzori, 2022)	41
Tablo 2-9 Değişken olarak kullanılamayan kelimeler (Atzori, 2022).....	41
Tablo 2-10 Kullanılabilecek operatörler (Atzori, 2022).....	52
Tablo 2-11 Matematik işlem örnekleri (Atzori, 2022)	52
Tablo 2-12 Matematik formülleri (Atzori, 2022)	53
Tablo 2-13 String tipi değişken fonksiyonları (Atzori, 2022)	53
Tablo 2-14 Dönüşüm fonksiyonları (Atzori, 2022)	54
Tablo 2-15 Sabitler (Atzori, 2022)	54

Şekiller Listesi

Şekil 1-1 Bilgi İşlemsel Düşünmenin Boyutları	13
Şekil 1-2 Problemin Anlaşılması ve Tanımlanması	17
Şekil 1-3 Girdi ve Çıktıların Belirlenmesi.....	17
Şekil 1-4 Çözüm Yöntemlerinin Belirlenmesi	18
Şekil 1-5 Algoritmanın Geliştirilmesi	19
Şekil 1-6 İki Sayının Toplamı Algoritması	19
Şekil 1-7 Algoritmadan Programlamaya 7 Temel Aşama	22
Şekil 1-8 Doğrusal, Mantıksal ve Döngüsel Algoritma Yapıları	27
Şekil 1-9 Doğrusal Algoritma Akış Diyagramı	27
Şekil 1-10 Mantıksal Algoritma Akış Diyagramı.....	28
Şekil 1-11 Döngüsel Algoritma Akış Diyagramı	28
Şekil 2-1 Programın Üst Sekmeleri	30
Şekil 2-2 Desteklenen Diller.....	30
Şekil 2-3 Dil Değiştirme Sekmesi	31
Şekil 2-4 Dil Değiştir Ekranı	31
Şekil 2-5 Dosya menüsündeki sekmelere ulaşma.....	33
Şekil 2-6 Düzenle menüsündeki sekmelere ulaşma.....	34
Şekil 2-7 Görünüm menüsündeki sekmelere ulaşma.....	35
Şekil 2-8 Program menüsündeki sekmelere ulaşma	36
Şekil 2-9 Değişken değerlerini anlık izleme.....	37
Şekil 2-10 Araçlar menüsündeki sekmelere ulaşma	37
Şekil 2-11 Yardım menüsündeki sekmelere ulaşma.....	38

Şekil 2-12 Değişkenin Yapısı	39
Şekil 2-13 Şekiller menüsü için tıklanması gereken ok.....	42
Şekil 2-14 Akış diyagramına eklenebilecek semboller	42
Şekil 2-15 Değişken atama	43
Şekil 2-16 Değişkenin isim ve türünü belirleme ekranı	43
Şekil 2-17 Değişken eklendikten sonra akış şemasındaki görünüm.....	43
Şekil 2-18 Akış şemasına Atama şekli eklendikten sonra oluşan görünüm	44
Şekil 2-19 Atama ekranında değişkenin değerini değiştirme görüntüsü	44
Şekil 2-20 Değişkenin değeri değiştikten sonra oluşan görüntü	45
Şekil 2-21 Akış şemasına çıktı şekli eklenince oluşan görüntü	45
Şekil 2-22 Çıktı ekranına yazdırmak istenilen bilginin girildiği kısım	46
Şekil 2-23 Bilgi girildikten sonra akış şemasının görünümü	46
Şekil 2-24 Program çalıştırıldığında oluşan çıktı	47
Şekil 2-25 Değişkenlerin tanımlama görüntüsü	48
Şekil 2-26 Çıktı şeklinin eklenmesi	48
Şekil 2-27 Girdi şeklinin eklenmesi	49
Şekil 2-28 Girdi özelliklerinin belirlenmesi	49
Şekil 2-29 Girdi girişinden sonra akış şemasındaki görüntü	50
Şekil 2-30 Çıktı şekillerinin eklenmesi	50
Şekil 2-31 Program çalıştırıldığında oluşan görüntü	51
Şekil 2-32 Program çalıştığında oluşan çıktı	55
Şekil 3-1 Kontrol yapısı (Atzori, 2022).....	56
Şekil 3-2 Tek-çift sayıların ayırmasını yapan algoritma.....	57
Şekil 3-3 Program çalıştırıldığında oluşan çıktı	57
Şekil 3-4 Başarı durumunu belirleyen algoritma.....	58
Şekil 3-5 For döngü şeması (Atzori, 2022)	59
Şekil 3-6 For döngüsünün şemaya eklenmesi	60
Şekil 3-7 Döngünün özelliklerinin belirlenmesi.....	60
Şekil 3-8 Döngü eklenince oluşan akış şeması.....	61
Şekil 3-9 Program çalıştırıldığında oluşan çıktı	61
Şekil 3-10 Belirli bir sayıya kadar girilen sayıların toplamını veren algoritma	62
Şekil 3-11 while döngü şeması (Atzori, 2022)	64
Şekil 3-12 While döngüsünün şemaya eklenmesi	65
Şekil 3-13 Döngü özelliklerinin belirlenmesi.....	65
Şekil 3-14 Döngü eklendikten sonra oluşan görüntü.....	66
Şekil 3-15 Program çalıştırıldıktan sonra oluşan görüntü	67
Şekil 3-16 Asal çarpan algoritması 1. kısım.....	67
Şekil 3-17 Asal çarpan algoritması 2. kısım.....	68
Şekil 3-18 Asal çarpan algoritması çıktı.....	69
Şekil 3-19 Do döngü şeması (Atzori, 2022)	71
Şekil 3-20 Do döngüsü algoritma örneği.....	73
Şekil 4-1 Dizi Tanımlama Ekranı	74
Şekil 4-2 Tanımlanmış dizinin akış şemasındaki görünümü.....	75
Şekil 4-3 Dizinin belirli bir elamanının değerini belirleme.....	75
Şekil 4-4 Dizi şekilleri eklendikten sonra oluşan akış şemasının görüntüsü.....	76
Şekil 4-5 Rakamları dizinin elemanı yapan akış şeması	76
Şekil 4-6 Beş farklı sayıyı sıralayan algoritma örneği.....	78
Şekil 4-7 Programın çıktısı	79
Şekil 5-1 Fonksiyon uygulaması Akış şeması 1. kısım	81
Şekil 5-2 Fonksiyon uygulaması Akış şeması 2. kısım	81

Şekil 5-3 Fonksiyon uygulaması Akış şeması 3. Kısım	82
Şekil 5-4 Fonksiyon uygulaması Akış şeması 4. kısım	82
Şekil 5-5 Fonksiyon uygulaması Akış şeması 5. kısım	83
Şekil 5-6 Fonksiyon uygulaması Akış şeması 6. kısım	83
Şekil 5-7 Fonksiyon uygulaması Akış şeması 7. kısım	84
Şekil 5-8 Fonksiyon uygulaması Akış şeması 8. kısım	84
Şekil 5-9 Fonksiyon uygulaması Akış şeması 9. Kısım	85
Şekil 5-10 Fonksiyon uygulaması Akış şeması 10. kısım	85
Şekil 5-11 Fonksiyon uygulaması Akış şeması 11. kısım	86
Şekil 5-12 Fonksiyon uygulaması Akış şemasının çıktısı	86
Şekil 5-13 Bankamatik uygulaması Ana Algoritma 1. Kısım	87
Şekil 5-14 Bankamatik uygulaması Menu fonksiyon algoritması	88
Şekil 5-15 Bankamatik uygulaması bekle fonksiyonu akış şeması	88
Şekil 5-16 Bankamatik uygulaması Ana Algoritma 2. Kısım	89
Şekil 5-17 Bankamatik uygulaması Ana Algoritma 3. Kısım	90
Şekil 5-18 Bankamatik uygulaması Ana Algoritma 4. Kısım	91
Şekil 5-19 Bankamatik uygulaması Ana Algoritma 5. Kısım	92
Şekil 5-20 Bankamatik uygulaması Ana Algoritma 6. Kısım	93
Şekil 6-1 Temizle	94
Şekil 6-2 İleri	94
Şekil 6-3 Ev	94
Şekil 6-4 Dönüş	95
Şekil 6-5 Çokgen çizme uygulaması akış şeması ana algoritma	95
Şekil 6-6 Çokgen çizme uygulaması for döngüsü	96
Şekil 6-7 Kaplumbağa grafik ekleme ekranı	96
Şekil 6-8 İleri şeklini ekleme ekranı	97
Şekil 6-9 Dönüş şeklini ekleme ekranı	97
Şekil 6-10 Programın çıktısı	98
Şekil 6-11 Ana fonksiyon 1. kısım	99
Şekil 6-12 Ana fonksiyon 2. kısım	99
Şekil 6-13 Programın bu kısma kadar olan çıktısı	100
Şekil 6-14 Ana program 3. kısım	100
Şekil 6-15 Programın son halinin çıktısı	101
Şekil 6-16 Fraktal örneği	103
Şekil 7-1 Açık	104
Şekil 7-2 Kapalı	104
Şekil 7-3 Okuma	104
Şekil 7-4 Yazma	105
Şekil 7-5 Ana algoritma 1. Kısım	105
Şekil 7-6 Okumak istediğimiz dosyayı akış şemasına ekleme ekranı	106
Şekil 7-7 Dosya okuma eklendikten sonra akış şemasındaki görünüm	107
Şekil 7-8 Program çalıştırıldıktan sonra oluşan çıktı	107
Şekil 7-9 Akış şemasına açık şekli ekleme ekranı	108
Şekil 7-10 Okumak istediğimiz dosyayı belirleme ekranı	109
Şekil 7-11 Akış şemasına dosya içinde yapılması istenen şekillerin ekleme ekranı	109
Şekil 7-12 Program çalıştıktan sonra oluşan txt dosyası	110
Şekil 8-1 Flowgorithm ile akış şemalarını dönüştürebileceğimiz diller	111
Şekil 8-2 Ana fonksiyonumuz 1. kısım	112
Şekil 8-3 Bekle Fonksiyonumuz	113
Şekil 8-4 Ana fonksiyon 2. kısım	113

Şekil 8-5 Ana fonksiyon 3. kısım	114
Şekil 8-6 Ana fonksiyon 4. kısım	114
Şekil 8-7 Menü fonksiyonu	115
Şekil 8-8 Ana fonksiyon 5. kısım	116
Şekil 8-9 Ana fonksiyon 6. kısım	117
Şekil 8-10 Ana fonksiyon 7. kısım	118
Şekil 8-11 Ana fonksiyon 8. kısım	119
Şekil 8-12 Ana fonksiyon 9. kısım	120
Şekil 8-13 Oyun Genel Yapısı	122
Şekil 8-14 Ana fonksiyon 1. kısım	122
Şekil 8-15 Ana fonksiyon 2. kısım	123
Şekil 8-16 Tahta çizme fonksiyonu	123
Şekil 8-17 Ana fonksiyon 3. kısım	124
Şekil 8-18 Oyuncu hamlesi çizme fonksiyonu	125
Şekil 8-19 Ana fonksiyon 4. kısım	125
Şekil 8-20 Oyuncu karesi seçme Fonksiyonu	126
Şekil 8-21 Oyuncunun geçerli kare seçtiğini kontrol eden fonksiyon.....	126
Şekil 8-22 Ana fonksiyon 5. kısım	127
Şekil 8-23 Tahtayı güncelleme fonksiyonu	128
Şekil 8-24 Ana fonksiyon 6. kısım	129
Şekil 8-25 Ana fonksiyon 7. kısım	129
Şekil 8-26 Ana fonksiyon 8. kısım	130
Şekil 8-27 Bilgisayarın hamle seçeceği fonksiyon.....	131
Şekil 8-28 Kazananın kontrol edildiği fonksiyon.....	132
Şekil 8-29 Satır sayısının hesaplandığı fonksiyon.....	133
Şekil 8-30 Sütun sayısının hesaplandığı fonksiyon.....	134
Şekil 8-31 Çapraz aynı sembollerin yan yana geldiğini kontrol eden fonksiyon.....	135
Şekil 8-32 Kazananı kontrol eden fonksiyonun son kısmı	136
Şekil 8-33 Ana fonksiyon 9. kısım	137
Şekil 8-34 Ana fonksiyon 10. kısım	138
Şekil 8-35 Ana fonksiyon 11. kısım	139
Şekil 8-36 Ana fonksiyon 1. kısım	144
Şekil 8-37 Çizdirilmek istenen saat tasarımı	145
Şekil 8-38 Saat yüzeyini çizen algoritma 1. kısım	146
Şekil 8-39 Saat yüzeyini çizen algoritma 2. kısım	147
Şekil 8-40 Saat yüzeyindeki çentikleri çizen algoritma 1. kısım.....	148
Şekil 8-41 Saat yüzeyindeki çentikleri çizen algoritma 1. kısım.....	149
Şekil 8-42 Saat yüzeyini çizen algoritma 3. kısım	149
Şekil 8-43 Ana fonksiyon 2. kısım	150
Şekil 8-44 saat animasyonunu gerçekleştiren algoritma 1. kısım.....	151
Şekil 8-45 Saat imlecinin konumunu ayarlayan algoritma.....	151
Şekil 8-46 saat animasyonunu gerçekleştiren algoritma 2. kısım.....	152
Şekil 8-47 İmleci merkeze taşıyan algoritma	153
Şekil 8-48 Programın genel çıktısı	154
Şekil 7-49 Program için gerekli olan değişken isimleri.....	155
Şekil 8-50 Kullanıcının girdiği kelimedeki harfleri diziye toplayan fonksiyon.....	156
Şekil 8-51 kelimedeki harfleri dizinin elemanı yapan fonksiyon.....	158
Şekil 8-52 Kullanıcının girdiği harfin doğruluğunu kontrol eden fonksiyon.....	159
Şekil 8-53 Ana fonksiyon 2. kısım	160
Şekil 8-54 Ana fonksiyon 3. kısım	160

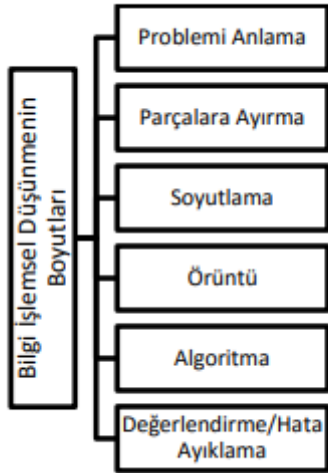
Şekil 8-55 Ana fonksiyon 4. kısım	161
Şekil 8-56 Kullanıcının kalan hakkını kontrol eden fonksiyon	162
Şekil 8-57 Ana fonksiyon 5. kısım	163
Şekil 8-58 Ana fonksiyon 6. kısım	164
Şekil 8-59 Ana fonksiyon 1. kısım	169
Şekil 8-60 Ana fonksiyon 2. kısım	169
Şekil 8-61 Ana fonksiyon 3. kısım	170
Şekil 8-62 Kelime arama fonksiyonu 1. kısım	171
Şekil 8-63 Kelime arama fonksiyonu 2. kısım	172
Şekil 8-64 Parça al fonksiyonu 1. kısım	173
Şekil 8-65 Parça al fonksiyonu 2. kısım	174
Şekil 8-66 Kelime arama fonksiyonu 3. kısım	175
Şekil 8-67 Parça al fonksiyonu 3. kısım	176
Şekil 8-68 Parça al fonksiyonu 4. kısım	177
Şekil 8-69 Ana fonksiyon 4. kısım	178
Şekil 8-70 Ana fonksiyon 5. kısım	179
Şekil 8-71 Programın çıktısı	180

1 Bölüm

1.1 BİLGİ İŞLEMSEL DÜŞÜNCE

Günümüzün hızla ilerleyen ve teknoloji odaklı 21. yüzyılında, bireylerden sahip olmaları beklenen bir dizi temel beceri bulunmaktadır. Bu becerilerden biri de düşünme becerileridir. Günümüzde eleştiril düşünme, yansıtıcı düşünme, yenilikçi düşünme, tasarımcı düşünme, algoritmik düşünme, analitik düşünme, üst düzey düşünme vb. düşünme becerilerinin farklı formlarıdır. Bilgisayar ve internetin hayatımıza girmesi ile Bilgi İşlemsel Düşünme (BİD) özellikle öne çıkan ve önemi giderek artan bir düşünme biçimidir. Bu kavram, Uluslararası Eğitim Teknolojileri Topluluğu (ISTE) tarafından 21. yüzyıl becerilerinin ön koşulu olarak kabul edilmekte ve insanların bilgisayarlarla iş birliği yapmasına olanak tanıyan bir düşünme biçimi olarak tanımlanmaktadır. (Sarıkaya, 2019)

BİD, bilgisayar bilimlerinin temel kavramlarından yararlanarak problemleri çözme, sistemler tasarlama ve insan davranışlarını anlama yaklaşımıdır. Bu beceri, sadece bilgisayar uzmanlarına özgü olmakla kalmayıp, toplumun tüm kesimlerindeki bireyler tarafından edinilmesi gereken evrensel bir yetenek olarak görülmektedir. Saniyede bir milyarın üstünde işlem yapabilen bilgisayarların işlem ve çalışma mantığı, düşünme becerilerinin farklı boyutlarına yönelik insanlara somut bir örnek olmaktadır. Şekil 1-1’de bilgi işlemsel düşüncenin boyutları görülmektedir. (Şefik & Urhan, 2023)



Şekil 1-1 Bilgi İşlemsel Düşünmenin Boyutları

İnsanoğlu için hangi yüzyılda yaşarsa yaşasın sahip olması gereken en önemli becerilerden biri de problem çözmedir. Bilgi işlemsel düşünme ve bileşenleri problem çözme becerilerinin güçlenmesini sağlamaktadır. Bilgi işlemsel düşünmenin bileşenleri genellikle şu şekilde ifade edilmektedir:

Problemi anlama: Problemin neden kaynaklandığını, hangi değişkenler ile ilgili olduğunu net bir şekilde tespit etme. Örneğin doktorlar hastalara sorular sorarak hastalığın tam olarak ne olduğunun belirlenmesi.

Problemi parçalara ayırma: Büyük problemleri daha küçük ve yönetilebilir parçalara bölme. Örneğin bir araçta sorun olduğunda araca bağlanan bir donanım sayesinde aracın tam olarak hangi parçadan ötürü çalışmadığını ortaya konması.

Örüntü tanıma: Verilerdeki veya problemlerdeki hareket şeklini belirleme. Örneğin trafik ışıklarında kırmızıdan sonra sarı ve sarıdan sonra yeşil ışığın yanması sürekli bir düzen içinde tekrarlanması.

Soyutlama: Gereksiz ayrıntıları göz ardı ederek problemin temel noktalarına odaklanma. Örneğin araç hata tespit testinde sorunun motorda olduğunun anlaşılması üzerine kaporta, elektrik, panel vb. bölümlerinin kapsam dışına alınması.

Algoritmik düşünme: Bir problemi çözmek için adım adım bir yol haritası oluşturma. Yol haritası oluşturulurken tüm giriş ve çıkış parametreleri dikkate alınmalıdır. Tüm detaylar göz önünde bulundurulmalıdır. Başla ile bitir arasında tüm yapılması gereken işlemler birer birer eksiksiz ve atlamadan listelenmelidir. Alternatif algoritmalara hedeflere ulaşma sürecinin etkililik ve verimlilik durumlarına göre karar verilmektedir.

Değerlendirme/Hata Ayıklama: Problem çözme sürecinin son aşaması genellikle değerlendirme aşamasıdır. Bu sayede daha iyi bir çözüm yönteminin olup olmadığı üzerinde durulur ve belirlenen çözüm yönteminin üstünlük ve zayıflıkları analiz edilir. Varsa daha iyi bir çözüm yöntemi geliştirilmeye çalışılır. Bilgisayar programlama sürecinde kodlama aşaması tamamlandıktan sonra program çalıştırılır ve varsa programın çalışmasını engelleyen hatalı kodlar ayıklanır. Bu şekilde program çalıştırılarak bilgisayar ortamındaki problem çözülmüş olur.

BİD; problem çözme başta olmak üzere yaratıcılık, düşünme becerileri, karar verme, sorumluluk, analiz, özgüven ve işbirlikli öğrenme gibi diğer önemli 21. yüzyıl becerileriyle de yakından ilişkilidir.

1.1.1 Eğitim Sistemindeki Önemi ve Uygulama Alanları

Eğitim sisteminin temel unsuru olan öğretmenlerin ve öğrencilerin BİD becerilerine sahip olması, eğitimin verimliliği ve etkililiği açısından büyük önem taşımaktadır. Hem Uluslararası Eğitimde Teknoloji Topluluğu (ISTE) hem de Millî Eğitim Bakanlığı (MEB) gibi ulusal ve uluslararası kuruluşlar, öğretmenlerin teknoloji okuryazarlığı ve BİD becerilerinin geliştirilmesi gerektiğini belirtmektedir.

Öğretmenlerin BİD becerileri, öğrencilerin öğrenmeye ve teknolojiye yönelik tutumlarını olumlu yönde etkileme gücüne sahiptir.

BİD'in eğitimdeki uygulama alanları oldukça geniştir. Özellikle bilgisayar programlama, matematik eğitimi ve yapay zeka BİD'in entegrasyonu için verimli bir zemin sunmaktadır. BİD tabanlı etkinlikler, öğrencilerin programlamayı, matematiği daha iyi anlamalarına, günlük hayatla ilişkilendirmelerine, karar verme, problem çözme, parçalara ayırma, örüntü tanımlama, soyutlama ve algoritma geliştirme vb. gibi önemli becerileri kazanmalarına yardımcı olmaktadır.

Günümüzde özellikle blok tabanlı programla herkesin anlaması gereken bir programlama türü olarak karşımıza çıkmaktadır. Arabalar, evler akıllanmakta ve temizlik robotları ev hanımları tarafından blok tabanlı programlanarak kullanılabilmektedir. Programlama, BİD becerilerinin geliştirilmesinde başvuru olan etkili bir yöntemdir. Öğrencilerin algoritmik düşünme ve problem çözme becerileri üzerinde programlama olumlu bir etki göstermektedir. Scratch, mblock, Python gibi kodlama platformları, simülasyon yazılımları ve robotik kitler, BİD tabanlı etkinliklerde sıkça kullanılan araçlardır.

1.1.2 BİD ve Bilgi ve İletişim Teknolojileri (BİT) Yeterlikleri

Bilgi ve İletişim Teknolojileri (BİT) yeterlikleri, öğretmenlerin eğitimde teknolojiyi etkin, verimli ve bilinçli bir şekilde kullanabilme becerilerini ifade etmektedir. Bilgi işlemsel düşünme ve BİT yeterlikleri birbiriyle yakından ilişkili kavramlardır. Yapılan araştırmalar, öğretmen adaylarının BİT yeterlikleri ile BİD becerileri arasında pozitif yönde ve zayıf düzeyde bir ilişki olduğunu göstermiştir (Akgün, 2020). Bu durum, bilgisayar donanım ve yazılım bilgisine sahip olmanın, BİT yeterliklerini ve dolayısıyla BİD becerilerini olumlu yönde etkileyebileceğini düşündürmektedir. Bilgisayarı tanımak ve onu programlayabilmek için nasıl çalıştığını bilmek ve işlemleri nasıl yaptığını anlamak gerekmektedir. Bu doğrultuda BİT yeterlikleri, BİD becerilerinin gelişmesine destek olmaktadır.

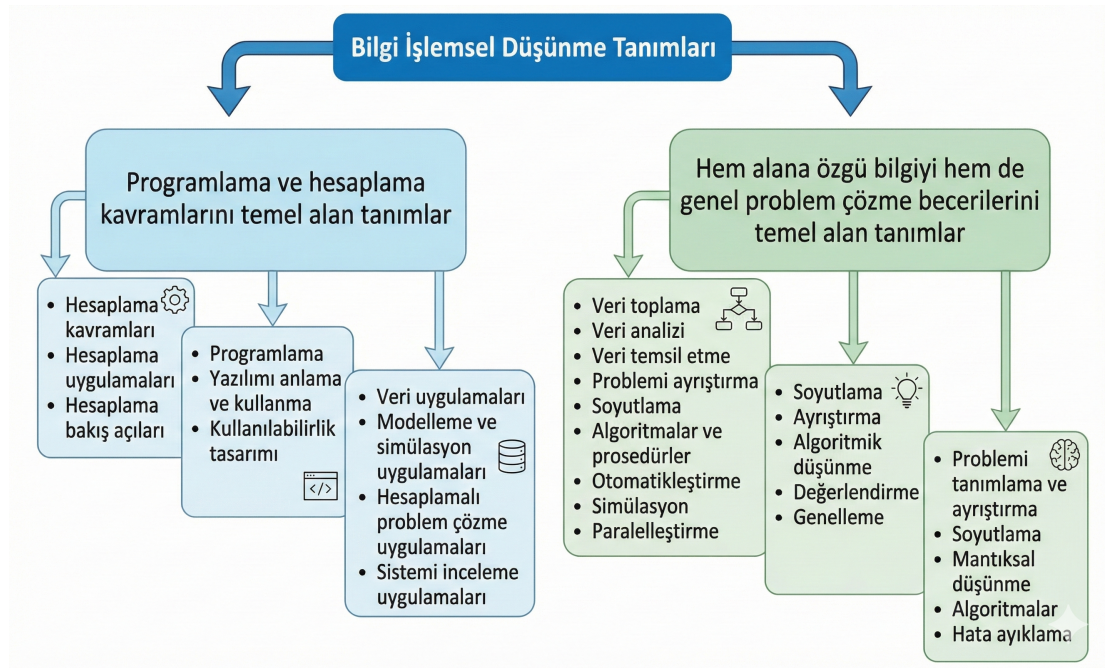
Bir kişi bilgisayarların ve diğer makinelerin dilini okuyabiliyor ve yazabiliyorsa kod okuryazarı olarak adlandırılmaktadır. Kod okuryazarlarının bir problemin çözüm sürecinde yürüttükleri akıl yürütme eylemlerinin temelinde bilgi işlemsel düşünce yer almaktadır (Şefik & Urhan, 2023). Papert 1980 tarafından ortaya atılan bilgi işlemsel düşünce kavramı ilk kez Wing 2006 tarafından “Problem çözmeye bir yaklaşım, sistem tasarlama programlamaya temel olan insan davranışlarını anlama” şeklinde tanımlamıştır.

Bilgi işlemsem düşünce insanın zekasını ve hayal gücünü kullanarak bilgisayarın düşünme tarzını ve kapasitesini geliştirmesi, problem çözmeye, iletişim ve etkileşim becerilerinin düzenlemeye odaklı mantıksal bir düşünme sürecidir (Şefik

& Urhan, 2023). Bilgi işlemsel düşünme becerisi, adından dolayı ilk başta bilgisayar ile ilgili çağrışımlarda bulunsa da çok daha geniş bir etki alanına sahiptir (Üzümcü, 2018). Bilgisayarların çalışma sistemleri, insanların günlük yaşamdaki problem çözme sistemi olarak düşünüldüğünde, sadece bilgisayar ile ilgili kişilerin değil, herkesin bu beceriye sahip olması gerektiği öne sürülmektedir (Wing, 2006)

Bilgi işlemsel düşünme sürecine yönelik tanımlamalarda yapılan vurguların sınıflandırılmasına ilişkin diyagram (Tang ve diğerleri, 2020) Tablo 1-1’de yer almaktadır.

Tablo 1-1 Bilgi işlemsel düşünme tanımları



Algoritmalar, bilgi işlemsel düşünmenin çoğu kaynakta yer alan boyutlarından biridir (Şefik & Urhan, 2023). Şimdi algoritma kavramını inceleyerek devam edelim.

1.2 ALGORİTMA

1.2.1 Algoritma Nedir?

Algoritma, bir işlemin gerçekleştirilmesi için izlenmesi gereken açık ve kesin talimatlar dizisidir (Levitin, 2003). Kısacası algoritmalar, sistemli ve planlı bir şekilde bir sorunu çözmek bir ihtiyacı karşılamak bir için adım adım izlenen bir dizi işlemidir. Yani belli bir problemi çözmek için veya çözüme ulaşmak için tasarlanan yoldur.

Algoritma kelimesi hesaplamalarda veya diğer problem çözme işlemlerinde izlenecek bir süreç veya kurallar kümesi anlamına da gelmektedir. Tüm programlama dillerinde temel olarak algoritma vardır. Algoritma yazılımın omurgasıdır. Bir programın

kullanıldığı dil ne olursa olsun temelinde mutlaka algoritmalar bulunmaktadır. Flowgorithm uygulamasında algoritma geliştirildikten sonra bu algoritmanın Java, Python, PHP, C++, Visual Basic vb. programlama dillerindeki yazılım karşılığı anında görülebilmektedir. Herhangi bir program yazmadan önce algoritma hazırlamak hatalı kodlama oranını azaltarak programı doğru bir şekilde yazmayı kolaylaştırmaktadır.

1.2.2 Algoritmanın Tarihi

Tarihte ilk algoritma örneği Türkistanlı bir bilim insanı tarafından yapılmıştır. El-Harezmi tarafından yazılan “Hisab el-cebir ve el-mukabala” adındaki kitabında ilk kez algoritma kullanılmıştır. Algoritma sözcüğü ise Avrupalılar tarafından El-Harezmi isminin telaffuzundan doğmuştur. Şimdilerde ise algoritma, programlama dilleri vasıtasıyla bilgisayarların, internetin ve yapay zekanın daha etkili bir şekilde çalışmasına öncülük etmektedir (Attila, 2022).

1.3 Algoritmadan Programlamaya Aşamalar

Algoritmanın tasarlanması ve geliştirilmesinde izlenen aşamalar aşağıdaki gibidir. (Aydoğdu, 2020)

1. Problemin tanımlanması: Algoritma bir problem çözme yöntemidir. Problemin etkili kolay ve hızlı bir şekilde çözülebilmesi öncelikle problemin net açık ve doğru bir şekilde anlaşılması ve tanımlanmasına bağlıdır. Örneğin hastalanıp hastaneye gittiğimizde doktor hemen belirli kontroller ve testler yaparak problemi doğru bir şekilde belirlemeye çalışmaktadır. Örnek olarak iki sayının toplamının sorulduğu bir problem ele alınabilir. Farkında olmadığımız dikkatimizi çekmeyen problemleri gündeme almamakta ve çözmeye çalışmamaktayız.



Şekil 1-2 Problemin Anlaşılması ve Tanımlanması

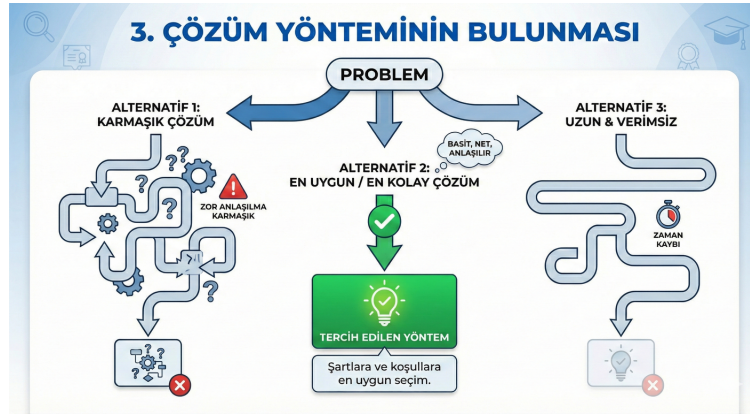
2. Girdi ve Çıktıların Belirlenmesi: Doğru ve net bir şekilde tanımlanan problemin neden kaynaklandığı üzerinde durulur. Algoritmalar ile problem çözümü aynı zamanda sistematik bir yaklaşımı gerektirmektedir. Probleme neden olan girdi değişkenleri ve problemin çözümü sonucunda ortaya çıkması gereken çıktı değişkeni belirlenmelidir. Girdi ve çıktılar belirlendikten sonra problemin çözümü için kullanılabilecek en kısa adımlar karşılaştırılmalıdır. Buna ek olarak probleme neden olan girdilerinin sayısı ve büyüklüğü, adımların toplam maliyeti



Şekil 1-3 Girdi ve Çıktıların Belirlenmesi

ile problemin çözümüne ulaşma hızı vb. dikkate alınabilecek ölçütlerdir. Örneğin yapay zeka girdi olarak kullanıcılardan istem (prompt) almakta ve çıktı olarak istemin yanıtını vermektedir. İstem yani girdiler ne kadar net olarak tanımlanırsa çıktı da o kadar amacına uygun ve hedefe yönelik olmaktadır. Örnek durumumuzu dikkate alarak iki sayının toplamının sorulduğu bir problem ele alındığında Girdiler: A, B ve Çıktılar: Toplam değişkeni ile ifade edilebilir.

3. Çözüm Yönteminin Bulunması: Bir problemin çözümü için birden fazla çözüm alternatifi olabilmektedir. Bu noktada programcının şartlara koşullara en uygun, en kolay olan çözümü tercih etmesi gerekmektedir. Çünkü karmaşık çözümlemeler programa dönüştürüldüğünde anlaşılabilirliği zorlaşabilmektedir. Bir sayının tek mi çift mi olduğu belirlenmek istendiğinde sayının 2'ye bölümünden kalana bakılarak çözüm bulunmaktadır. Kalan 0 olduğunda sayı “çift” ve kalan 1 olduğunda sayı “tek” denmektedir. Problem durumunu ele alırsak çözüm yöntemi $Toplam=A+B$ olacaktır.



Şekil 1-4 Çözüm Yöntemlerinin Belirlenmesi

4. Algoritmanın Geliştirilmesi: Çözüm yöntemi bulunduktan sonra çözüm için tüm işlemler başlangıç ve bitiş adımları arasında tamamlanmaktadır. Giriş değişkenleri üzerinde çeşitli işlemler yapılarak istenen çıktıya ulaşılması hedeflenmektedir. Ortaya çıkan çözüm yolu algoritmamızı oluşturmaktadır. Algoritma oluşturulduktan sonra kontrol edilmelidir. Kontrol esnasında bir eksiklik ya da bir hata ile karşılaşılır ise bu sorunun düzeltilmesi gerekmektedir. Bu eksiklikler ve hatalar giderildikten sonra algoritma kâğıt üzerinde değerler vererek test edilmelidir. Bu şekilde algoritmanın istenen çıktıları üretme durumu garanti altına alınmaktadır. Aşağıda iki sayının toplamının algoritması görülmektedir.

4. ALGORİTMANIN GELİŞTİRİLMESİ

A1-BAŞLA

A2-Girdi ve Çıktıları tanımla (A, B, Toplam)

A3-A'nın değerini gir

A4-B'nin değerini gir

A5-Toplam=A+B

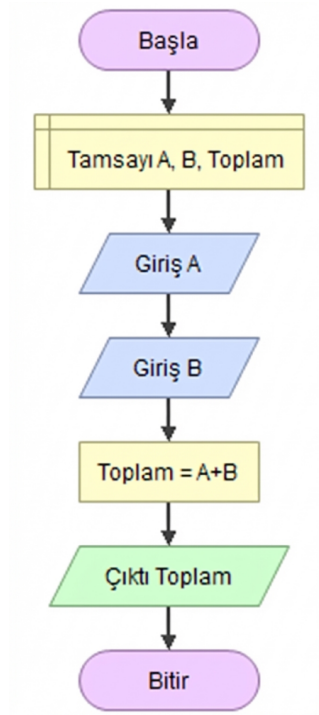
A6-Toplam'ı yaz

A7-BİTİR



Şekil 1-5 Algoritmanın Geliştirilmesi

5. Akış Diyagramının Oluşturulması: Problemi çözebilmek için gerekli tüm işlemler başlangıç ve bitiş adımları arasında tamamlanmaktadır. Akış diyagramlarında algoritma kapsamında belirlenen her bir işlem bir sembol ile ifade edilerek algoritmaya ait akış diyagramı oluşturulmaktadır. Şekil 1-6'da iki sayısının toplamına yönelik akış diyagramı görülmektedir.



Şekil 1-6 İki Sayının Toplamı Algoritması

Tablo 1-2'de akış diyagramlarında yapılacak işleme göre kullanılması gereken semboller görülmektedir (Şefik & Urhan, 2023).

Tablo 1-2 Akış şemalarında kullanılan semboller (Vatansever, 2017)

İşlem	Sembol
Başlangıç-bitiş	
Veri girişi	
Atama-işlemler	
Denetim-karar	
Döngü	
Akış yönü	
Bağlaç	
Önceden tanımlı işlem	
Çıktı	

6.Algoritmanın Program Kodlarına Dönüştürülmesi: İlk 4 aşama genel olarak tüm programlama dillerinde aynıdır denilebilir. Algoritmaların geliştirilmesi programlama dilinden bağımsızdır. Ancak ifadeler, gösterimler ve fonksiyonlar birbirlerinden farklılık göstermektedir. Algoritmanın geliştirilmesi ve doğruluğunun test edilmesinin ardından algoritmanın kodlanması aşamasına geçilmektedir. Bu aşamada, programcı algoritmanın yayınlanacağı ortam, kullanıcı ihtiyaçları ve sistem gereksinimlerini dikkate alarak uygun programlama dilini seçer ve programlama dilinin kendine has ifade, gösterim, komut ve fonksiyonları program kodlarını yazmaktadır. Aşağıda iki sayının toplamına yönelik algoritmanın JavaScript, VBasic, C# ve Phyton dillerindeki kod karşılığı görülmektedir.

- İki Sayının Toplamı Algoritmasının JavaScript Kodu

```
function main() {  
    var a, b, toplam;  
  
    a = Number(window.prompt('Enter a value for a'));  
    b = Number(window.prompt('Enter a value for b'));  
    toplam = a + b;  
    window.alert(toplam);  
}
```

- İki Sayının Toplamı Algoritmasının Visual Basic Kodu

Public Module MyProgram

```
Sub Main  
    Dim a As Integer, b As Integer, toplam As Integer  
  
    a = inputValue()  
    b = inputValue()  
    toplam = a + b  
    Console.WriteLine(toplam)  
End Sub
```

- İki Sayının Toplamı Algoritmasının C# Kodu

public class MyProgram

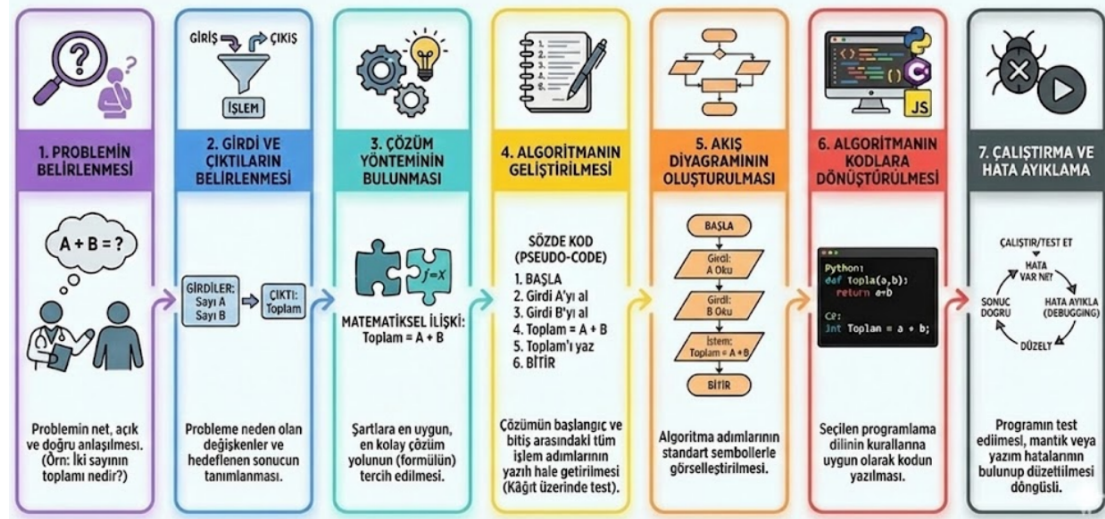
```
{  
    public static void Main(string[] args)  
    {  
        int a, b, toplam;  
  
        a = (int) inputValue();  
        b = (int) inputValue();  
        toplam = a + b;  
        Console.WriteLine(toplam);  
    }  
}
```

- İki Sayının Toplamı Algoritmasının Phyton Kodu

```
a = int(input())  
b = int(input())  
toplam = a + b  
print(toplam)
```

7.Çalıştırma ve Hata Ayıklama: Algoritmanın belirli bir programlama diline dönüştürülmesiyle oluşturulan kodlar, bilgisayar ortamında çalıştırılarak test edilmelidir. Bu aşamanın temel amacı, yazılan programın, en başta net ve açık bir şekilde tanımlanan problemi doğru çözüp çözmediğini ve hedeflenen çıktıları üretip üretmediğini kontrol etmektir. Programlama sürecinde, algoritmanın geliştirilmesi aşamasında kâğıt üzerinde yapılan testlere rağmen gözden kaçan mantık hataları veya seçilen programlama dilinin kurallarına (sözdizimi) aykırı yazılımlar nedeniyle hatalarla karşılaşılabilir. Tıpkı problemin tanımlanması aşamasında bir doktorun problemi belirlemek için testler yapması gibi, programcı da bu aşamada karşılaşılan hataların kaynağını tespit etmek ve düzeltmek için "hata ayıklama" (debugging) işlemini gerçekleştirmektedir. Örneğin, iki sayının toplamı probleminde, eğer program doğru girdiler verildiğinde $A+B$ işlemi yerine yanlış bir çıktı üretiyorsa, bu durum algoritmadaki bir mantık hatasını veya programlama dilinin kurallarına uygun olmayan bir durumun varlığını göstermektedir. Hata ayıklama sürecinde programın algoritması kontrol edilerek bir mantık hatası varsa tespit edilmeye çalışılır. Algoritma ile ilgili bir sorun yoksa programlama dilinin kurallarına uygun olmayan ifadeler, gösterimler, komutlar kontrol edilerek yazım hataları varsa düzeltilir. Hatalar giderildikten sonra program tekrar çalıştırılır ve istenen sonuç garanti altına alınana kadar bu döngü devam eder.

Algoritmalar özellikle bir problemin çözümü ve problem çözümünden hareketle bir yazılımın ortaya çıkarılabilmesi için geliştirilmektedir. Şekil 1-7'de iki sayının toplamını ifade eden algoritmadan programlamaya 7 temel aşama görülmektedir.



Şekil 1-7 Algoritmadan Programlamaya 7 Temel Aşama

Algoritmanın Özellikleri Nelerdir?

- Her algoritmanın bir başlangıç ve bir bitiş noktası vardır.
- Algoritma başlangıç ve bitiş noktası arasında işlemlerden oluşur.

- İşlemler, giriş değişkenlerinin işlenerek adım adım istenen çıktının üretilmesine yöneliktir.
- Her bir işlem açık, net ve anlaşılır olmak zorundadır.
- Her adımın bilgisayarda uygulanabilir olması gerekir.
- Algoritma sonsuz bir döngüye girmemelidir.
- Algoritma problemi çözerken en az süreyi ve en az belleği kullanmayı hedeflemelidir.

1.4 Algoritmalarda Kullanılan Terimler

1.4.1 Tanımlayıcı

Programı yazan kişi tarafından oluşturulan ve programdaki değişkenleri, sabitleri, kayıt alanlarına verilen isimdir. İsimler harfle başlamalıdır ve boşluk içeremezler.

Örnek:



Doğru: VizeNotu, toplamFiyat, sayac



Yanlış: 1.Yazılı (Sayıyla başlamaz), toplam fiyat (Boşluk olmaz)

1.4.2 Değişken

Program her çalıştırıldığında farklı değerler alabilen ve değişebilen alanlardır. Değişkenler, verilerin hafızada saklandığı adresleri temsil ederler. Flowgorithm'de bir değişkeni kullanmadan önce, programın onu tanıması için mutlaka oluşturmamız gerekir. Buna tanımlama denir. Değişken tanımlarken değişkenin tipi de mutlaka belirtilmelidir. integer, string, boolean vb.

Örnek:

İsim: KullaniciAdi -> **Tür:** String (Metin)

İsim: DaireAlan -> **Tür:** Real (Ondalıklı Sayı)

1.4.3 Atama

Bir işlemin sonucunu başka bir değişkende göstermesine atama denir. Atama, oluşturduğumuz değişken içine bir değer koyma veya var olan değeri değiştirme işlemidir. Bilgisayar bu işlemi her zaman **sağdan sola** doğru yapar.

Örnekler:

Yas = 18 (Yas kutusuna 18 değerini koy.)

Alan = KisaKenar * UzunKenar (Sağ taraftaki çarpma işlemini yap, sonucu Alan değişkenine ata.)

1.4.4 Sayaç

Bazı programlamalarda işlemlerin belirli sayıda yapılması için sayaçlar konulur. Genellikle bir döngünün kaç kez döndüğünü takip etmek veya bir olayın kaç kez gerçekleştiğini saymak için kullanılan özel bir *atama* işlemidir. Değişkenin kendi

değeri üzerine ekleme yapılır. Sayaç olarak tanımlanan değişken döngü her döndüğünde (her bir çevrimde) değeri bir artarak sonraki çevrime başlar. Sayaçlar programın duruma göre ikiye, üçer vb. artırılıp azaltılabilir.

Örnekler:

sayac = sayac + 1 (En yaygın kullanım - birer birer artırma)
 toplam = toplam + sayi (Kümülatif toplama - birikimli sayaç)
 faktoriyel=faktoriyel * sayac

1.4.5 Döngü

Programlamada işlem blokları ardışık bir şekilde işlenebilmektedir. Programda belirlenen sayıda gerçekleştiren çevrim yapılarına döngü adı verilmektedir. Döngü, belirli bir bloğun (birden fazla kod satırı), bir koşul sağlandığı sürece (True/Doğru olduğu sürece) tekrar tekrar çalıştırılması işlemidir.

Örnekler:

While Döngüsü: sayac < =100 olduğu sürece toplama işlemi yapılmaya devam edebilir. While döngüsünde ilk işlem koşulun sağlanıp sağlanmadığıdır. Koşul sağlanıyorsa döngü başlar. Bloktaki işlemlerin tamamlanmasının ardından çevrim başında tekrar koşul kontrolü yapılır ve koşullun sağlanma durumunu göre döngü devam eder ya da tamamlanır.

Do-While Döngüsü: sayac < 100 olduğu sürece ekrana toplama işlemini yapmaya devam edebilir. Do-While döngüsünde koşulun sağlanıp sağlanmadığına bloğun en sonunda bakılır. Öncesinde bazı işlemler yapılmaktadır. Koşul sağlanıyorsa döngü sürer, koşul sağlanmıyorsa döngü tamamlanır.

For Döngüsü: 1'den 100'e kadar olan sayılar toplanabilir. For döngüsünde bir sayaç kullanılır ve genelde bu sayaç i harfi ile isimlendirilir. Sayaca bir başlangıç değeri, bir artış değeri verilmekte ve bir de döngünün istenilen zamanda tamamlanabilmesi için koşul ifadesi yer almaktadır.

1.5 Algoritmada Kullanılan Operatörler

Algoritma tasarım ve geliştirmede matematiksel işlem, karşılaştırma ve mantık operatörleri kullanılmaktadır. Ayrıca metin tanımlayan değişkenler için metin operatörü bulunmaktadır.

1.5.1 Matematiksel İşlem Operatörleri

Matematiksel işlem operatörleri toplama, çıkarma, çarpma, bölme, üst alma ve mod işlemlerini ifade etmekte ve bu işlemlerin yapılmasını sağlamaktadır. Tablo 1-3'te matematiksel işlem operatörleri sembolleri, örnekleri ve açıklamaları ile beraber görülmektedir.

Tablo 1-3 Matematiksel işlem operatörleri (Aydoğdu, 2020)

Sembol	İşlem Adı	Örnek Gösterim	Açıklama
^	Üs Alma	2^3	2'nin 3. kuvvetini hesaplar ($2 \times 2 \times 2 = 8$)
*	Çarpma	$3 * 6$	İki değeri çarpar (Sonuç: 18).
/	Bölme	$18 / 3$	İlk değeri ikinci değere böler (Sonuç: 6)
+	Toplama	$6 + 3$	İki değeri toplar (Sonuç: 9)
-	Çıkarma	$16 - 6$	İkinci değeri ilk değerden çıkarır (Sonuç: 10)
%	Mod	$3\%2$	İlk değerın ikinci değere bölümünden kalanı verir (Sonuç: 1)
.	Ondalık Ayracı	3.14	Tam sayı ile kesirli kısmı ayırır (örneğin Pi sayısı).

1.5.2 Karşılaştırma Operatörleri

Karşılaştırma operatörleri algoritmaların karar verme adımlarında koşul yazmak için kullanılırlar. Değişkenlerin sabit bir değerden veya başka bir değişkenin değerinden büyük mü, küçük mü, eşit mi, farklı mı olduğunu belirlemek amacıyla kullanılırlar. Tablo 1-4'te karşılaştırma operatörleri sembolleri, örnekleri ve açıklamaları ile görülmektedir.

Tablo 1-4 Karşılaştırma operatörleri (Aydoğdu, 2020)

Sembol(ler)	İşlem Adı	Örnek Gösterim	Örnek Sonucu
>	Büyük mü?	$5 > 2$	Doğru
<	Küçük mü?	$3 < 8$	Doğru
==	Eşit mi?	$10 = 10$	Doğru
<>, !=	Eşit değil mi, Farklı mı?	$5 <> 5$	Yanlış
>= veya =>	Büyük eşit mi?	$7 >= 7$	Doğru
<= veya =<	Küçük eşit mi?	$4 <= 9$	Doğru

1.5.3 Mantık Operatörleri

Mantık operatörleri algoritmaların karar verme adımlarında koşullar arasında mantık bağlantıları kurmak için kullanılırlar. Koşullar birbirlerine Ve, Veya, Değil mantığı ile bağlanabilirler. Ve mantığı ile bağlanan koşullar için koşullardan yalnızca birinin sağlanamaması koşulun sağlanmadığını göstermektedir. Ancak tüm koşulların sağlandığı durumda sonuç lojik 1 yani doğru olabilmektedir. Veya mantığında ise sonucun lojik 1 yani doğru olabilmesi için koşullardan herhangi birinin sağlanması yeterli olmaktadır. Ancak hiçbir koşul sağlanmıyorsa o zaman sonuç lojik 0 yani yanlış olarak kabul edilmektedir. Değil mantığında ise koşul sağlanmıyorsa sonuç lojik 1 yani doğru, koşul sağlanıyorsa sonuç lojik 0 yani yanlış olarak kabul edilmektedir. Mantıksal ve döngüsel algoritmalarda koşulların sonuçları kullanılan mantık operatörüne göre belli olmakta ve bu sonuca göre karar bloklarına ve döngü bloklarına ait kod satırları çalıştırılmaktadır. Tablo 1-5'te mantık operatörleri ve örnek gösterimleri verilmektedir.

Tablo 1-5 Mantık operatörleri (Şefik & Urhan, 2023)

Operatör	Anlamı	Örnek Gösterim	Örnek Sonucu
AND	Ve	Doğru Ve Yanlış	Yanlış
OR	Veya	Doğru Veya Yanlış	Doğru
NOT	Değil	Değil Doğru	Yanlış

1.5.4 Metin Operatörü

& : Birleştirme (Concatenation) İşlemi = Girilen metinlerin birleştirilerek yan yana getirilmesini sağlamaktadır. Örneğin “Merhaba” & “Ali” işleminin sonucu “Merhaba Ali” olmaktadır. “11” & “11” işlemi yapıldığında sonuç “1111” olmaktadır.

1.6 Algoritma Örnekleri

Bu kitapta doğrusal (sıralı yapı), mantıksal (karar/seçim yapısı) ve döngüsel (tekrarlı/yinelemeli) algoritmalar üzerinde durulmaktadır. Belirtilen algoritmaların sahip olduğu yapı Şekil 1-8’de görülmektedir.



Şekil 1-8 Doğrusal, Mantıksal ve Döngüsel Algoritma Yapıları

Doğrusal Algoritmalar: Algoritmanın ilk adımdan başlayıp, hiçbir sapma yapmadan, yukarıdan aşağıya doğru adım adım ilerleyip sonlanmasıdır. Sadece başla, işlemler ve bitir vardır. Doğrusal algoritmalar içinde hiçbir koşul (if) ve döngü (for, while, do-while) yer almamaktadır. Doğrusal algoritmaya ve akış diyagramına (Şekil 1-9) aşağıdaki üç sayının toplamı örneği verilebilir:

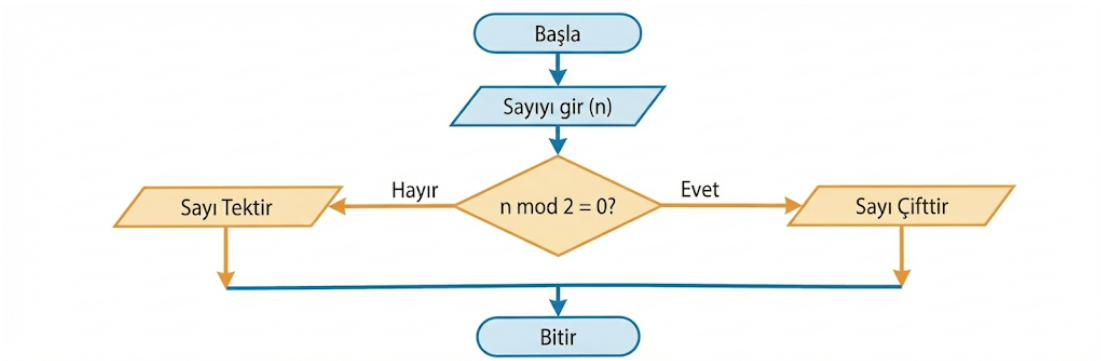
- A1 : Başla.
 A2 : Birinci sayıyı gir.
 A3 : İkinci sayıyı gir.
 A4 : Üçüncü sayıyı gir.
 A5 : Sayıların üçünü topla.
 A6 : Toplam sonucunu üçe böl.
 A7 : Sonucu yaz.
 A8 : Bitir.



Şekil 1-9 Doğrusal Algoritma Akış Diyagramı

Mantıksal (Karar/Seçim) Algoritmalar: Algoritmanın, adımları içinde bir soruya "Evet" veya "Hayır" cevabı vererek yolunu değiştirebilmesidir. Algoritma akışı, belirli bir koşula (if) göre ikiye veya daha fazlaya ayrılır. Başla, işlemler ve bitir adımları yanında evet hayır vb. farklı yanıtları olabilen mutlaka bir koşullu ifade bulunmaktadır. Mantıksal algoritmaya ve akış diyagramına (Şekil 1-10) aşağıdaki sayı tek mi çift mi örneği verilebilir:

- A1 : Başla.
 A2 : Kullanıcıdan bir sayı iste ve bu sayıyı n değişkenine kaydet.
 A3 : **Karar Anı:** n sayısının 2'ye bölümünden kalan 0 mı diye kontrol et.
 • **Eğer Evet ise (Koşul Doğru):** Ekranı "Girdiğiniz sayı Çifttir" yaz.
 • **Eğer Hayır ise (Koşul Yanlış):** Ekranı "Girdiğiniz sayı Tektir" yaz.
 A4 : Bitir.



Şekil 1-10 Mantıksal Algoritma Akış Diyagramı

Döngüsel Algoritmalar: Belirli bir işlemin, bir koşul sağlandığı sürece tekrar tekrar yapılmasıdır. Program akışı doğrusal algoritmada olduğu gibi sürekli ileri gitmez, belirli bir döngü içinde yukarıdan aşağıya doğru koşul sağlanıncaya kadar işlemler yapılmaya devam etmektedir. Döngüsel algoritmalarda doğrusal ve mantıksal algoritma yapıları beraber kullanılmakla birlikte aynı zamanda koşulun durumuna göre tekrar tekrar yukarıdan aşağıya doğru bir işlem akışı söz konusudur. Bu süreç döngüsel algoritmanın daha karmaşık işlemleri yapmasını sağlamaktadır. Döngüsel algoritmaya ve akış diyagramına (Şekil 1-11) 1' den 10'a kadar olan sayıların toplamı örneği verilebilir:

A1 : Başla

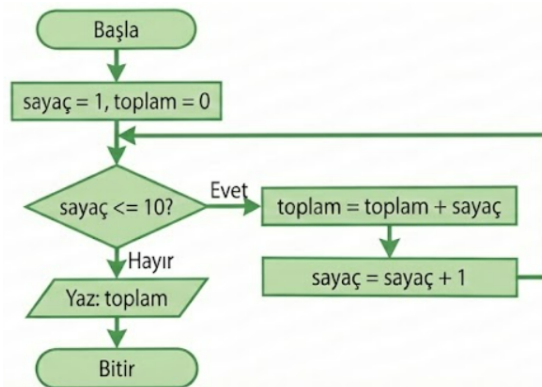
A2 : sayac = 1, toplam = 0

A3 : **Karar (Döngü Koşulu):** Elimizdeki sayac değeri **10'dan küçük veya 10'a eşit mi?** diye kontrol et.

- **Eğer Evet ise (Döngüye Gir):**
 - a. toplam = toplam + sayac
 - b. sayac = sayac + 1
 - c. Tekrar A3 adımına (Karar anına) geri dön.
- **Eğer Hayır ise (Döngüden Çık):** Artık sayac 11 olmuş demektir, şart bozulmuştur. Döngüden çık.

A4 : toplam sonucunu yaz

A5: Bitir



Şekil 1-11 Döngüsel Algoritma Akış Diyagramı

2 Bölüm

2.1 Flowgorithm İndirme Linki

Flowgorithm programı aşağıdaki linkten indirilebilmektedir:

<http://www.flowgorithm.org/download/index.html>



2.2 Program Hakkında

Flowgorithm, akış diyagramları kullanarak program oluşturmayı sağlayan ücretsiz bir uygulamadır. Bilgisayar bilimlerinde akış diyagramı (İngilizcede *flowchart* olarak da bilinir), algoritmaların, prosedürlerin ve işletme talimatlarının kontrol ve yürütme akışını temsil etmek için kullanılan grafiksel bir modelleme dilidir.

Akış diyagramları, şematik veya grafik biçimde aşağıdaki unsurları ifade etmenizi sağlar:

- Yapılacak işlemler, her biri belirli bir mantıksal anlam taşıyan geleneksel şekillerle (dikdörtgen, elips, altıgen, paralelkenar vb.) gösterilir. Bu şekiller genellikle gerçekleştirilecek faaliyeti açıklayan metin içerir.
- İşlemlerin sıralaması, şekiller arasındaki bağlantı oklarıyla temsil edilir.

Bu topolojik özellikleri nedeniyle akış diyagramları, daha geniş bir sınıf olan blok diyagramlarının bir alt türü olarak kabul edilir. Blok diyagramları da bilgiyi tanımlamak ve temsil etmek için kullanılan kavram haritalarının bir parçasıdır.

Genellikle programlar bir metin düzenleyici kullanılarak yazılır. Programlama diline bağlı olarak bu süreç, özellikle yeni başlayanlar için kolay olabileceği gibi oldukça zor da olabilir. Birçok dilde "Merhaba Dünya!" şeklindeki basit bir ifadeyi ekrana yazdırmak bile birkaç satır kod gerektirebilir.

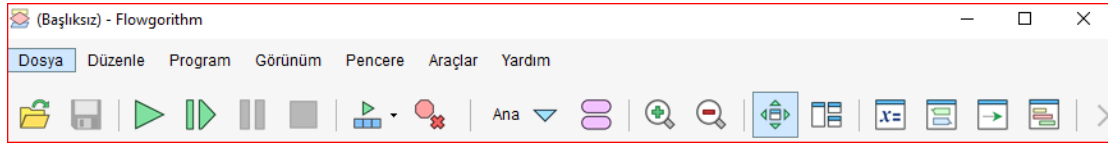
Flowgorithm ise bu süreci kolaylaştırarak, programın gerçekleştirmesini istediğiniz işlemleri farklı sembollerle temsil etmenize olanak tanır. Böylece kullanıcı, bir programlama dilinin ayrıntılarına takılmadan algoritma geliştirmeye (Bilgisayarca Düşünme) odaklanabilir. Flowgorithm ile oluşturulan programlar doğrudan uygulama içinde çalıştırılabilir. Ayrıca, yüksek seviyeli bir programlama dili öğrenmek isteyenler için Flowgorithm akış diyagramlarını birçok popüler dile dönüştürebilir.

Bu diller arasında C#, C++, Delphi/Pascal, Java, JavaScript, Lua, Perl, Python, QBasic, Ruby, Swift 2, Visual Basic .NET, Visual Basic for Applications (Microsoft Office’de kullanılır), TypeScript ve Scala yer almaktadır.

2.3 Programın Özellikleri ve İşlevleri

2.3.1 Arayüz

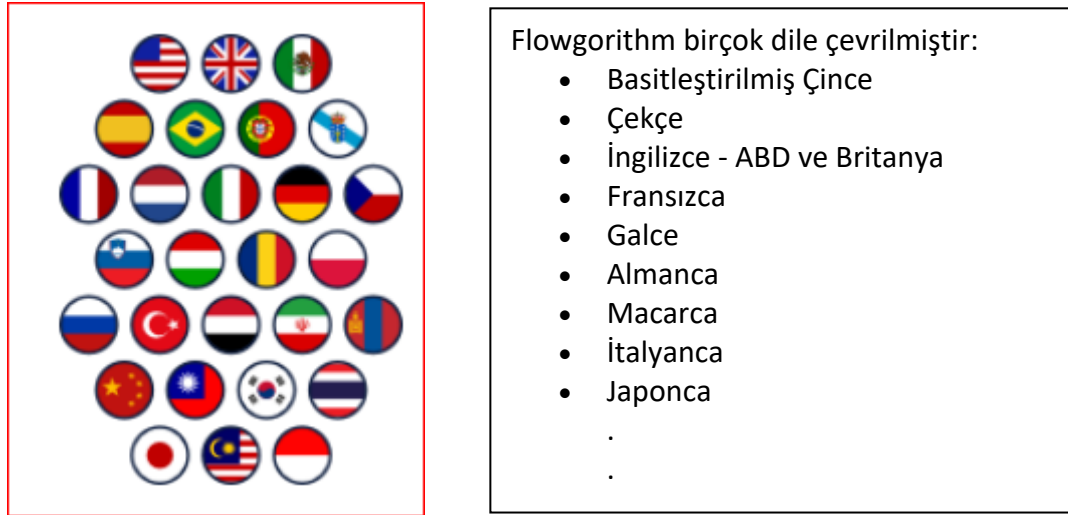
Flowgorithm'ın arayüzü oldukça basit ve sezgisel bir yapıya sahiptir. Menü bölümü, yazılım tarafından sunulan tüm işlevleri içerirken; sekmeler bölümü, algoritmanın kolaylıkla uygulanması, çalıştırılması ve kontrol edilmesine yönelik temel işlevleri barındırır. Aşağıda verilen Şekil 2-1, programın üst sekme görünümünü göstermektedir.



Şekil 2-1 Programın Üst Sekmeleri

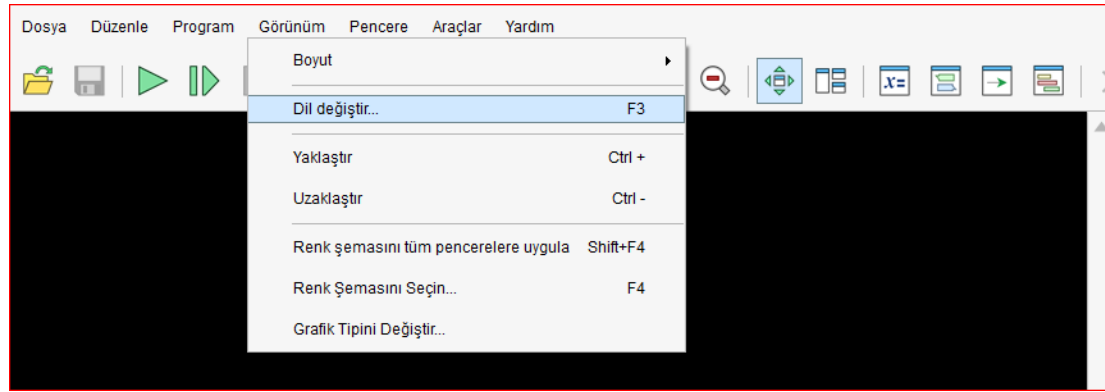
2.3.2 Çoklu Dil Desteği

Aşağıda verilen Şekil 2-2'de Flowgorithm programının desteklediği diller görülmektedir.



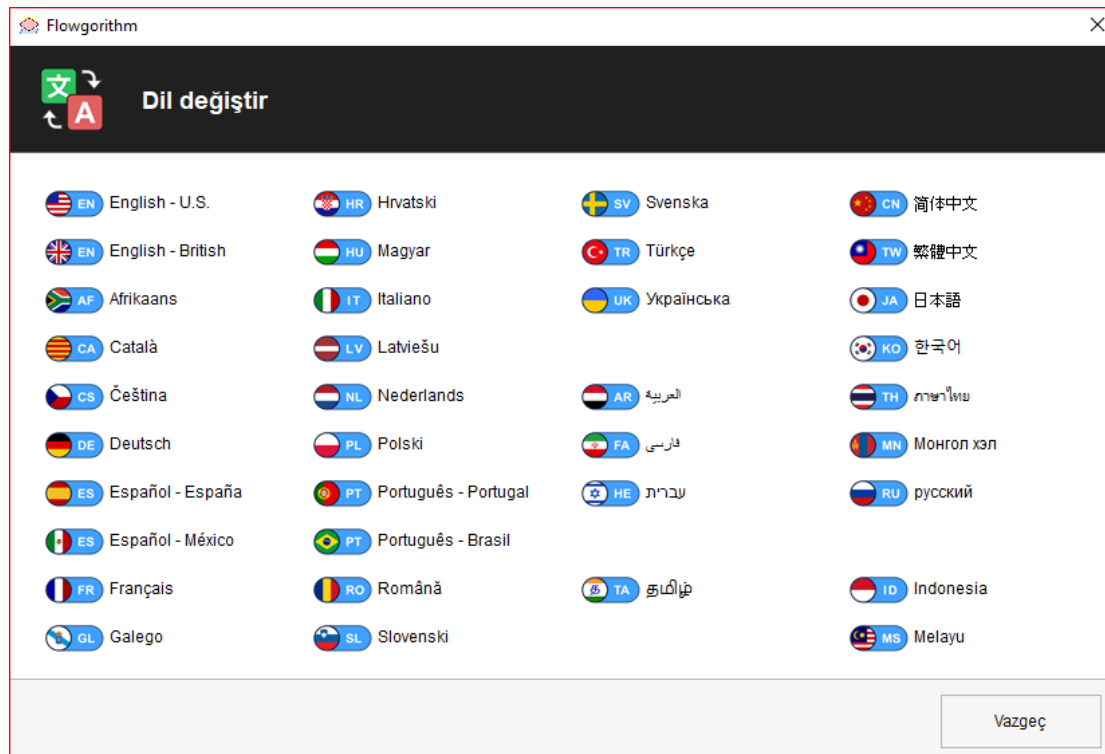
Şekil 2-2 Desteklenen Diller

Görünüm sekmesi altındaki “Dili Değiştir” seçeneğine tıklayarak dil seçme ekranını açabilirsiniz (Şekil 2-3).



Şekil 2-3 Dil Değiştirme Sekmesi

Şekil 2-4’te görüldüğü gibi “Dil değiştir” ekranından istediğiniz dili seçebilirsiniz.



Şekil 2-4 Dil Değiştir Ekranı

2.3.3 Araç Çubuğu

Menü çubuğunun hemen altında, en sık kullanılan komutlara hızlı erişim sağlayan standart araç çubuğu yer alır. Bu araç çubuğu, dosya yönetimi ve önemli işlemlerin yürütülmesi için kullanılan kısayol düğmelerini içerir. Aşağıdaki Tablo 2.1’de bu sekmelere ilişkin açıklamalar verilmiştir.

Tablo 2-1 Programın Üst Sekmeleri

Sekme	Açıklama
	Önceden oluşturulmuş ve kaydedilmiş bir akış diyagramı yükleyin.
	Akış diyagramını kaydedin.
	Akış diyagramını otomatik olarak belirlenen hızda yürütün.
	Sonraki adıma el ile yürütme
	Otomatik modda duraklama yapın.
	Akış diyagramı yorumlayıcısını durdurun.
	Yürütme hızını otomatik modda ayarla.
	Otomatik çalışma hızını ayarla
	Ana program/fonksiyon seçici
	Yakınlaştırmayı artır
	Yakınlaştırmayı azalt
	Renk kombinasyonları
	Otomatik Düzen (pencere ön plana gelecek şekilde yerleştirilecektir)
	Pencere Düzeni ve Seçimi
	"Değişkenler" Penceresi
	Konsol Penceresi
	"Grafik Kaplumbağa" penceresi
	Kaynak kod görüntüleme
	Dil seçimi

Dikkat: Standart araç çubuğundaki düğmeler, menü çubuğunda yer alan en sık kullanılan öğelerle eşleşir. Bu nedenle standart araç çubuğunun amacı, bu öğelere daha hızlı ve kolay erişim sağlamaktır. Örneğin, üzerinde çalıştığınız belgeyi kaydetmek için her seferinde **Dosya** menüsünü açıp **Kaydet** seçeneğini tıklamak yerine, standart araç çubuğundaki ilgili düğmeyi kullanmanız yeterlidir.

Standart araç çubuğundaki düğmelere klavyeyle erişmek de mümkündür:

- “Alt” tuşuna basarak menü çubuğunu etkinleştirin.
- Ardından “Ctrl” + “Tab” tuşlarını kullanarak standart araç çubuğuna geçiş yapın.

Bu işlemten sonra standart araç çubuğundaki ilk düğme seçilecektir. Düğmeler arasında gezinmek için sola doğru ilerlemek amacıyla **Tab**, sağa doğru ilerlemek

için ise **Shift + Tab** tuşlarını kullanabilirsiniz. İstediğiniz düğmeyi bulduğunuzda **Enter** tuşuna basarak ilgili işlemi gerçekleştirebilirsiniz.

2.3.4 Menü Çubuğu

Menü çubuğu, programdaki komutların büyük bölümünü içeren ve menü başlıklarına göre düzenlenmiş seçenek listelerinden oluşur. Menü çubuğunu klavyeyle kullanmak için aşağıdaki adımları izleyebilirsiniz:

- “Alt” tuşuna basarak menü çubuğunu etkinleştirin.
- Sağ veya sol ok tuşlarını kullanarak menü başlıkları arasında gezinin.
- İstediğiniz menüyü bulduğunuzda, menüdeki seçenekleri görmek için yukarı veya aşağı ok tuşlarına basın.
- Uygun seçeneği bulduğunuzda, komutu çalıştırmak için “Enter” tuşuna basın.

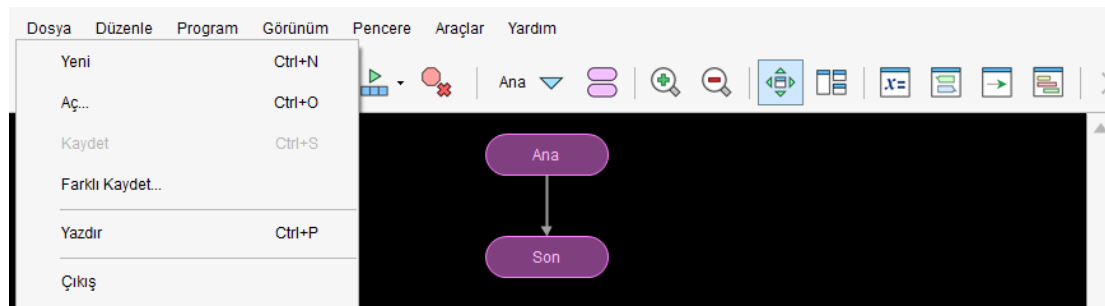
Ayrıca, belirli menüleri hızlıca açmak için aşağıdaki klavye kısayollarını kullanabilirsiniz:

- **Dosya Menüsü:** Alt + F
- **Düzen Menüsü:** Alt + E
- **Görünüm Menüsü:** Alt + V
- **Yürütme Menüsü:** Alt + R
- **Yardım Menüsü:** Alt + H

Bir menüyü kapatmak için menüden bir komut seçebilir, başka bir menüyü etkinleştirebilir veya “Alt” tuşuna basabilirsiniz.

2.3.5 Dosya menüsü

Şekil 2-5’te dosya menüsünde yer alan sekmeler görülmektedir.



Şekil 2-5 Dosya menüsündeki sekmelere ulaşma

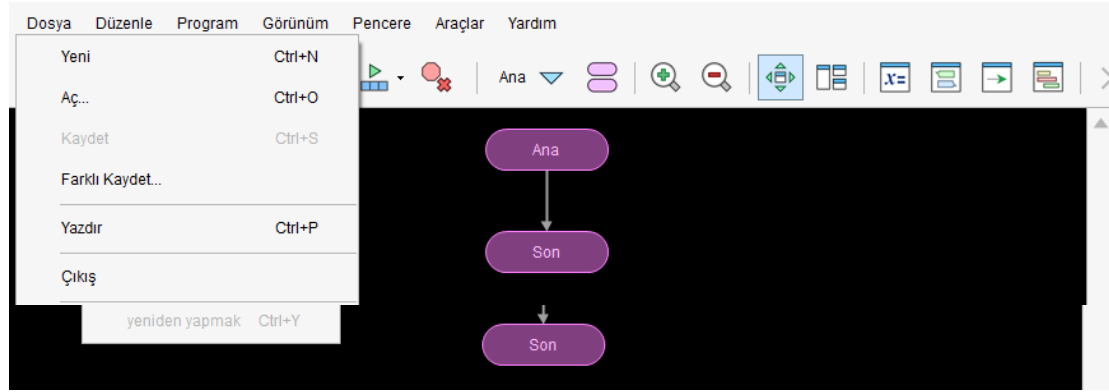
Aşağıda verilen Tablo 2-2’de ilgili sekmelerin açıklamaları verilmiştir.

Tablo 2-2 Tablo sekmelerinin açıklamaları (Atzori, 2022)

Sekme	Açıklaması
Yeni	Yeni bir akış diyagramı oluşturun.
Aç	Önceden oluşturulmuş ve kaydedilmiş bir akış diyagramı yükleyin.
Kaydet	Akış diyagramını kaydedin.
Farklı Kaydet	Dosyanın adını ve kaydedileceği klasörü değiştirmenize olanak tanır.
Yazdır	Çıktı almak için kullanılır
Çıkış	Mevcut sayfayı kapatır

2.3.6 Düzenle Menüsü

Şekil 2-6’da düzenle menüsünde yer alan sekmeler görülmektedir.



Şekil 2-6 Düzenle menüsündeki sekmelere ulaşma

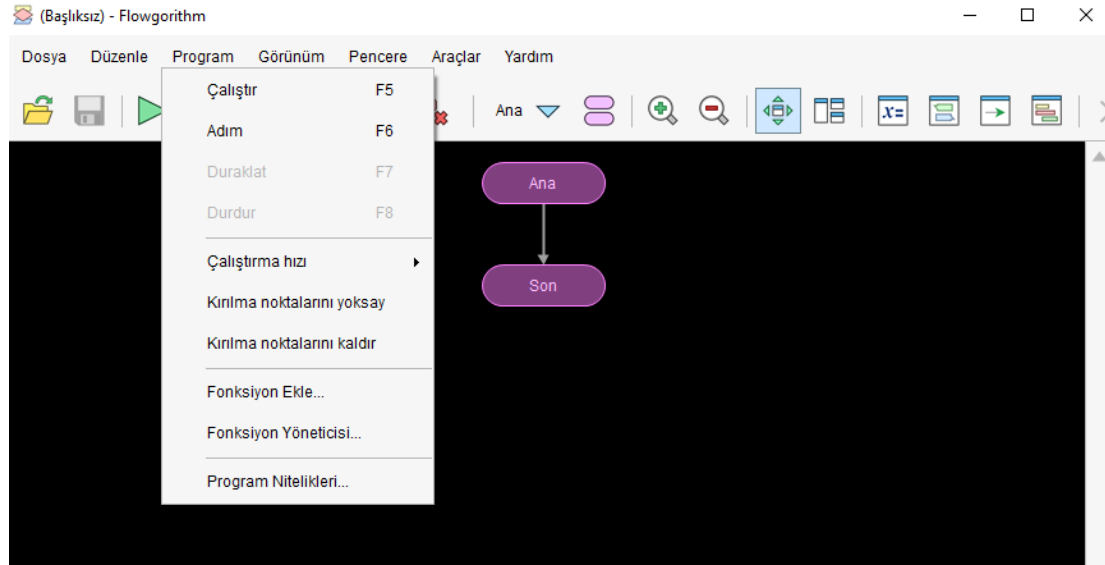
Aşağıda verilen Tablo 2-3’te düzenle menüsüne ait sekmelerin açıklamaları verilmiştir.

Tablo 2-3 Düzenle menüsü sekmelerin açıklamaları (Atzori, 2022)

Menü seçeneği	Tanım
Kes	Seçilen öğeyi belleğe kopyalayarak silin
Kopyala	Seçilen öğeyi belleğe kopyalar
Sil	Seçilen öğeyi sil
Geri Alma	Şekil ekleme, yazma işlemlerini iptal edin...
Yeniden yapma	İptal edilen işlemi yeniden yap

2.3.7 Görünüm Menüsü

Aşağıda görülen Şekil 2-7’de görünüm menüsüne ait sekmeler yer almaktadır.



Şekil 2-7 Görünüm menüsündeki sekmelere ulaşma

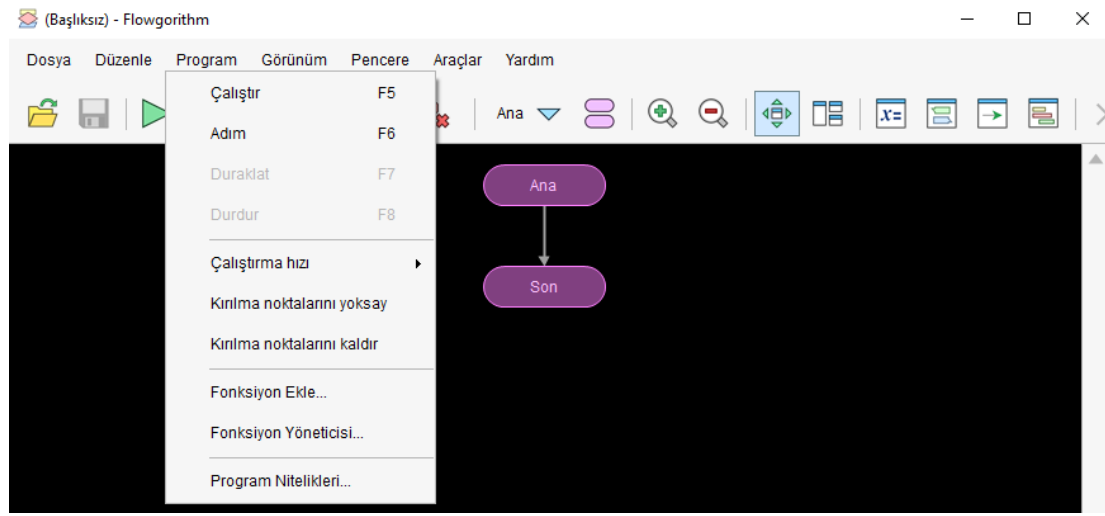
Aşağıda verilen Tablo 2-4’te görünüm menüsüne ait sekmelerin açıklamaları verilmiştir.

Tablo 2-4 Görünüm menüsü sekmelerin açıklamaları (Atzori, 2022)

Menü seçeneği	Tanım
Dili değiştir	Dili değiştirmenizi sağlar
Eylemi Sıfırla	İptal edilen işlemi yeniden yap
Büyütmeyi artır	Akış şeması alanını genişletir
Büyütmeyi azalt	Akış şeması alanını azaltır
Boyutlandırma	Flowgorithm penceresinin tamamını yeniden boyutlandır
Renk şemasını tüm pencerelere uygula	
Renk kombinasyonunu seçin	Renk düzeninizi seçin
Diyagram stili...	Diyagram stilini seçin
Renk Şemasını Düzenle...	Renklendirme temalarını düzenleyin
Diyagram Stilini Düzenle...	Diyagram stilini değiştirme

2.3.8 Program Menüsü

Aşağıda görülen Şekil 2-8’de program menüsüne ait sekmeler yer almaktadır.



Şekil 2-8 Program menüsündeki sekmelere ulaşma

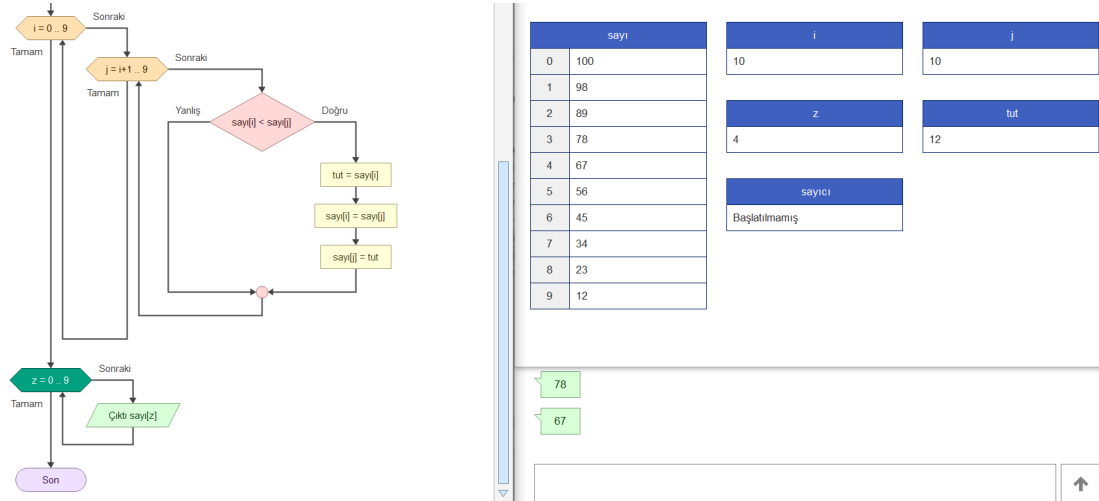
Aşağıda verilen Tablo 2-5’te program menüsüne ait sekmelerin açıklamaları verilmiştir.

Tablo 2-5 Program menüsü sekmelerin açıklamaları (Atzori, 2022)

Menü seçeneği	Tanım
Çalıştır	Akış şemasını yürütür (yorumlama)
Adım	Akış şemasını adım adım yürütün
Duraklat	Yürütmeyi duraklat
Durdur	Yürütmeyi durdurur
Çalıştırma Hızı	Otomatik modda yürütme hızının seçimi
Kırılma noktalarını kaldır	Eklenen tüm kesme noktalarını kaldırır
Fonksiyon Ekle	Yeni fonksiyon ekler
Fonksiyon Yöneticisi	Fonksiyon yönetimi yapar
Program Nitelikleri	Program özellikleri hakkında bilgi verir

Değişken değerlerini anlık izleme: Flowgorithm programında algoritma geliştirildikten sonra değişkenlerin her adım sonrasındaki değerlerini görebilmek Adım sekmesi ile mümkün olmaktadır. **Pencere düzeninden Değişken&Konsol pencere yapısı** seçilerek akış diyagramı adım adım ilerlerken değişkenlerin değerleri anlık olarak takip edilebilmektedir. Bu sayede hata ayıklama ve analiz

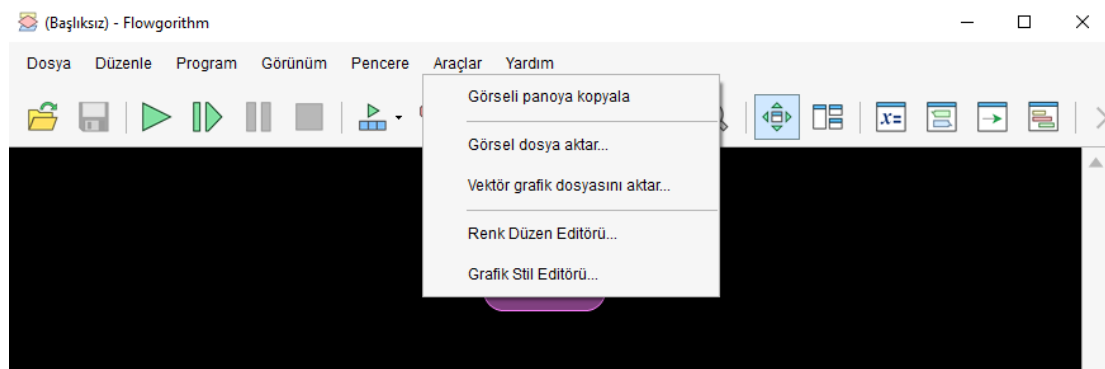
işlemi oldukça kolaylaşmaktadır. Şekil 2-9’da değişken değerlerinin izlenmesine yönelik örnek bir görsel yer almaktadır.



Şekil 2-9 Değişken değerlerini anlık izleme

2.3.9 Araçlar Menüsü

Şekil 2-10’da araçlar menüsüne yönelik sekmeler yer almaktadır.



Şekil 2-10 Araçlar menüsündeki sekmelere ulaşma

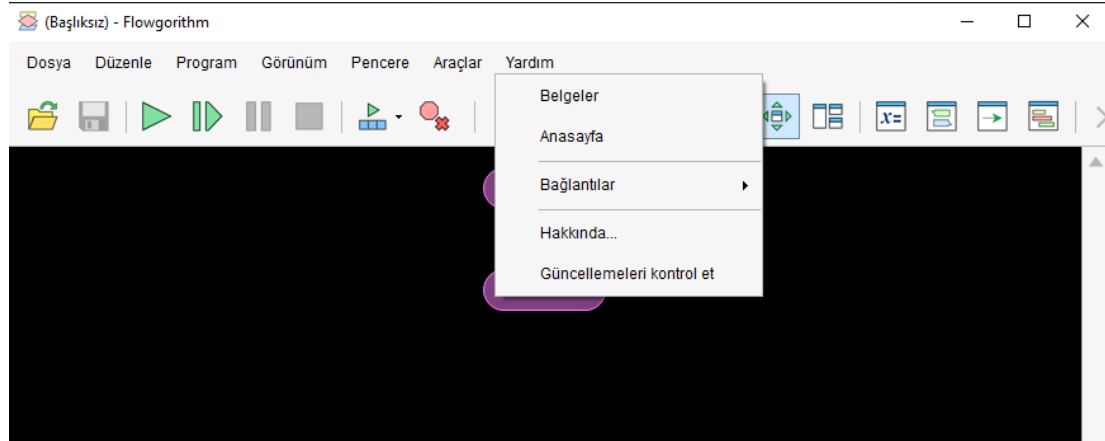
Aşağıda verilen Tablo 2-6’da araçlar menüsüne ait sekmelerin açıklamaları verilmiştir.

Tablo 2-6 Araçlar menüsü sekmelerin açıklamaları (Atzori, 2022)

Menü seçeneği	Tanım
Otomatik düzenleme	Flowgorithm otomatik pencere düzenlemesi
Pencere düzeni...	Masaüstündeki pencerelerin düzenlenmesi
'Değişkenler' penceresi...	Programda kullanılan değişkenleri kontrol etmek için pencereyi etkinleştirir
'Yürütme' penceresi...	Programın yürütülmesini izlemek ve klavyeden değer girmek için pencereyi etkinleştirir
Kaplumbağa grafik penceresi	Kaplumbağa Programlamayı Kullanma (LOGO Stili)
'Kodlama' penceresi...	Flowgorithm tarafından sağlanan dillerde otomatik kodlamayı görüntülemek için pencereyi etkinleştirir
Görüntüyü belleğe kopyala	Görüntüyü belleğe kopyalayın
Resmi dosyaya aktar...	Aktif alan görüntüsünü bir PNG dosyasına aktarın
Görüntüyü vektör formatında dışa aktar	Aktif alan görüntüsünü .SVG (Ölçeklenebilir Vektör Grafikleri) veya .EMF (Gelişmiş Meta Dosyası Formatı)

2.3.10 Yardım Menüsü

Şekil 2-11'da yardım menüsüne yönelik sekmeler yer almaktadır.



Şekil 2-11 Yardım menüsündeki sekmelere ulaşma

Aşağıda verilen Tablo 2-7'de yardım menüsüne ait sekmelerin açıklamaları verilmiştir.

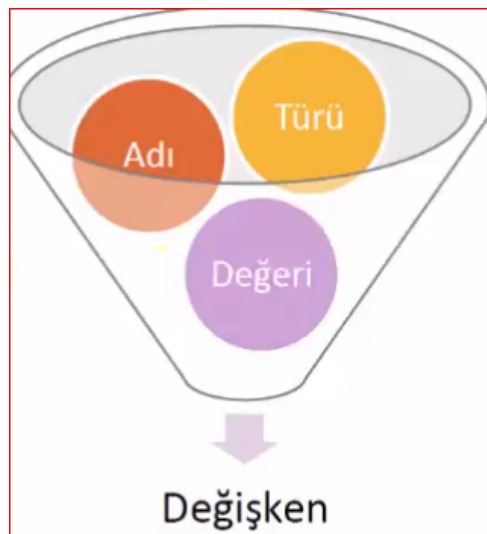
Tablo 2-7 Yardım menüsü sekmelerin açıklamaları (Atzori, 2022)

Menü seçeneği	Tanım
Dokümantasyon	Çevrimiçi dokümantasyon sayfasını açar http://www.flowgorithm.org/documentation/
Flowgorithm – Web Sitesi	Flowgorithm sitesinin çevrimiçi ana sayfasını açar http://www.flowgorithm.org
Bağlantılar	Flowgorithm yazarının ait olduğu Üniversiteye bağlantıların bulunduğu bir alt menüyü açar
Bilgi	Yazılım hakkında bilgi penceresi açılır, çeviriler ve bu yazılımın geliştirilmesinde emeği geçenlere teşekkürler
Güncellemeleri kontrol et	Yazılımın yeni sürümlerinin mevcut olup olmadığını kontrol etmenizi sağlar

2.4 Programlamada Değişken Kavramı

Programlamada değişken kavramı, bir değeri saklamak ve bu değere ihtiyaç duyulduğunda erişmek için kullanılan temel bir yapıdır. Değişkenler, programın çalışması boyunca değerleri değiştirilebilen veya güncellenebilen öğelerdir. Her değişken, programın belleğinde adlandırılmış belirli bir konumda saklanır ve program tarafından verilen bir adla tanımlanır.

Bir değişkenin kullanılabilmesi için önce tanımlanması gerekir. Tanımlama işlemi, genellikle değişkenin adının ve türünün belirlenmesiyle yapılır. Değişkenin türü, hangi türden verileri saklayabileceğini ifade eder. Örneğin, bir değişken tamsayı değerlerini saklayacaksa **tamsayı (integer)** türünde, ondalık değerler saklayacaksa **ondalık (float)** türünde tanımlanır. Aşağıda verilen Şekil 2-12, bir değişkenin yapısını göstermektedir.



Şekil 2-12 Değişkenin Yapısı

2.4.1 Değişken Türleri

Flowgorithm programında 4 farklı değişken türü kullanılabilmektedir. Bu değişkenler ve özellikleri şu şekilde ifade edilmektedir.

Tamsayı (Integer)

- **Açıklama:** Tamsayılar, ondalık kısmı olmayan tam sayı değerlerini temsil eden değişken türüdür. Negatif, pozitif veya sıfır olabilirler.
- **Örnek:** 1, -5, 100, 0.

Ondalık Sayı (Real)

- **Açıklama:** Ondalık sayılar, ondalık kısmı bulunan sayıları temsil eden değişken türüdür. Genellikle reel sayıların saklanması için kullanılır.
- **Örnek:** 3.14, -0.5, 2.71828.

Metin Dizisi (String)

- **Açıklama:** Metin dizileri, karakter veya metinlerden oluşan değerleri saklamak için kullanılan değişken türüdür. Bu değerler tek tırnak (' ') veya çift tırnak (" ") içinde yazılır.
- **Örnek:** "Hello, World!", "Merhaba dünya!", "123".

Mantıksal (Boolean)

- **Açıklama:** Mantıksal değişkenler, yalnızca iki farklı değer alabilen değişken türüdür: **True** veya **False**. Genellikle koşullu ifadelerde kullanılır.
- **Örnek:** True, False.

2.4.2 Değişkenlere İsim Verme Kuralları

Bir işlev veya değişken tanımladığınızda, ona tanımlayıcı adı verilen benzersiz bir ad verilir. Tanımlayıcıların ifadelerdeki diğer öğelerle karıştırılmaması için belirli adlandırma kurallarına uyması gerekir. Her programlama dilinin kendine özgü kuralları olmakla birlikte, bu kurallar genellikle diller arasında büyük ölçüde benzerdir.

Flowgorithm'de tanımlayıcıların aşağıdaki kurallara uygun olması gerekir:

- Bir harfle başlamalıdır.
- İlk harften sonra ek harfler veya rakamlar içerebilir.
- Boşluk içeremez.

- Anahtar kelimeler veya Flowgorithm tarafından önceden tanımlanmış kelimeler kullanılamaz.

Visual Basic ve C gibi bazı dillerde tanımlayıcıların içinde “_” (alt çizgi) karakteri kullanılabilir; ancak Flowgorithm bu karaktere izin vermez. Ayrıca, Flowgorithm’de tanımlayıcılar büyük/küçük harf duyarlı değildir. Aşağıdaki Tablo 2-8, bir değişkenin yapısını göstermektedir.

Tablo 2-8 Değişkenin yapısını veren tablo (Atzori, 2022)

Geçerli Tanımlayıcılar	Notlar
x	Tanımlayıcılar tek harf olabilir.
isim2	İlk harften sonra sayılara izin verilir

Programlama dillerinde okunabilirliği artırmak amacıyla genellikle **class**, **public** veya **if** gibi özel sözcükler kullanılır. Bu sözcükler çoğu zaman tanımlayıcıların adlandırma kurallarıyla benzerlik gösterdiğinden, çakışmaları önlemek için ayrılmış kelime (reserved word) olarak kabul edilir ve değişken adı olarak kullanılamaz. Ayrıca birçok programlama dili, bazı fonksiyonları ve sabitleri önceden tanımlar; bu tanımlı ifadeler de değişken ismi olarak kullanılamaz.

Aşağıda verilen Tablo 2-9’da yer alan kelimeler, değişken adı olarak kullanılamazlar.

Tablo 2-9 Değişken olarak kullanılamayan kelimeler (Atzori, 2022)

and	false	mod	not	or	pi	true
-----	-------	-----	-----	----	----	------

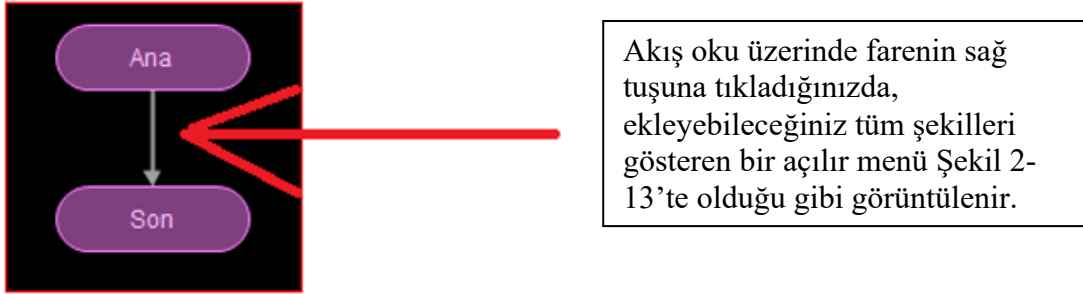
boolean	integer	real	string
---------	---------	------	--------

abs	cos	random	tan	tostring
arccos	int	sgn	tochar	toreal
arcsin	len	sin	to code	
arctan	log	size	to fixed	
char	log10	sqrt	to integer	

arccosh	cosh
arcsinh	sinh
arctanh	tanh

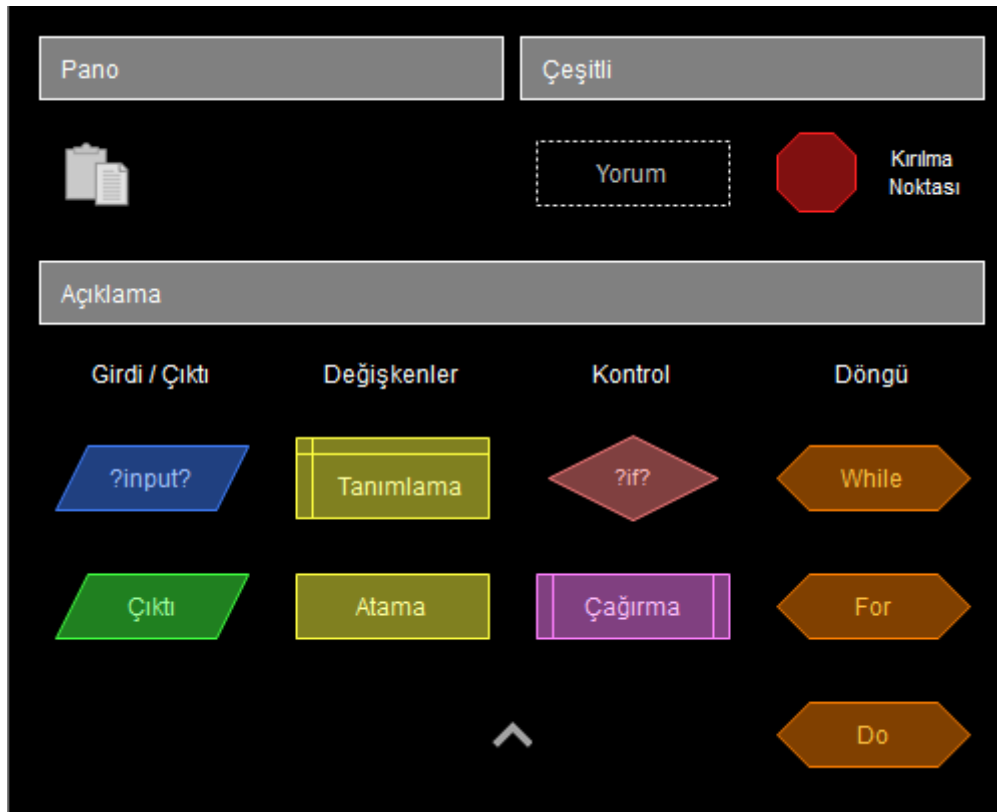
2.4.3 Değişken Atama ve Değer Verme

Değişkenleri atamak ve değer verebilmek için öncelikle değişkeni tipi ile tanımlamak ve daha sonra klavyeden giriş yapmak veya atama ile bir değere eşitlemek gerekmektedir.

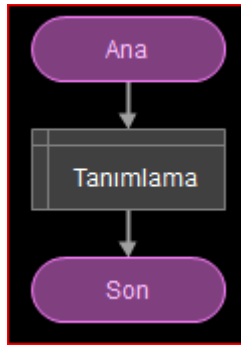


Şekil 2-13 Şekiller menüsü için tıklanması gereken ok

Aşağıda verilen Şekil 2-14, akış diyagramına eklenebilecek sembolleri göstermektedir.



Şekil 2-14 Akış diyagramına eklenebilecek semboller

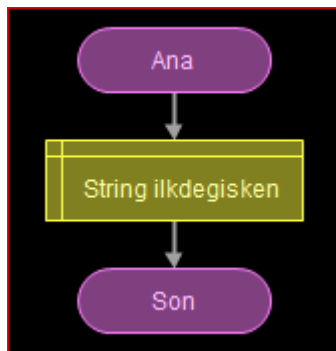


Değişkenler sütununun altında yer alan Tanımlama sekmesine tıklayarak, değişkenimizin türünü belirleyeceğimiz tanımlama adımını akış diyagramına ekliyoruz (Şekil 2-15). Ardından, eklediğimiz Tanımlama sekmesine tekrar tıklayarak aşağıdaki pencerenin açılmasını sağlıyoruz (Şekil 2-16).

Şekil 2-15 Değişken atama

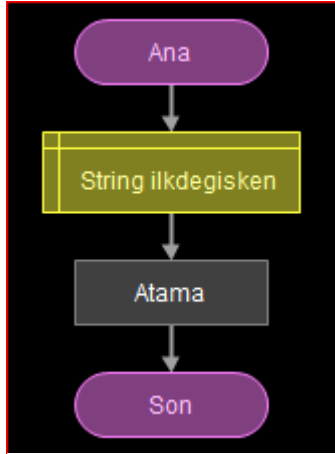
Şekil 2-16 Değişkenin isim ve türünü belirleme ekranı

Bu pencere, değişkenin adını ve türünü belirleyeceğimiz ekrandır. Örneğin, değişken türü olarak **String** seçebilir ve değişken adına **ilkdegisken** yazabilirsiniz.



Akış diyagramımız, Şekil 2-17’da gösterildiği gibi oluşturulmuştur. Artık değişkenimizin adı ve türü belirlenmiştir. Son olarak, değişkenin değerini atamak için yeniden akış okuna tıklıyor ve açılan pencereden **Atama** sekmesini seçiyoruz.

Şekil 2-17 Değişken eklendikten sonra akış şemasındaki görünüm



Atama sekmesine tıklayarak Şekil 2-19’de gösterilen pencerenin açılmasını sağlarız.

Şekil 2-18 Akış şemasına Atama şekli eklendikten sonra oluşan görünüm

Atama Özellikleri

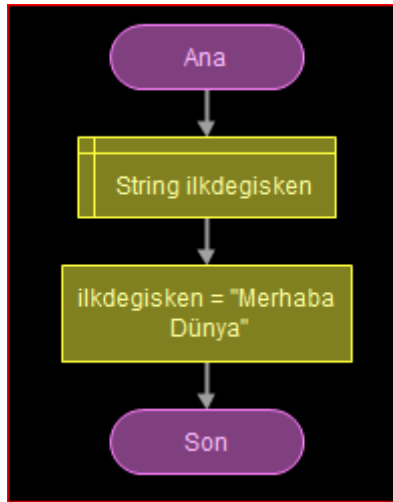
Atama Atama Komutu bir ifade hesaplar ve sonucu bir değişkende saklar.

Değişken: = Açıklama:

OK Vazgeç

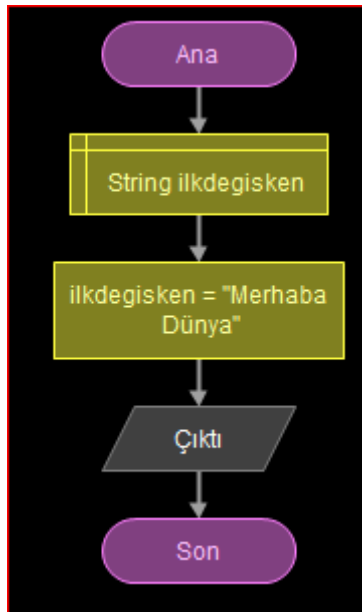
Şekil 2-19 Atama ekranında değişkenin değerini değiştirme görüntüsü

Değişken kısmına daha önce belirlediğimiz değişken adını, **Açıklama** kısmına ise değişkene vereceğimiz değeri yazıyoruz. Değişkenimiz **String** türünde olduğu için değeri yazarken metni mutlaka " " (**çift tırnak**) işaretleri arasına almaya dikkat ediyoruz.



Artık değişken için gerekli olan üç adımı tamamlamış olduk (Şekil 2-20). Son olarak, değişkenimizin değerini konsol ekranında görüntülemek istiyoruz. Bunun için akış diyagramındaki ilgili oka tekrar tıklayarak özellik penceresini açıyoruz ve **Girdi/Çıktı** başlığı altındaki **Çıktı** sekmesini seçiyoruz (Şekil 2-21).

Şekil 2-20 Değişkenin değeri değiştikten sonra oluşan görüntü

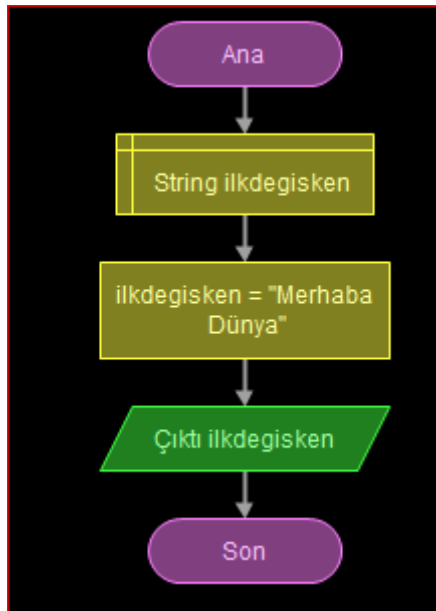


Çıktı sekmesine çift tıklayarak Şekil 2-22'de görülen pencerenin açılmasını sağlıyoruz.

Şekil 2-21 Akış şemasına çıktı şekli eklenince oluşan görüntü

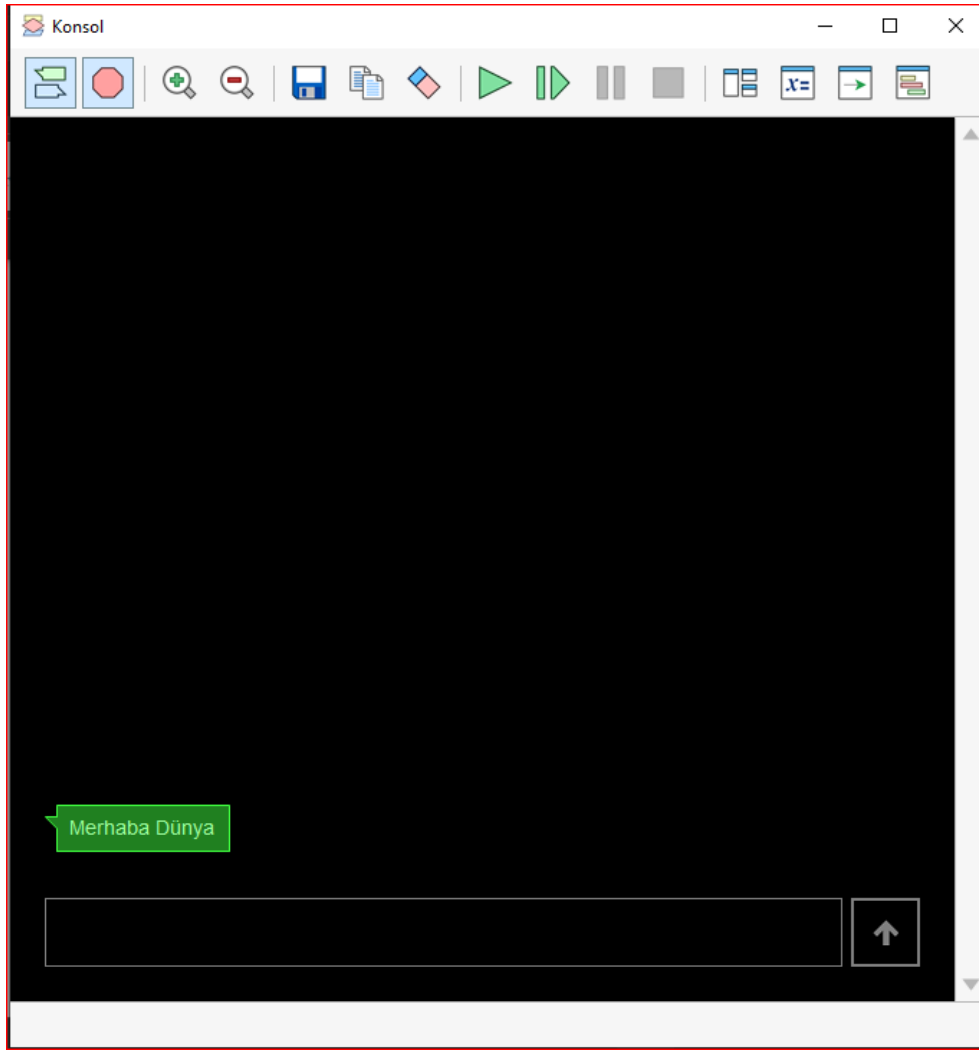
Şekil 2-22 Çıktı ekranına yazdırmak istenilen bilginin girildiği kısım

Çıktı Özellikleri penceresine değişkenimizin adını yazarak OK sekmesine tıklıyoruz.



Artık ilk programımız için her şey Şekil 2-23'te görüldüğü gibi hazır. Çalıştır sekmesine tıklayarak değişkenimizin değerini Şekil 2-24'te görüldüğü gibi çıktı olarak görüntüleyebiliriz.

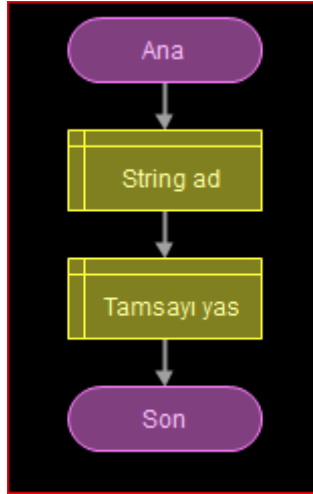
Şekil 2-23 Bilgi girildikten sonra akış şemasının görünümü



Şekil 2-24 Program çalıştırıldığında oluşan çıktı

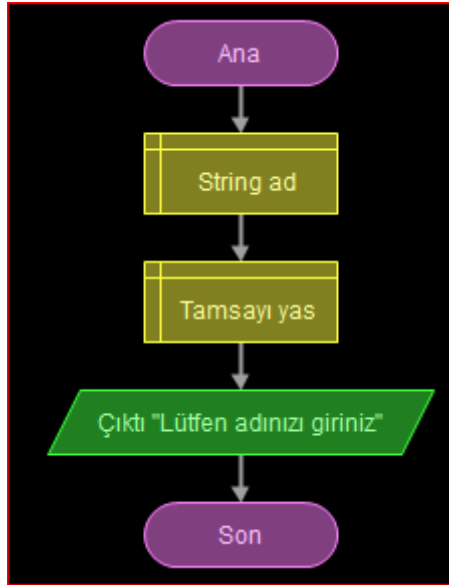
2.5 Değişken Değer Bilgisini Kullanıcıdan Alma(Input)

Programlamada **input**, kullanıcıdan veri almak için kullanılan bir fonksiyondur. Kullanıcının bir metin, sayı ya da başka bir veri türünü girmesi gerektiğinde input fonksiyonundan yararlanılır. Örneğin, bir kullanıcıdan adını veya yaşını girmesini istemek için input fonksiyonu kullanılabilir.



Kullanıcının isim ve yaş bilgilerini saklamak için **String (metin)** ve **Integer (tamsayı)** türlerinde iki farklı değişken tanımlayalım (Şekil 2-25).

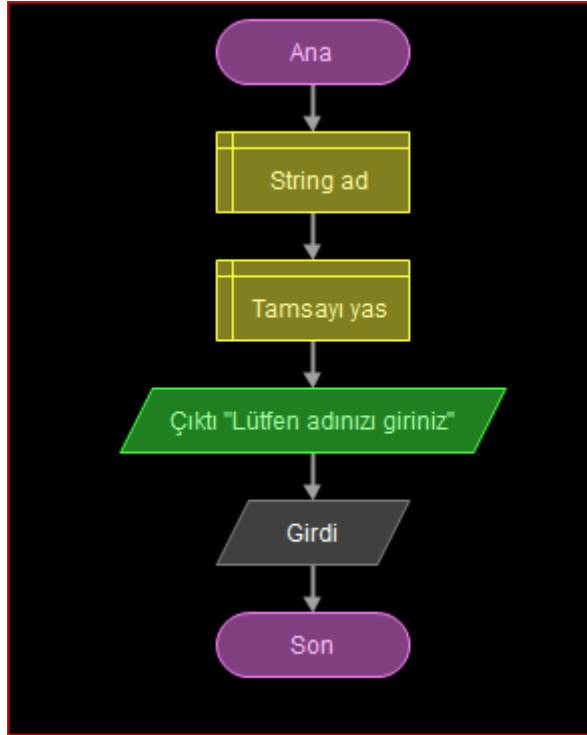
Şekil 2-25 Değişkenlerin tanımlama görüntüsü



“Kullanıcıdan isim bilgisini almak için akış şemamıza bir *Çıktı (Output)* bloğu ekliyoruz. Bu blokta kullanıcıya Şekil 2-26’te görüldüğü gibi ‘Lütfen adınızı giriniz:’ şeklinde bir mesaj göstererek yönlendirme yapıyoruz.”

Şekil 2-26 Çıktı şeklinin eklenmesi

Kullanıcıdan isim bilgisini almak için, akış şemasında *Girdi/Çıktı* kategorisi altında bulunan *Input (Girdi)* bloğunu şemamıza ekliyoruz (Şekil 2-27). Bu blok aracılığıyla kullanıcıdan isim bilgisinin girilmesini sağlıyoruz.

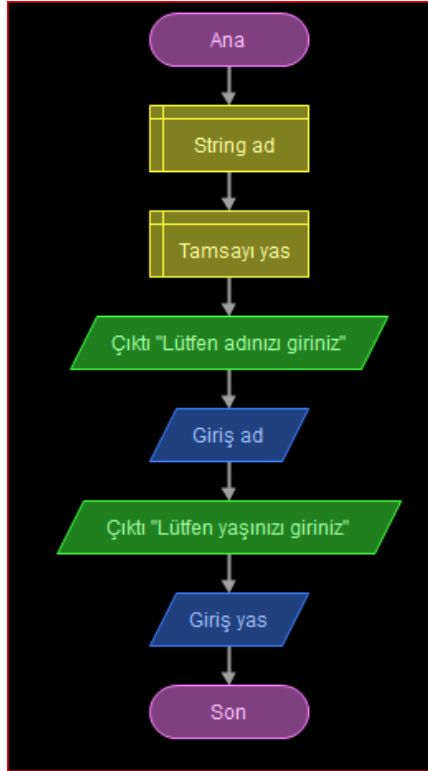


Şekil 2-27 Girdi şeklinin eklenmesi

Girdi bloğuna çift tıklayarak, kullanıcıdan alınacak veriyi tanımlamamızı sağlayan aşağıdaki pencerenin (Şekil 2-28) açılmasını sağlayalım.

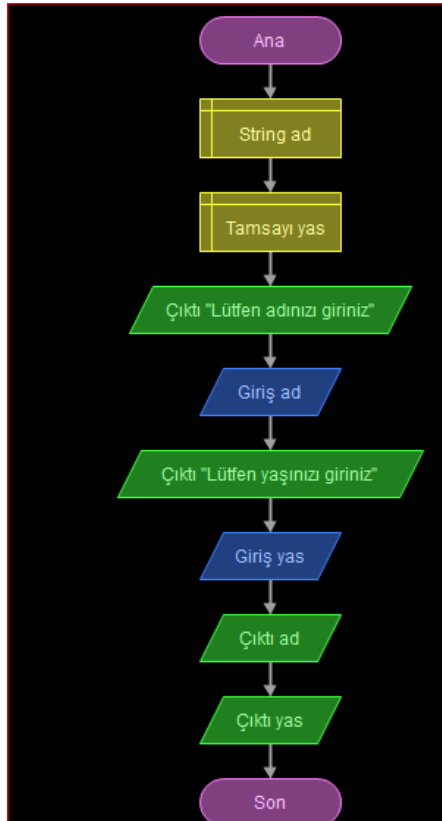
Şekil 2-28 Girdi özelliklerinin belirlenmesi

Bu pencerede, kullanıcıdan değer girmesini istediğimiz değişkenin adını yazıyoruz. Aynı işlemleri *yas* değişkeni için de yaptıktan sonra akış şemamız Şekil 2-29'deki gibi oluşmaktadır.



Yandaki program çalıştırıldığında *ad* ve *yas* değişkenlerimize kullanıcı tarafından girilen değerler atanmış olur. Son adım olarak, bu bilgileri ekranda gösterebilmek için akış şemamızın en altına bir *Çıktı* bloğu ekleyelim (Şekil 2-30).

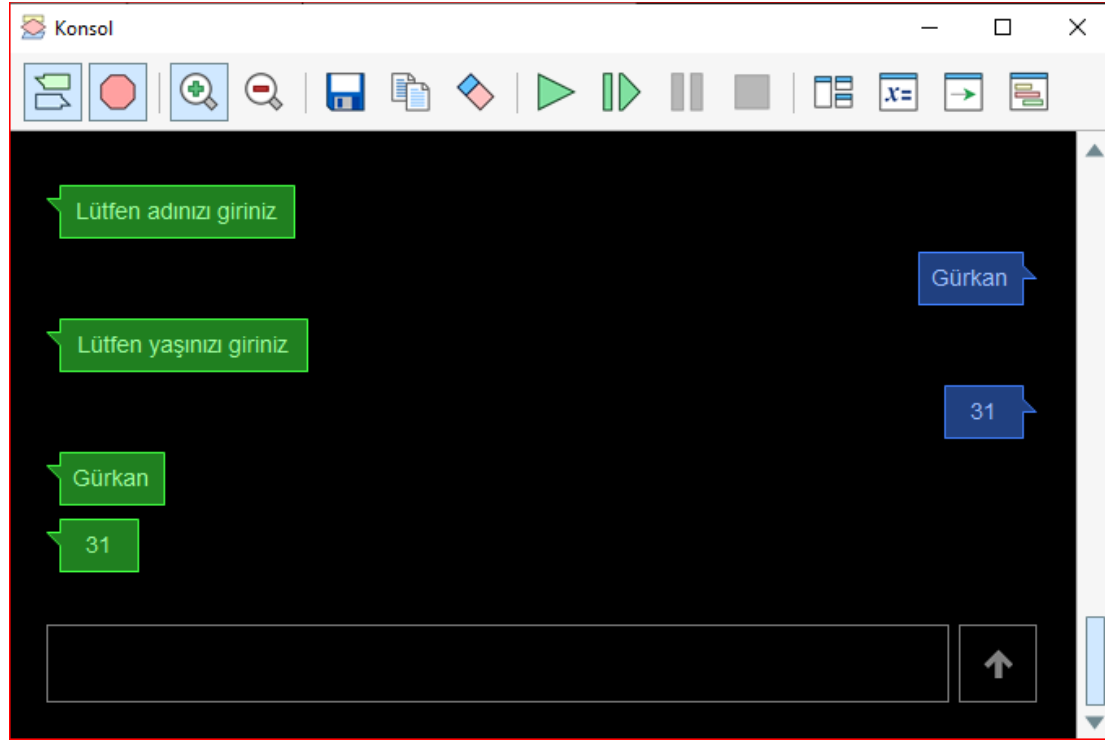
Şekil 2-29 Girdi girişinden sonra akış şemasındaki görüntü



Yandaki program çalıştırıldığında kullanıcıdan ad ve yaş bilgisi istenecek, ardından bu bilgiler en son ekrana çıktı olarak yazdırılacaktır (Şekil 2-31).

Şekil 2-30 Çıktı şekillerinin eklenmesi

Tüm adımlar tamamlandığında programın konsol çıktısı aşağıdaki Şekil 2-31'deki gibidir.



Şekil 2-31 Program çalıştırıldığında oluşan görüntü

2.6 Operatörler

Mantıksal operatörler ve karşılaştırma operatörleri, kullanılan programlama diline göre farklılık gösterir. Flowgorithm, bu çeşitliliği dikkate alarak hem matematikte kullanılan sembolleri (Unicode karakterleriyle) hem de iki temel programlama dili ailesini destekler.

- **BASIC ailesi**, İngilizce anahtar kelimeler ve sözcüksel operatörler içerir.
- **C ailesi** (C, Java, C# gibi dilleri kapsar) daha sembolik operatörler kullanır.

Flowgorithm hem matematiksel gösterimleri hem de bu iki dil ailesine ait operatörleri desteklediği için bazı işlemlerin birden fazla yazım biçimi vardır. Program, bu alternatif gösterimlerin hepsini eşdeğer kabul eder. Böylece öğrenci hangi dili öğrenmeyi hedefliyorsa o dilin operatör yapısını kullanma özgürlüğüne sahip olur.

Tablo 2-10'da Flowgorithm'de kullanılabilen operatörler gösterilmiştir.

Tablo 2-10 Kullanılabilecek operatörler (Atzori, 2022)

Operator	C Family	BASIC Family	Mathematics (Unicode)
Eşitlik	==	=	=
Eşitsizlik	!=	<>	≠
Az veya eşit	<=	<=	≤
Büyüktür veya Eşittir	>=	>=	≥
Mantıksal Değil	!	not	¬
Mantıksal ve	&&	and	∧
Mantıksal Veya		or	∨
Çarpmak	*	*	×
Bölmek	/	/	÷
Kalan	%	mod	

Tablo 2-11'de Flowgorithm'de % (mod) ve ^ (üs alma/power) vb. matematiksel operatörlerin nasıl kullanıldığını, işlem önceliğini ve işlem sonuçlarını gösteren örnekler yer almaktadır.

Tablo 2-11 Matematik işlem örnekleri (Atzori, 2022)

İfade	Sonuç	Notlar
$1 + 3 \wedge 2$	10	
$10 * 2 + 5 * 6$	50	$10 * 2$ ve $5 * 6$ toplamaya göre daha yüksek önceliğe sahiptir. Ekleme en son yapılır.
$7 * (4 - 1)$	21	Bir bütün olarak değerlendirilen alt ifadeler için parantez kullanılır.
$6 / 3 * 2$	4	Matematikte çarpma ve bölme aynı öncelik düzeyine sahiptir. Yani soldan sağa değerlendirilirler.
$10 \bmod 3$	1	Modulo matematiği bölmeden kalanı verir

2.7 İçsel Fonksiyonlar

Tablo 2-12'de Flowgorithm'de kullanılan matematiksel fonksiyonlar ve fonksiyonların anlamı yer almaktadır. Fonksiyonlarda n değişkeninin içine yazılan değer üzerinden işlemler yapılmaktadır.

Tablo 2-12 Matematik formülleri (Atzori, 2022)

Abs(n)	Mutlak değer
Arcsin(n)	Trigonometrik Arksinüs
Arccos(n)	Trigonometrik Arcco'lar
Arctan(n)	Trigonometrik Arktanjan
Cos(n)	Trigonometrik Kosinüs
Int(n)	Gerçek bir sayının integrali (tam değeri)
Ln(n)	Doğal Log
Log(n)	Doğal Log (Ln ile aynı)
Log10(n)	Günlük Tabanı 10
Sgn(n)	Matematiksel işaret (n negatifse -1 , sıfırsa 0, pozitifse 1)
Sin(n)	Trigonometrik Sinüs
Sqrt(n)	Kare kök
Tan(n)	Trigonometrik Tanjan

Aşağıdaki Tablo 2-13, string türündeki değişkenlerde kullanılabilen fonksiyonları ve fonksiyonların anlamını göstermektedir.

Tablo 2-13 String tipi değişken fonksiyonları (Atzori, 2022)

Len(s)	Bir dizinin uzunluğu
Char(s, i)	i dizindeki s dizesinden bir karakter döndürür . Karakterler 0'dan başlayarak indekslenir.

Tablo 2-14'te dosya okumak, rastgele sayı üretmek, dizinin boyutu öğrenmek ve değişkenler arasındaki dönüşümleri yapabilmek için gerekli olan fonksiyonlar ve fonksiyonların anlamı verilmiştir.

Tablo 2-14 Dönüşüm fonksiyonları (Atzori, 2022)

Fonksiyon	Fonksiyonun Anlamı
ToChar(<i>n</i>)	Bir karakter kodunu <i>n</i> bir karaktere dönüştürün.
ToCode(<i>c</i>)	<i>c</i> karakterini karakter koduna (tamsayı) dönüştürün.
ToFixed(<i>r</i> , <i>i</i>)	Gerçek sayı <i>r</i> 'yi, ondalık noktadan sonra <i>i</i> rakamı olan bir metne dönüştürün. Bu işlev para birimi için kullanışlıdır.
ToInteger(<i>n</i>)	Bir metni tam sayıya dönüştürme
ToReal(<i>n</i>)	Bir metni gerçeğe dönüştürün
ToString(<i>n</i>)	Sayıyı metne dönüştürme
EOF()	Dosyanın sonuna ulaşıldıysa true değerini döndürür. Bu, okumak için açılan dosyalarda kullanılır.
Random(<i>n</i>)	0 ile (<i>n</i> - 1) arasında rastgele bir sayı
Size(<i>a</i>)	Bir dizideki boyut (öge sayısı)

2.8 Yerleşik Sabitler

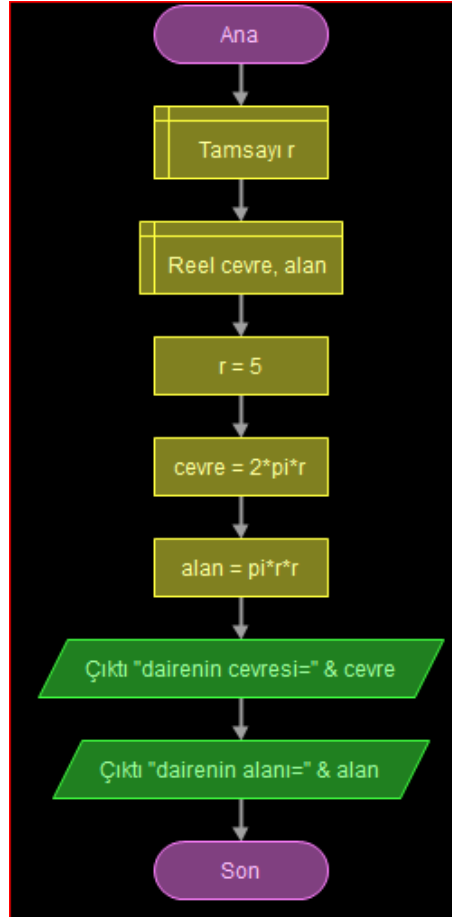
Flowgorithm, yaygın olarak kullanılan üç sabiti tanımlar. True ve False sabitleri genellikle Boolean türündeki değişkenleri başlatmak için kullanılır. Pi sabiti ise matematiksel işlemlerde sıkça başvurulanan bir sabittir. Aşağıdaki Tablo 2-15, bu sabitlere ilişkin bilgileri içermektedir.

Tablo 2-15 Sabitler (Atzori, 2022)

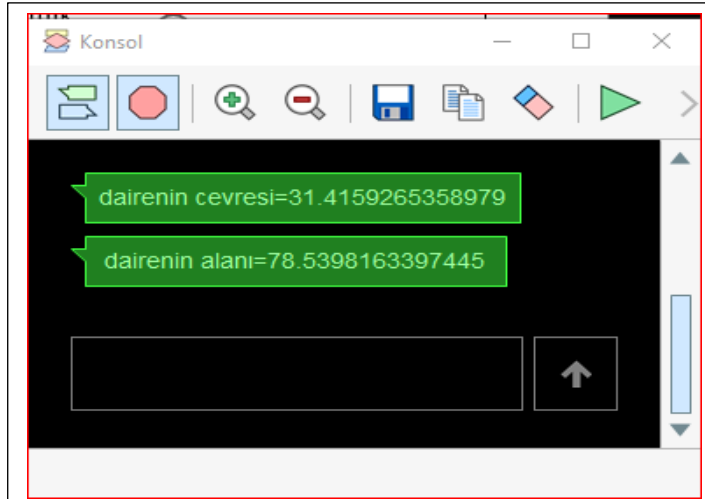
true	Boolean Doğru
false	Boolean Yanlış
pi	Matematiksel Pi. Yaklaşık 3.1415.
π	Matematiksel Pi. Yukarıdakinin aynısı.

2.9 Değişken Uygulaması Örneği (Dairenin Çevre ve Alanını Hesaplama)

Flowgorithm'de değişken tanımlamak ve değişkenlere değer vermeyi daha anlayabilmek için aşağıdaki örnek uygulama incelenebilir.



Yandaki akış şemasında, integer türünde bir *r* değişkeni ile reel sayı türünde *cevre* ve *alan* değişkenleri tanımlanmıştır. *r* değişkenine 5 değeri atanmış, *cevre* ve *alan* değişkenlerinin değerleri ise ilgili formüllere uygun şekilde hesaplanmıştır (Şekil 2-32). Son olarak, string birleştirme operatörü (&) kullanılarak çevre ve alan değerleri konsol ekranına yazdırılmıştır (Şekil 2-33).



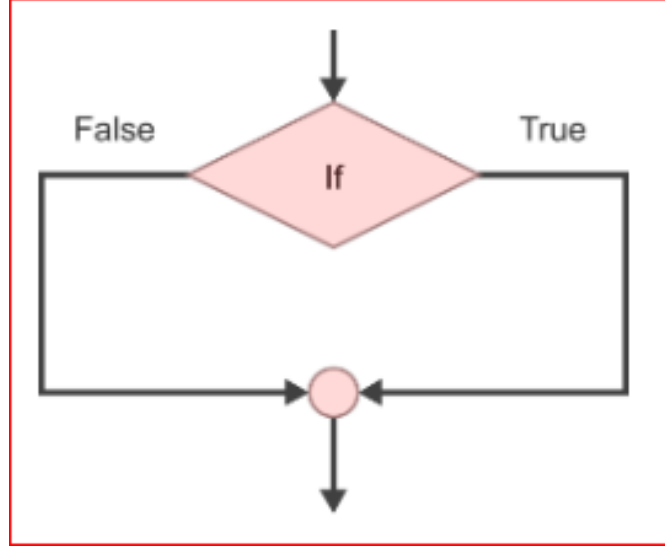
Şekil 2-33 Program çalıştığında oluşan çıktı

Şekil 2-32 Dairenin çevre ve alanını hesaplayan algoritma

3 Bölüm

3.1 Kontrol Yapıları (If)

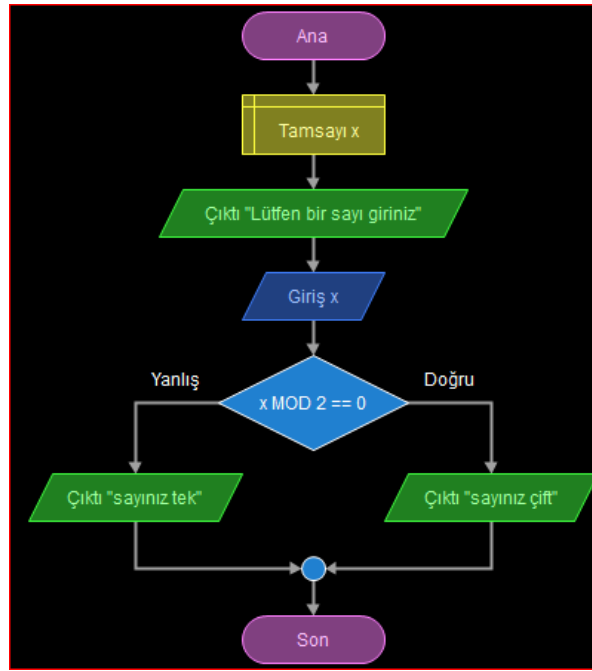
Kontrol ve karar yapıları programlama ve dolayısı ile algoritmalar için olmazsa olmazdır. Döngü yapılarının içinde de mutlaka kontrol ifadeleri bulunmaktadır. Kontrol yapıları sayesinde istenen koşullara uygun bloklar çalıştırılabilmektedir. Şekil 3-1, Flowgorithm’de kullanılan if yapısını göstermektedir.



Şekil 3-1 Kontrol yapısı (Atzori, 2022)

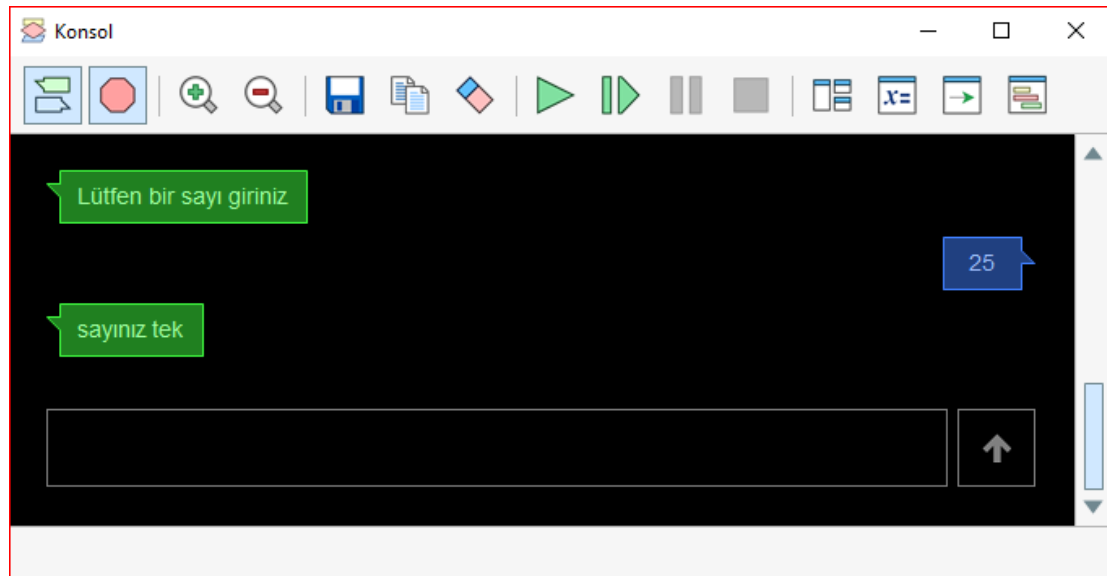
Programlama terimlerinde *if* yapısı, bir programın belirli bir koşula bağlı olarak farklı davranışlar sergilemesini sağlayan temel kontrol yapılarından biridir. *If* ifadesi, verilen koşulu değerlendirir; koşul doğru olduğunda ilgili kod bloğunu çalıştırır. Koşul yanlış ise bu kod bloğu atlanır veya alternatif bir işlem gerçekleştirilir.

Şekil 3-2’ de verilen akış şemasında x adında bir değişken tanımlanmıştır. Bu değişkenin değeri kullanıcıdan alınmış ve şemaya bir if bloğu eklenmiştir. Orta kısımdaki koşul ifadesine $x \text{ MOD } 2 == 0$ yazılmıştır. Bu ifade, kullanıcının girdiği sayının 2’ye bölümünden kalanını kontrol eder.



Şekil 3-2 Tek-çift sayıların ayırımı yapan algoritma

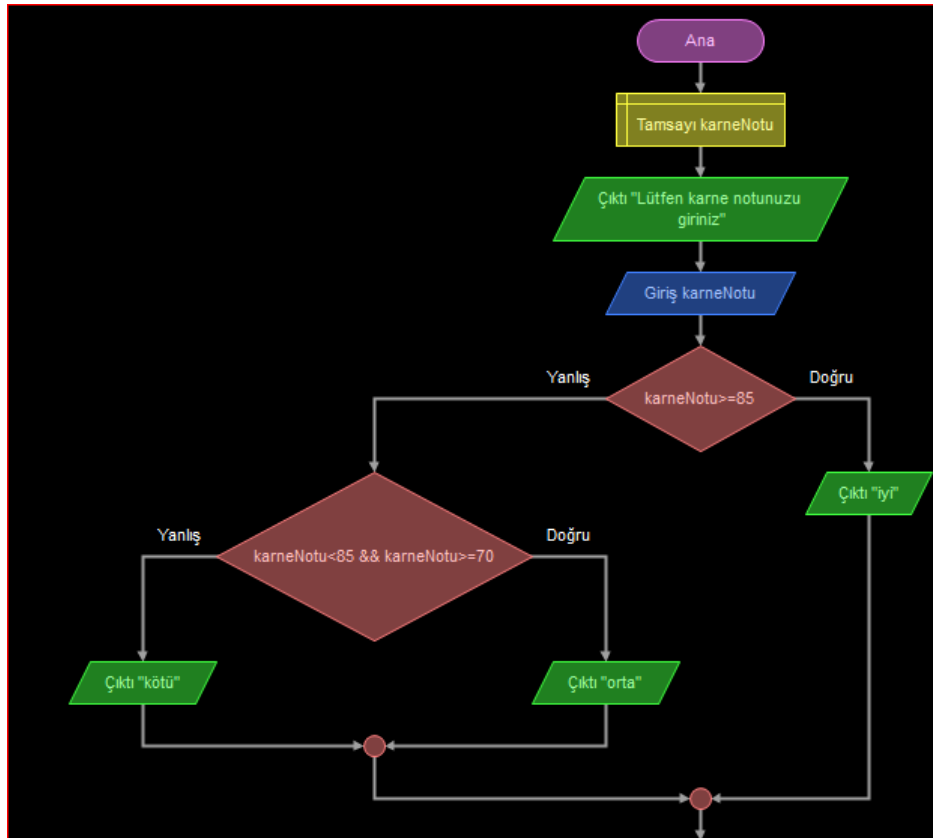
Eğer kalan 0 ise akış şeması sağ taraftan devam ederek ‘Sayınız çift’ çıktısını üretir; kalan 0 değilse akış sol taraftan ilerleyerek ‘Sayınız tek’ ifadesi konsol ekranına yazdırılır (Şekil 3-3).



Şekil 3-3 Program çalıştırıldığında oluşan çıktı

3.1.1 If Uygulaması Örneği (Öğrencinin karne notuna göre başarı durumu)

Verilen akış şemasında (Şekil 3-4) kullanıcıdan karne notu bilgisi alınmıştır. Eğer not 85'ten büyük veya 85'e eşit ise program sağ taraftan ilerleyerek çıktı olarak 'İyi' ifadesini üretmiştir. Not 85'ten küçükse akış sol tarafa yönelmiş ve ikinci bir kontrol adımına geçilmiştir. Bu aşamada notun belirli bir aralıkta olup olmadığı değerlendirilmiştir. Not bu aralıkta ise akış sağ taraftan devam ederek 'Orta' çıktısı verilmiştir. Not belirtilen aralığa uymuyorsa akış sol taraftan ilerlemiş ve çıktı olarak 'Kötü' ifadesi yazdırılmıştır.



Şekil 3-4 Başarı durumunu belirleyen algoritma

Flowgorithm dosyasına ulaşmak için karekodu okutunuz.



3.1.2 Vücut Kitle Indexi Hesaplayan Program

Kullanıcıdan kilo (kilogram) ve boy (metre) bilgileri alınarak $VKİ = \text{kilo} / (\text{boy} \times \text{boy})$ formülüne göre bir akış diyagramı oluşturabilirsiniz. Karar koşulları şu şekilde değerlendirilecektir: $VKİ > 40$ ise '3. derece obezite', $30 < VKİ \leq 40$ ise '2. derece obezite', $20 < VKİ \leq 30$ ise '1. derece obezite', $10 < VKİ \leq 20$ ise 'ortalama kilo', $VKİ \leq 10$ ise 'sağlıklı kilo'. Aşağıda verilen bağlantıda programın PDF çıktısı bulunmaktadır:

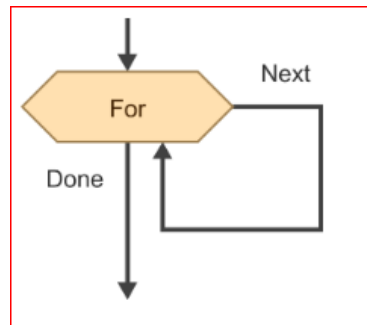
<https://drive.google.com/file/d/1o0w5bHze-S-G4qETF9Ln0oUJ40SWuvQN/view>

Flowgorithm dosyasına ulaşmak için kodu okutunuz:



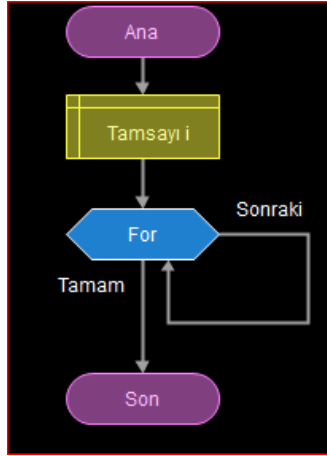
3.2 For Döngüsü

Şekil 3-5, Flowgorithm'de kullanılan *for* döngüsünün yapısını göstermektedir.



Şekil 3-5 For döngü şeması (Atzori, 2022)

For döngüsü, programlamada tekrar eden işlemleri gerçekleştirmek için kullanılan temel kontrol yapılarından biridir. Genellikle bir dizi üzerindeki elemanları işlemek veya belirli bir koşul sağlandığı sürece aynı kod bloğunu yinelemek amacıyla kullanılır. Bir *for* döngüsü, çoğunlukla başlangıç değeri, bitiş değeri ve artış miktarı gibi parametrelerin tanımlanmasıyla başlatılır.



Yanda verilen akış şemasında (Şekil 3-6) integer türünde bir i değişkeni tanımlanmış ve hemen altına bir *for* döngüsü eklenmiştir. *For* ifadesinin bulunduğu alana çift tıklanarak aşağıda gösterilen pencere açılmaktadır (Şekil 3-7).

Şekil 3-6 For döngüsünün şemaya eklenmesi

For Özellikleri

For For döngüsü bir değer aralığı boyunca değişken artırır ya da azaltır. Bu yaygındır ve While döngüsüne göre kullanışlıdır.

Değişken:
i

Başlangıç Değeri: 0 Bitiş Değeri: 10

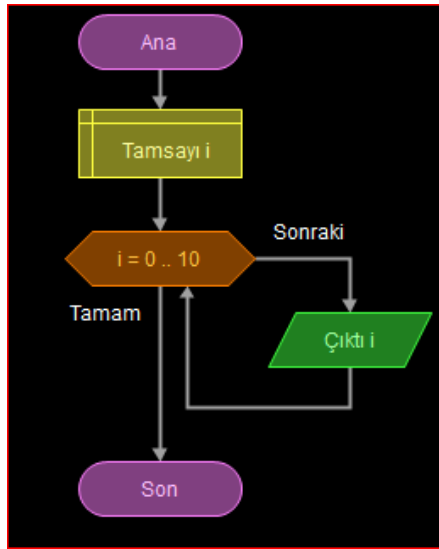
Yön: ☒ Artan ☐ Azalan

Adım: 1

OK Vazgeç

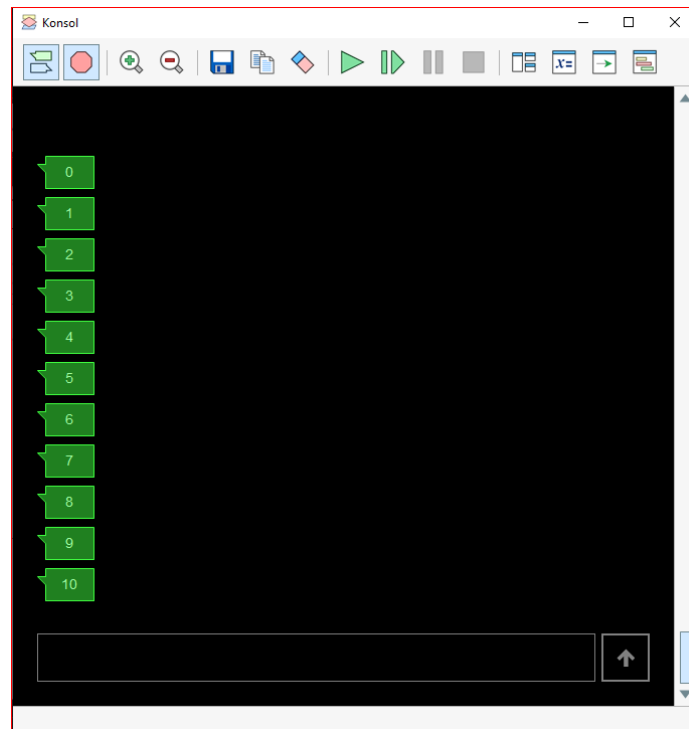
Şekil 3-7 Döngünün özelliklerinin belirlenmesi

Açılan pencerede döngünün işlem yapacağı değişken i olarak belirlenmiştir. Başlangıç değeri 0, bitiş değeri ise 10 olarak ayarlanmıştır. Artan bir döngüde, her adımda i değişkeninin değeri 1 artacaktır.



Şekil 3-8’de görülen akış şemasında program toplam 11 kez çalışacak; her döngü adımında i değişkeninin değeri 1 artırılarak konsol ekranına yazdırılacaktır. Programın çıktısı alındığında konsol ekranı Şekil 3-9’daki gibi görünür.

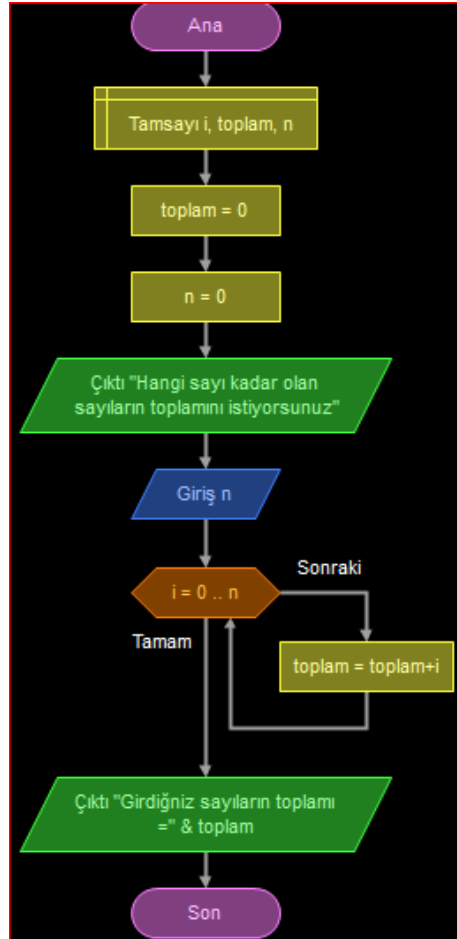
Şekil 3-8 Döngü eklenince oluşan akış şeması



Şekil 3-9 Program çalıştırıldığına oluşan çıktı

3.2.1 Örnek for döngüsü uygulaması (1'den n'ye kadar olan sayıları toplayan program)

Şekil 3-10'da belirli bir sayıya kadar girilen sayıların toplamını veren algoritma görülmektedir.



1. İlk olarak *toplam* ve *n* adlı iki değişken tanımlanır; *toplam* değişkeni başlangıçta sıfıra eşitlenir.
 2. Kullanıcıya “Hangi sayıya kadar olan sayıların toplamını istiyorsunuz?” şeklinde bir mesaj gösterilir ve kullanıcının girdiği değer *n* değişkenine atanır.
 3. Ardından, bir *for* döngüsü kullanılarak 0’dan başlayıp *n* değerine kadar olan sayılar üzerinde iterasyon yapılır.
 4. Her iterasyonda mevcut sayı *toplam* değişkenine eklenir.
 5. Döngü tamamlandığında *toplam* değişkeninin son değeri ekrana yazdırılır.
- İterasyon**, bir döngü içindeki her bir tekrar veya adımdır. Döngüler programlamada belirli bir işlemi, belirli bir koşul sağlanana kadar ya da belirli bir aralık boyunca tekrar etmek amacıyla sıkça kullanılır.

Şekil 3-10 Belirli bir sayıya kadar girilen sayıların toplamını veren algoritma



Flowgorithm dosyasına ulaşmak için kodu okutunuz.

3.2.2 Örnek For Uygulaması 2(Mükemmel Sayıyı Bulan Program)

Matematikte, kendisi hariç pozitif bölenleri toplamı sayının kendisine eşit olan sayılara *mükemmel sayılar* denir. Örneğin 6 sayısının kendisi hariç pozitif bölenleri 1, 2 ve 3'tür; çünkü $1 + 2 + 3 = 6$ olduğundan 6 bir mükemmel sayıdır.

Aşağıdaki akış şeması, kullanıcıdan bir sayı alıp bu sayının mükemmel sayı olup olmadığını kontrol eden bir programı göstermektedir. Yapılacak işlemler adım adım şöyledir:

1. *sayi*, *carpanToplam* ve *sayac* adlarında üç adet integer değişken tanımlanır. Başlangıçta *carpanToplam* değişkenine 0 değeri atanır. Bu değişken, sayının bölenlerinin toplamını tutacaktır.
2. Kullanıcıya 'Lütfen bir sayı giriniz.' mesajı gösterilir.
3. Kullanıcının girdiği değer alınarak tam sayıya dönüştürülür ve *sayi* değişkenine atanır.
4. Bir *for* döngüsü oluşturulur. Bu döngü, 1'den başlayarak kullanıcının girdiği sayının yarısına kadar devam eder. Çünkü bir sayının, kendisinden büyük bir pozitif böleni olmayacağı için bu aralık yeterlidir.
5. Döngü içerisinde, *sayi* değişkeninin *sayac* değişkenine tam bölünüp bölünmediği kontrol edilir. Kalan sıfır ise *sayac*, *sayi*'nin bir bölenidir.
6. Her bulunan bölen, *carpanToplam* değişkenine eklenir. Böylece sayının bölenleri toplamı elde edilir.
7. Döngü tamamlandıktan sonra *carpanToplam* ile *sayi* karşılaştırılır. Eğer bu iki değer eşitse kullanıcıdan alınan sayı bir mükemmel sayıdır.
8. Eğer sayı mükemmel bir sayı ise ekrana 'Tebrikler, girmiş olduğunuz sayı bir mükemmel sayıdır.' mesajı yazdırılır.
9. *carpanToplam* ile *sayi* eşit değilse kullanıcının girdiği sayının mükemmel sayı olmadığı belirlenir.
10. Bu durumda ekrana 'Maalesef, girmiş olduğunuz sayı bir mükemmel sayı değildir.' mesajı yazdırılır.

Bu adımları izleyerek kullanıcıdan alınan bir sayının mükemmel olup olmadığını kontrol eden bir akış şeması oluşturmanız gerekmektedir. Aşağıdaki bağlantıyı tıklayarak programın PDF çıktısına ulaşabilirsiniz.

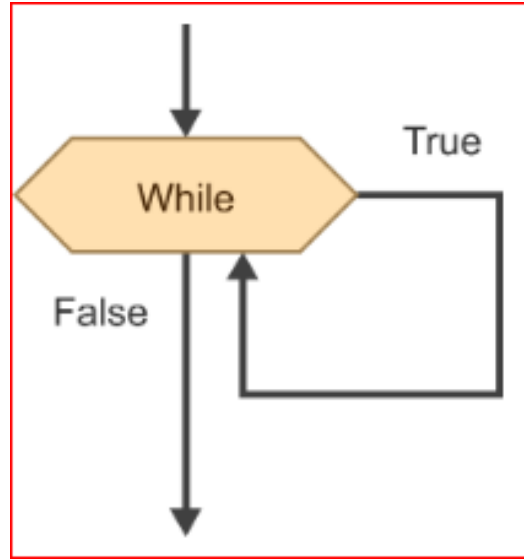
https://drive.google.com/file/d/1CS4JkA50ooudmz7uoTjONfwn5ur3vnc4/view?usp=drive_link



Flowgorithm dosyasına ulaşmak için kodu okutunuz.

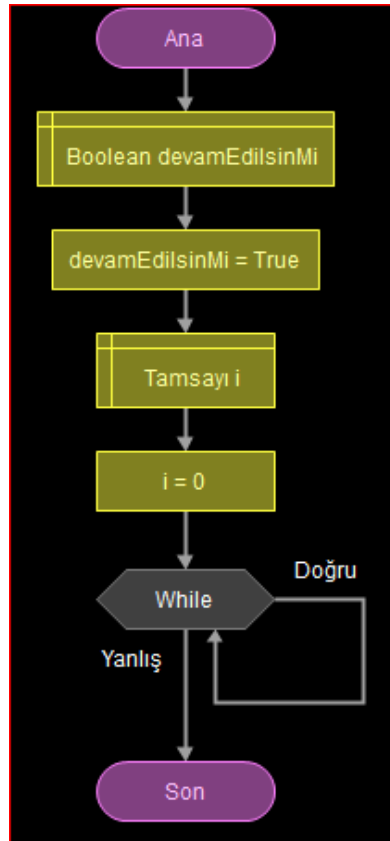
3.3 While Döngüsü

Şekil 3-11, Flowgorithm de yer alan while döngüsünün yapısını göstermektedir.



Şekil 3-11 while döngü şeması (Atzori, 2022)

While döngüsü, programlamada belirli bir koşul sağlandığı sürece bir kod bloğunun tekrar tekrar çalıştırılmasını sağlayan bir kontrol yapısıdır. Bu döngüde koşul, her döngü adımının başında değerlendirilir ve koşul doğru olduğu sürece döngü içindeki işlemler yürütülür. Koşul yanlış olduğunda ise döngü sona erer.



Şekil 3-12’de görülen akış şemasında Boolean türünde *devamEdilsinMi* adında bir değişken tanımlanmış ve başlangıç değeri *True* (doğru) olarak ayarlanmıştır. Ayrıca integer türünde *i* değişkeni tanımlanmış ve değeri 0 olarak belirlenmiştir.

Gelecek örnekte, kullanıcı *devamEdilsinMi* değişkenini *False* (yanlış) yapana kadar *i* değişkenine her adımda 1 ekleyen bir program oluşturacağız. *While* bloğuna çift tıklanarak şekil 3-13’te görülen pencere açılır.

Şekil 3-12 While döngüsünün şemaya eklenmesi

While Özellikleri

While While döngüsü bir Boolean ifadeyi değerlendirir, eğer doğruysa ifadeleri çalıştırır. İfadeyi denetler ve yanlış olana kadar döngüde tutar.

Aşağıya koşullu bir ifade giriniz:

devamEdilsinMi==true

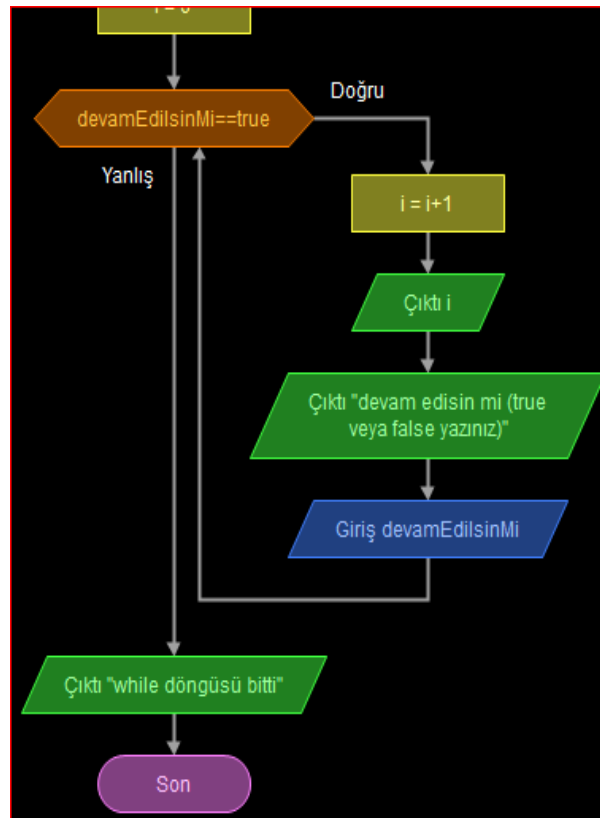
OK Vazgeç

Şekil 3-13 Döngü özelliklerinin belirlenmesi

Açılan pencereye, *devamEdilsinMi* değişkeni *True* olduğu sürece döngünün devam etmesi için *devamEdilsinMi == true* ifadesi yazılmıştır.

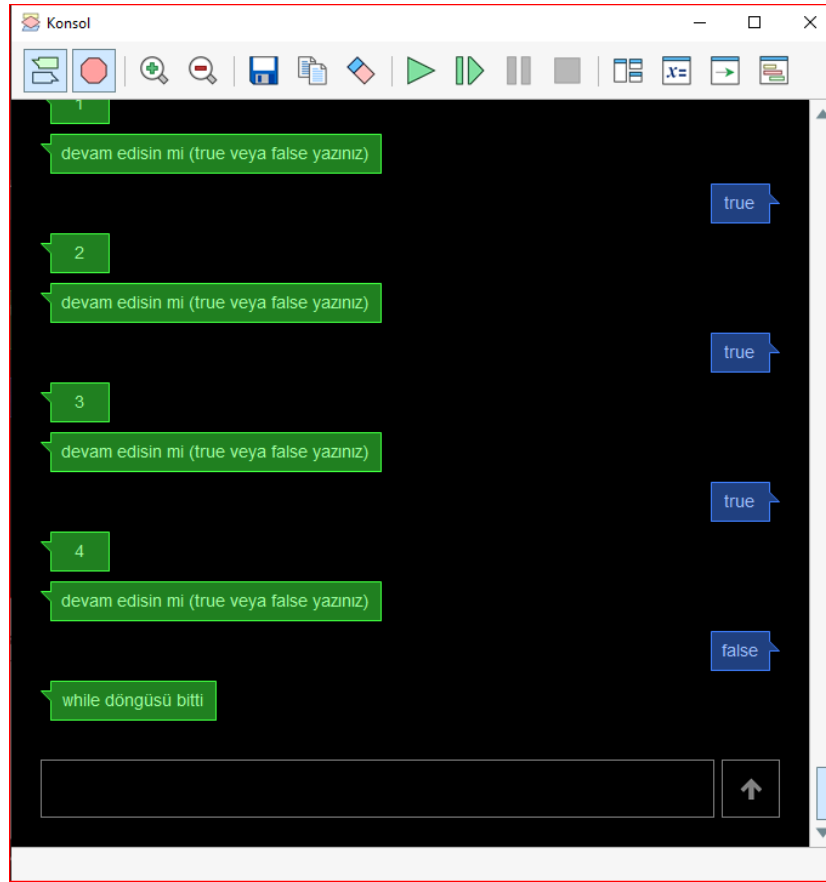
Daha sonra Şekil 3-14'te görüldüğü gibi while döngüsünün içine $i = i + 1$ ifadesi yazılarak i değişkeninin değeri her döngü adımında 1 artırılmıştır. Sonraki adımda i değeri konsol ekranına yazdırılır. Ardından kullanıcıya 'Devam edilsin mi?' şeklinde bir soru yöneltilir ve konsola true veya false yazması istenir.

Kullanıcının girdiği bu değer, input komutu ile devamEdilsinMi değişkenine atanır. Kullanıcı true yazdığı sürece döngü devam eder ve i değişkeni her seferinde 1 artırılarak ekrana yazdırılır. Kullanıcı false yazdığı anda döngü sonlanır ve konsol ekranına 'While döngüsü bitti.' mesajı yazdırılır.



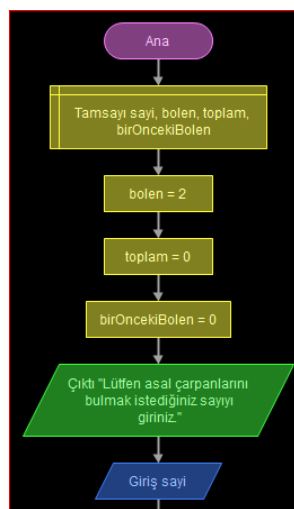
Şekil 3-14 Döngü eklendikten sonra oluşan görüntü

Program çalıştırıldığında, kullanıcı birkaç kez konsol ekranına *true* girdikten sonra *false* girdiğinde, konsol ekranı Şekil 3-15'teki gibi görüntülenir.



Şekil 3-15 Program çalıştırıldıktan sonra oluşan görüntü

3.3.1 While Döngüsü Örneği(Asal Çarpanları Bulma Uygulaması)

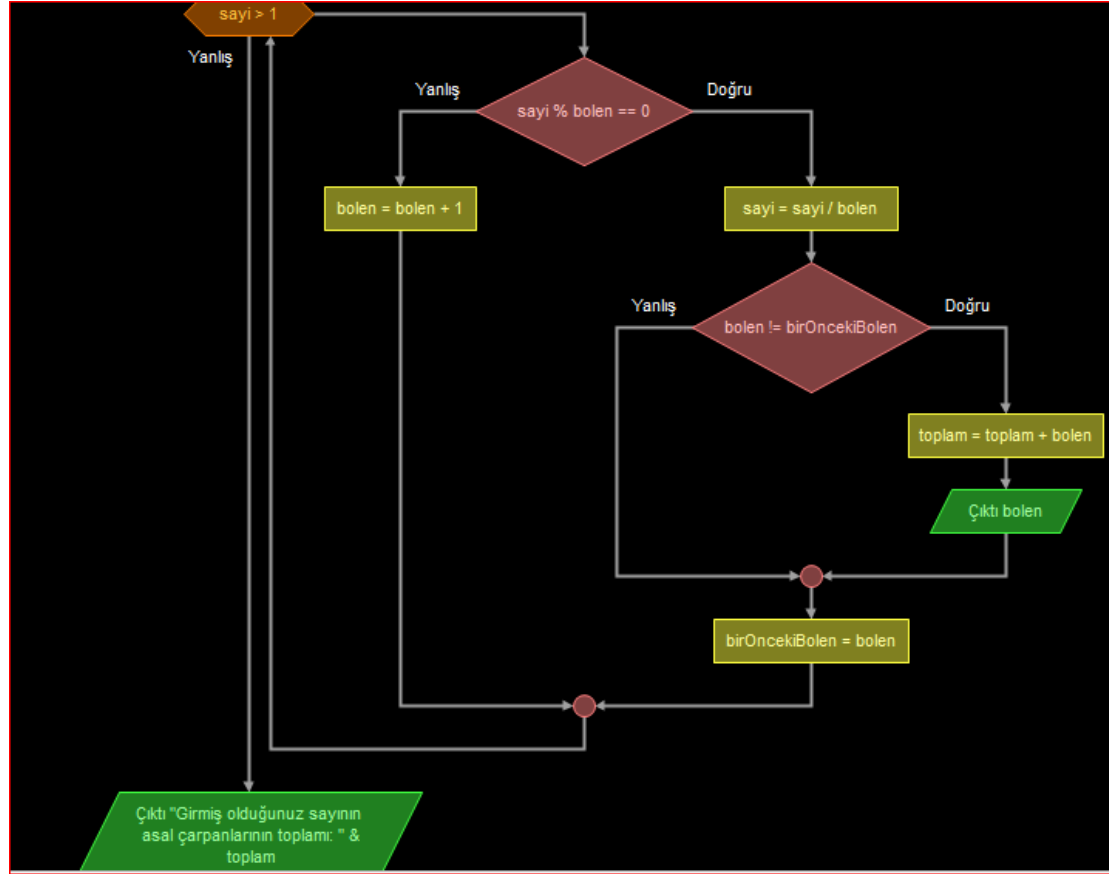


Şekil 3-16'da görülen akış şemasında, kullanıcının girdiği sayının asal çarpanlarını bulan ve bu çarpanların toplamını hesaplayan programın adım adım açıklaması şunlardır:

1. $bolen = 2$, $toplam = 0$ ve $birOncekiBolen = 0$ değişkenleri tanımlanır. Bu değişkenler, asal çarpanları ve bunların toplamını tutmak için kullanılacaktır.
2. Kullanıcıdan bir sayı girmesi istenir ve girilen değer $sayi$ değişkenine atanır.

Şekil 3-16 Asal çarpan algoritması 1. kısım

Şemanın devamında Şekil 3-17’de görüldüğü gibi bir *while* döngüsü tanımlanır ve *sayi* değişkeni 1’den büyük olduğu sürece döngü çalışır.



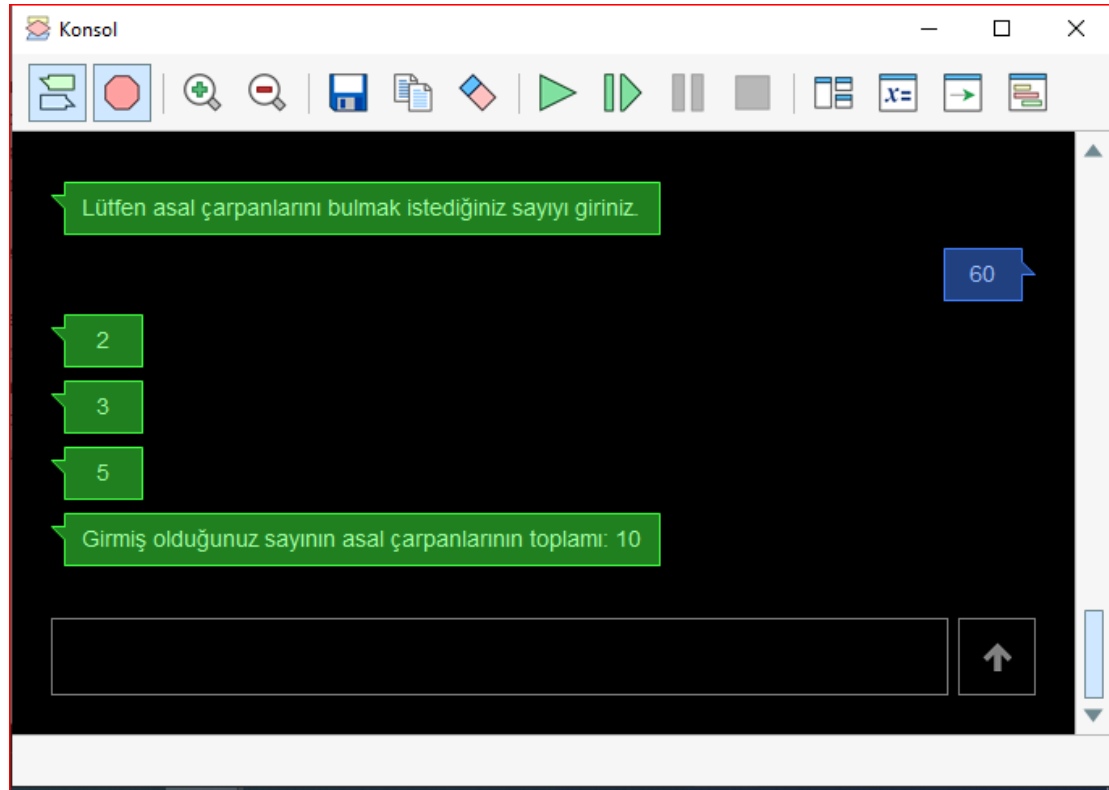
Şekil 3-17 Asal çarpan algoritması 2. kısım

Akış diyagramının adım adım çalışma açıklaması aşağıdaki gibidir:

1. *While* döngüsü, *sayi* 1’den büyük olduğu sürece devam eder; bu, sayının tüm asal çarpanlarını bulmak için gerekli bir döngü sağlar.
2. Döngü içinde, *sayi* değişkeninin *bolen* ile tam bölünüp bölünmediği kontrol edilir ($sayi \% bolen == 0$). Bölünüyorsa, *bolen* sayının bir asal çarpanıdır.
3. *Sayi*, *bolen* ile bölünerek güncellenir; böylece sayının değeri küçülür ve kalan asal çarpanlar üzerinde işlem yapılabilir.
4. Eğer *bolen*, bir önceki bölenden farklıysa, yeni bir asal çarpan bulunmuş demektir. Bu çarpan *toplam* değişkenine eklenir ve ekrana yazdırılır.
5. *birOncekiBolen* değişkeni, *bolen* değeriyle güncellenir; böylece bir sonraki iterasyonda bir önceki asal çarpan hatırlanmış olur.
6. Eğer *sayi*, *bolen* ile tam bölünmüyorsa, *else* bloğu çalışır ve *bolen* bir artırılır; bu, bir sonraki olası asal çarpanı kontrol etmek için yapılır.

7. *While* döngüsü, *sayi* 1 veya daha küçük olduğunda sona erer; böylece tüm asal çarpanlar bulunmuş olur.
8. Son olarak, bulunan asal çarpanların toplamı *toplam* değişkeninden alınır ve ekrana yazdırılır.

Program çalıştırıldığında konsol ekranı Şekil 3-18’de olduğu gibi görüntülenir.



Şekil 3-18 Asal çarpan algoritması çıktı



Flowgorithm dosyasına ulaşmak için kodu okutunuz.

3.3.2 While Döngüsü Örneği (Ebob bulma)

Matematikte, EBOB (En Büyük Ortak Bölen) bulmak için iki sayının bölenleri ayrı ayrı bulunup, ortak olan en büyük bölen seçilebilir. Alternatif bir yöntem ise, iki sayıyı birbirine eşit olana kadar büyük olandan küçük olanı çıkarmaktır; iki sayı eşit olduğunda bu sayı EBOB olur. Örneğin:

- Sayı1 = 9, Sayı2 = 6 olsun.
- Sayı1 > Sayı2 olduğu için Sayı1 = Sayı1 – Sayı2 $\Rightarrow$ Sayı1 = 9 – 6 = 3 olur.
- Sayı1 = 3 ve Sayı2 = 6 olduğunda Sayı2 > Sayı1 olduğundan Sayı2 = Sayı2 – Sayı1 $\Rightarrow$ Sayı2 = 6 – 3 = 3 olur.
- Son durumda Sayı1 = 3 ve Sayı2 = 3 olduğundan EBOB(9, 6) = 3 olur.

Bu kurala uygun bir akış şeması adım adım şöyle oluşturulabilir:

1. Kullanıcıdan “Lütfen EBOB hesaplanmasını istediğiniz 1. sayıyı giriniz.” mesajı ile sayı alınır ve *sayi1* değişkenine atanır.
2. Kullanıcıdan “Lütfen EBOB hesaplanmasını istediğiniz 2. sayıyı giriniz.” mesajı ile sayı alınır ve *sayi2* değişkenine atanır.
3. Bir *while* döngüsü başlatılır; bu döngü, *sayi1* ve *sayi2* birbirine eşit olana kadar devam eder. Yani, iki sayının EBOB’u bulunana kadar çalışır.
4. Döngü sırasında iki sayının farklı olup olmadığı kontrol edilir. Eğer *sayi1* > *sayi2* ise, *sayi1* = *sayi1* – *sayi2* işlemi yapılır. Eğer *sayi2* > *sayi1* ise, *sayi2* = *sayi2* – *sayi1* işlemi uygulanır.
5. Döngü, iki sayı birbirine eşit olduğunda sona erer; bu noktada *sayi1* ve *sayi2* aynı değerdedir ve bu değer EBOB’u temsil eder.
6. Son olarak, "EBOB: " & *sayi2* ifadesi ile EBOB değeri ekrana yazdırılır; EBOB, *sayi2* değişkeninde bulunur ve bir dizeye dönüştürülerek kullanıcıya gösterilir.

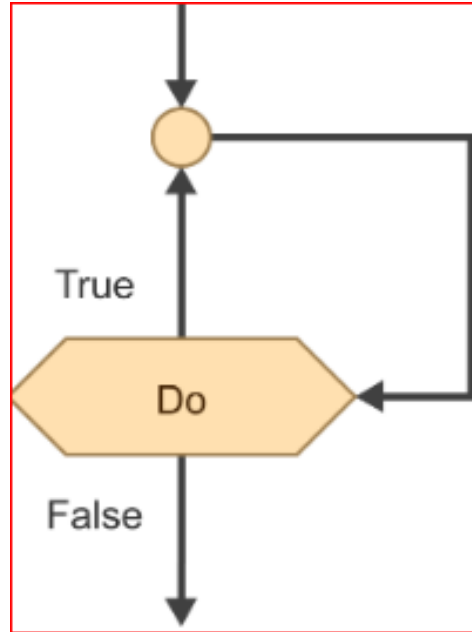
Oluşturulan şemanın linki: https://drive.google.com/file/d/1gYFmn1k-GCRacbHo8HmfA3XJWRBycbMz/view?usp=drive_link



Flowgorithm dosyasına ulaşmak için kodu okutunuz.

3.4 DO Döngüsü

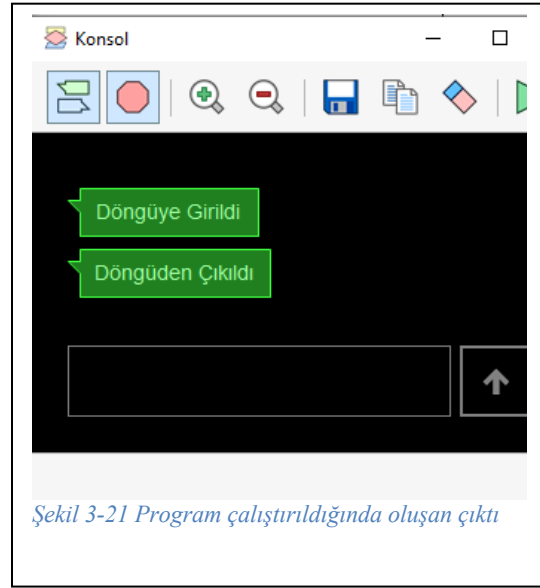
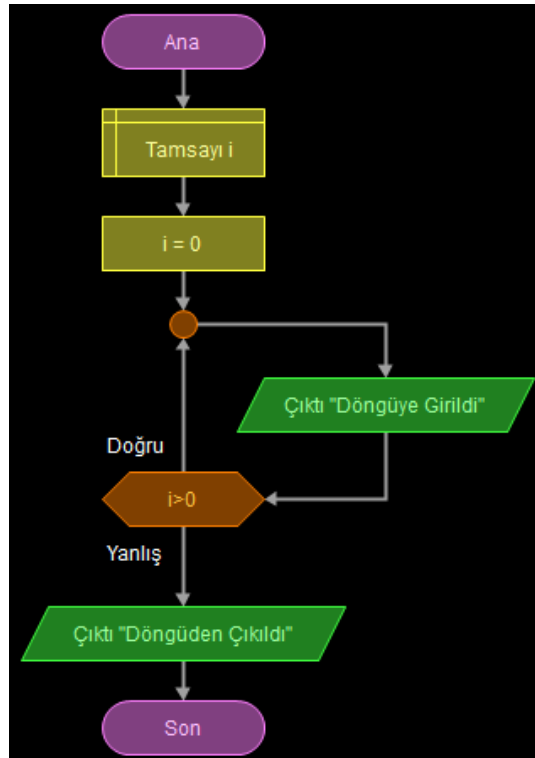
Aşağıda bulunan Şekil 3-19, Flowgorithm’de kullanılan *do* döngüsünün yapısını göstermektedir.



Şekil 3-19 Do döngü şeması (Atzori, 2022)

DO döngüsü, çalışma prensibi açısından *while* döngüsüne benzer. Ancak temel fark şudur: *While* döngüsünde koşul sağlanmazsa döngü bloğuna hiç girilmezken, *DO* döngüsünde döngü bloğu en az bir kez çalıştırılır. Döngü bloğu çalıştıktan sonra koşul kontrol edilir; koşul doğru ise döngü çalışmaya devam eder, yanlış ise döngü sona erer.

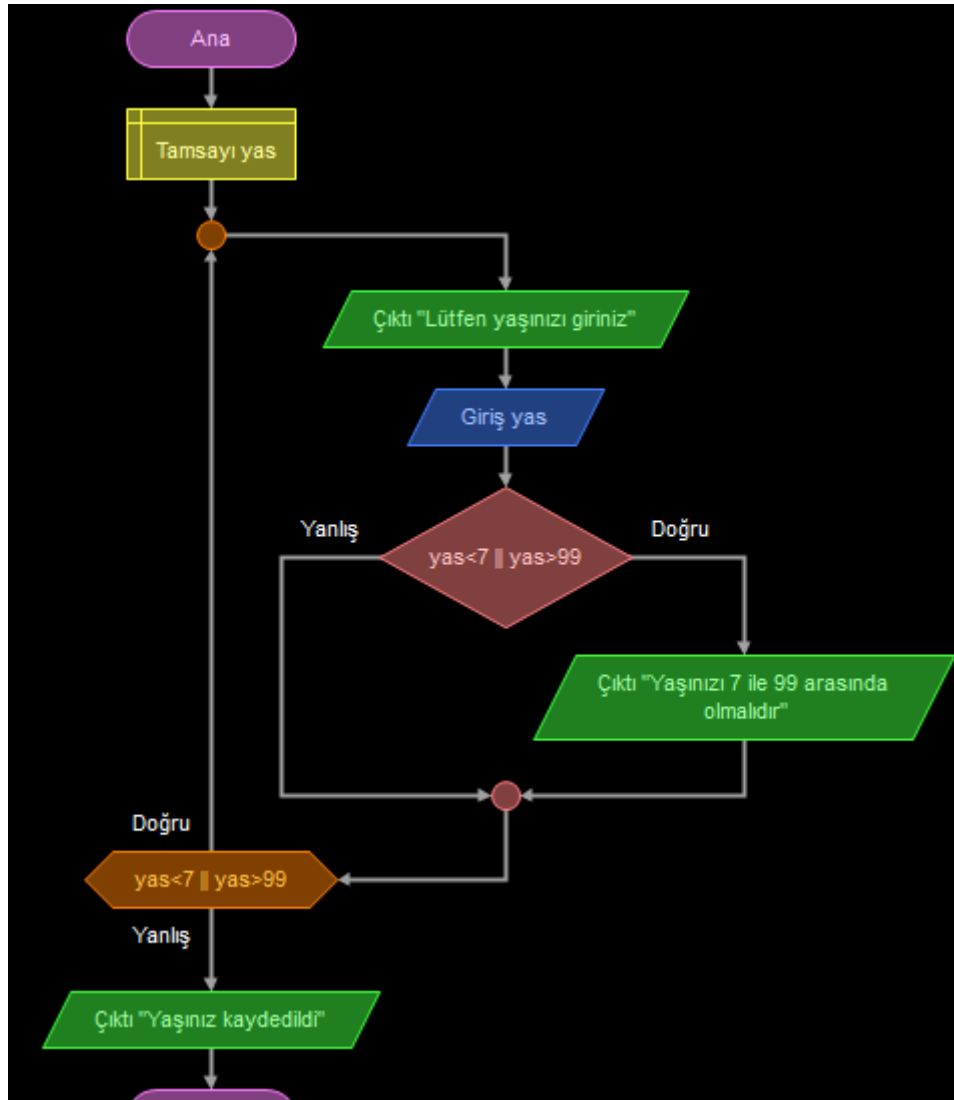
Aşağıda verilen akış şemasında (Şekil 3-20), *i* tamsayısının değeri 0 olarak atanmıştır. *DO* döngüsünde koşul, *i* 0'dan büyük olduğu sürece döngünün devam etmesi şeklinde belirlenmiştir. Koşul sağlanmasa bile döngü bloğu bir kez çalıştırılır ve hemen ardından döngüden çıkılır. Program çalıştırıldığında konsol ekranı Şekil 3-21'de olduğu gibi görüntülenir.



Şekil 3-20 DO döngüsünün akış şemasına eklenmesi

Do döngüsü, kullanıcıdan istenilen bilgiyi belirli kurallara göre girmediği sürece tekrar tekrar istemek için kullanılabilir. Döngüye bir kez girmesi, bu durumda büyük bir avantaj sağlar. Örneğin, kullanıcıdan yaşını istediğimiz bir uygulamada, kullanıcının yalnızca 7 ile 99 arasında bir değer girmesi kabul edilsin; yanlış bir değer girildiğinde *do* döngüsü devreye girerek doğru değer girilene kadar kullanıcıdan tekrar bilgi alır.

Aşağıda verilen program (Şekil 3-22) incelendiğinde, kullanıcı 7 ile 99 arasındaki bir sayı girene kadar döngü devam eder. Kullanıcı istenilen aralıkta bir değer girdiğinde döngü sona erer ve program ‘Yaşınız kaydedildi.’ mesajını ekrana yazdırarak tamamlanır.



Şekil 3-22 Do döngüsü algoritma örneği

4 Bölüm

4.1 DİZİLER

Aynı türden verileri belirli bir düzen içinde saklamak için kullanılan yapılar *dizi* (*array*) olarak adlandırılır. Diziler tanımlanırken, kaç eleman depolayacakları ve boyutları genellikle tanımlama aşamasında belirtilir. Dizideki her bir elemana erişim için *indis* kullanılır ve indisler her zaman 0'dan başlar.

Şimdi, tanımlama sekmesine tıklayarak Şekil 4-1'de görülen pencereyi açalım.

Tanımlama Özellikleri

Tanımlama Komutu değişkenler ve diziler oluşturmak için kullanılır. Bunlar program çalışırken veriyi saklar

Değişken isimleri:

stringDizisi

Tip:

☐ Tamsayı ☒ Dizi?

☐ Reel

☒ String

☐ Boolean

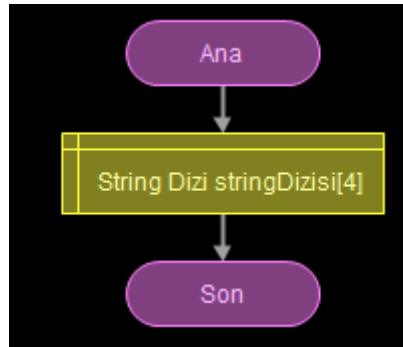
Dizinin boyutu:

4

OK Vazgeç

Şekil 4-1 Dizi Tanımlama Ekranı

Verilen pencerede veri tipi olarak *String* seçilir. Ardından *Dizi* sekmesine tıklandığında, hemen altında dizinin boyutunu girebileceğiniz bir alan açılır. Dizimizin kaç elemanlı olacağını bu alana girip *OK* düğmesine tıklıyoruz.



Şekil 4-2’de görüldüğü gibi 4 elemanlı dizimiz akış şemasına eklenmiştir. Artık dizinin elemanlarına değer atayabiliriz. Atama ekranına giderek aşağıda verilen pencereyi (Şekil 4-3) açıyoruz.

Şekil 4-2 Tanımlanmış dizinin akış şemasındaki görünümü

Atama Özellikleri

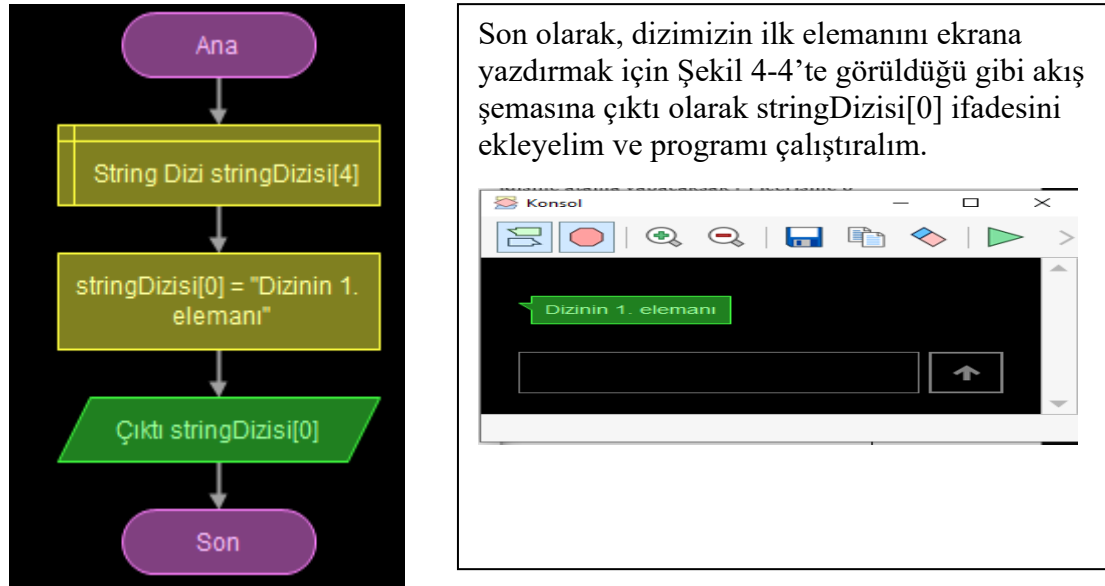
Atama Atama Komutu bir ifade hesaplar ve sonucu bir değişkende saklar.

Değişken: `stringDizisi[0]` = Açıklama: `"Dizinin 1. elemanı"`

OK Vazgeç

Şekil 4-3 Dizinin belirli bir elemanının değerini belirleme

Değişken olarak dizimizin ismini ve atama yapılacak indisi köşeli parantez [] içinde belirtiyoruz. Örneğin, 0. indise ‘Dizinin 1. elemanı’ değerini atamak için `stringDizisi[0] = "Dizinin 1. elemanı"` şeklinde yazıyoruz ve ardından *OK* düğmesine tıklıyoruz.

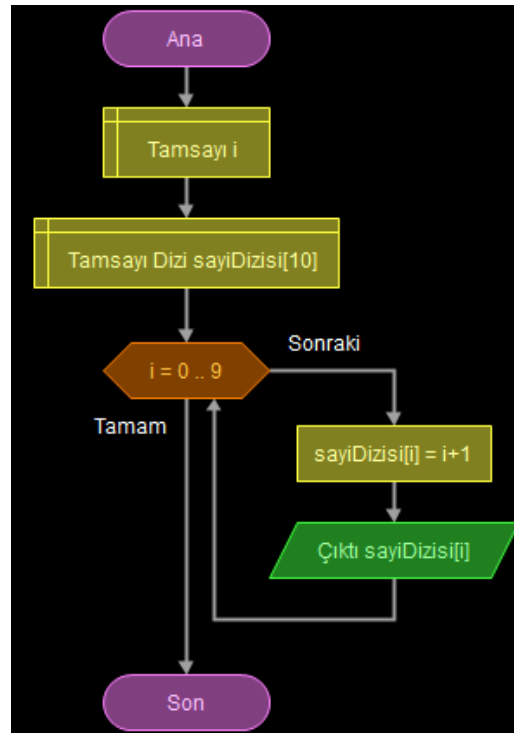


Şekil 4-4 Dizi şekilleri eklendikten sonra oluşan akış şemasının görüntüsü

Konsol ekranında çıktı olarak 'Dizinin 1. Elemanı' ifadesini aldık. Böylece *stringDizisi* dizimizin 0. indisi belirlenmiş oldu.

4.1.1 Dizi Örneği 1 (Rakamları dizinin elemanı yapan program)

Aşağıda yer alan Şekil 4-5, rakamları dizinin elemanı yapan algoritmanın akış diyagramını göstermektedir.



Şekil 4-5 Rakamları dizinin elemanı yapan akış şeması

Yukarıda verilen akış şemasında `sayiDizisi[10]` olarak boyutu 10 olan bir tam sayı dizisi tanımlanmıştır. Dizinin elemanlarını atarken *for* döngüsünden yararlanılmıştır; döngü 0'dan 9'a kadar birer birer artmaktadır.

`SayiDizisi[i] = i + 1` ifadesiyle, her döngü adımında *i* değeri kullanılarak dizinin ilgili indisine (yani *i*'inci indise) *i* + 1 değeri atanır. *i*'nin değeri 0'dan başladığı için, *i* + 1 işlemi sayesinde diziye 1'den başlayarak 10'a kadar olan sayılar yerleştirilmiş olur.

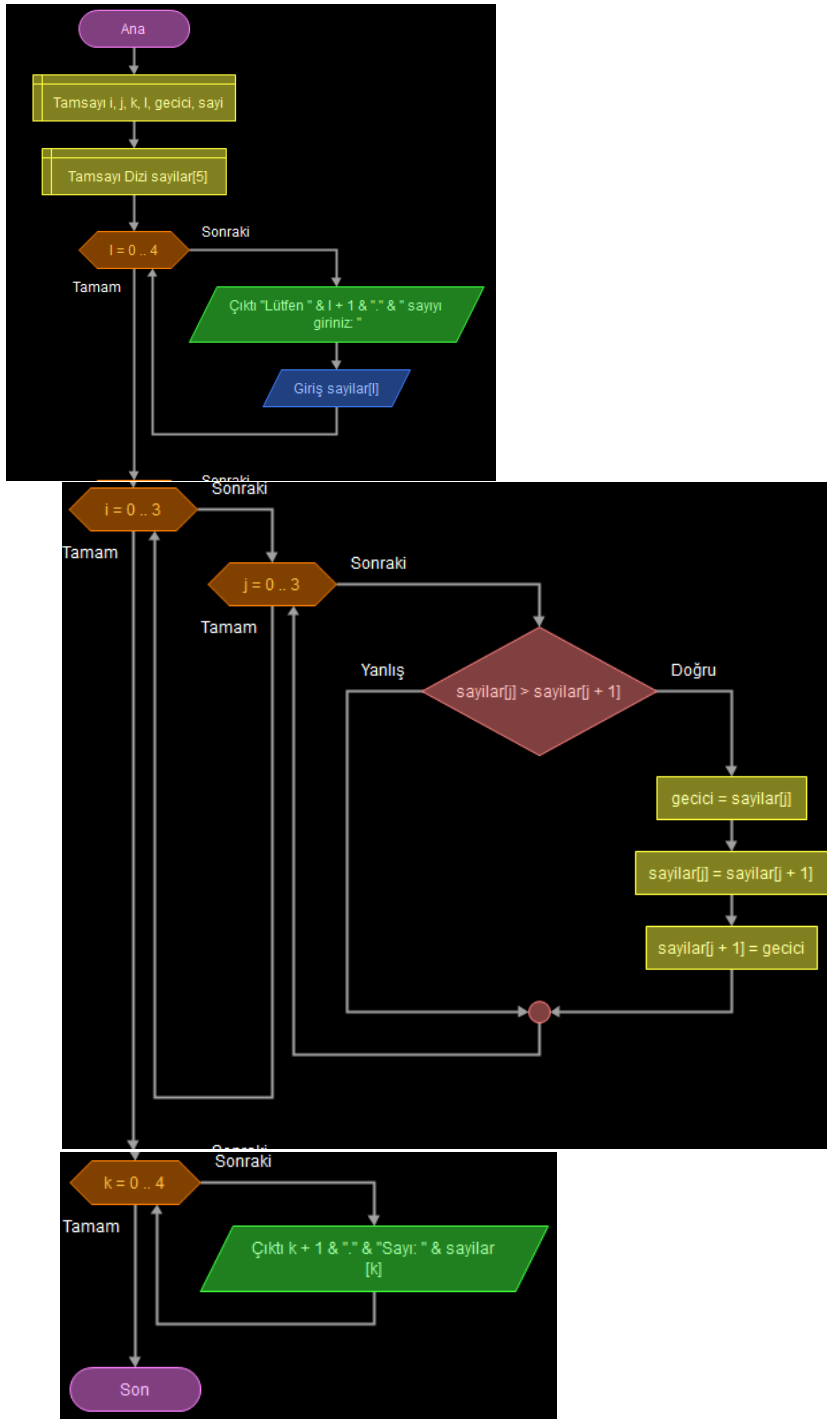
Son olarak, her bir sayı diziye atandıktan sonra, o anki *i* değerine karşılık gelen dizi elemanı konsol ekranına yazdırılır.



Flowgorithm dosyasına ulaşmak için kodu okutunuz.

4.1.2 Dizi Örneği 2 (Kullanıcıdan Alınan 5 farkı sayıyı sıralayan program)

Aşağıda Şekil 4-6'da görülen akış şeması incelendiğinde, adım adım yapılan işlemler şemanın altında açıklanmıştır.

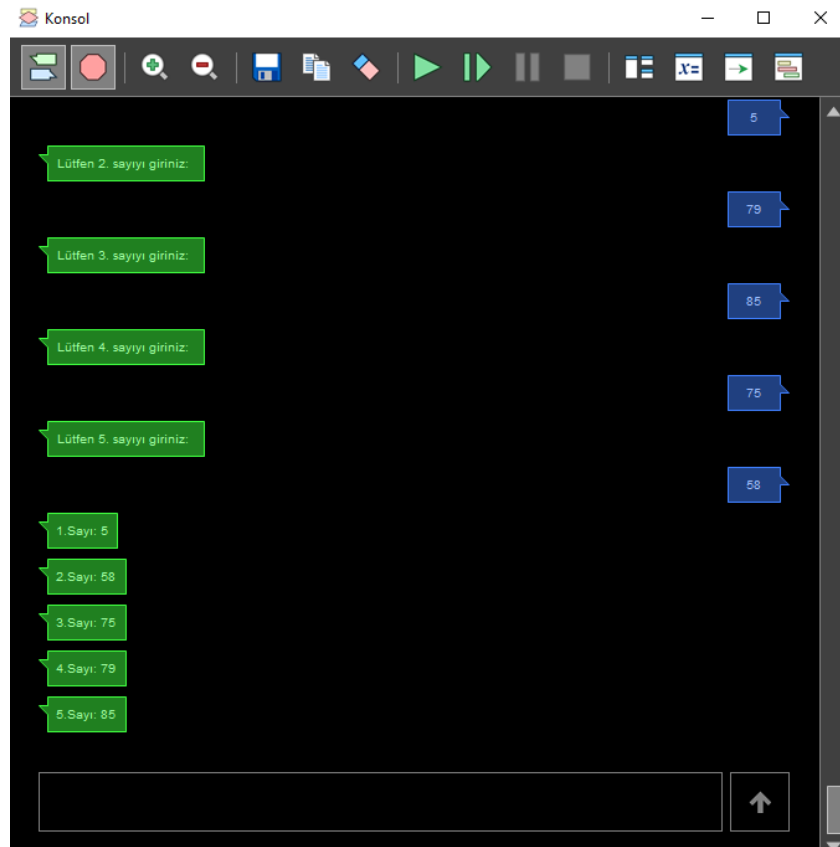


Şekil 4-6 Beş farklı sayıyı sıralayan algoritma örneği

1. sayilar[5]: sayilar adında bir liste oluşturulur ve başlangıçta beş elemanı sıfır olan bir liste olarak tanımlanır. Yani, [0, 0, 0, 0, 0] şeklinde bir liste oluşur.
2. 1. For döngüsü: Bu döngü, kullanıcıdan beş adet sayı girişi alır. l değişkeni 0'dan 4'e kadar değerler alır; yani döngü beş kez çalışır.
3. Çıktı("Lütfen " & i + 1 & ". sayıyı giriniz: "): Her döngü adımında kullanıcıya hangi sıradaki sayıyı girmesi gerektiği sorulur. i + 1 ifadesi,

insanların saymaya 1'den başladığını dikkate alır ve sayının kaçınıcı olduğunu gösterir.

4. Giriş sayılar(1): Kullanıcıdan alınan değer, 1 değişkenine göre sayılar listesinin ilgili indeksine atanır.
5. 2. ve 3. For döngüsü: Bu döngüler, kabarcık sıralama algoritması kullanılarak girilen sayıları küçükten büyüğe doğru sıralar. İç içe iki döngü, liste elemanlarını karşılaştırır ve gerekirse yer değiştirir.
6. if sayılar[j] > sayılar[j + 1]: Listenin ardışık elemanları karşılaştırılır. Eğer bir önceki eleman bir sonraki elemandan büyükse, yer değişimi yapılır.
7. 4. For döngüsü (0, 4 + 1, 1): Son olarak, sıralanmış sayılar Şekil 4-7'de olduğu gibi ekrana yazdırılır. Bu döngü sayesinde sıralı liste konsola basılır. Benzer şekilde, k + 1 ifadesi, insanların saymaya 1'den başladığını dikkate alır ve sayının kaçınıcı olduğunu gösterir.



Şekil 4-7 Programın çıktısı



Flowgorithm dosyasına ulaşmak için kodu okutunuz.

4.1.3 Dizi Örneği 3 (Kullanıcının Girdiği En Büyük Sayıyı Bulan Program)

Şimdi, kullanıcıdan alınan 5 sayıyı bir dizide saklayan ve bu sayılardan en büyüğünü belirleyen bir programı oluşturmak için aşağıdaki adımları takip edebilirsiniz:

1. `sayi[5]` adında bir liste oluşturulur ve başlangıçta beş elemanı sıfır olan bir liste olarak tanımlanır. Yani, `[0, 0, 0, 0, 0]` şeklinde bir liste oluşur.
2. Tamsayı türünde `i`, `enb` ve sayıcı değişkenleri tanımlanır.
3. `enb = 0`: `enb` değişkeni, en büyük sayıyı takip etmek için kullanılır ve başlangıçta 0 değeri atanır.
4. `for (0...4, 1)`: Bu döngü, kullanıcıdan beş adet sayı girişi alır. `i` değişkeni 0'dan 4'e kadar değerler alır (5 hariç). Döngü beş kez çalışır.
5. `Çıktı(i + 1 & ". sayıyı giriniz")`: Her döngü adımında, kullanıcıya hangi sıradaki sayıyı girmesi gerektiği sorulur. `i + 1` ifadesi, insanların saymaya 1'den başladığını dikkate alır.
6. Giriş `sayi[i]`: Kullanıcıdan alınan değer, `i` değişkenine göre `sayi` listesinin ilgili indeksine atanır.
7. `for (0...4, 1)`: Yeni bir döngü tanımlanır. Bu döngü, girilen sayılar içinden en büyük olanı bulmak için kullanılır. `i` değişkeni 0'dan 4'e kadar değerler alır (5 hariç). Döngü beş kez çalışır.
8. `if enb < sayi[i]`: Listenin her elemanı, `enb` değişkenindeki en büyük sayı ile karşılaştırılır. Eğer listenin elemanı `enb`'den büyükse, `enb` değişkeni bu elemana eşitlenir (`enb = sayi[i]`).
9. `sayıcı = i + 1`: `sayıcı` değişkeni, en büyük sayının kaçınıcı sırada olduğunu gösterir. `i + 1` ifadesi, insanların saymaya 1'den başladığını dikkate alır.
10. `Çıktı("Girdiğiniz sayılar içerisinde en büyük sayı " & sayıcı & ". sayı olan " & enb & " sayısıdır.")`: En büyük sayı ve sırası ekrana yazdırılır.

İstenen programın akış diyagramına aşağıdaki bağlantıdan ulaşılabilir:

https://drive.google.com/file/d/1bmQU0mP0Kee9nLVc22LQTvz9sBe8ml7E/view?usp=drive_link

Verilen linke tıklayarak programın pdf şemasına ulaşabilirsiniz.

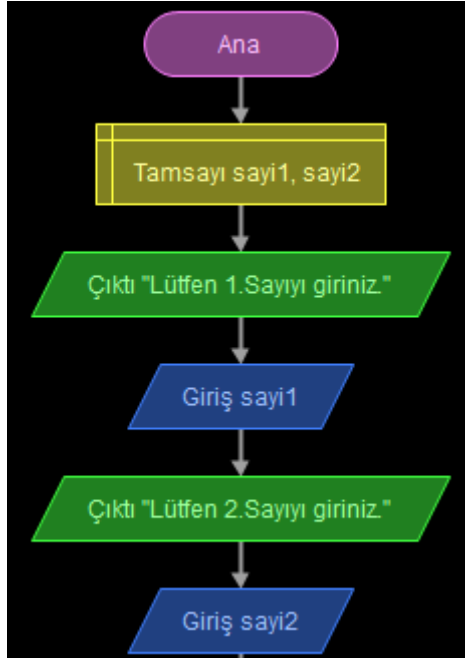


Flowgorithm dosyasına ulaşmak için kodu okutunuz.

5 Bölüm

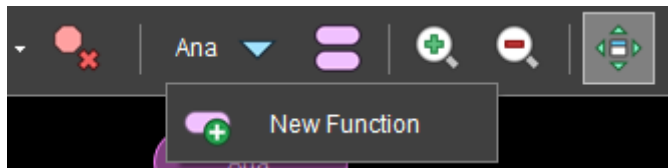
5.1 Fonksiyonlar

Programlamada bir işlev veya fonksiyon, belirli bir görevi yerine getiren ve bir veya birden fazla girdi alarak bir çıktı üreten kod bloğudur. Fonksiyonlar, kodun modüler hâle getirilmesi ve tekrar kullanılabilirliğinin artırılması açısından önemlidir. Bir fonksiyon genellikle bir isimle çağrılır ve fonksiyon bloğu içinde tanımlanır.



Şekil 5-1’de görüldüğü gibi, kullanıcıdan iki sayı almak için sayi1 ve sayi2 adında tamsayı değişkenleri tanımlanmış ve Giriş ile kullanıcıdan değerleri alınmıştır.

Şekil 5-1 Fonksiyon uygulaması Akış şeması 1. kısım



Şekil 5-2 Fonksiyon uygulaması Akış şeması 2. kısım

Daha sonra Şekil 5-2’de görüldüğü gibi *Ana* sekmesinin altında yer alan *New Function* sekmesine tıklayarak aşağıda verilen pencereyi açıyoruz.

Açılan pencerede, sol tarafta bulunan *Ekle* kısmına tıklayarak fonksiyonumuzun parametrelerini ekleyeceğimiz pencereyi Şekil 5-3'de görebilirsiniz.

Şekil 5-3 Fonksiyon uygulaması Akış şeması 3. Kısım

Aşağıda verilen Şekil 5-4 fonksiyon parametrelerinin ekleneceği pencereyi göstermektedir.

Şekil 5-4 Fonksiyon uygulaması Akış şeması 4. kısım

Parametre olarak iki farklı tamsayı ekleyelim; bu tamsayılar a ve b olsun. Parametreleri belirledikten sonra fonksiyon penceremiz Şekil 5-5'deki gibi olur.

Fonksiyon Özellikleri

Fonksiyon Bir fonksiyon programların kodları tekrar kullanmasını ve mantığı basitleştirmesini sağlar. Veri, parametreler kullanarak fonksiyonlardan geçer.

Fonksiyon Adı:

Parametreler:

- ☐ Tamsayı a
- ☐ Tamsayı b

Ekle Düzenle Sil

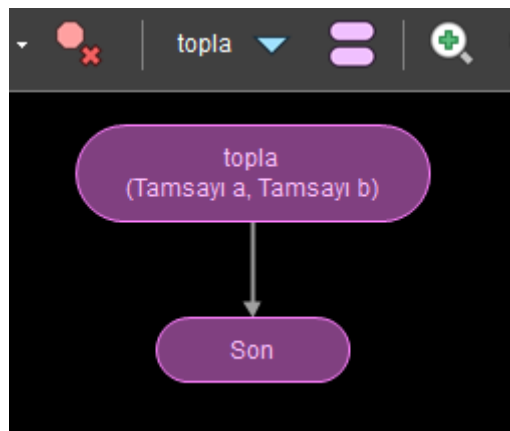
Dönüş Tipi:

- ☐ Tamsayı
- ☐ Reel
- ☐ String
- ☐ Boolean
- ☒ Yok

OK Vazgeç

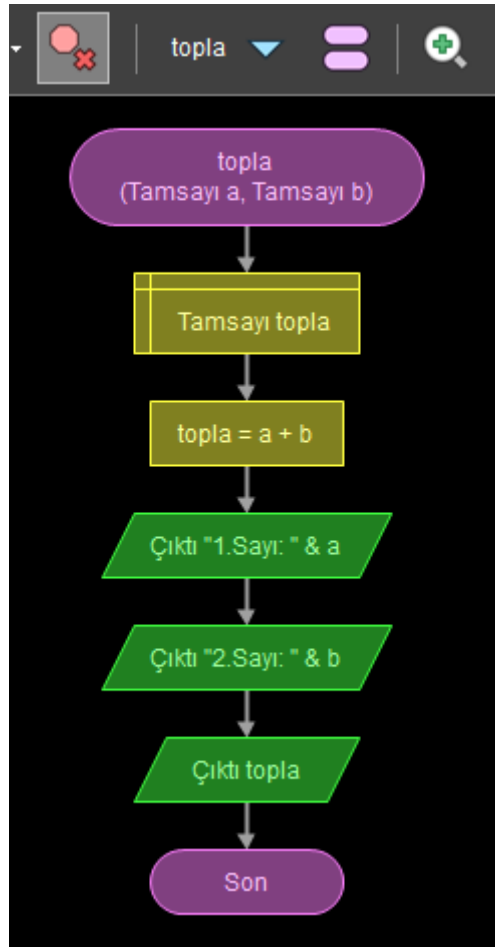
Şekil 5-5 Fonksiyon uygulaması Akış şeması 5. kısım

OK sekmesine tıkladıktan sonra, üst sekmedeki ana ekranda fonksiyonumuzun adı Şekil 5-6’te olduğu gibi topla olarak değişecektir.



Artık, topla fonksiyonunda yapılacak işlemleri buradan düzenleyebiliriz.

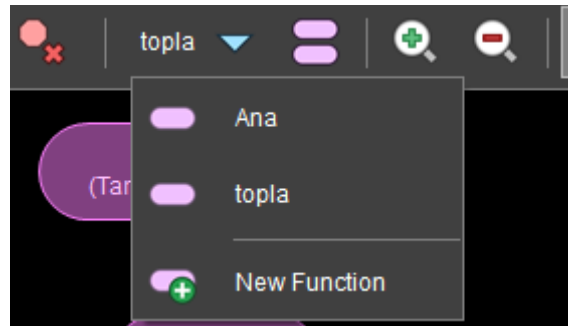
Şekil 5-6 Fonksiyon uygulaması Akış şeması 6. kısım



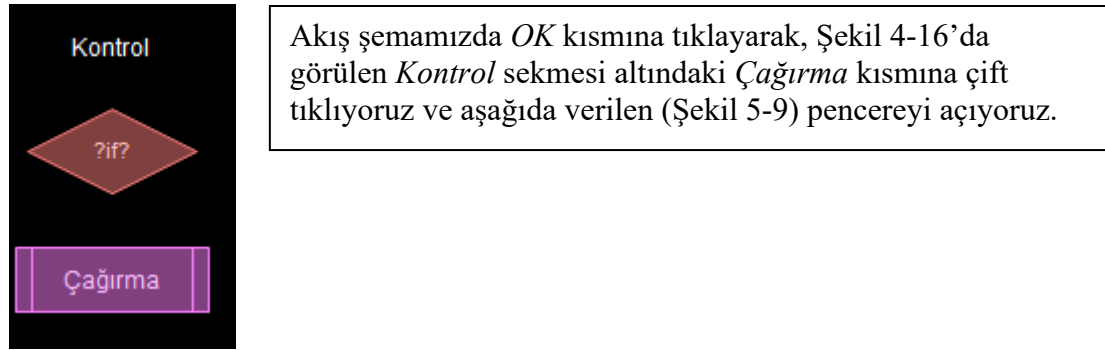
Şekil 5-7’te görüldüğü gibi topla fonksiyonunun içinde topla adında bir tamsayı değişkeni tanımladık. Daha sonra girilen a ve b sayılarının toplamını topla değişkenine atadık. Son olarak, 1. sayı a, 2. sayı b ve topla değişkeninin değeri ekrana çıktı olarak yazdırıldı.

Şekil 5-7 Fonksiyon uygulaması Akış şeması 7. kısım

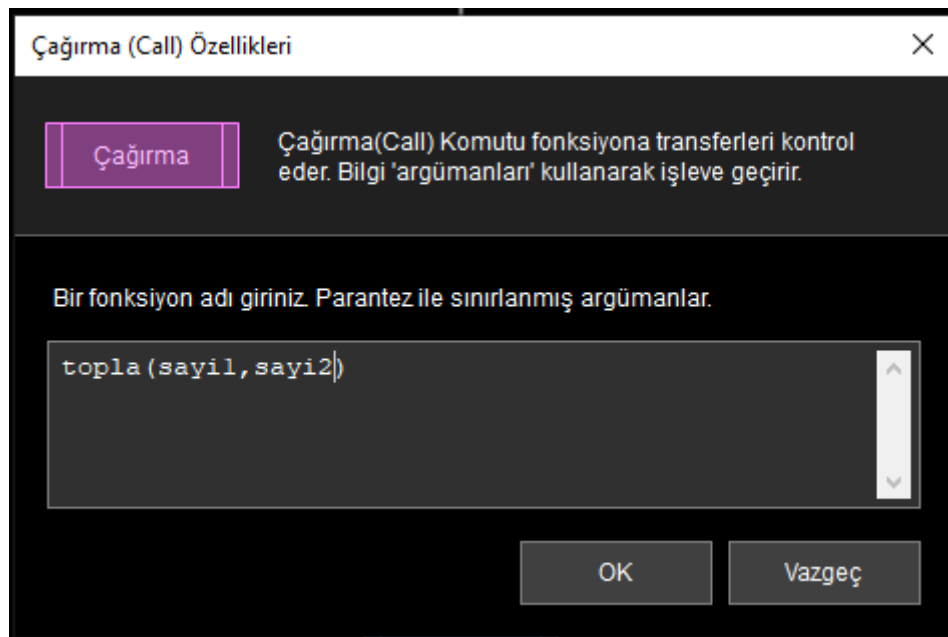
Artık topla fonksiyonunda işimiz tamamlandı. Bu fonksiyonu çalıştırmak için üst sekmedeki topla kısmına tıklayarak burayı ana fonksiyonumuzla değiştiriyoruz (Şekil 5-8).



Şekil 5-8 Fonksiyon uygulaması Akış şeması 8. kısım

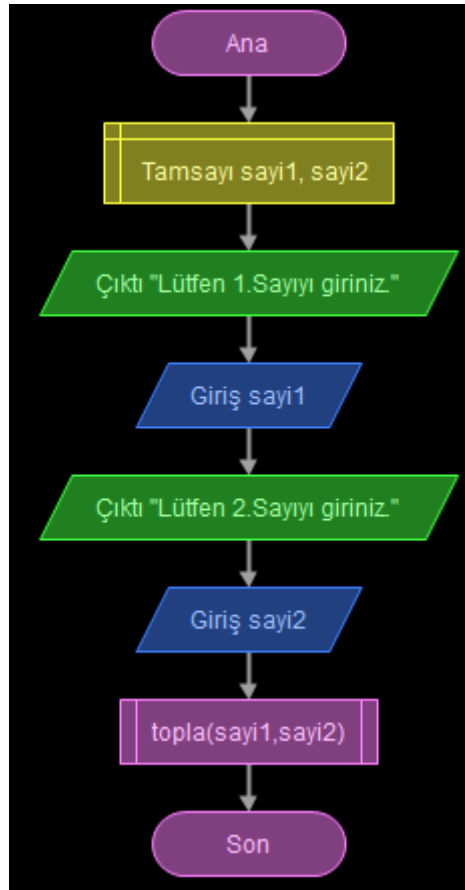


Şekil 5-9 Fonksiyon uygulaması Akış şeması 9. Kısım



Şekil 5-10 Fonksiyon uygulaması Akış şeması 10. kısım

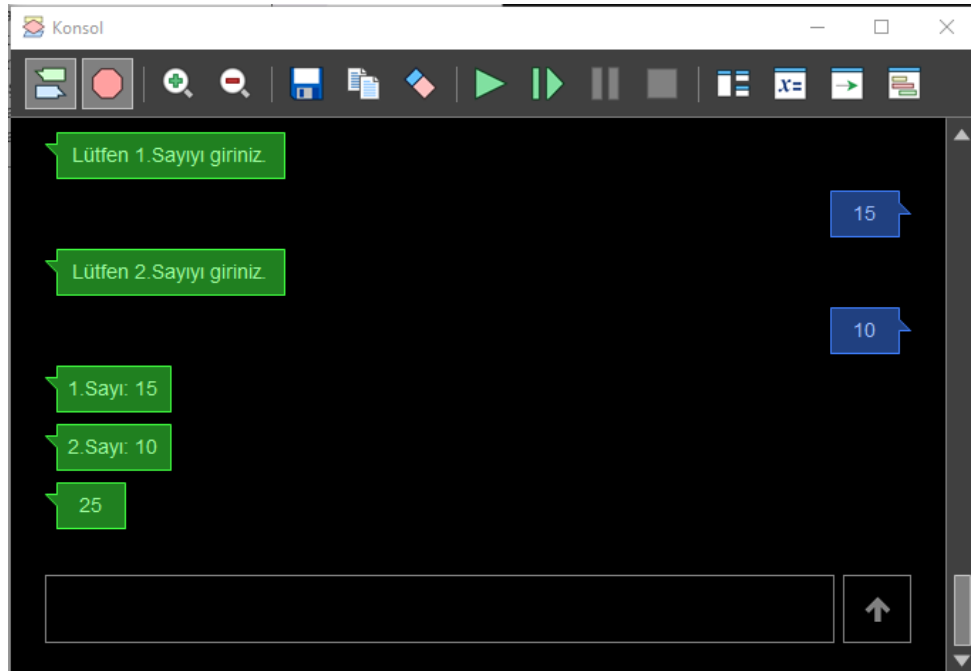
Şekil 5-10'da görülen pencerede, topla fonksiyonunu çağırmak ve kullanıcıdan alınan sayı1 ile sayı2 değişkenlerini toplamak için topla (sayı1, sayı2) yazıyoruz ve ardından *OK* sekmesine tıklıyoruz.



Son durumda akış şemamız Şekil 5-11'deki gibi görünmektedir. Artık kullanıcıdan alınan iki sayı, topla fonksiyonunda tanımladığımız a ve b değişkenlerinin yerine geçerek toplanacak ve çıktı olarak sayıların toplamı ekrana yazdırılacaktır.

Şekil 5-11 Fonksiyon uygulaması Akış şeması 11. kısım

Program çalıştırıldığında, konsol ekranında Şekil 5-12'deki çıktıyı görürüz.



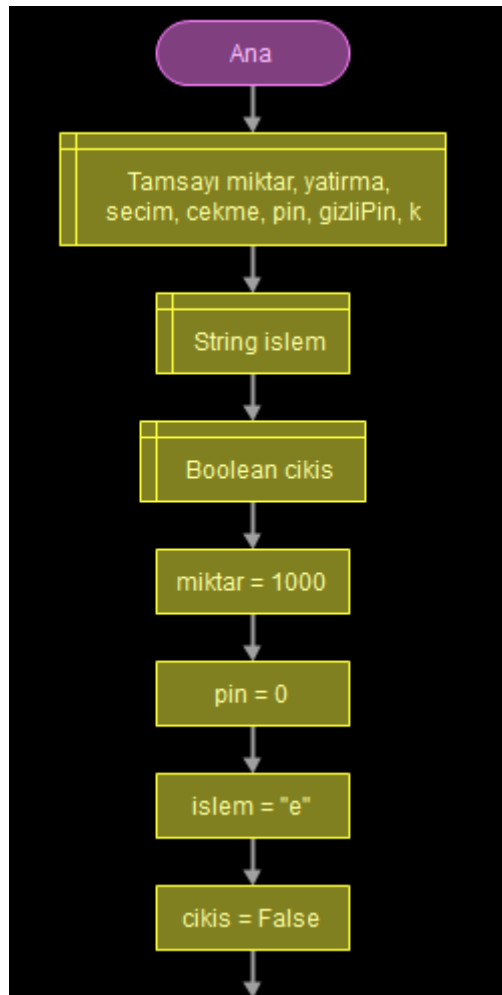
Şekil 5-12 Fonksiyon uygulaması Akış şemasının çıktısı



Flowgorithm dosyasına ulaşmak için kodu okutunuz

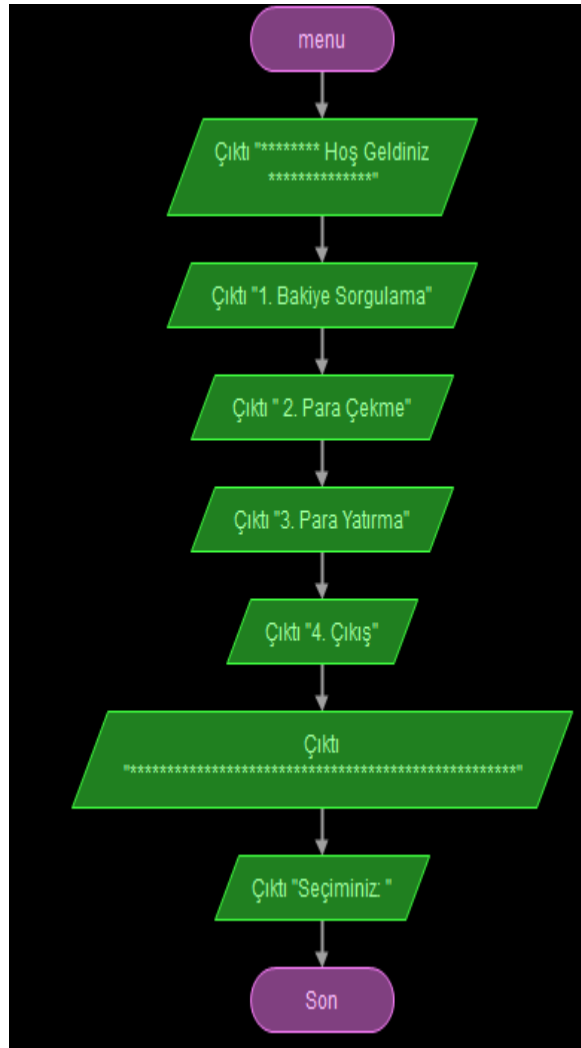
5.1.1 Fonksiyon kullanımı örneği (Bankamatik Uygulaması)

Fonksiyonları anlamak için bir bankamatik düşünün; para çekme, para yatırma ve bakiye sorgulama işlemlerinin her biri, arka planda çalışan ayrı birer fonksiyondur. Şimdi gerekli akış şemalarını (Şekil 5-13) oluşturarak başlayalım.



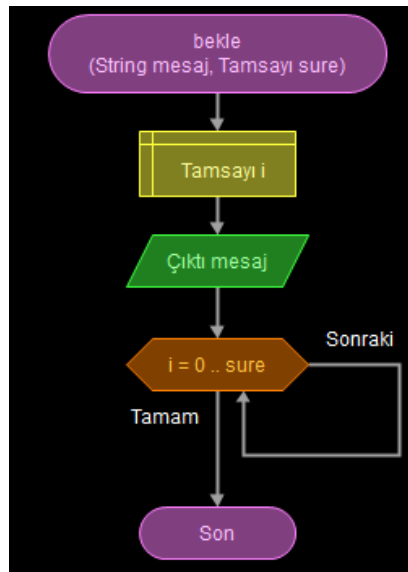
Değişken Tanımlamaları: miktar, yatırma, secim, cekme, pin, gizliPin, k ve işlem adında değişkenler tanımlanır.
Değişkenlerin Başlangıç Değerleri: miktar başlangıçta 1000 olarak atanır; pin, işlem ve cikis değişkenleri ise sırasıyla 0, "e" ve false olarak başlatılır.

Şekil 5-13 Bankamatik uygulaması Ana Algoritma 1. Kısım



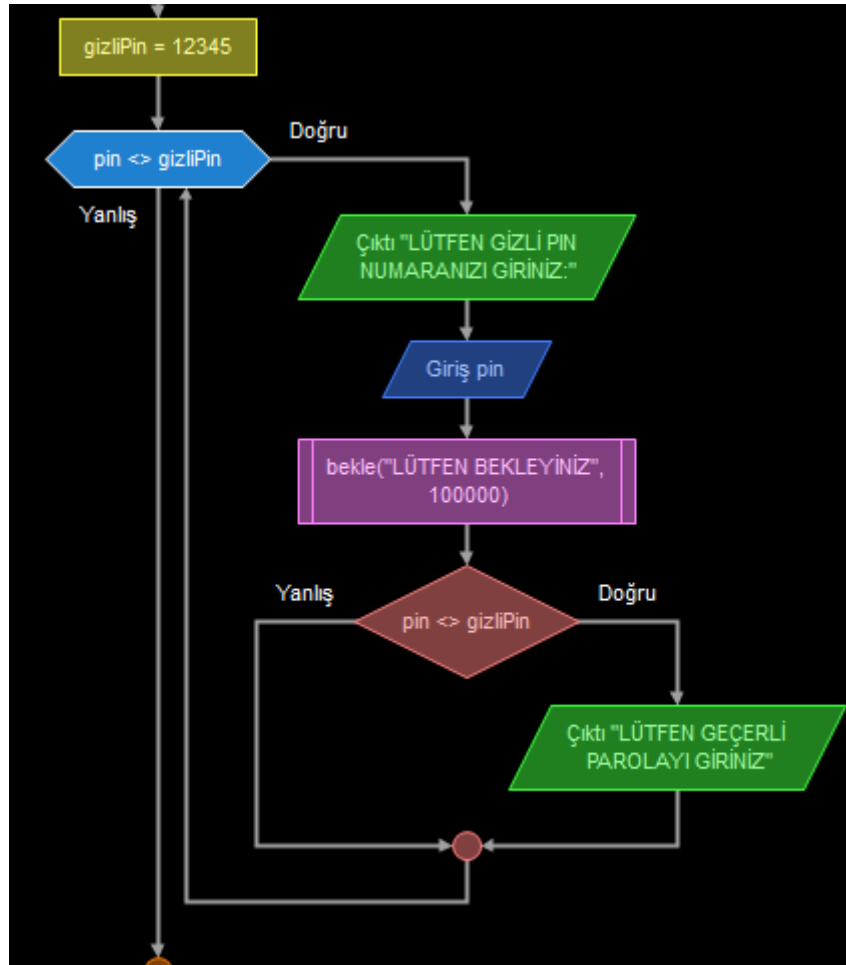
Kullanıcıyı yönlendirecek çıktılar, menu fonksiyonu olarak şekil 5-14 gibi tanımlanır.

Şekil 5-14 Bankamatik uygulaması Menu fonksiyon algoritması



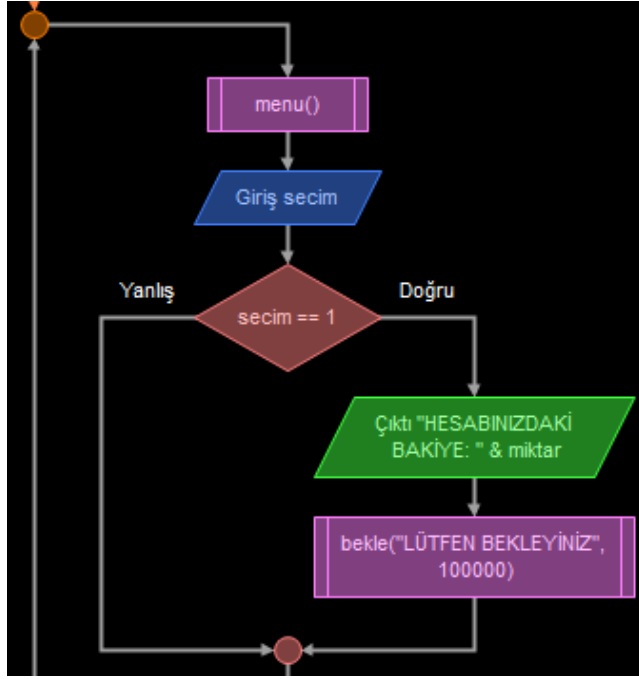
Bekle Fonksiyonu (Bekle()): Belirli bir süre beklemek amacıyla kullanılır. Ancak, fonksiyon içindeki döngü boş olduğu için aslında herhangi bir işlem gerçekleştirmez. Oluşan şema Şekil 5-15'teki gibi olmaktadır.

Şekil 5-15 Bankamatik uygulaması bekle fonksiyonu akış şeması



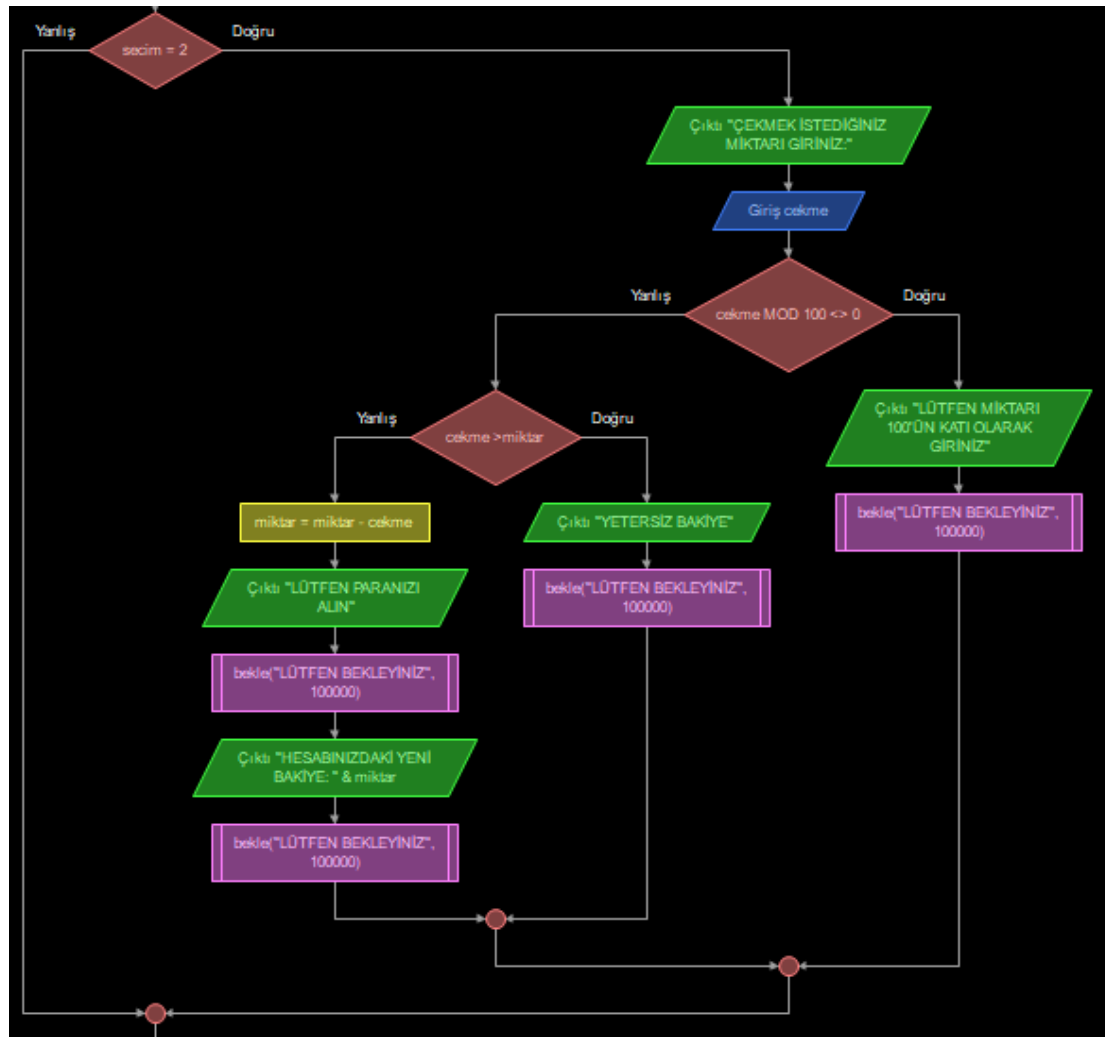
Şekil 5-16 Bankamatik uygulaması Ana Algoritma 2. Kısım

Bu noktadan Şekil (5-16) itibaren gizliPin değişkeni 12345 olarak atanır ve pin, kullanıcıdan alınan pin kodu ile sisteme atadığımız gizliPin değerine eşit olduğu sürece devam edecek bir while döngüsü başlatılır.



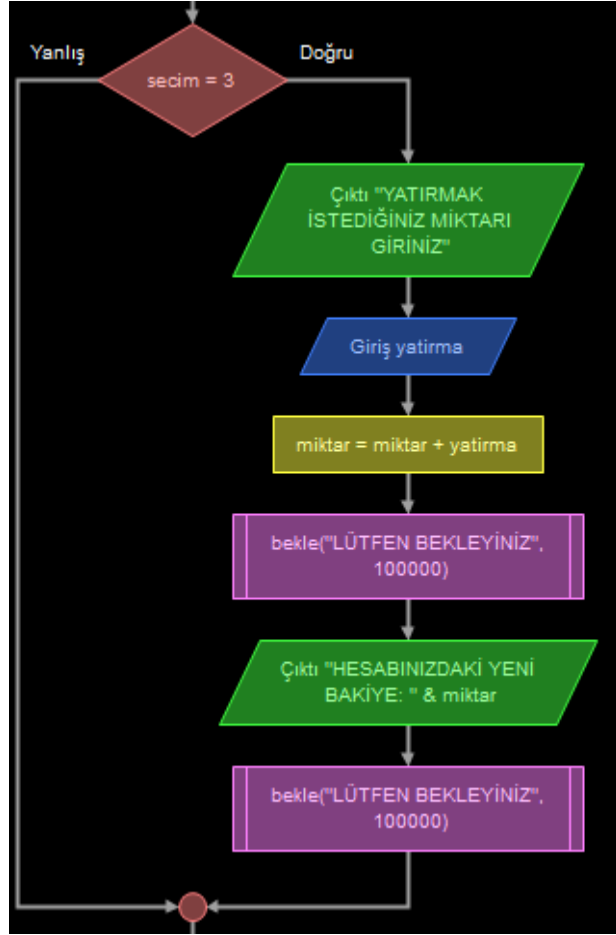
menu fonksiyonu ile (Şekil 5-17) seçenekler listelenir ve kullanıcıdan bir seçim yapması istenir. 1. seçim, kullanıcının hesabındaki bakiyeyi gösterir.

Şekil 5-17 Bankamatik uygulaması Ana Algoritma 3. Kısım



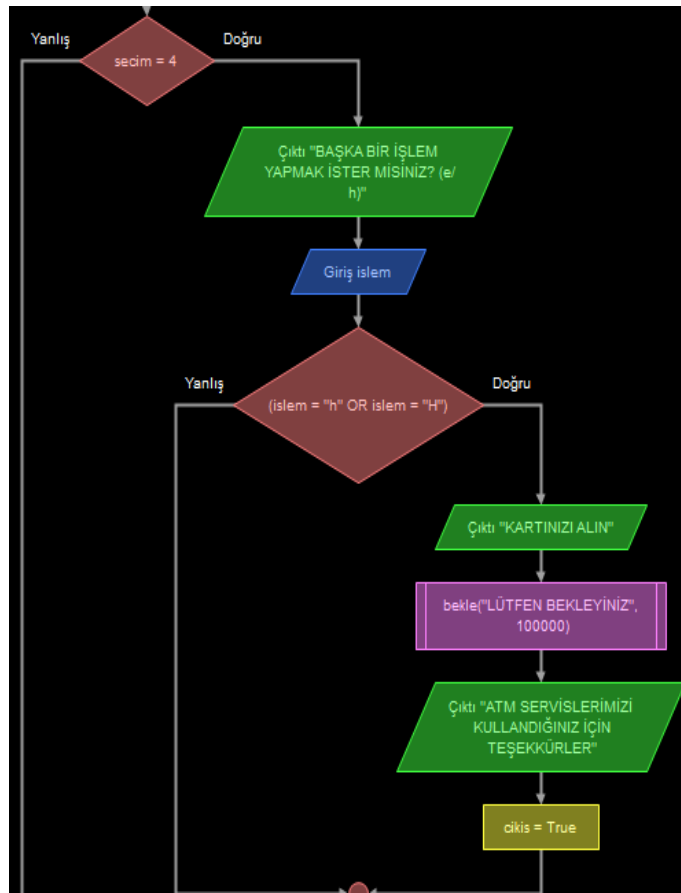
Şekil 5-18 Bankamatik uygulaması Ana Algoritma 4. Kısım

2. seçim, kullanıcının para çekebileceği seçenektir. Burada (Şekil 5-18) ilk olarak, çekmek istediği miktarın 100'ün katı olması için çekme değişkeninin 100 ile kalansız bölünebilmesi şartı konmuştur. Ardından, kullanıcının çekmek istediği miktarın miktar değişkenindeki bakiyeden fazla olması durumunda 'yetersiz bakiye' uyarısı verilir; aksi halde, kullanıcı parasını çekebilir.



3. seçim, kullanıcının yatırdığı parayı anaparaya, yani miktar değişkenine ekleyen seçenek Şekil 5-19'da gösterilmiştir.

Şekil 5-19 Bankamatik uygulaması Ana Algoritma 5. Kısım



4. seçim (Şekil 5-20), while döngüsünü sonlandırmak için kullanıcıya sunulur; bu, cikis değişkenini True yapmak amacıyla yapılır. Kullanıcı 'h' harfine bastığında döngü sona erer ve programdan çıkılır.

Şekil 5-20 Bankamatik uygulaması Ana Algoritma 6. Kısım



Flowgorithm dosyasına ulaşmak için kodu okutunuz

6 Bölüm

6.1 Grafik Şekilleri

6.1.1 Şekli Temizle

Bu sembol (Şekil 6-1), grafik penceresi için kullanılır. Arka planı temizler ve imleci başlangıç noktasına (Ana Sayfa konumuna) geri döndürür.



Şekil 6-1 Temizle

6.1.2 İleri

İleri sembolü (Şekil 6-2) grafik penceresiyle birlikte kullanılır. Bir kalem görevi gören ‘kaplumbağayı’ (imleci) belirli bir mesafe ileri hareket ettirir. Eğer ‘Kalem Aşağı’ seçili ise kaplumbağa ilerlerken bir çizgi çizer; seçili değilse kaplumbağa herhangi bir iz bırakmadan hareket eder.



Şekil 6-2 İleri

6.1.3 Ev

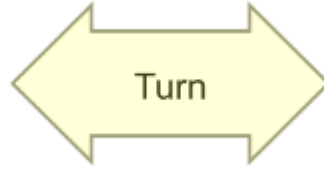
Ev şekli (Şekil 6-3), Turtle Graphics penceresinde kullanılır ve kaplumbağayı (imleci) başlangıç konumuna, yani ‘Ev’ noktasına geri götürür.



Şekil 6-3 Ev

6.1.4 Dönüş

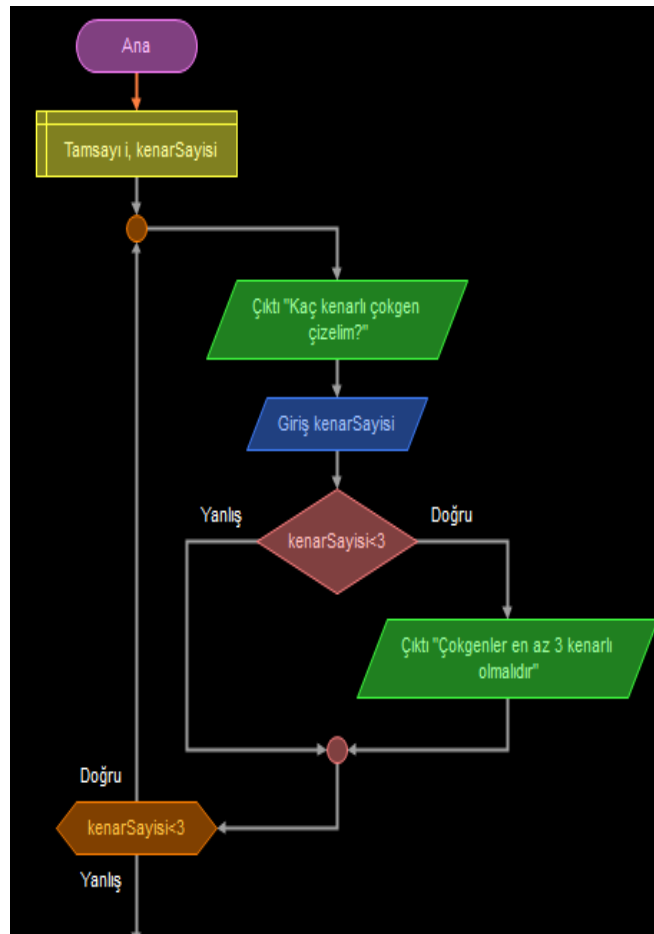
Bu sembol (Şekil 6-4), grafik ekranında kaplumbağanın (imlecin) kaç derece ve hangi yönde döneceğini belirtir.



Şekil 6-4 Dönüş

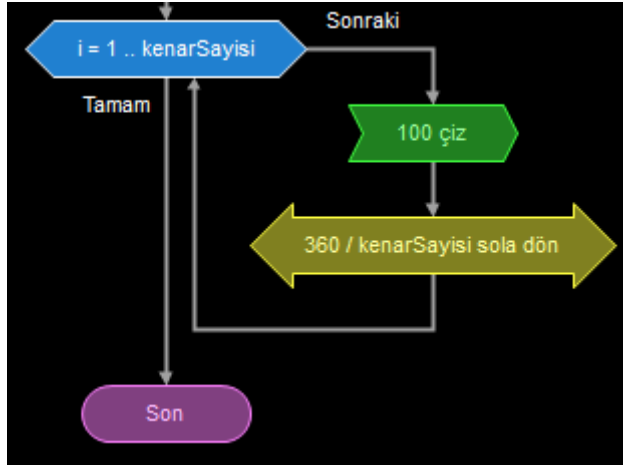
6.1.5 Çokgen Çizme Uygulaması

Çokgen çizerken kenar sayısı değişse de 'ilerle ve dön' algoritması hep aynıdır; fonksiyonlar işte bu ortak mantığı tek bir kalıba dökmemizi sağlar. Şekil 6-5 ile verilen şema ile uygulamamızı oluşturmaya başlayabiliriz.



Şekil 6-5 Çokgen çizme uygulaması akış şeması ana algoritma

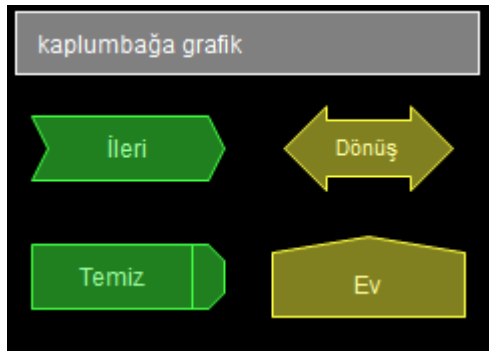
Öncelikle tamsayı türünde i ve $kenarSayisi$ değişkenlerini tanımladık. Ardından bir DO döngüsü kullanarak, kullanıcı $kenarSayisi$ değişkenine 3'ten küçük bir değer girdiğinde yeniden giriş yapmasını sağladık. Kullanıcı $kenarSayisi$ değerini 3 veya daha büyük girdiğinde DO döngüsü sona ermektedir.



Şekil 6-6 Çokgen çizme uygulaması for döngüsü

Yanda gösterildiği gibi (Şekil 6-6) bir for döngüsü tanımladık. Bu döngü, 1'den başlayarak kullanıcının girdiği kenar sayısına kadar çalışacaktır. Şimdi, for döngüsünün içerisinde yapılan işlemleri inceleyelim.

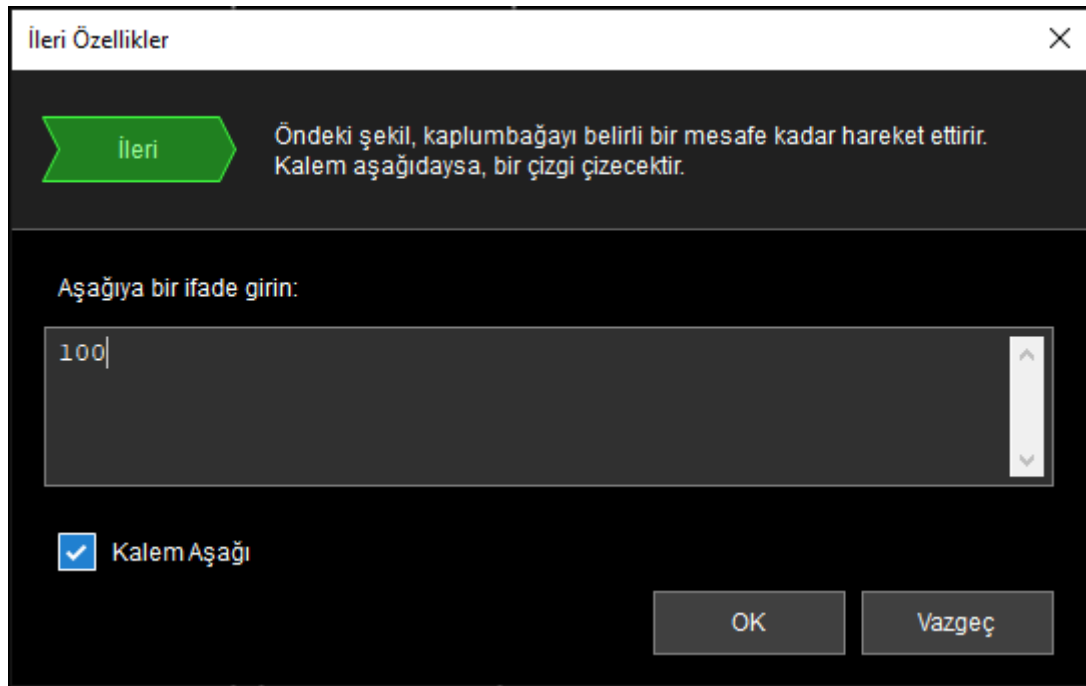
Akış şemasındaki ok simgesine tıkladığımızda, Kaplumbağa Grafik Penceresi bölümünde yer alan *İleri* sembolünü seçip şemamıza ekliyoruz.



Şekil 6-7 Kaplumbağa grafik ekleme ekranı

Şemaya eklediğimiz *İleri* sembolüne (Şekil 6-7) çift tıkladığımızda, kaplumbağanın kaç birim ilerleyeceğini belirleyebileceğimiz ayar penceresi açılır.

Aşağı bir ifade girin” kısmına 100 yazdık. For döngüsünün her adımında imleç 100 birim ileri hareket edecek ve *Kalem Aşağı* seçili olduğu için ilerlerken çizim yapacaktır. Eğer *Kalem Aşağı* seçili değilse imleç yine ilerler, ancak bu ilerleme sırasında herhangi bir çizim oluşmaz.



İleri Özellikler

İleri Öndeki şekil, kaplumbağayı belirli bir mesafe kadar hareket ettirir. Kalem aşağıdaysa, bir çizgi çezecektir.

Aşağıya bir ifade girin:

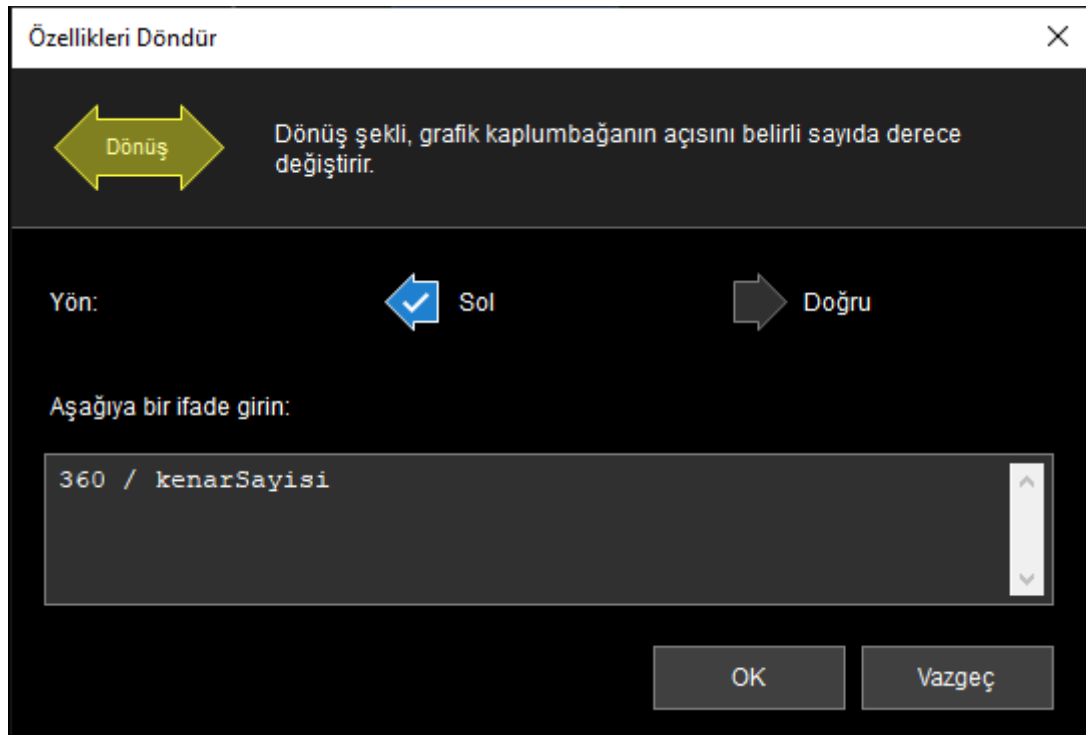
100

☒ Kalem Aşağı

OK Vazgeç

Şekil 6-8 İleri şeklini ekleme ekranı

Sonrasında, kaplumbağa grafik sekmelerinin altındaki dönüş sembolünü (Şekil 6-8) şemamıza ekliyoruz. Bu sembole çift tıkladığımızda aşağıdaki pencere (Şekil 6-9) açılmaktadır.



Özellikleri Döndür

Dönüş Dönüş şekli, grafik kaplumbağanın açısını belirli sayıda derece değiştirir.

Yön: ☒ Sol ☐ Doğru

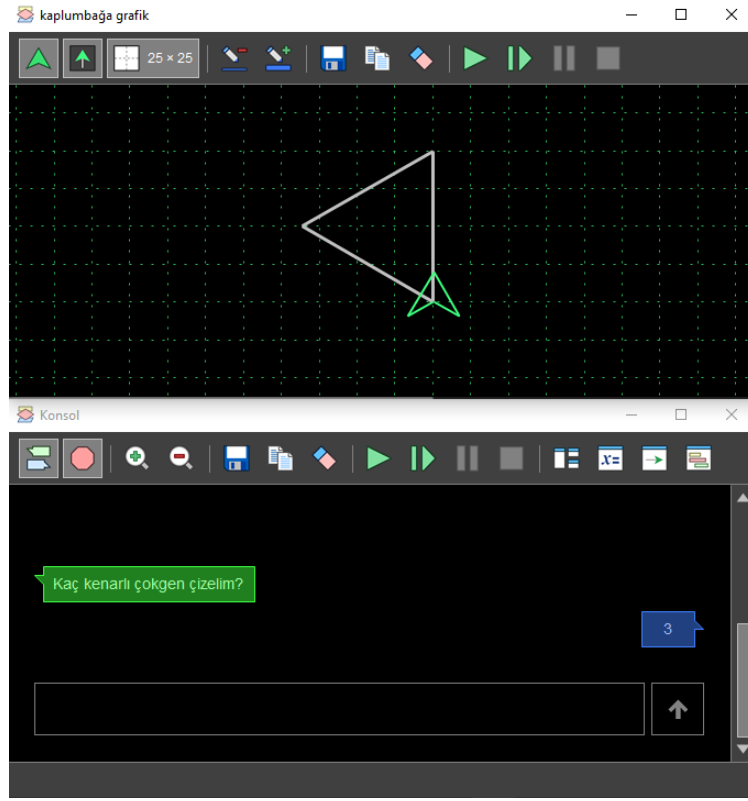
Aşağıya bir ifade girin:

360 / kenarSayısı

OK Vazgeç

Şekil 6-9 Dönüş şeklini ekleme ekranı

Bu pencere, imlecin hangi yöne döneceğini ve dönüş açısını belirlediğimiz ekrandır. Yön olarak sol, açı olarak ise çokgenlerde kullanılan dış açı formülü olan $360/n$ kuralı seçilmiştir. Kullanıcı kaç kenar girecekse, 360 sayısını o kenar sayısına bölerek dış açıyı elde etmiş oluyoruz. Örneğin kullanıcı üç kenarlı bir çokgen seçerse dış açı 120 derece olacaktır. Böylece her döngüde imleç 120 derece dönerek Şekil 6-10'daki gibi bir üçgen oluşturur.



Şekil 6-10 Programın çıktısı



Flowgorithm dosyasına ulaşmak için kodu okutunuz

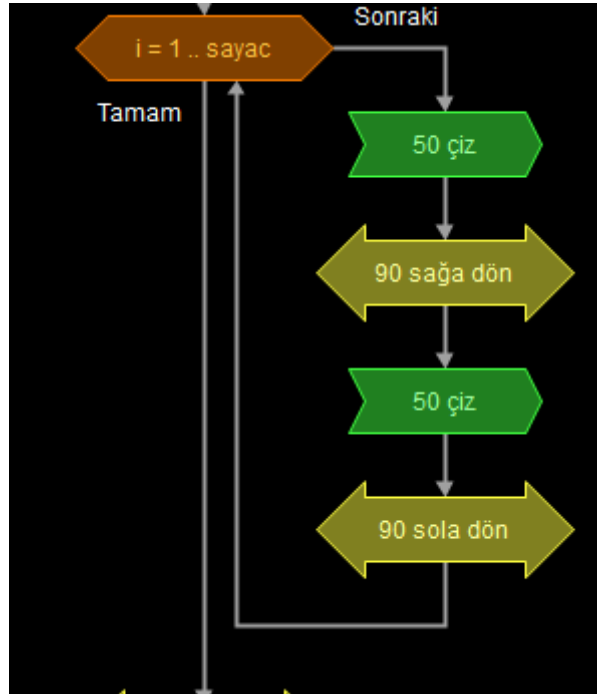
6.1.6 Merdiven çizme uygulaması

Karmaşık görünen bir merdiven yapısı, aslında basit bir 'Basamak Çiz' fonksiyonunun belirli bir düzende tekrar tekrar çağrılmasından ibarettir. Şekil 6-11 ile verilen akış şemasın oluşturarak fonksiyonumuza başlayalım.



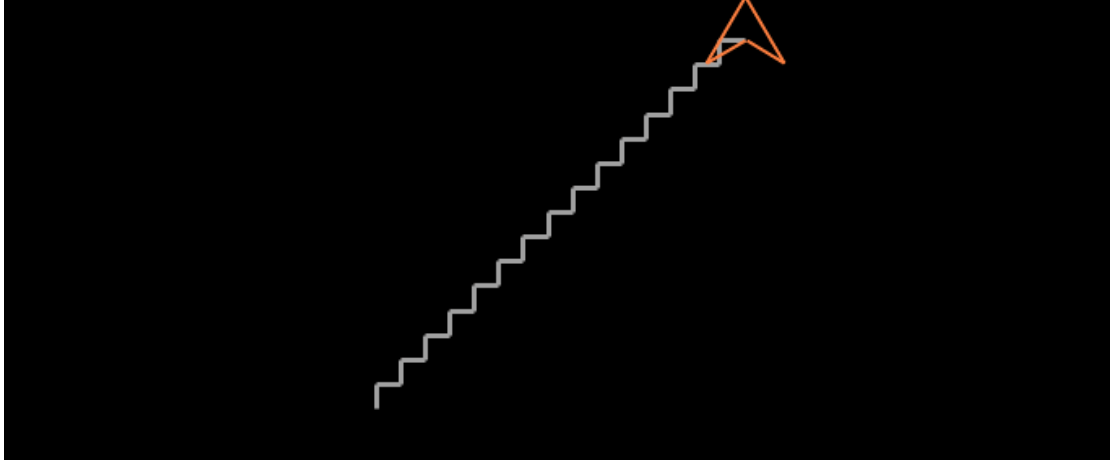
İhtiyacımız olan *sayac* ve *i* değişkenlerini tanımladık. *sayac* değişkeni, merdivenin kaç basamaklı olacağını kullanıcıdan almak için kullanılacak olan değişkendir.

Şekil 6-11 Ana fonksiyon 1. kısım

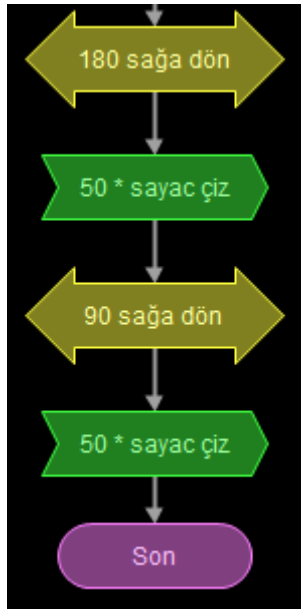


Ardından bir for döngüsü (Şekil 6-12) oluşturuyoruz ve bu döngü 1'den başlayıp *sayac* değişkeninin değerine kadar devam ediyor. *Kalem Aşağı* seçeneği işaretliyken, imlecin 50 birim çizip 90 derece sağa dönmesini; ardından tekrar 50 birim çizip bu kez 90 derece sola dönmesini sağlayan bir döngü tanımladık. Bu işlem 15 kez tekrarlandığında Şekil 6-13 elde edilmektedir.

Şekil 6-12 Ana fonksiyon 2. kısım

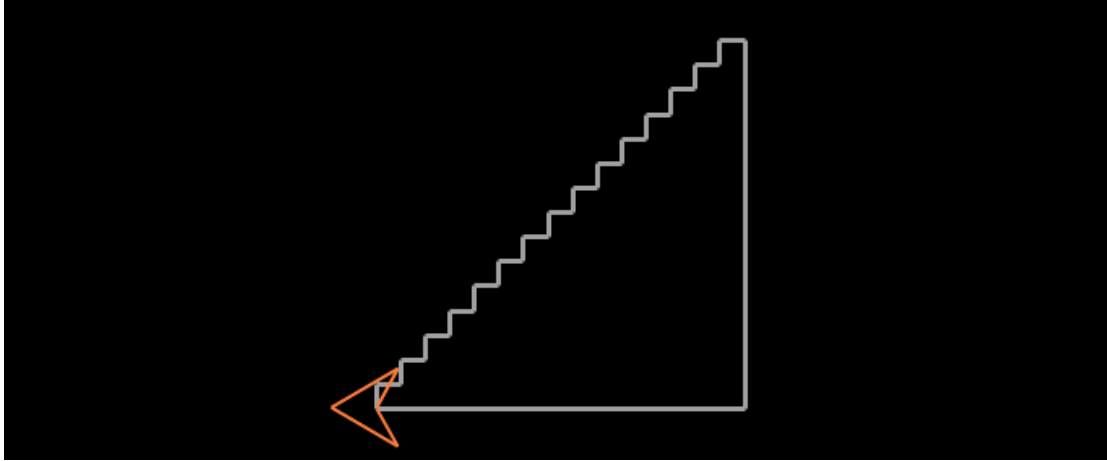


Şekil 6-13 Programın bu kısma kadar olan çıktısı



Son olarak grafiği tamamlamak için imleci 180 derece sağa döndürdük ve merdivenin her basamağı 50 birim olduğundan, yukarıya doğru toplamda $\text{sayaç} \times 50$ birim ilerlettik. Aynı şekilde, merdiven 15 basamak için sağa doğru da 15×50 birim kaydığı için imleci geri konumlandırmak adına önce 90 derece sağa döndürdük ve ardından tekrar $\text{sayaç} \times 50$ birim ileri gönderdik (Şekil 6-14). Bu işlemler tamamlandığında merdiven şekli oluşturulmuş olur. Son durumda grafiğimiz Şekil 6-15'teki gibi görünmektedir.

Şekil 6-14 Ana program 3. kısım



Şekil 6-15 Programın son halinin çıktısı



Flowgorithm dosyasına ulaşmak için
kodu okutunuz

6.1.7 Fraktal Çizdirme Uygulaması

Fraktallar, parçanın bütüne benzediği şekillerdir; biz de fonksiyonumuza 'bir dal çiz ve ucuna daha küçük iki dal daha çizmesi için kendini tekrar çağır' diyerek bu büyüleyici desenleri oluştururuz.

1. Üçgen adında bir fonksiyon tanımlanır.

Bu fonksiyon, fraktal üçgenin oluşturulmasından sorumludur. İki parametre alır:

uzunluk: Üçgenin kenar uzunluğu

derinlik: Fraktalın kaçınıcı derinlik seviyesine kadar oluşturulacağı

2. **İf(derinlik>0)** koşulu uygulanır.

Bu koşul, derinlik değerinin 0'dan büyük olup olmadığını kontrol eder.

Derinlik 0'dan büyükse fraktal üçgenin oluşturulması devam eder. Koşulun içinde bir for döngüsü tanımlanır.

3. i değişkeni 1'den başlayarak 3'e kadar artan bir döngü kuruludur. Bu döngü, üçgenin üç kenarı için yapılacak işlemleri tekrarlar. Bir üçgen üç kenardan oluştuğu için döngü 3 kez çalışır.
4. uzunluk / 3 kadar çiz:
Üçgenin her bir kenarı için, kenar uzunluğunun üçte biri kadar çizgi çizilir.
5. $Ucgen(uzunluk / 3, derinlik - 1)$ şeklinde fonksiyon kendini yeniden çağırır. Bu çağrıda kenar uzunluğu üçte bir düşürülür, derinlik ise bir azaltılır. Bu işlem, her seviyede üçgenin küçülerek daha detaylı hale gelmesini sağlar.
6. Yeniden uzunluk / 3 kadar çizilir.
Böylece üçgenin her kenarı için toplam üç segment çizilmiş olur.
7. 120 derece sağa dön:
Üç kenardan her biri çizildikten sonra imleç 120 derece sağa döner. Bu dönüş üçgenin yapısını oluşturur.

Ana Fonksiyonun Tanımlanması

1. Öncelikle derinlik isimli bir tamsayı değişkeni tanımlanır. Kullanıcıdan bu değişkeni 1 ile 7 arasında bir değer girmesi istenir.
2. $Ucgen$ fonksiyonu $Ucgen(yükseklik, derinlik)$ şeklinde çağrılır. Burada yükseklik 200 olarak belirlenmiştir. Yani fonksiyon şu şekilde çalıştırılır:
 $Ucgen(200, derinlik)$
3. Belirlenen derinlikte oluşturulmuş fraktal üçgen ekranda Şekil 6-16'daki gibi görüntülenir.



Şekil 6-16 Fraktal örneği

https://drive.google.com/file/d/1QHlleZxLW2dJkQ4Y2fX0S8XLtr_V-49V/view?usp=drive_link linkini tıklayarak uygulamanın pdf dosyasına ulaşabilirsiniz.



Flowgorithm dosyasına ulaşmak için
kodu okutunuz

7 Bölüm

7.1 Dosya I/O Şekilleri

7.1.1 Açık

Belirli bir dosyanın okunmasına veya yazılmasına izin vermek için Şekil 7-1 kullanılır. Dosya adı tırnak işaretleri içerisinde belirtildikten sonra, alt sekmelerden dosya modu seçilerek okuma ya da yazma işlemi belirlenir.



Şekil 7-1 Açık

7.1.2 Kapalı

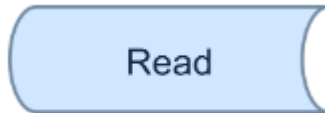
Kapatma ifadesi (Şekil 7-2), açık olan bir dosyanın okunması veya dosyaya yazılması işlemini sonlandırır. Program sona ermeden önce tüm dosyaların kapatılması gerekir.



Şekil 7-2 Kapalı

7.1.3 Okuma

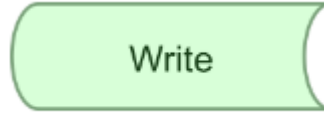
Bu sembol (Şekil 7-3), açılan dosyadan bir değeri okur ve okunan bu ifadeyi bir değişkende saklar.



Şekil 7-3 Okuma

7.1.4 Yazma

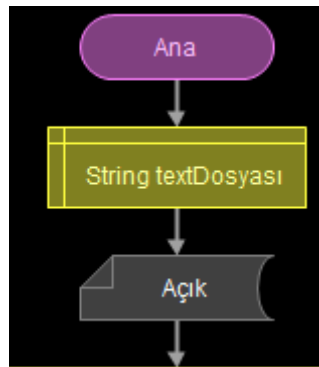
Bir ifadeyi değerlendirir ve sonucunu açık olan dosyaya kaydetmek için verilen Şekil 7-4 kullanılır.



Şekil 7-4 Yazma

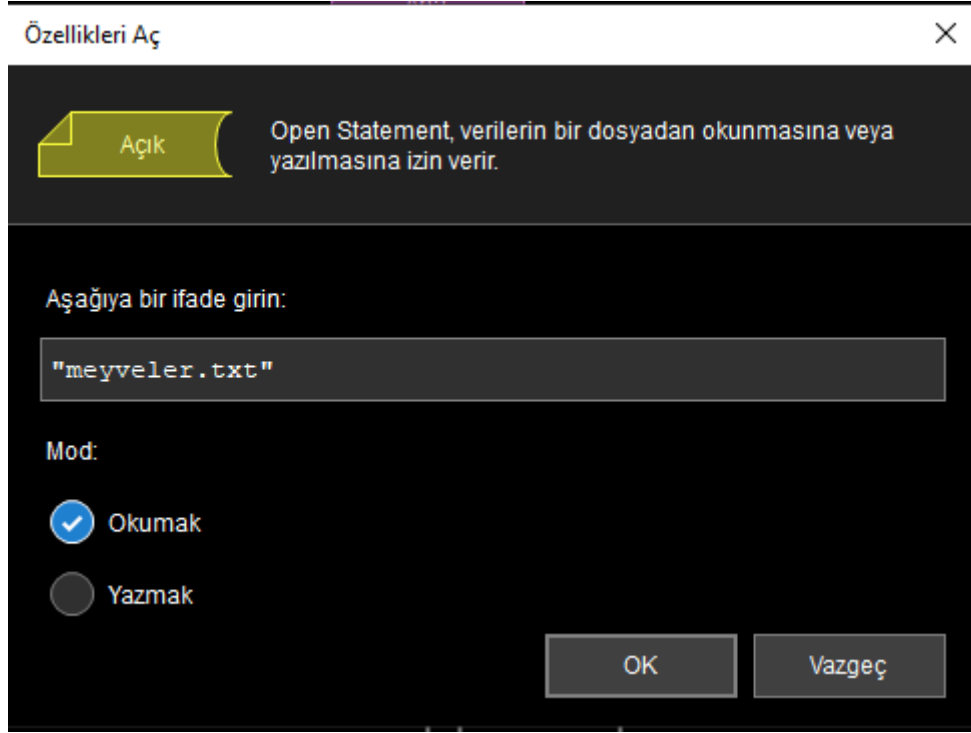
7.1.5 Dosya Okuma Örneği

İçinde meyve isimleri bulunan bir .txt uzantılı metin belgesi oluşturalım. Dosyamızın adı meyveler.txt olacak ve Flowgorithm dosyamız ile bu metin dosyasının aynı klasörde yer alması gerekir. Örneğin her iki dosyayı da masaüstünde tutabilirsiniz.



Sonrasında akış şemamıza (Şekil 7-5) bir String değişken tanımlamamız gerekir. Bu değişkenin adı *textDosyası* olsun. Bu tanımlamayı yaptıktan sonra şemamıza *Açık* ifadesini ekliyoruz. Üzerine çift tıkladığımızda ise aşağıdaki pencere açılacaktır.

Şekil 7-5 Ana algoritma 1. Kısım

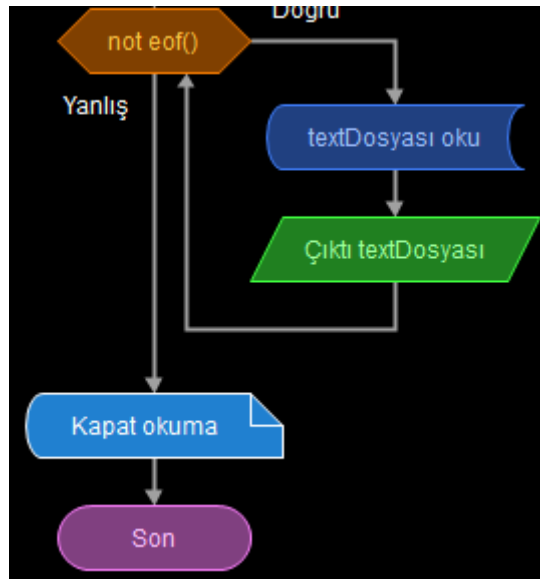


Şekil 7-6 Okumak istediğimiz dosyayı akış şemasına ekleme ekranı

Açılan bu pencerede (Şekil 7-6), tırnak işaretleri içerisine okumak istediğimiz dosyanın adını uzantısıyla birlikte yazıyoruz. Okuma işlemi yapacağımız için *Mod* bölümünden “Okumak” seçeneğini işaretliyoruz ve ardından *OK* sekmesine tıklayarak akış şemasına geri dönüyoruz.

Bu adımla birlikte meyveler.txt dosyasını okumak üzere açmış olduk. Şimdi, bu dosyanın içindeki verileri daha önce tanımladığımız *String textDosyası* değişkenine aktarmamız gerekiyor. Bunun için akış şemamıza bir *while* döngüsü ekleyelim.

While döngüsünde şart olarak *not eof()* özel fonksiyonunu kullanalım. Bu fonksiyon, dosyanın sonuna ulaşılan kadar metin dosyası içindeki verilere erişmemizi sağlar.

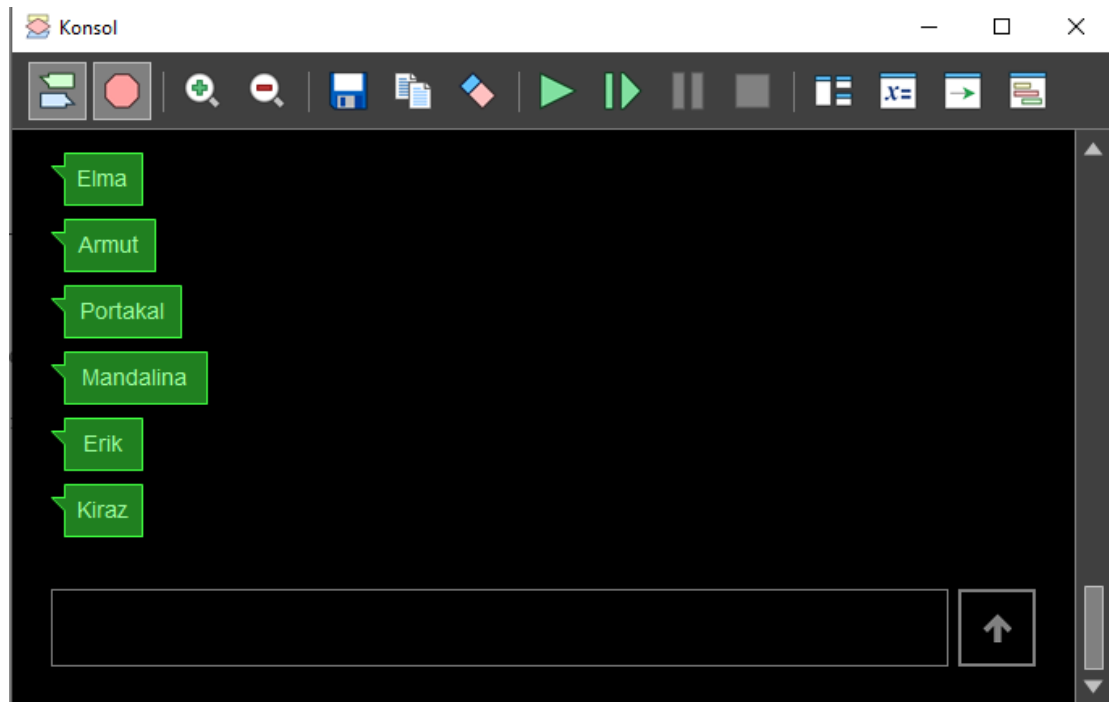


Şekil 7-7 Dosya okuma eklendikten sonra akış şemasındaki görünüm

Artık *textDosyası* değişkeninde bulunan verileri okuyarak çıktı olarak alabiliriz. Şemada da görüldüğü gibi (Şekil 7-7), while döngümüzün içine *Okumak* ifadesini ekliyoruz. Bu ifadeye çift tıkladığımızda açılan pencereye, okumak istediğimiz String değişkeninin adını yani *textDosyası* yazıp OK sekmesine tıklıyoruz.

Verileri konsol ekranına yazdırmak için ise *textDosyası* değişkenini çıktı olarak while döngüsünün içine ekliyoruz. Böylece dosyada yer alan her satır, döngü ilerledikçe ekranda görüntülenmiş olur.

Son olarak, Flowgorithm'de açılan her dosyanın kapatılması zorunludur. Bu nedenle akış şemamıza bir adet Kapat ifadesi ekliyoruz. Bu ifadeye çift tıklayıp açılan pencerede Mod: Okumak seçeneğini işaretleyerek OK sekmesine tıklıyoruz. Böylece program tamamlanmış olur. Program çalıştırıldığında konsol ekranı aşağıdaki gibi (Şekil 7-8) görünmektedir.



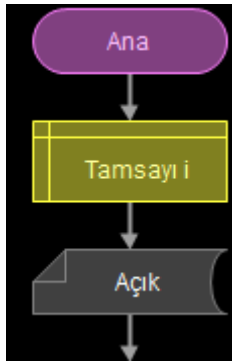
Şekil 7-8 Program çalıştırıldıktan sonra oluşan çıktı



Flowgorithm dosyasına ulaşmak için kodu okutunuz

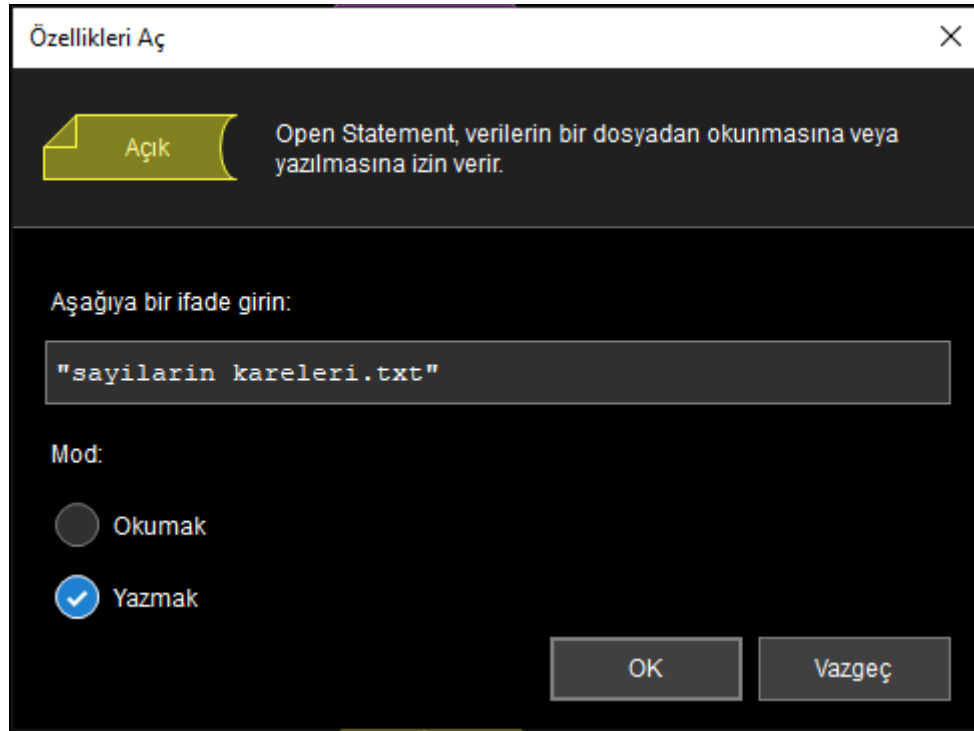
7.1.6 1'den 10'a kadar olan sayıların karelerini alıp txt dosyasına yazdıran program

Dosya işlemlerinin mantığını kavramak adına, döngüleri ve yazdırma komutlarını birleştirerek 1'den 10'a kadar olan sayıların karelerini harici bir dosyaya kaydeden şu örneği (Şekil 7-9) inceleyelim.



Akış şemamıza tamsayı türünde bir *i* değişkeni ekleyelim. Ardından, sayıların karelerini yazdırmak için kullanılacak olan .txt dosyasını açmamız gerekir. Bu dosya, program tarafından otomatik olarak oluşturulacaktır. Bu nedenle şemamıza *Açık* ifadesini ekleyelim ve üzerine çift tıklayarak aşağıdaki pencereyi açalım.

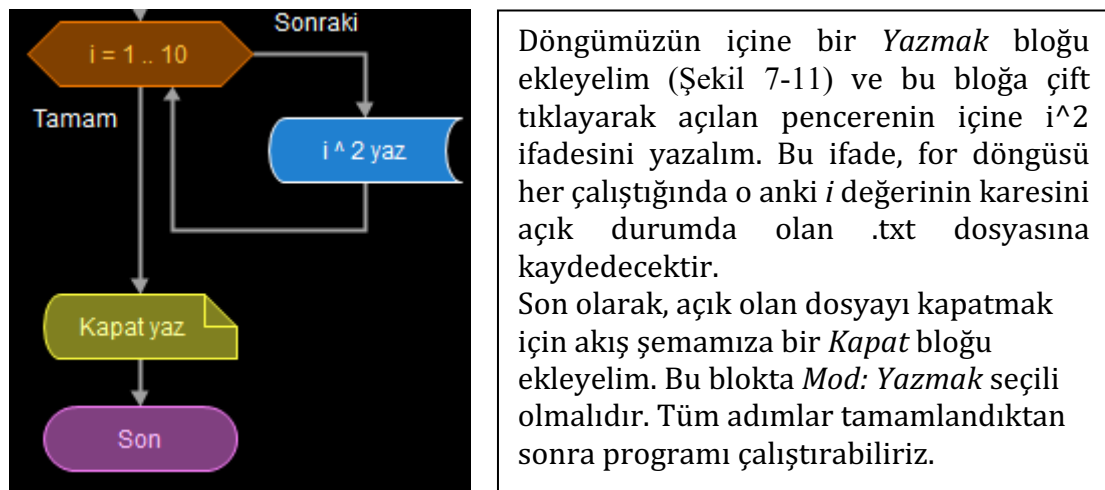
Şekil 7-9 Akış şemasına açık şekli ekleme ekranı



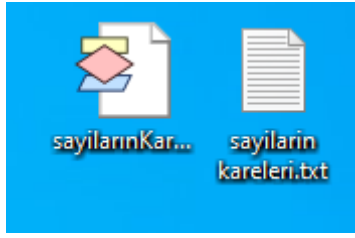
Şekil 7-10 Okumak istediğimiz dosyayı belirleme ekranı

Açılan pencerede (Şekil 7-10), oluşturmak istediğimiz dosyanın adını tırnak işaretleri içinde yazalım. Dosyamıza “sayıların kareleri.txt” şeklinde bir isim verdikten sonra, dosyaya veri yazacağımız için *Mod* bölümünden “Yazmak” seçeneğini işaretleyelim ve *OK* sekmesine tıklayalım.

Ardından akış şemamıza bir *for* döngüsü ekleyelim. Bu döngü, *i* değişkeni için 1’den başlasın ve 10 değerine kadar çalışsın.



Şekil 7-11 Akış şemasına dosya içinde yapılması istenen şekillerin ekleme ekranı



Program çalıştıktan sonra, Flowgorithm dosyamızın bulunduğu konumda “sayıların kareleri.txt” adlı bir metin belgesi oluşur (Şekil 6-12). Bu dosyanın içinde, 1’den 10’a kadar olan sayıların kareleri alt alta gelecek şekilde yazılmış olarak bulunur.

Şekil 7-12 Program çalıştıktan sonra oluşan txt dosyası



Flowgorithm dosyasına ulaşmak için kodu okutunuz

8 Bölüm

8.1 Farklı Yazılım Dillerinde Çıktı Alma

Flowgorithm’de oluşturduğumuz akış şemalarını farklı yazılım dillerinde çıktı olarak alabiliriz. Aşağıdaki Şekil 8-1’de, çıktı olarak dönüştürebileceğimiz yazılım dilleri gösterilmektedir.

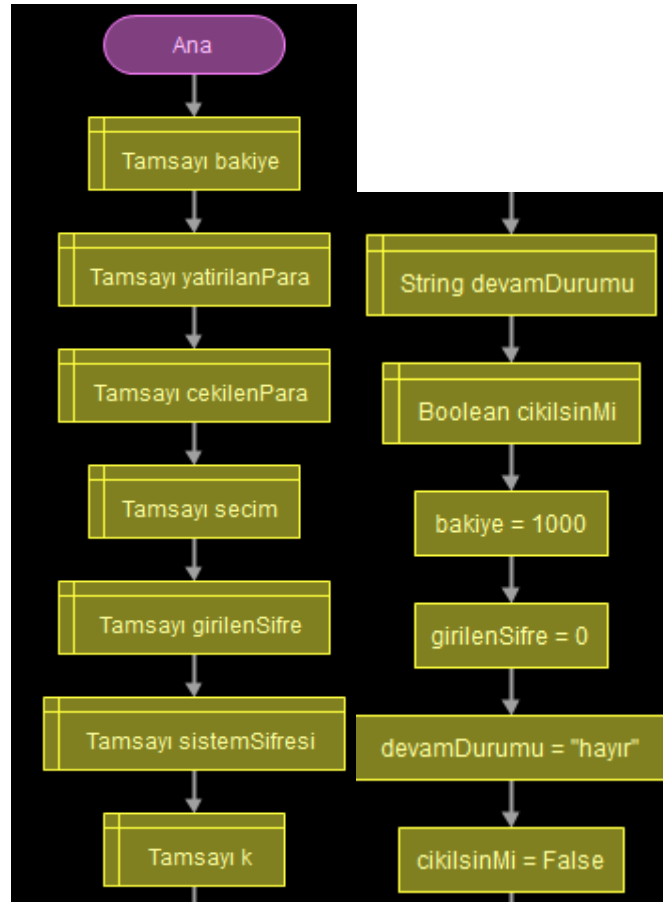


Şekil 8-1 Flowgorithm ile akış şemalarını dönüştürebileceğimiz diller

8.2 Genel Örnek Uygulamalar

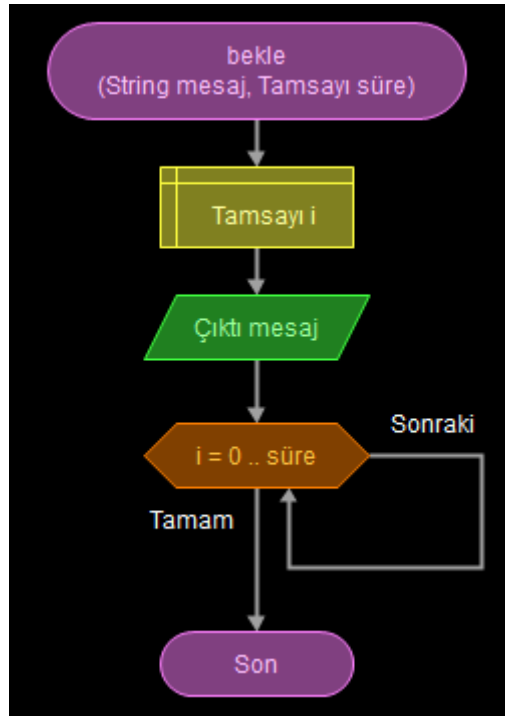
8.2.1 ATM Simülasyonu

Sınıflar (Classes) ve Nesneler (Objects) arasındaki ilişkiyi anlamamanın en iyi yollarından biri, her müşterinin ve hesabın ayrı birer nesne olarak ele alındığı bir bankamatik sistemi tasarlamaktır. Şekil 8-2 de ihtiyacımız olan tüm değişkenler tanımlanmıştır. Bu değişkenlerin nasıl kullanılacağı ise programın ilerleyen aşamalarında açıklanacaktır.



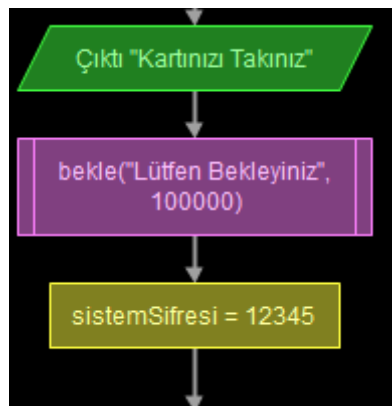
Şekil 8-2 Ana fonksiyonumuz 1. kısım

Şekil 8-2 de ihtiyacımız olan tüm değişkenler tanımlanmıştır. Bu değişkenlerin nasıl kullanılacağı ise programın ilerleyen aşamalarında açıklanacaktır.



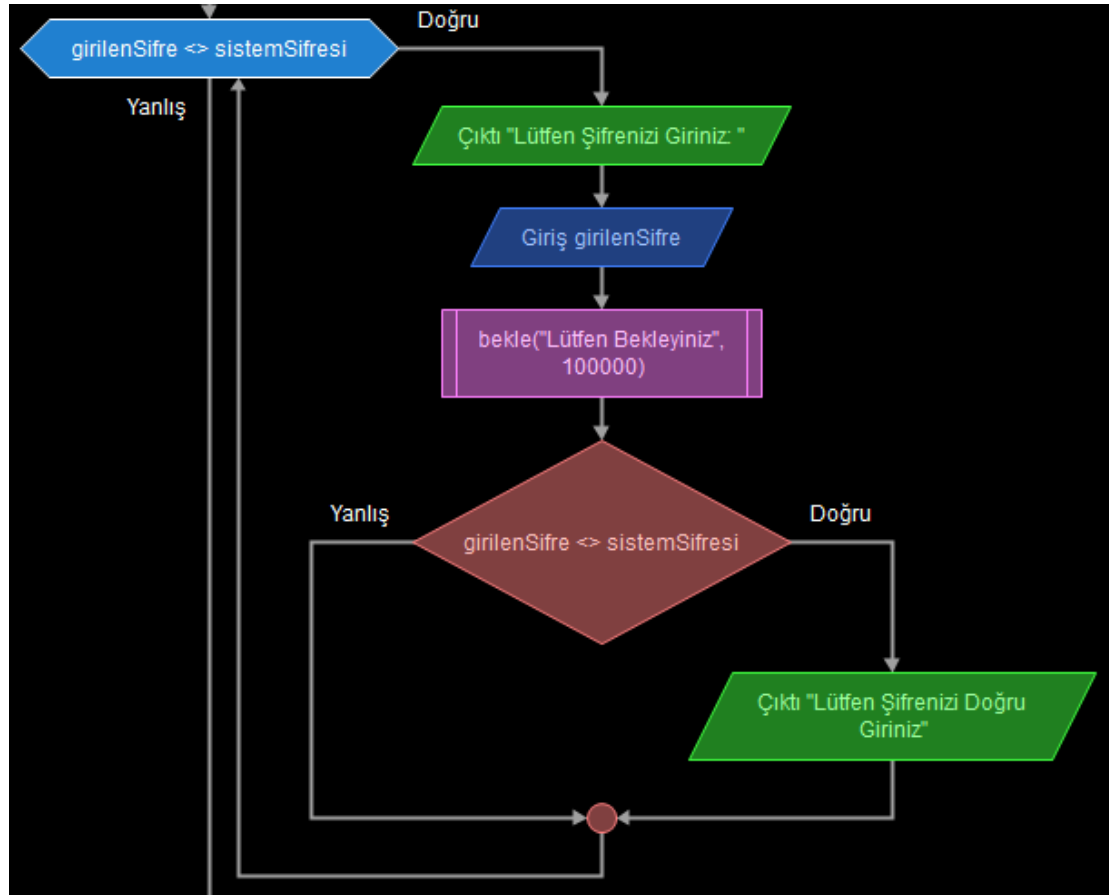
Ana programımıza başlamadan önce “bekle” şeklinde bir fonksiyon (Şekil 8-3) tanımladık. Bu fonksiyonun kullanılabilmesi için, mesaj ve süre bilgilerini girerek tanımlama yapılması gerekir. Fonksiyonu, programın farklı bölümleri arasında kısa bir bekleme süresi oluşturmak için kullanacağız. Fonksiyon çağrıldığında, süre değişkenine hangi değer atanmışsa, içindeki boş *for* döngüsü o değer kadar çalışacak ve program akışında belirli bir gecikme oluşacaktır.

Şekil 8-3 Bekle Fonksiyonumuz



Ana fonksiyonumuza geri döndük. Kullanıcıya, Şekil 8-4’te gösterildiği gibi “Kartınızı takınız” şeklinde bir çıktı verdik. Ardından “Lütfen bekleyiniz” ifadesinin konsol ekranında bir süre görünmesi için *bekle* fonksiyonunu, süre değişkeni 100000 olacak şekilde çağırdık. Fonksiyon içindeki *for* döngüsü 100000 kez boş döneceği için kısa bir bekleme süresi oluşacaktır. Ayrıca kullanıcının kart şifresi 12345 olarak belirlenmiştir.

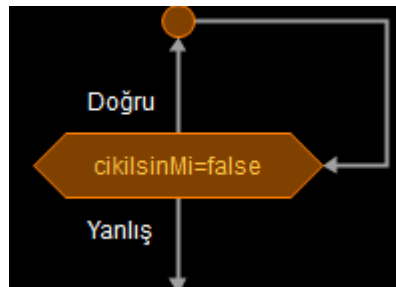
Şekil 8-4 Ana fonksiyon 2. kısım



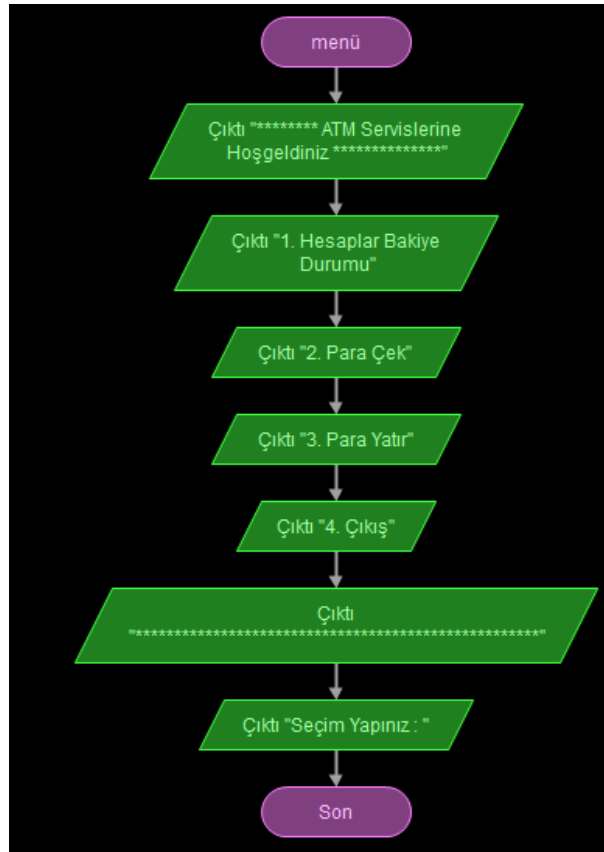
Şekil 8-5 Ana fonksiyon 3. kısım

Programın devamında bir while döngüsü (Şekil 8-5) tanımladık. Bu döngü içinde kullanıcıdan bir şifre girmesi istenir ve girilen değer girilenSifre değişkenine atanır. Kullanıcı, sistemSifresi ile aynı şifreyi girene kadar döngü devam eder. Her yanlış girişte kullanıcı "Lütfen şifrenizi doğru giriniz." şeklinde uyarılır. Şifre doğru girildiğinde ise program döngüden çıkar.

Değişkenleri tanımlarken Boolean türünde cikilsinMi = false şeklinde bir değişken oluşturmuştuk. Bu değişken, kullanıcı istemediği sürece programın kapanmamasını sağlamak için kullanılacaktır. Bu amaçla bir do döngüsü (Şekil 8-6) tanımlıyoruz ve bu adımdan sonra tüm bloklarımızı bu döngünün içine yerleştiriyoruz.

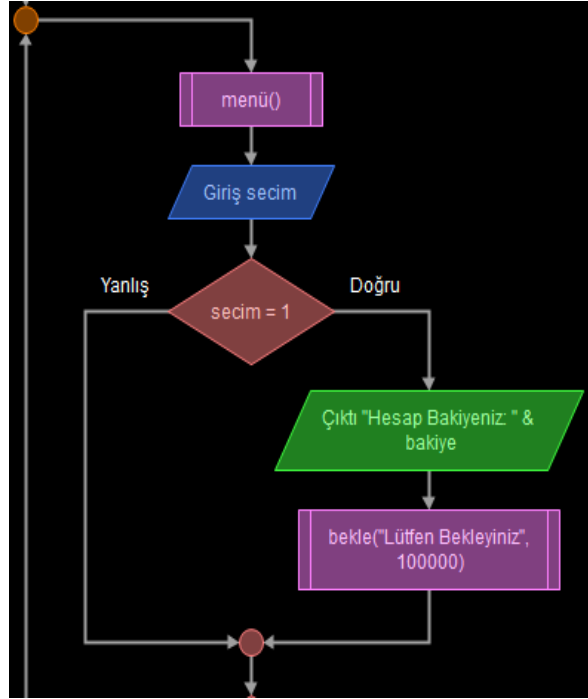


Şekil 8-6 Ana fonksiyon 4. kısım



Şekil 8-7 Menü fonksiyonu

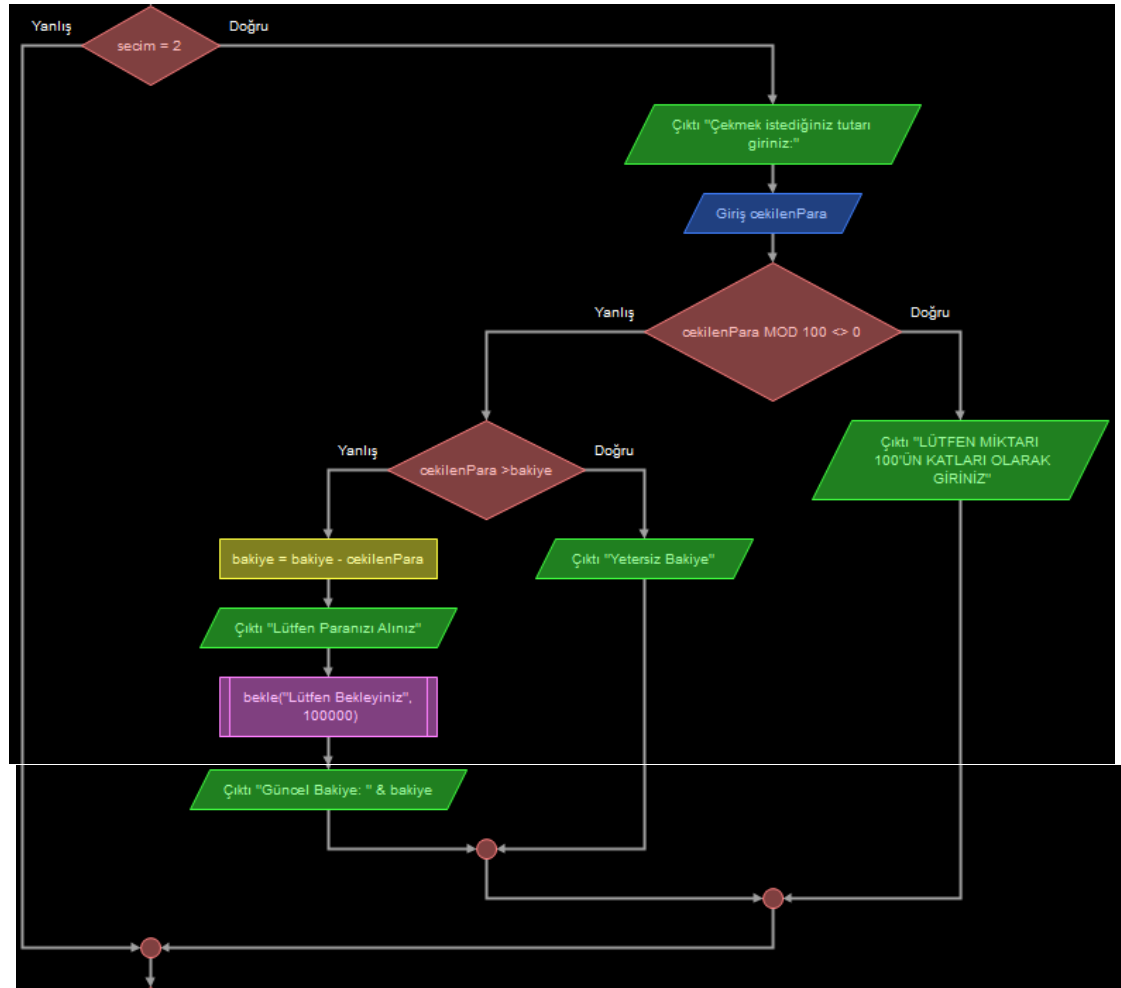
Şekil 8-7’de gösterildiği gibi menu adlı bir fonksiyon tanımlıyoruz. Bu fonksiyonun özel bir işlevi bulunmamaktadır; yalnızca ana fonksiyonun çok kalabalık görünmesini önlemek amacıyla menüyü ayrı bir fonksiyon olarak tasarladık.



Şekil 8-8 Ana fonksiyon 5. kısım

Ana fonksiyonumuza yeniden döndük ve *do* döngüsünün içerisinde *menu()* fonksiyonunu (Şekil 8-8) çağırdık. Kullanıcı programı çalıştırdığında, *menu* fonksiyonunun içindeki çıktılar konsol ekranında görüntülenecektir. Ardından kullanıcıya dört farklı seçenek sunulmuştur. Kullanıcıdan alınan değer *secim* değişkenine atanır ve bu dört durum sırayla değerlendirilir.

Eğer *secim = 1* ise, kullanıcı hesap bakiyesini görüntülemek istemiş olur. Şemada görüldüğü gibi, bu durumda hesap bakiyesi konsol ekranına yazdırılır ve kısa bir beklemenin ardından *do* döngüsü başa döner. Böylece menü yeniden ekrana yazdırılır ve kullanıcıdan tekrar bir seçim yapması istenir.



Şekil 8-9 Ana fonksiyon 6. kısım

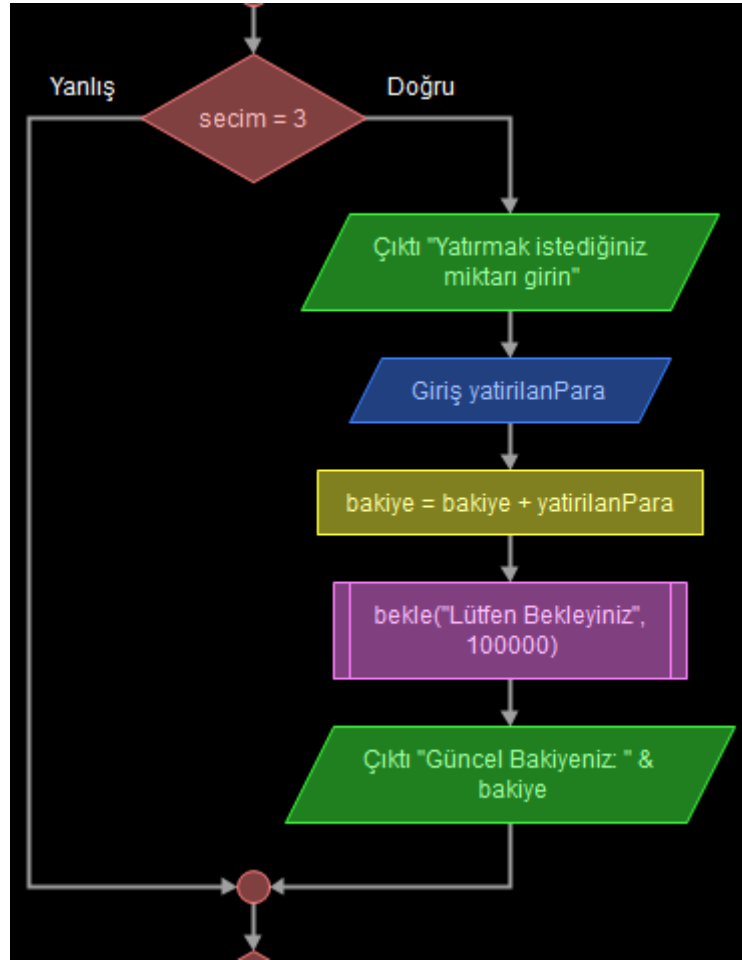
Kullanıcı secim değişkenine 2 değerini girdiğinde, konsol ekranında “*Lütfen çekmek istediğiniz tutarı giriniz.*” şeklinde bir bilgilendirme mesajı görüntülenir ve ardından kullanıcıdan cekilenPara değişkenine bir değer girmesi istenir.

Girilen tutarın işleme uygun olup olmadığı bir dizi koşulla kontrol edilir. İlk olarak, çekilmek istenen tutarın 100’ün katı olup olmadığı incelenir. Eğer cekilenPara % 100 != 0 ise, konsol ekranına “*Lütfen çektiğiniz tutar 100’ün katı olsun.*” uyarısı yazdırılır.

Tutar 100’ün katıysa, bu kez farklı bir koşulla çekilmek istenen miktarın mevcut bakiyeden büyük olup olmadığı kontrol edilir. Eğer cekilenPara > bakiye ise kullanıcıya “*Yetersiz bakiye.*” uyarısı verilir.

Bu iki koşul sağlanmıyorsa, işlem başarıyla gerçekleştirilir. Bakiye, bakiye = bakiye- cekilenPara şeklinde güncellenir; kullanıcıya kartını alması gerektiği bildirilir ve kısa bir bekleme süresinin ardından güncel bakiye konsol ekranına yazdırılır. Oluşan algoritma Şekil 8-9’daki gibi olmaktadır.

Tüm bu işlemler tamamlandıktan sonra do döngüsü başa döner ve menü fonksiyonu yeniden çağrılarak kullanıcıdan tekrar bir seçim yapması Şekil 8-10'daki gibi istenir.



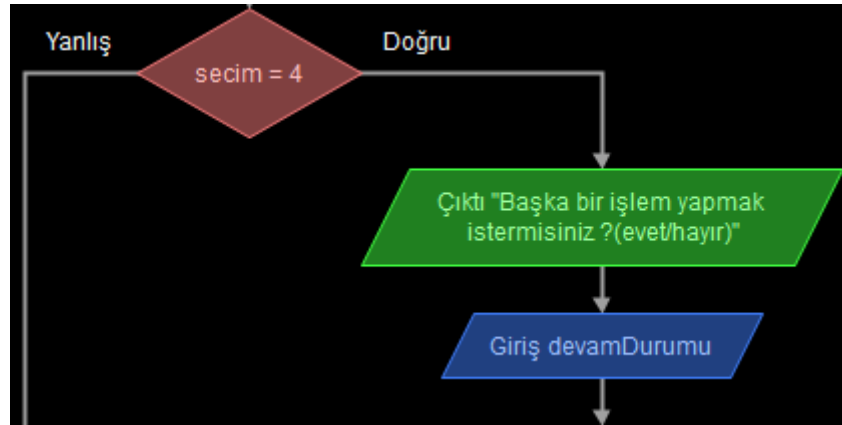
Şekil 8-10 Ana fonksiyon 7. kısım

Secim değişkeninin değeri 3 olarak girildiğinde, kullanıcı konsol ekranında para yatırma işlemi için yönlendirilir ve “Lütfen yatırmak istediğiniz tutarı giriniz.” şeklinde bir mesaj görüntülenir. Kullanıcının girdiği değer yatırılanPara değişkenine atanır.

Bu değer alınmasının ardından, mevcut bakiye $bakiye = bakiye + yatırılanPara$ ifadesiyle güncellenir ve böylece yeni bakiye miktarı elde edilmiş olur. Ardından

bekle fonksiyonunun çalışmasıyla kısa bir duraklama sağlanır ve kullanıcıya güncel bakiye bilgisi konsol ekranında gösterilir.

Tüm bu işlemler tamamlandıktan sonra do döngüsü tekrar başa döner ve menü fonksiyonu yeniden çalıştırılarak kullanıcıdan tekrar bir seçim yapması Şekil 8-11'deki gibi istenir.

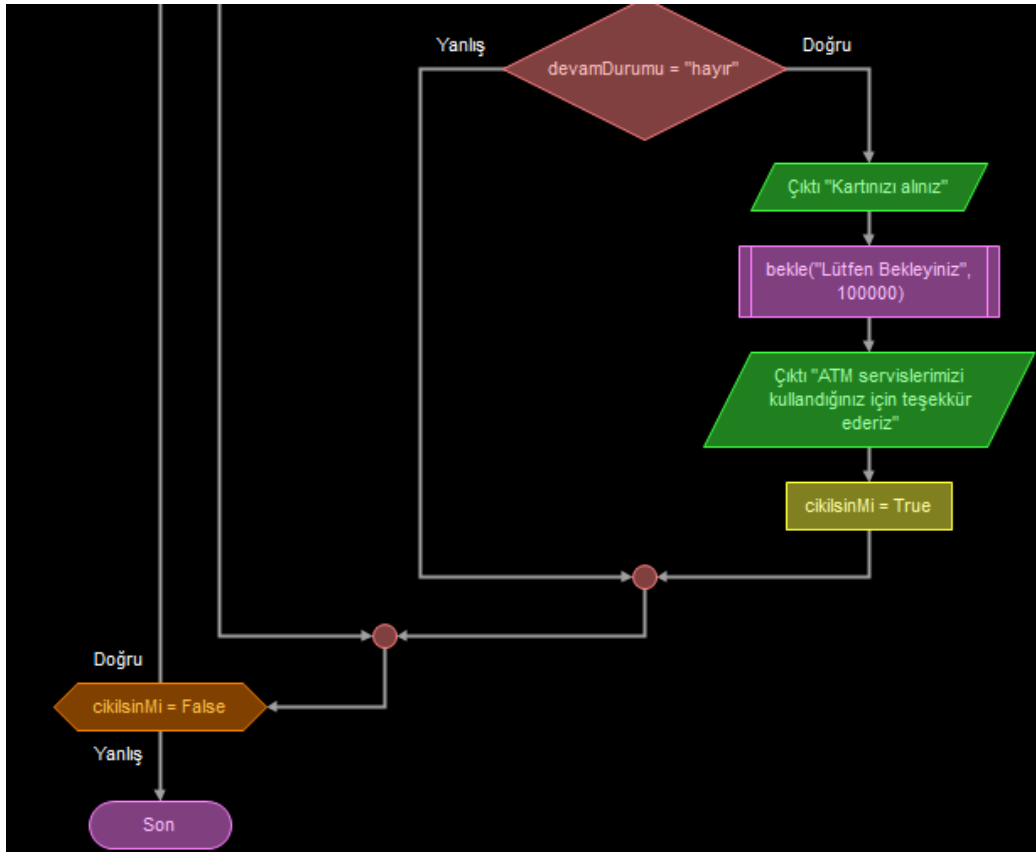


Şekil 8-11 Ana fonksiyon 8. kısım

Secim değişkeninin 4 olarak girilmesi, kullanıcının programdan çıkmak istediği anlamına gelen menü seçeneğidir. Bu aşamada kullanıcıya “Programdan çıkmak istiyor musunuz? (evet/hayır)” şeklinde bir soru yöneltilir. Kullanıcının girdiği yanıt, devamDurumu değişkenine atanır. Bu değişkenin değeri, do döngüsünün sonlandırılıp sonlandırılmayacağını belirleyeceği için programın kapanma koşulunu kontrol eden temel unsur olarak kullanılır.

Kullanıcı devamDurumu değişkenine “hayır” değerini girdiğinde, konsol ekranında “Kartınızı alınız.” ifadesi görüntülenir ve kısa bir bekleme süresi uygulanır. Ardından sistem, “ATM servislerimizi kullandığınız için teşekkür ederiz.” mesajını ekrana yazdırır. Bu işlemten sonra cikilsinMi değişkeninin değeri true olarak güncellenir.

Programın başında do döngüsünün çalışma koşulu cikilsinMi değişkeni false olduğu sürece döngünün devam etmesi şeklinde tanımlandığı için, değer true olarak değişmesiyle birlikte döngü sonlanır ve program kapanır. Oluşan akış şeması Şekil 8-12'deki gibi olmaktadır.



Şekil 8-12 Ana fonksiyon 9. kısım

8.2.2 Verilen şemanın Python çıktısı

```
def bekle(mesaj, süre):
```

```
    print(mesaj)
```

```
    for i in range(0, süre + 1, 1):
```

```
        pass
```

```
def menü():
```

```
    print("***** ATM Servislerine Hoşgeldiniz *****")
```

```
    print("1. Hesaplar Bakiye Durumu")
```

```
    print("2. Para Çek")
```

```
    print("3. Para Yatır")
```

```
    print("4. Çıkış")
```

```
    print("*****")
```

```
    print("Seçim Yapınız : ")
```

```
# Main
```

```
bakiye = 1000
```

```
girilenSifre = 0
```

```
devamDurumu = "hayır"
```

```
cikilsinMi = False
```

```
print("Kartınızı Takınız")
```

```
bekle("Lütfen Bekleyiniz", 100000)
sistemSifresi = 12345
```

```
while girilenSifre != sistemSifresi:
    print("Lütfen Şifrenizi Giriniz: ")
    girilenSifre = int(input())
    bekle("Lütfen Bekleyiniz", 100000)
    if girilenSifre != sistemSifresi:
        print("Lütfen Şifrenizi Doğru Giriniz")
```

```
while True:
    menü()
    secim = int(input())
```

```
if secim == 1:
    print("Hesap Bakiyeniz: " + str(bakiye))
    bekle("Lütfen Bekleyiniz", 100000)
if secim == 2:
    print("Çekmek istediğiniz tutarı giriniz:")
    cekilenPara = int(input())
    if cekilenPara % 100 != 0:
        print("LÜTFEN MİKTARI 100'ÜN KATLARI OLARAK GİRİNİZ")
    else:
        if cekilenPara > bakiye:
            print("Yetersiz Bakiye")
        else:
            bakiye = bakiye - cekilenPara
            print("Lütfen Paranızı Alınız")
            bekle("Lütfen Bekleyiniz", 100000)
            print("Güncel Bakiye: " + str(bakiye))
if secim == 3:
    print("Yatırmak istediğiniz miktarı girin")
    yatırılanPara = int(input())
    bakiye = bakiye + yatırılanPara
    bekle("Lütfen Bekleyiniz", 100000)
    print("Güncel Bakiyeniz: " + str(bakiye))
if secim == 4:
    print("Başka bir işlem yapmak istermisiniz?(evet/hayır)")
    devamDurumu = input()
    if devamDurumu == "hayır":
        print("Kartınızı alınız")
        bekle("Lütfen Bekleyiniz", 100000)
        print("ATM servislerimizi kullandığınız için teşekkür ederiz")
        cikilsinMi = True
if cikilsinMi != False: break
```



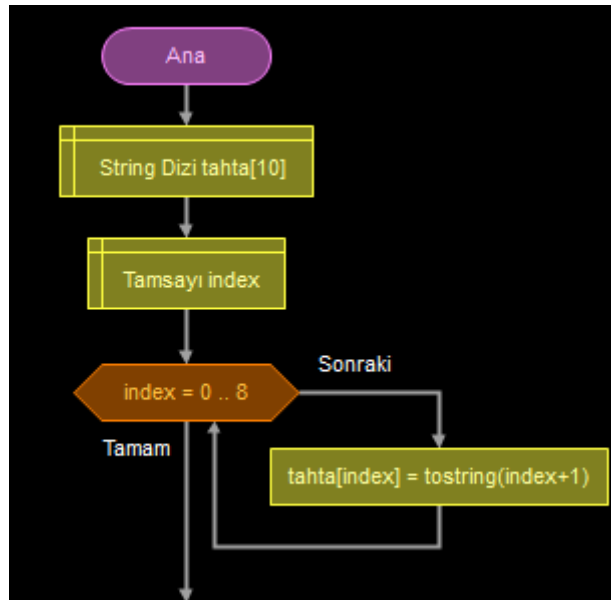
Flowgorithm dosyasına ulaşmak için kodu okutunuz

8.2.3 XOX Oyununun Akış Şeması

Oyunumuzu tasarlarken, aşağıdaki görselde (Şekil 8-13) yer aldığı gibi **3×3 boyutunda bir matrisi** kullanıcıya çıktı olarak sunacağız. Kullanıcı bir konum seçtiğinde seçilen hücreye “X”, bilgisayar bir konum seçtiğinde ise “O” sembolü yerleştirilecek ve oyun bu şekilde ilerleyecektir.

7 8 9
4 5 6
1 2 3

Şekil 8-13 Oyun Genel Yapısı



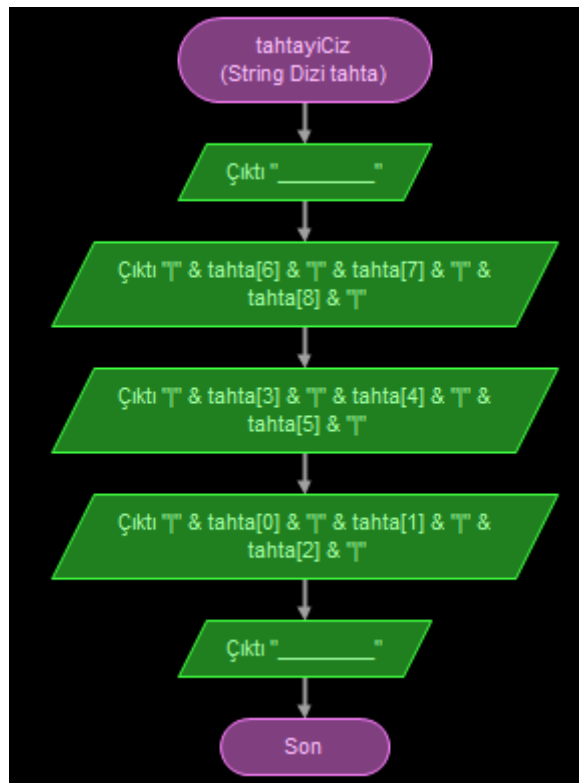
Şekil 8-14 Ana fonksiyon 1. kısım

Tahta isimli, String türünde bir dizi tanımladık (Şekil 8-14) ve bu dizinin boyutunu 10 olarak belirledik. Bir for döngüsü ve *index* değişkeni kullanarak dizinin her bir elemanına karşılık gelen indeks numaralarını oluşturmuş olacağız. Program çalıştığında for döngüsü sırasıyla: tahta[0], tahta[1], tahta[2], tahta[3], tahta[4], tahta[5], tahta[6], tahta[7], tahta[8] şeklinde dizinin elemanlarını oluşturur ve oyun tahtasının başlangıç durumunu hazırlar.

Ana programda kullanacağımız değişkenler Şekil 8-15’de gösterilmiştir. Bu değişkenler, oyunun akışını kontrol etmek, kullanıcıdan alınan değerleri saklamak ve oyun tahtasının güncel durumunu takip etmek amacıyla kullanılacaktır.

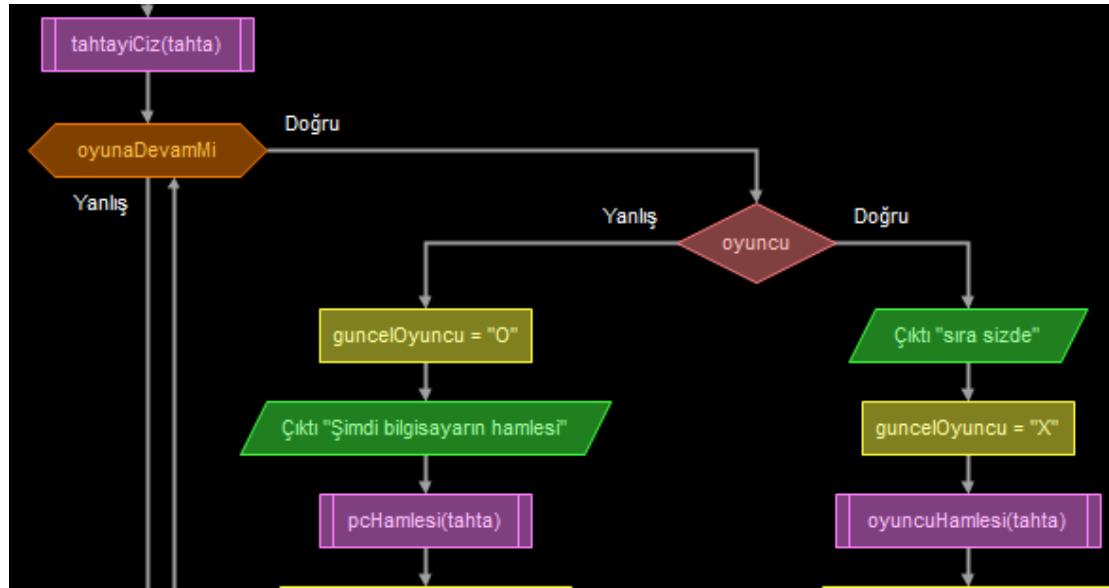


Şekil 8-15 Ana fonksiyon 2. kısım



Oyun tahtasını konsol ekranında görüntülemek için tahtayıCiz(tahta) isimli bir fonksiyon (Şekil 8-16) tanımlıyoruz. Bu fonksiyon, ana programdan gönderilen tahta[] dizisindeki güncel değerleri kullanarak oyun tahtasını konsol ekranına çizer. Böylece kullanıcı, hamlelerden sonra tahtanın son durumunu görebilir. Fonksiyon tanımlandıktan sonra ana programa geri dönerek tahtayıCiz fonksiyonunu çağırıyoruz.

Şekil 8-16 Tahta çizme fonksiyonu

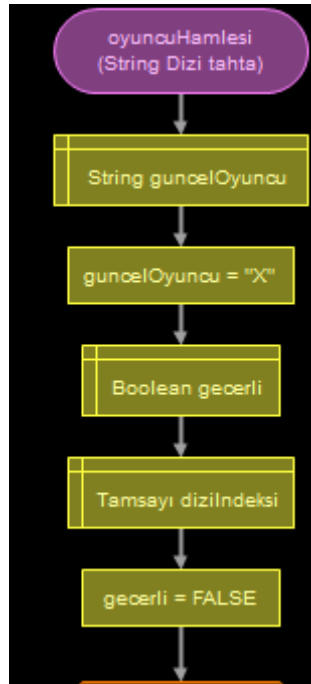


Şekil 8-17 Ana fonksiyon 3. kısım

tahtayıCiz fonksiyonunu çağırdıktan sonra oyunumuzu programlamaya devam etmek için bir while döngüsü (Şekil 8-17) oluşturuyoruz. Döngünün koşulu olarak, daha önce tanımlamış olduğumuz *boolean* türündeki oyunaDevamMi değişkenini kullanıyoruz. Bu değişkenin değeri true olduğu sürece döngü çalışmaya devam edecek; değer false olduğunda ise döngü sonlanacaktır.

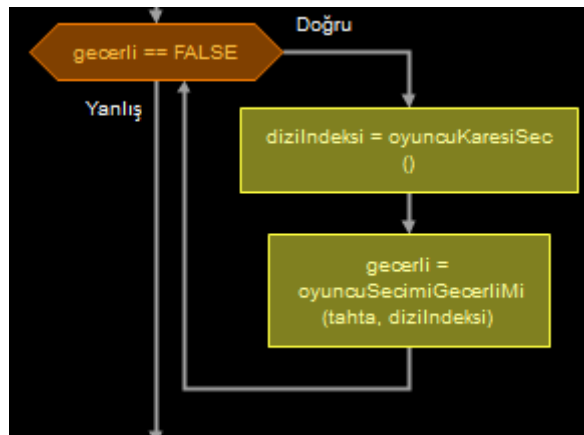
Bu aşamada while(oyunaDevamMi) döngüsünün içerisine bir if yapısı ekliyoruz ve bu yapıya koşul olarak *boolean* türündeki oyuncu değişkenimizi veriyoruz. Eğer oyuncu değişkeninin değeri true ise oynama sırası kullanıcıda; değer false ise oynama sırası bilgisayardadır. O an hangi oyuncunun hamle yapacağını belirleyebilmek için string türünde tanımladığımız guncelOyuncu değişkenini kullanıyoruz. Kullanıcı için sembolü "X", bilgisayar için ise "O" olarak belirlemiş bulunuyoruz.

Artık oyuncu hamlelerinin kodlanmasına geçebiliriz. Bunun için oyuncuHamlesi(tahta) isimli bir fonksiyon oluşturarak kullanıcının hamle yapma sürecini programlayacağız.



Fonksiyonumuzu oluşturmaya, ihtiyaç duyduğumuz değişkenleri tanımlayarak (Şekil 8-18) başladık. Bu değişkenlerin her biri oyunun akışı içerisinde belirli görevler üstlenecektir. Her değişkenin nasıl kullanılacağı ise fonksiyonun ilerleyen aşamalarında ayrıntılı olarak

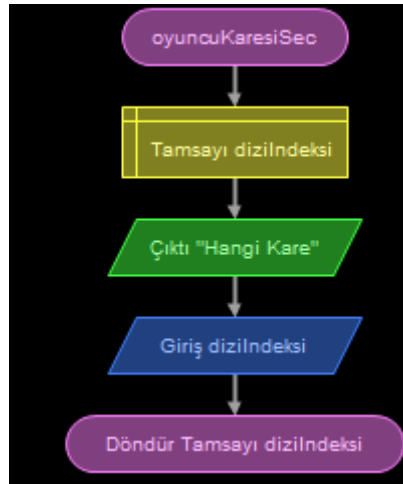
Şekil 8-18 Oyuncu hamlesi çizme fonksiyonu



Bir while döngüsü oluşturuyoruz (Şekil 8-19) ve bu döngü, boolean türündeki *gecerli* değişkeninin değeri false olduğu sürece çalışmaya devam edecektir. Bu yapının kullanılmasındaki amaç, kullanıcının daha önce işaretlenmiş olan tahta[] elemanlarından birini seçmesi durumunda, geçersiz hamleyi engelleyip ondan yeni bir seçim yapmasını sağlamaktır.

Şekil 8-19 Ana fonksiyon 4. kısım

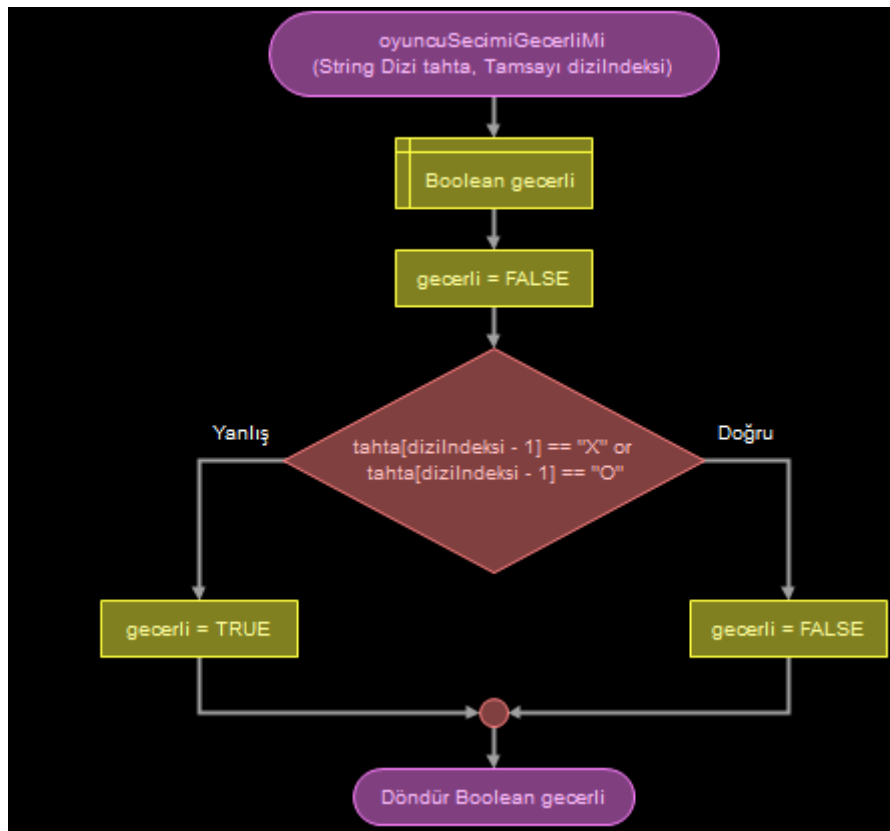
Şimdi, oyuncunun bir hamle yapmasını sağlamak için *oyuncuKaresiSec()* isimli ayrı bir fonksiyon tanımlayalım. Bu fonksiyondan dönen değer, *diziIndexi* değişkenine atanacak ve böylece oyuncunun seçtiği kare belirlenmiş olacaktır.



Yandaki fonksiyonda (Şekil 8-20), **diziIndexsi** değişkeninin değerini kullanıcının belirlemesi için konsol ekranına bir çıktı verdik. Kullanıcının girdiği değer, **diziIndexsi** değişkenine atanır. Fonksiyonun dönüş değeri olarak da **diziIndexsi** belirlenmiştir, böylece fonksiyon çağrıldığında kullanıcının seçimi elde edilmiş olur.

Şekil 8-20 Oyuncu karesi seçme Fonksiyonu

Artık kullanıcıya bir kare seçtirdiğimize göre, seçilen karenin geçerli olup olmadığını kontrol etmemiz gerekiyor. Bunun için oyuncuSecimiGecerliMi(tahta, diziIndexsi) şeklinde bir fonksiyon oluşturalım. Bu fonksiyon, kullanıcının seçtiği karenin daha önce işaretlenip işaretlenmediğini kontrol edecek ve geçerli olup olmadığını belirleyecektir.



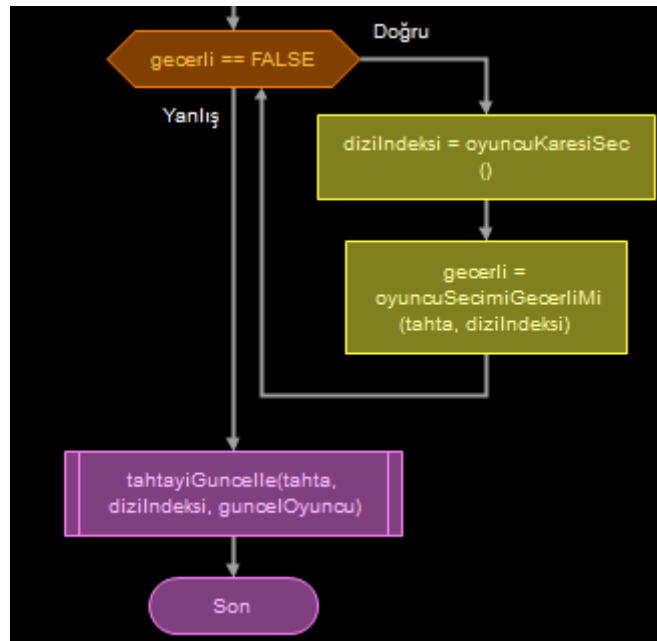
Şekil 8-21 Oyuncunun geçerli kare seçtiğini kontrol eden fonksiyon

Fonksiyonumuzda (Şekil 8-21) gerekli değişkenleri tanımladıktan sonra bir if yapısı oluşturduk. Bu yapıda, eğer `tahta[diziIndeksi - 1] == "X"` ise —yani kullanıcının seçtiği indeksin bir eksiğinde “X” sembolü varsa— geçerli değişkeninin değeri false olacak şekilde kontrol sağlanır.

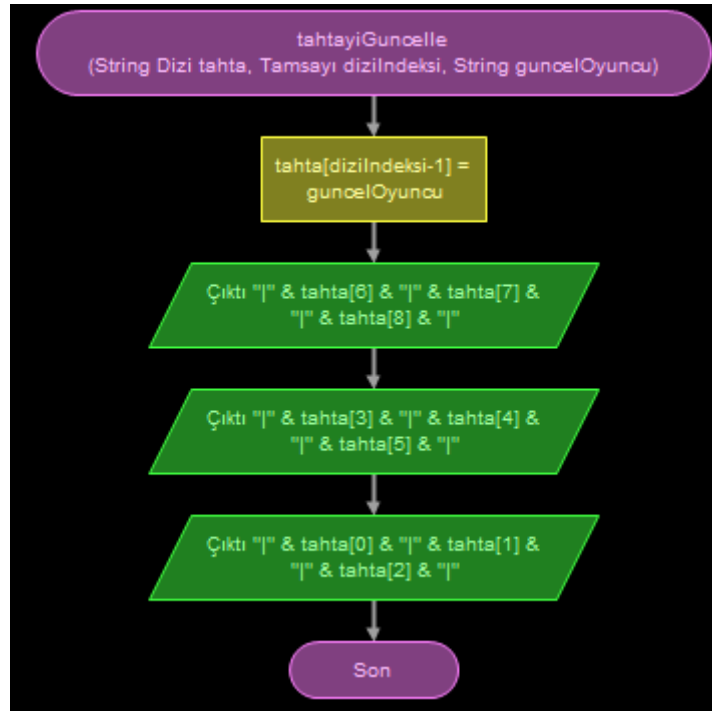
Dizi indeksinde 1 çıkarmamızın nedeni, konsol ekranında tahtanın 1–9 arası karelerden oluşurken, programımızdaki dizinin indekslerinin 0–8 arasında olmasıdır. Kullanıcının girdiği değerden 1 çıkararak, dizinin indeksleri ile eşleşmesini sağlıyoruz.

Bu fonksiyon `while(gecerli == false)` döngüsünün içerisinde tanımlandığı için, geçerli değişkeninin değeri false olduğu sürece program bu kontrolü tekrar edecektir. Fonksiyonun dönüş değeri de geçerli olarak belirlenmiştir.

Kullanıcıya kare seçtirip geçerliliğini kontrol ettikten sonra, artık tahtayı kullanıcının seçtiği kareye “X” sembolü gelecek şekilde güncellememiz gerekiyor. Bunun için `oyuncuHamlesi(tahta)` fonksiyonuna geri dönerek, `while(gecerli == false)` döngüsünün hemen (Şekil 8-22) altına `tahtayiGuncelle(tahta, diziIndeksi, guncelOyuncu)` fonksiyonumuzu çağırıyoruz.



Şekil 8-22 Ana fonksiyon 5. kısım

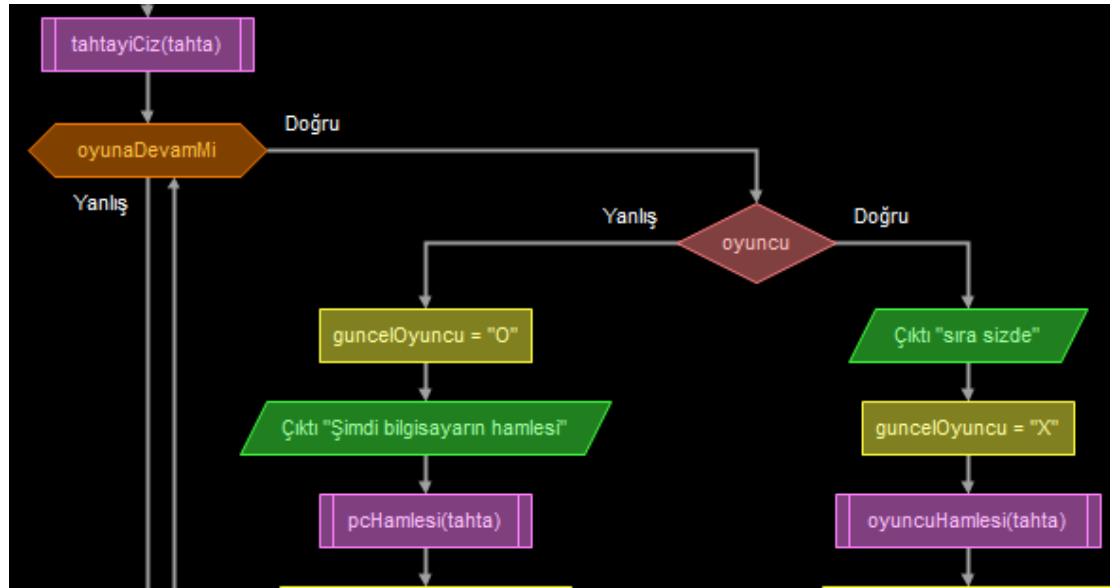


Şekil 8-23 Tahtayı güncelleme fonksiyonu

Fonksiyonumuzu (Şekil 8-23) incelediğimizde, `tahta[diziIndeksi - 1] = guncelOyuncu` şeklinde bir değer ataması yaptığımızı görüyoruz. Buradaki amaç, örneğin kullanıcı 9 numaralı kareyi seçerse, `tahta[8] = "X"` olacak şekilde güncelleme yapmaktır. Çünkü ana fonksiyonda sıra kullanıcıda olduğu için, `guncelOyuncu = "X"` olarak tanımlanmıştır.

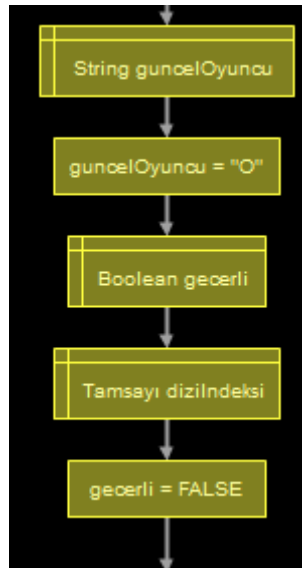
Bu atama işlemi tamamlandıktan sonra, tahtanın güncel durumu konsol ekranına yazdırılacak ve `tahta[8]` konumunda "X" sembolü görüntülenecektir.

Artık kullanıcı hamlelerini kodladığımıza göre, bir sonraki adım olarak (Şekil 8-24) bilgisayarın hamlelerini kodlamaya geçebiliriz.



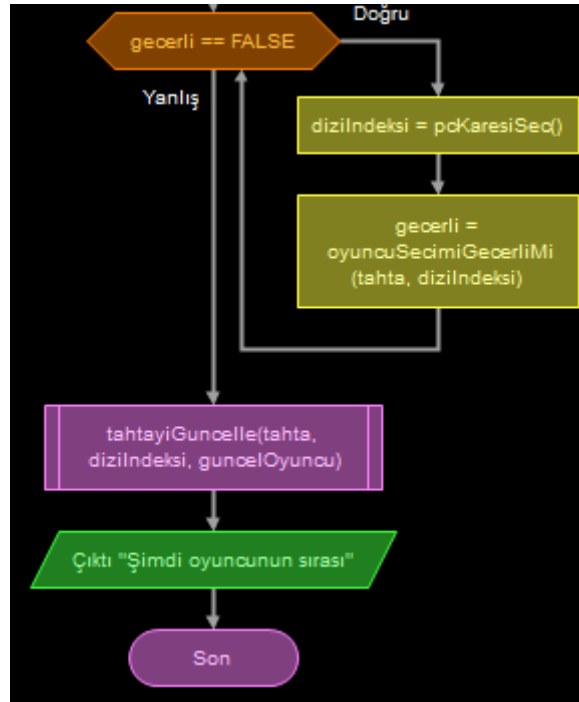
Şekil 8-24 Ana fonksiyon 6. kısım

Şemamızın sol tarafına geçmek için oyuncu değişkeninin değeri false olmalıdır. İlerleyen aşamalarda bu değeri değiştireceğiz; şimdilik oynama sırası bilgisayarda olduğu için, guncelOyuncu değişkeninin değerini “O” olarak atıyoruz. Konsol ekranına, sıranın bilgisayarda olduğunu belirten bir çıktı verdikten sonra, pcHamlesi(tahta) fonksiyonunu (Şekil 8-25) kodlamaya başlayabiliriz.



Bilgisayar hamlesinde de, oyuncu hamlesinde fonksiyonumuzda yaptığımız benzer işlemleri gerçekleştireceğiz. Öncelikle, bu fonksiyonda kullanacağımız gerekli değişkenleri tanımlayarak işe başlayalım.

Şekil 8-25 Ana fonksiyon 7. kısım



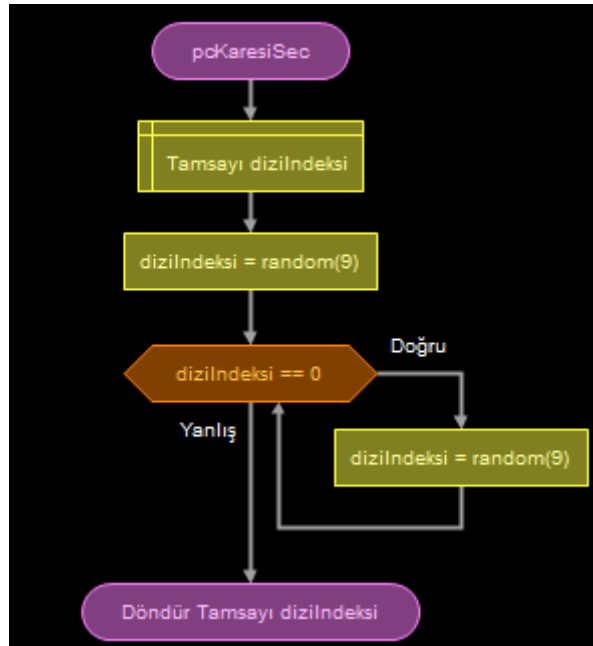
Şekil 8-26 Ana fonksiyon 8. kısım

Bu aşamada (Şekil 8-26), while(gecerli == false) döngüsü oluşturuyoruz. Döngü içerisinde üç fonksiyon kullanıyoruz.

İlk fonksiyon, `diziIndeksi = pcKaresiSec()`, bilgisayarın hamle yapmasını sağlamak için kullanılacaktır. Bu fonksiyon, bilgisayarın rastgele bir kare seçmesini sağlayacak şekilde kodlanacaktır.

Diğer iki fonksiyon ise, `oyuncuSecimiGecerliMi(tahta, diziIndeksi)` ve `tahtayiGuncelle(tahta, diziIndeksi, guncelOyuncu)`, kullanıcı için tanımladığımız işlemlerle aynıdır. Burada da aynı şekilde, seçilen karenin geçerliliğini kontrol etmek ve tahtayı güncellemek amacıyla kullanıyoruz.

Özetle, bu döngüde bilgisayar bir kare seçer, seçimin geçerliliği kontrol edilir ve ardından tahtadaki durum güncellenir; bu işlem geçerli bir seçim yapılana kadar tekrarlanır.



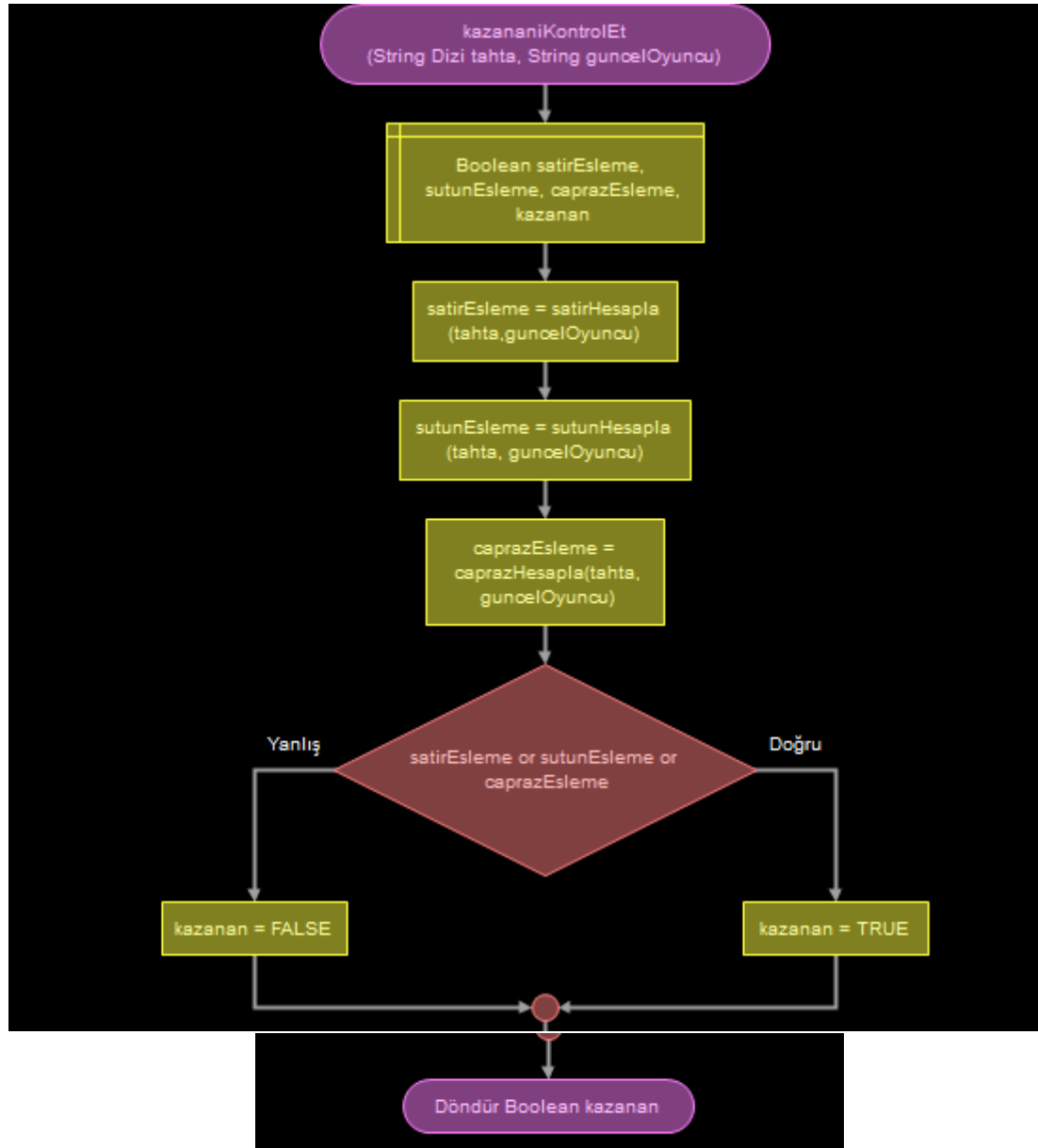
Bilgisayara bir kare seçtirmek için, `dizilIndeksi` değişkeninin değerini 0 ile 9 arasında rastgele bir sayı olacak şekilde (Şekil 8-27) atadık. Burada, bir `while` döngüsü kullanarak bilgisayarın 0 numaralı kareyi seçmesini engelledik. Yani `dizilIndeksi` 0 olduğu sürece döngü devam eder; değer 0 olmadığında döngü sonlanır.

Fonksiyonun dönüş değeri olarak `dizilIndeksi` belirlenmiş olup, böylece bilgisayarın seçtiği geçerli kare elde edilmiş ve fonksiyon tamamlanmıştır.

Şekil 8-27 Bilgisayarın hamle seçeceği fonksiyon

Hem oyuncu hem de bilgisayar hamlelerini yaptırдық; ancak bu hamleler bir döngü içerisinde devam ediyor. Oyun sona erdiğinde, döngüyü sonlandırmamız gerekiyor.

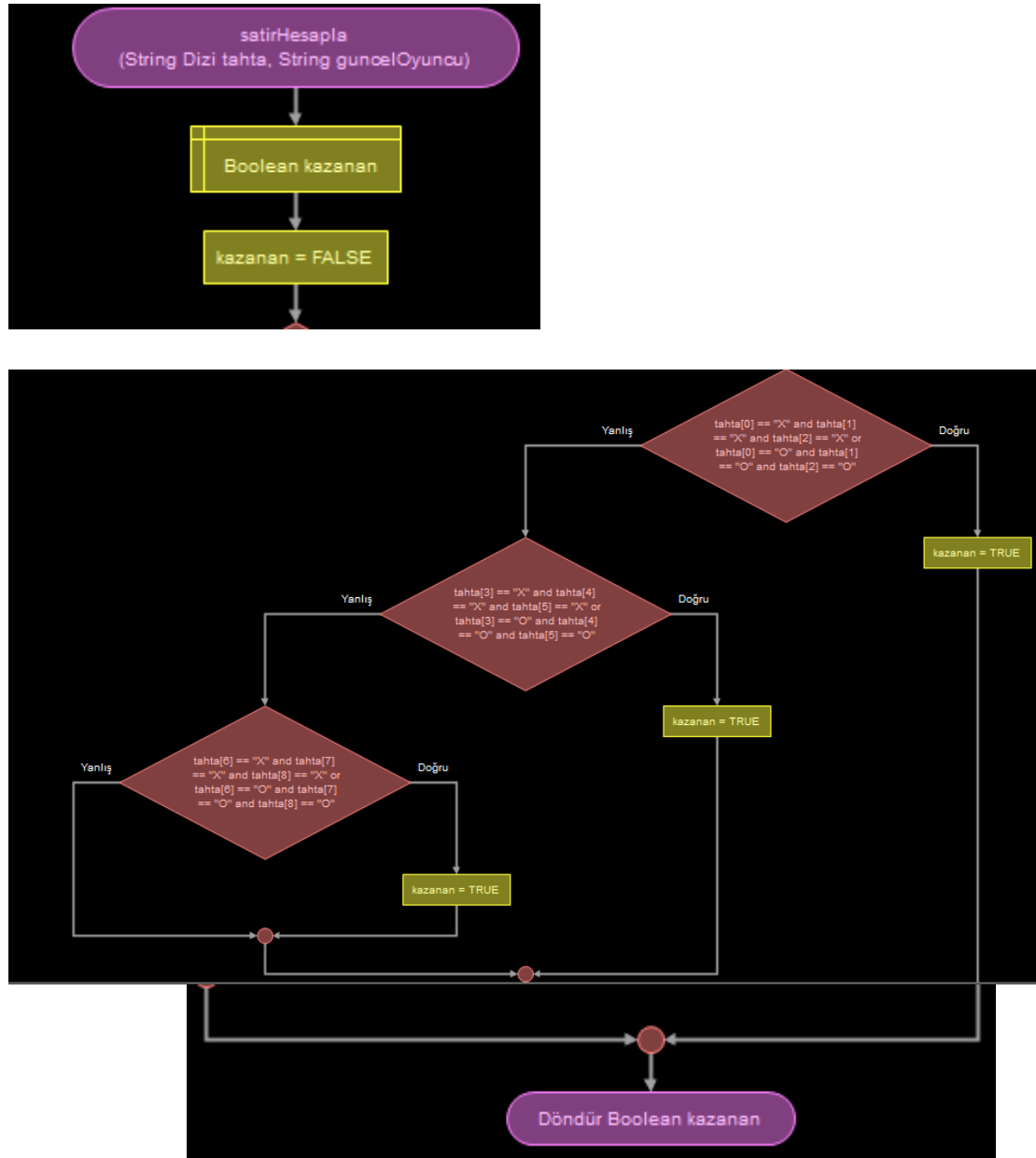
XOX oyunu kuralları gereği, aynı satırda, sütunda veya çaprazda sembolünü ilk yerleştiren oyuncu oyunu kazanır. Bu nedenle, oyunun her aşamasında bu üç durumu da kontrol etmemiz gerekmektedir. Bu kontrolleri, `kazananiKontrolEt(tahta, guncelOyuncu)` fonksiyonu içinde ayrı ayrı fonksiyonlar aracılığıyla Şekil 8-28’de gerçekleştireceğiz.



Şekil 8-28 Kazananın kontrol edildiği fonksiyon

satirEsleme, sutunEsleme, caprazEsleme ve kazanan isimli *boolean* türünde değişkenler oluşturduk. Başlangıçta bu değişkenlerin tümü false değerine sahiptir. Ancak oyunda aynı satırda, sütunda veya çaprazda aynı semboller yan yana geldiğinde, ilgili değişkenin değeri true olacak şekilde güncellenmelidir.

Kontrole satırlarla başlayalım. Bunun için satirHesapla(tahta, guncelOyuncu) fonksiyonunu (Şekil 8-29) kullanarak tahtadaki satırları inceleyip, aynı sembollerin yan yana gelip gelmediğini belirleyeceğiz.



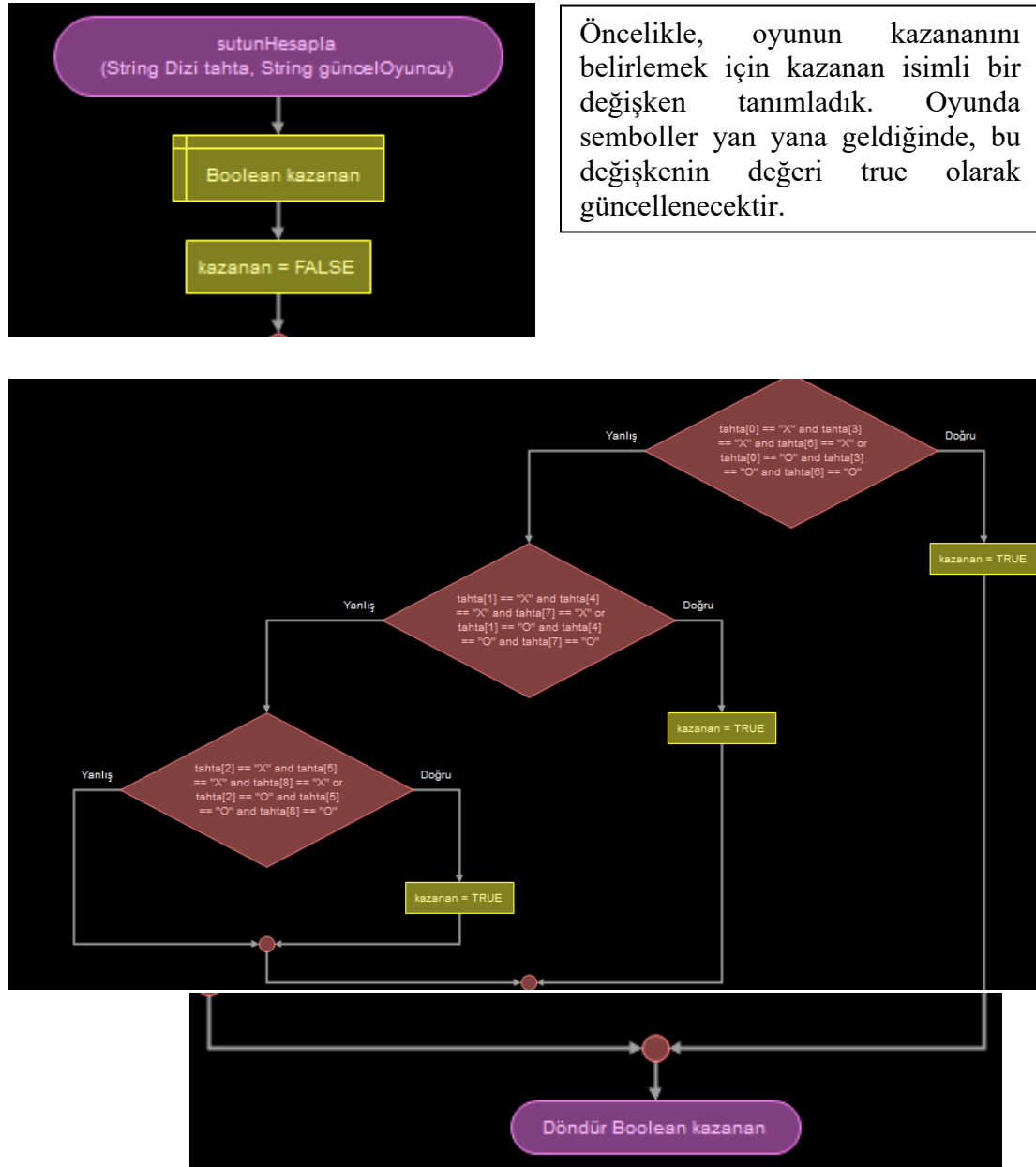
Şekil 8-29 Satır sayısının hesaplandığı fonksiyon

Üç satırımız olduğu için, her satırı ayrı ayrı kontrol etmek amacıyla üç adet if yapısı oluşturduk.

- Eğer `tahta[0] == "X" ve tahta[1] == "X" ve tahta[2] == "X"` veya `tahta[0] == "O" ve tahta[1] == "O" ve tahta[2] == "O"` ise, yani ilk satırda “XXX” veya “OOO” oluşmuşsa, kazanan değişkeninin değeri true olarak atanır.
- İkinci satır için, `tahta[3] == "X" ve tahta[4] == "X" ve tahta[5] == "X"` veya `tahta[3] == "O" ve tahta[4] == "O" ve tahta[5] == "O"` durumu sağlandığında, kazanan değişkeni true olarak güncellenir.
- Üçüncü satır için ise, `tahta[6] == "X" ve tahta[7] == "X" ve tahta[8] == "X"` veya `tahta[6] == "O" ve tahta[7] == "O" ve tahta[8] == "O"` kontrolü ile kazanan değişkeni true olur.

Bu şekilde, satırlar üzerinde kontrollerimizi tamamladık ve fonksiyonun dönüş değişkeni olarak kazanan belirledik.

Sütunlarda aynı sembollerin alt alta gelip gelmediğini kontrol etmek için ise `sutunHesapla(tahta, guncelOyuncu)` fonksiyonunu (Şekil 8-30) oluşturabiliriz.



Şekil 8-30 Sütun sayısının hesaplandığı fonksiyon

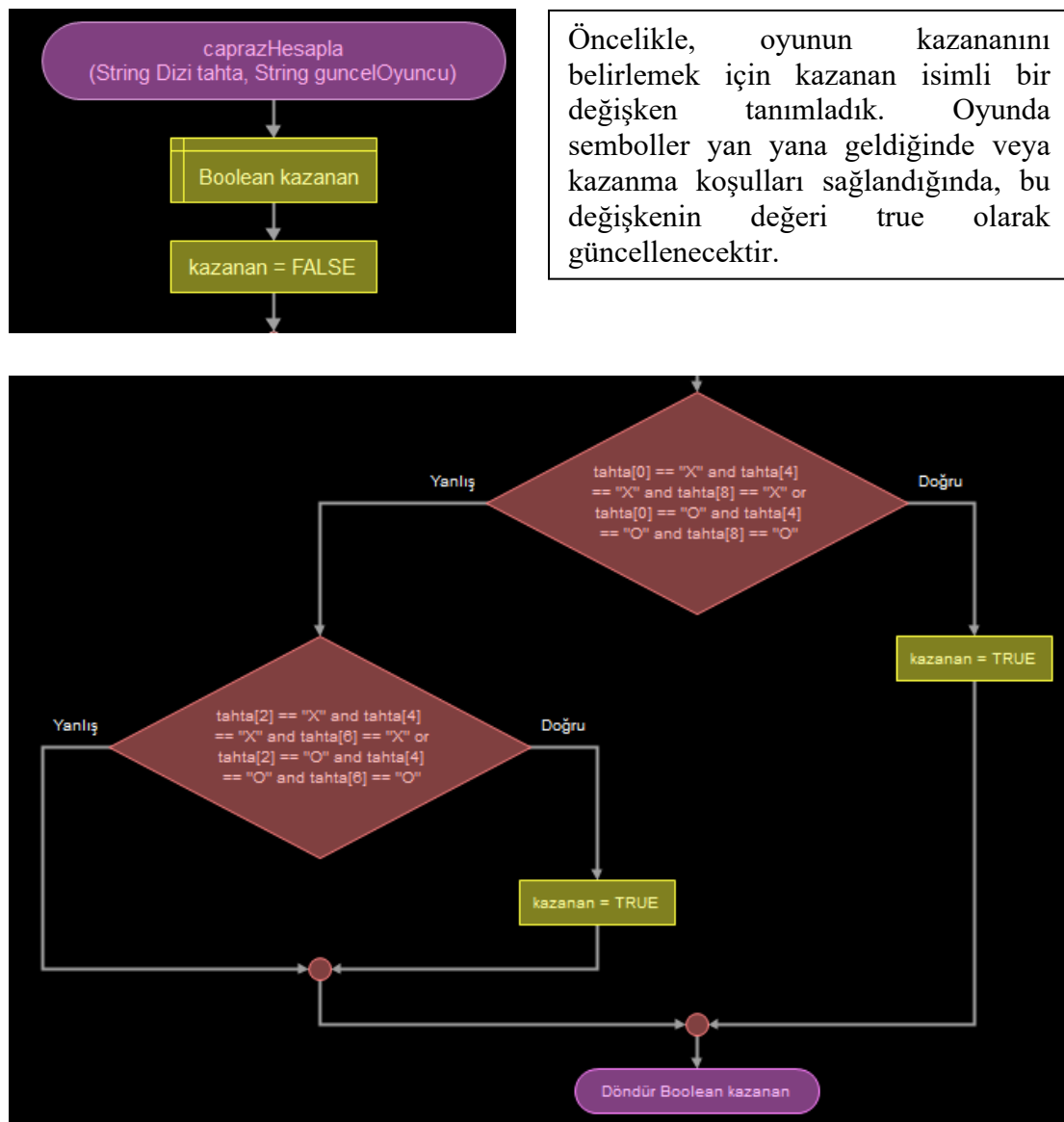
Üç sütunumuz olduğu için, her sütunu ayrı ayrı kontrol etmek amacıyla üç adet if yapısı oluşturduk.

- Eğer `tahta[0] == "X" ve tahta[3] == "X" ve tahta[6] == "X"` veya `tahta[0] == "O" ve tahta[3] == "O" ve tahta[6] == "O"` ise, yani ilk sütunda “XXX” veya “OOO” oluşmuşsa, kazanan değişkeni true olarak atanır.

- İkinci sütun için, tahta[1] == "X" ve tahta[4] == "X" ve tahta[7] == "X" veya tahta[1] == "O" ve tahta[4] == "O" ve tahta[7] == "O" durumu sağlandığında, kazanan değişkeni true olur.
- Üçüncü sütun için ise, tahta[2] == "X" ve tahta[5] == "X" ve tahta[8] == "X" veya tahta[2] == "O" ve tahta[5] == "O" ve tahta[8] == "O" kontrolü ile kazanan değişkeni true değerini alır.

Bu şekilde sütun kontrollerini tamamladık ve fonksiyonun dönüş değeri olarak kazanan belirlenmiştir.

Son olarak, çaprazda aynı semboller yan yana gelmiş mi diye kontrol etmek için caprazHesapla(tahta, guncelOyuncu) fonksiyonunu (Şekil 8-31) oluşturabiliriz.



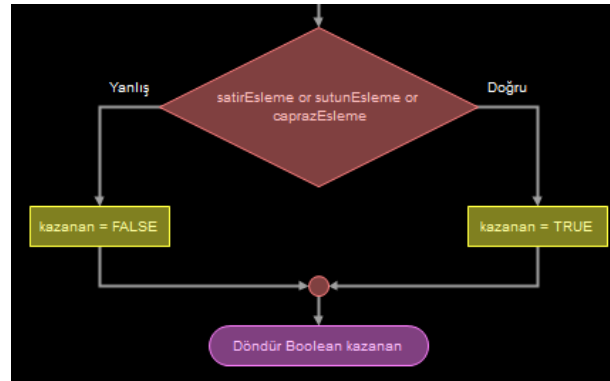
Şekil 8-31 Çapraz aynı sembollerin yan yana geldiğini kontrol eden fonksiyon

İki adet çapraz bulunduğu için, her çaprazı ayrı ayrı kontrol etmek amacıyla iki adet if yapısı oluşturduk.

- Eğer `tahta[0] == "X" ve tahta[4] == "X" ve tahta[8] == "X"` veya `tahta[0] == "O" ve tahta[4] == "O" ve tahta[8] == "O"` ise, yani ilk çaprazda “XXX” veya “OOO” oluşmuşsa, kazanan değişkeni true olarak atanır.
- İkinci çapraz için, `tahta[2] == "X" ve tahta[4] == "X" ve tahta[6] == "X"` veya `tahta[2] == "O" ve tahta[4] == "O" ve tahta[6] == "O"` durumu sağlandığında, kazanan değişkeni true olarak güncellenir.

Bu şekilde, `kazananiKontrolEt` fonksiyonunun alt fonksiyonlarını tamamlamış olduk.

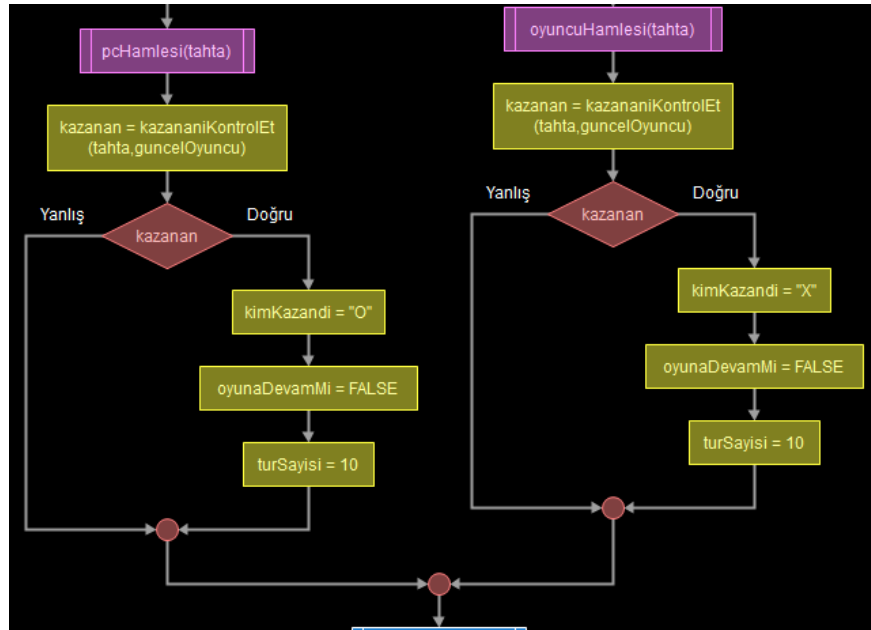
Son olarak, `satirEsleme`, `sutunEsleme` ve `caprazEsleme` değişkenlerinin değerlerini kontrol ederek, herhangi birinin değeri true ise, kazanan değişkenini true yapacak bir if kontrolü ekleyerek fonksiyonu Şekil 8-32’deki gibi sonlandırıyoruz.



Şekil 8-32 Kazanani kontrol eden fonksiyonun son kısmı

Artık `kazananiKontrolEt` fonksiyonumuzu ana fonksiyonumuzda çağırabiliriz. Hem oyuncu hem de bilgisayar için kazanma durumunu kontrol etmemiz gerektiğinden, her iki tarafın hamle fonksiyonlarının hemen altında bu fonksiyonu çalıştırıyoruz.

Fonksiyondan dönen değeri kazanan değişkenimize atayarak, oyunun hangi aşamada kazanıldığını takip edebiliriz.

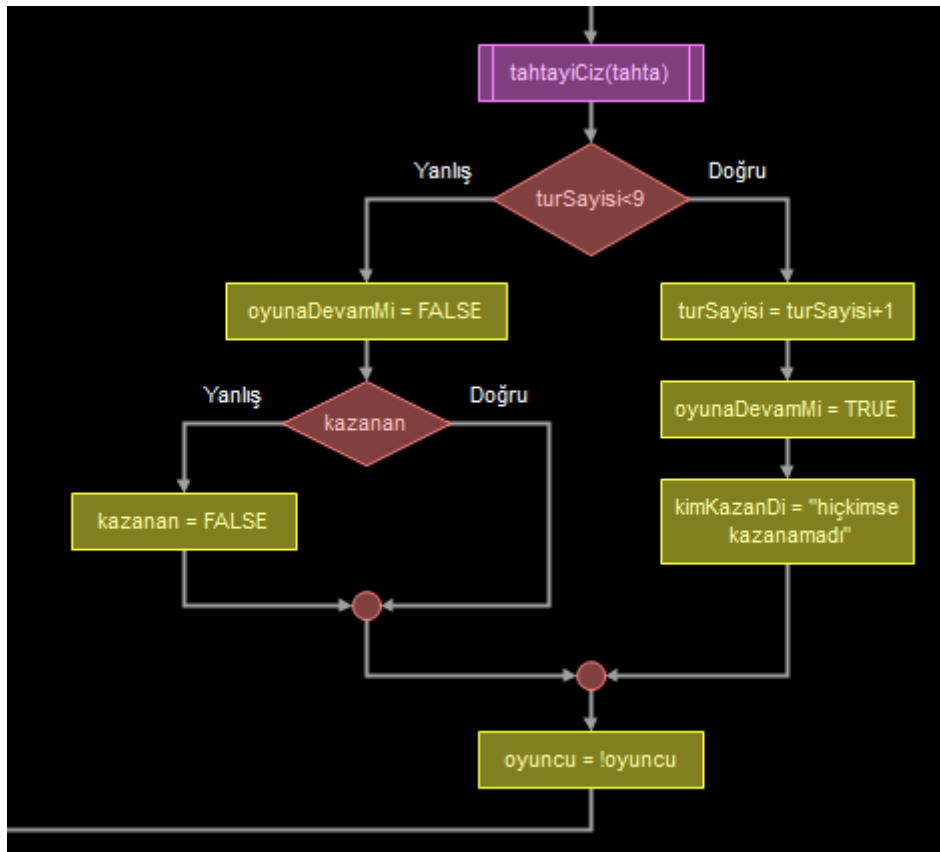


Şekil 8-33 Ana fonksiyon 9. kısım

Şekil 8-33’de görüldüğü gibi, ana fonksiyonumuz içinde kazanan değişkeninin true olması durumunda, while(oyunaDevamEdilsinMi) döngüsünü sonlandırmak için her iki taraf için ayrı if kontrolleri ekledik. Kazanan değişkeni true olduğunda, oyunaDevamEdilsinMi değişkenini false yaparak döngünün bitmesini sağlıyoruz.

Kodun ilerleyen aşamalarında, oyunu bilgisayarın mı yoksa kullanıcının mı kazandığını konsol ekranına yazdırmak için kimKazandi isimli bir değişkene ihtiyaç duyacağız. Bu değişken, kullanıcı kazanmışsa "X", bilgisayar kazanmışsa "O" değeri alacak şekilde ayarlanmıştır.

Ayrıca, oyunun maksimum 8 turda kazanılması gerekmektedir. Eğer 8 tur sonunda oyun bitmemişse, oyunun berabere sonuçlandığı kabul edilir. Bu kontrolü daha sonra ekleyeceğiz. Şimdilik, kazanın belirlenebilmesi için turSayisi değişkenini 10 olarak belirledik.



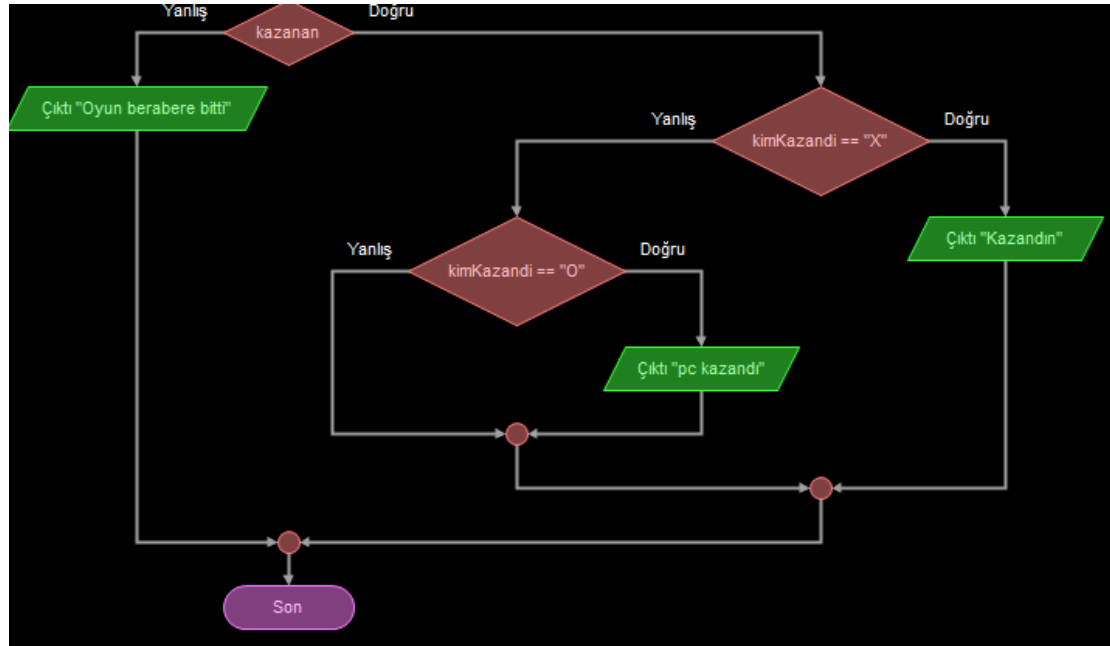
Şekil 8-34 Ana fonksiyon 10. kısım

Kazanan kontrollerini tamamladıktan sonra, tekrar **tahtayı çiz** fonksiyonunu Şekil 8-34’de görüldüğü gibi çağırdık. Oyunun 8 turdan fazla sürmemesi için, **turSayisi** değişkeninin 9’dan küçük olduğu durumlarda bu değişkeni 1 artırıyor ve oyunun devam etmesini sağlıyoruz.

Ancak, tur sayısı 8’i geçtiyse, artık yapılacak hamle kalmadığından **kazanan** değişkenini **false** olarak ayarlıyoruz ve oyunun berabere sonuçlandığını kabul ediyoruz.

Oynama sırasının kimin olduğunu belirlemek için **oyuncu = !oyuncu** ifadesini kullanıyoruz. Döngü devam ettiği sürece, **oyuncu** değişkeni her adımda **true** ve **false** arasında değişir. **True** olduğunda hamle sırası kullanıcıda, **false** olduğunda ise hamle sırası bilgisayardadır.

Bu kontroller ve güncellemeler ile oyunun kimin kazandığını belirleyerek oyunu tamamen Şekil 8-35’de kodlamış oluyoruz.



Şekil 8-35 Ana fonksiyon 11. kısım

Eğer kazanan değişkeninin değeri false ise, konsol ekranında “Oyun berabere bitti” mesajı yazdırılacaktır.

Eğer kazanan değişkeni true ise, kimKazandi değişkeninin değeri kontrol edilir:

- kimKazandi == "X" ise, oyunu kullanıcı kazanmıştır.
- kimKazandi == "O" ise, oyunu bilgisayar kazanmıştır.

Program çalıştırıldığında, konsol ekranında oyun sonucuna ilişkin çıktı bu şekilde görüntülenecektir.

_____	7 8 9	Şimdi	7 O 9
		bilgisayarın	
7 8 9	4 5 6	hamlesi	4 5 6
4 5 6	X 2 3	7 O 9	X 2 3
1 2 3	_____	4 5 6	_____
_____	7 8 9	X 2 3	sıra sizde
sıra sizde	4 5 6	Şimdi	Hangi Kare
Hangi Kare	X 2 3	oyuncunun	
		sırası	
1	_____		

8.2.4 Oyunun Python Çıktısı

```

import random

def caprazHesapla(tahta, guncelOyuncu):
    kazanan = False
    if tahta[0] == "X" and tahta[4] == "X" and tahta[8] == "X" or tahta[0] == "O"
    and tahta[4] == "O" and tahta[8] == "O":
        kazanan = True
    else:
        if tahta[2] == "X" and tahta[4] == "X" and tahta[6] == "X" or tahta[2] == "O"
        and tahta[4] == "O" and tahta[6] == "O":
            kazanan = True

    return kazanan

def kazananiKontrolEt(tahta, guncelOyuncu):
    satirEsleme = satirHesapla(tahta, guncelOyuncu)
    sutunEsleme = sutunHesapla(tahta, guncelOyuncu)
    caprazEsleme = caprazHesapla(tahta, guncelOyuncu)
    if satirEsleme or sutunEsleme or caprazEsleme:
        kazanan = True
    else:
        kazanan = False

    return kazanan

def oyuncuHamlesi(tahta):
    guncelOyuncu = "X"
    gecerli = False
    while gecerli == False:
        diziIndeksi = oyuncuKaresiSec()
        gecerli = oyuncuSecimiGecerliMi(tahta, diziIndeksi)
        tahtayiGuncelle(tahta, diziIndeksi, guncelOyuncu)

def oyuncuKaresiSec():
    print("Hangi Kare")
    diziIndeksi = int(input())

    return diziIndeksi

def oyuncuSecimiGecerliMi(tahta, diziIndeksi):
    gecerli = False

```

```
if tahta[diziIndeksi - 1] == "X" or tahta[diziIndeksi - 1] == "O":
    gecerli = False
else:
    gecerli = True

return gecerli

def pcHamlesi(tahta):
    guncelOyuncu = "O"
    gecerli = False
    while gecerli == False:
        diziIndeksi = pcKaresiSec()
        gecerli = oyuncuSecimiGecerliMi(tahta, diziIndeksi)
    tahtayiGuncelle(tahta, diziIndeksi, guncelOyuncu)
    print("Şimdi oyuncunun sırası")

def pcKaresiSec():
    diziIndeksi = int(random.random() * 9)
    while diziIndeksi == 0:
        diziIndeksi = int(random.random() * 9)

    return diziIndeksi

def satirHesapla(tahta, guncelOyuncu):
    kazanan = False
    if tahta[0] == "X" and tahta[1] == "X" and tahta[2] == "X" or tahta[0] == "O"
    and tahta[1] == "O" and tahta[2] == "O":
        kazanan = True
    else:
        if tahta[3] == "X" and tahta[4] == "X" and tahta[5] == "X" or tahta[3] == "O"
        and tahta[4] == "O" and tahta[5] == "O":
            kazanan = True
        else:
            if tahta[6] == "X" and tahta[7] == "X" and tahta[8] == "X" or tahta[6] ==
            "O" and tahta[7] == "O" and tahta[8] == "O":
                kazanan = True

    return kazanan

def sutunHesapla(tahta, guncelOyuncu):
    kazanan = False
```

```

    if tahta[0] == "X" and tahta[3] == "X" and tahta[6] == "X" or tahta[0] == "O"
    and tahta[3] == "O" and tahta[6] == "O":
        kazanan = True
    else:
        if tahta[1] == "X" and tahta[4] == "X" and tahta[7] == "X" or tahta[1] == "O"
        and tahta[4] == "O" and tahta[7] == "O":
            kazanan = True
        else:
            if tahta[2] == "X" and tahta[5] == "X" and tahta[8] == "X" or tahta[2] ==
            "O" and tahta[5] == "O" and tahta[8] == "O":
                kazanan = True

    return kazanan

def tahtayiCiz(tahta):
    print("_____")
    print("|" + tahta[6] + "|" + tahta[7] + "|" + tahta[8] + "|")
    print("|" + tahta[3] + "|" + tahta[4] + "|" + tahta[5] + "|")
    print("|" + tahta[0] + "|" + tahta[1] + "|" + tahta[2] + "|")
    print("_____")

def tahtayiGuncelle(tahta, diziIndeksi, guncelOyuncu):
    tahta[diziIndeksi - 1] = guncelOyuncu
    print("|" + tahta[6] + "|" + tahta[7] + "|" + tahta[8] + "|")
    print("|" + tahta[3] + "|" + tahta[4] + "|" + tahta[5] + "|")
    print("|" + tahta[0] + "|" + tahta[1] + "|" + tahta[2] + "|")

# Main
tahta = [""] * (10)

for index in range(0, 8 + 1, 1):
    tahta[index] = str(index + 1)
oyuncu = True
turSayisi = 1
kazananVarMi = True
dahaFazlaKare = False
oyunaDevamMi = True
tahtayiCiz(tahta)
while oyunaDevamMi:
    if oyuncu:
        print("sıra sizde")
        guncelOyuncu = "X"

```

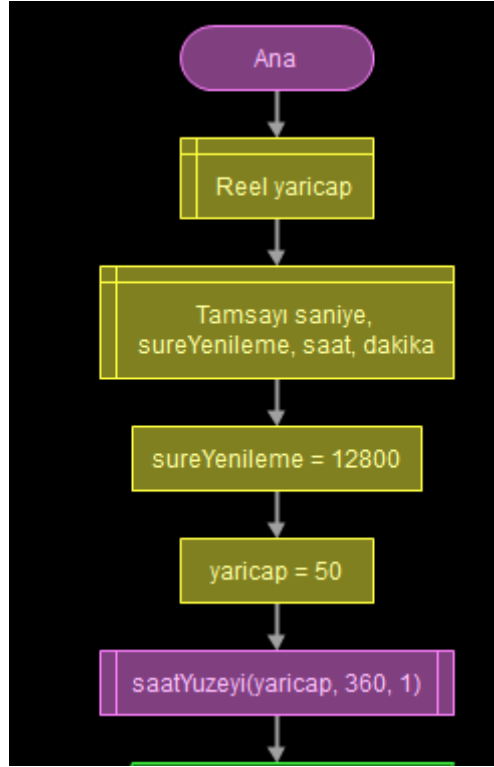
```
oyuncuHamlesi(tahta)
kazanan = kazananiKontrolEt(tahta, guncelOyuncu)
if kazanan:
    kimKazandi = "X"
    oyunaDevamMi = False
    turSayisi = 10
else:
    guncelOyuncu = "O"
    print("Şimdi bilgisayarın hamlesi")
    pcHamlesi(tahta)
    kazanan = kazananiKontrolEt(tahta, guncelOyuncu)
    if kazanan:
        kimKazandi = "O"
        oyunaDevamMi = False
        turSayisi = 10
    tahtayiCiz(tahta)
    if turSayisi < 9:
        turSayisi = turSayisi + 1
        oyunaDevamMi = True
        kimKazandi = "hiçkimse kazanamadı"
    else:
        oyunaDevamMi = False
        if kazanan:
            pass
        else:
            kazanan = False
    oyuncu = not oyuncu
if kazanan:
    if kimKazandi == "X":
        print("Kazandın")
    else:
        if kimKazandi == "O":
            print("pc kazandı")
else:
    print("Oyun berabere bitti")
```



Flowgorithm dosyasına ulaşmak için
karekodu okutunuz

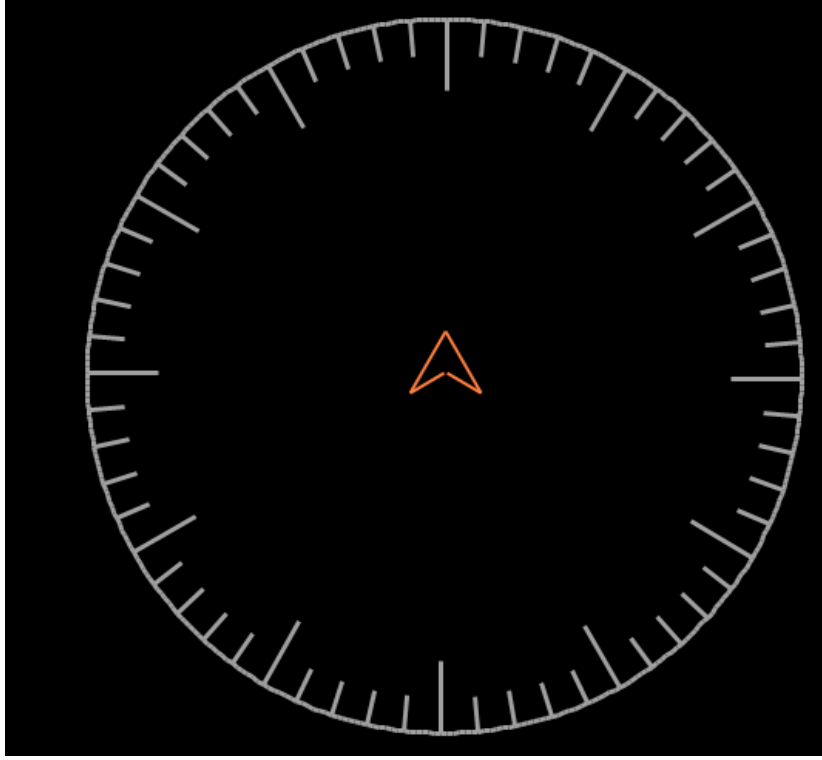
8.2.5 Ocak Saati Uygulaması

Bir sayacın belirlenen dakikadan geriye doğru akması ve sıfıra ulaştığında bir eylemi tetiklemesi prensibine dayanan bu uygulama, while döngülerinin pratik kullanımını göstermektedir. İhtiyacımız olan değişkenler, akış şemamızda Şekil 8-36'da görüldüğü gibi tanımlanmıştır.



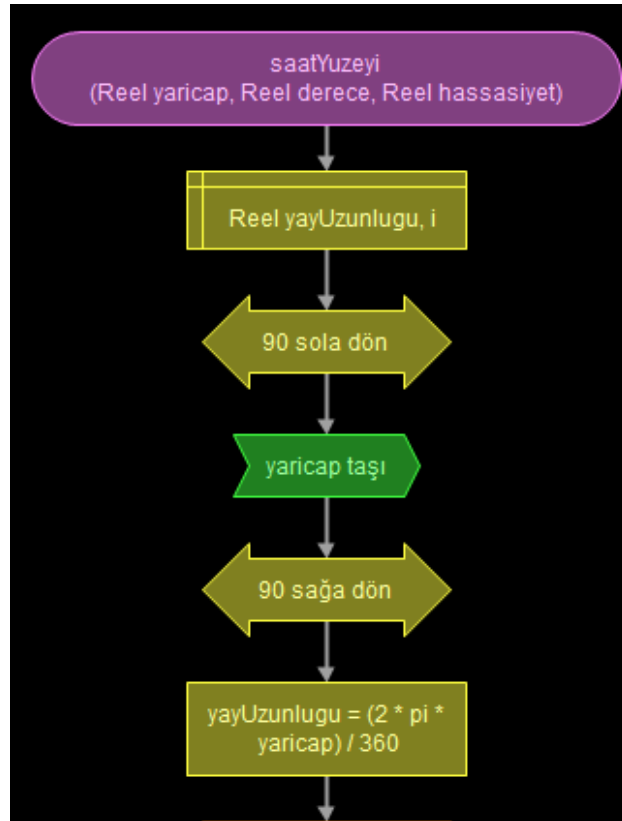
Değişkenler tanımlandıktan sonra, saat yüzeyini çizebilmek amacıyla saatYuzeyi(yaricap, derece, hassasiyet) isimli bir fonksiyon oluşturuldu. Bu fonksiyonu hemen kodlayarak saat yüzeyini oluşturmaya başlayabilir ve programımızı başlatabiliriz.

Şekil 8-36 Ana fonksiyon 1. kısım



Şekil 8-37 Çizdirilmek istenen saat tasarımı

Saatimizin tasarımını, Şekil 8-37’de gösterildiği gibi oluşturmak istiyoruz. İmlecimiz başlangıç konumunda saat yüzeyinin tam ortasında yer almakta ve ileri doğru bakacak şekilde ayarlanmıştır.

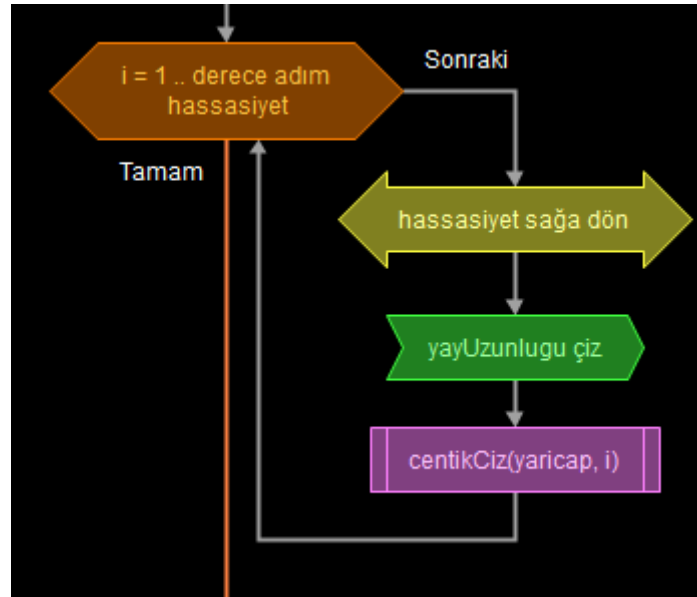


Şekil 8-38 Saat yüzeyini çizen algoritma 1. kısım

Gerekli değişkenler Şekil 8-38'deki şemada tanımlandıktan sonra, imlecimizi başlangıç noktasında sola doğru döndürerek yarıçap uzunluğu kadar ilerlettik. Daha sonra imleci tekrar ileri doğru döndürerek çemberi çizmeye hazır hale getirdik.

Çemberdeki iki derece arasında toplamda 60 yay uzunluğu bulunmaktadır. Bu uzunlukları hesaplayabilmek için çemberin çevresini 360'a bölerek yayUzunlu değişkeninin değerini belirledik.

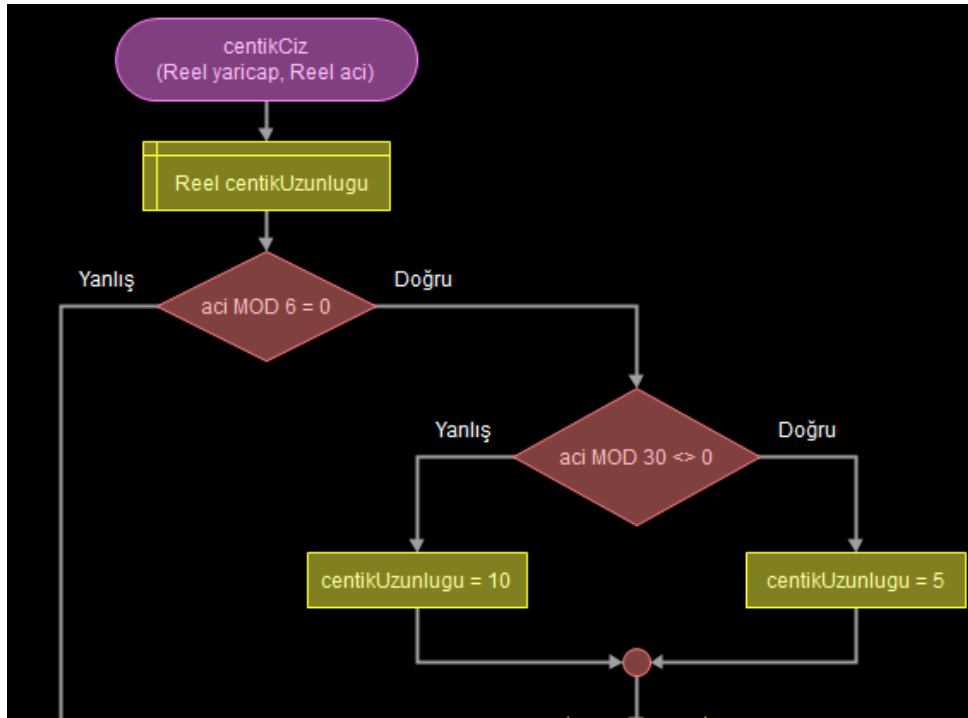
Çemberin çevresi ise, $\text{Çevre} = 2 \times \pi \times r$ formülü ile hesaplanmaktadır.



Şekil 8-39 Saat yüzeyini çizen algoritma 2. kısım

Şekil 8-38’de verilen şemada bir for döngüsü tanımlanmıştır. Bu döngü, 1’den başlayarak, girdiğimiz derece değişkeninin değeri kadar, yani 360 kez tekrar edecektir. Ana programda hassasiyet değişkeninin değeri 1 olarak tanımlandığından, döngü her adımda 1 derece sağa dönüp, hesaplanan yayUzunlugu kadar çizim yapacaktır.

Ancak bu şekilde yalnızca düz bir çember çizmiş oluruz. Çember üzerindeki çentikleri oluşturmak için centikCiz fonksiyonunu (Şekil 8-40) tanımlamamız gerekmektedir.



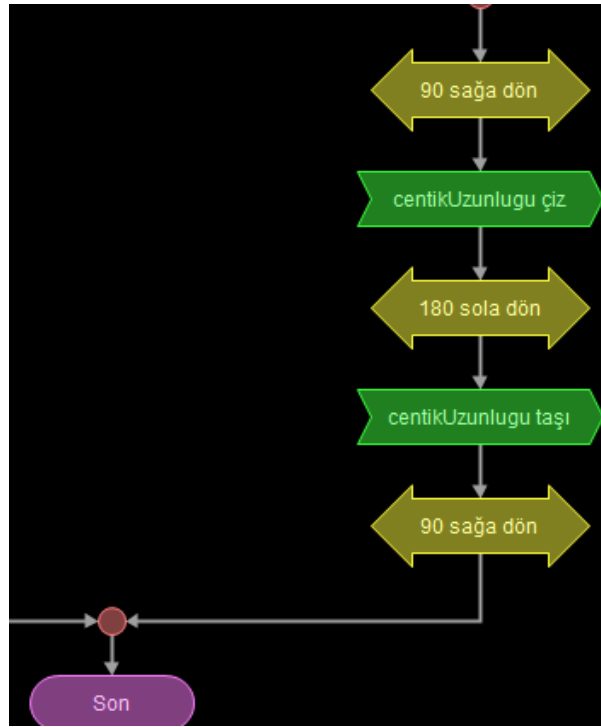
Şekil 8-40 Saat yüzeyindeki çentikleri çizen algoritma 1. kısım

İlk olarak centikUzunlugu isimli bir değişken tanımladık. Bu değişken, saat 1–12 arasındaki sayıları gösteren çizgilerde 10 birim, ara değerlerde ise 5 birim uzunluğunda olacak şekilde ayarlanacaktır.

saatYuzeyi fonksiyonundaki for döngüsünde, döngüdeki her adımı i değişkeni ile göstermiştik. Bu i değişkenini, centikCiz fonksiyonunun aci parametresine atayarak fonksiyonu çağırdık. Böylece aci değişkeni 1’den 360’a kadar değer alacaktır.

Ancak her açı için çentik çizmek istemiyoruz. Bunun için bir if kontrolü ile yalnızca 6’nın katı olan derecelerde çentik çizilmesini sağladık ($\text{aci} \bmod 6 = 0$). Ayrıca saat 1–12’yi temsil eden uzun çizgiler için, 30’un katı olan açılarda centikUzunlugu değerini 10, diğer açılarda 5 olacak şekilde ayarladık ($\text{aci} \bmod 30 \leq 0$).

Gerekli ayarlamaları yaptıktan sonra çentik çizimine geçebiliriz.



saatYuzeyi fonksiyonundaki döngü sırasında, i değeri yalnızca 6'nın katı olduğunda çizim işlemi gerçekleşecektir. İmlecimiz düz ilerlerken sağa dönecek ve i değeri 6'nın katı ise ara çizgi olarak 5 birim, 30'un katı ise saatleri gösteren uzun çizgi olarak 10 birim uzunluğunda centikUzunlugu kalem aşağı konumundayken çizilecektir. Çizim tamamlandıktan sonra imleç, çizilen uzunluk kadar geri gidip kalem yukarı konumunda ilerleyecek ve tekrar sağa dönerek saatin çemberini çizme işlemine devam edecektir. Süreç sonrası oluşan algoritma Şekil 8-41'deki gibi olmaktadır.

Şekil 8-41 Saat yüzeyindeki çentikleri çizen algoritma 1. kısım

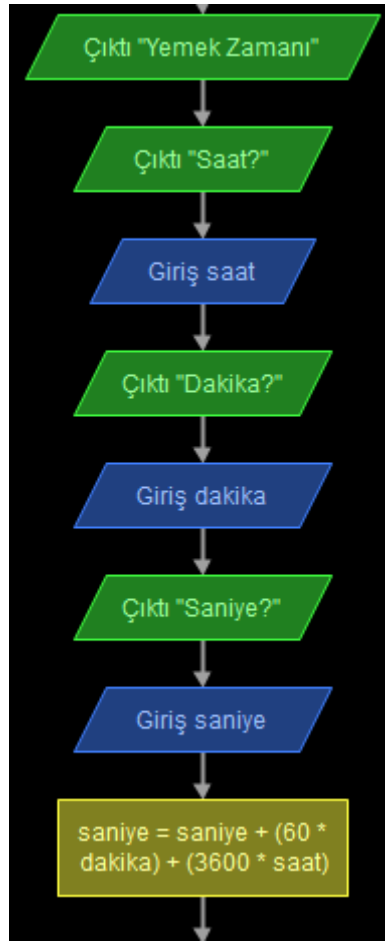
Çentik çizme işlemi tamamlandıktan sonra saatYuzeyi fonksiyonumuza geri dönerek, fonksiyonun kalan işlemlerini Şekil 8-42'deki gibi tamamlayabiliriz.



Yazdığımız for döngüsü, 360 kez tekrarlanarak çember ve çentik çizimlerini tamamlayacaktır. Ancak döngü sona erdiğinde imleç, çemberi çizmeye başladığı konumda kalmaktadır. İmleci tekrar çemberin merkezine taşımak için, sağa döndürme, kalemi yukarı konuma alma, yarıçap uzunluğu kadar ileri gitme ve sola dönme bloklarını uygun şekilde ekleyerek saatYuzeyi fonksiyonumuzu tamamlıyoruz.

Şekil 8-42 Saat yüzeyini çizen algoritma 3. kısım

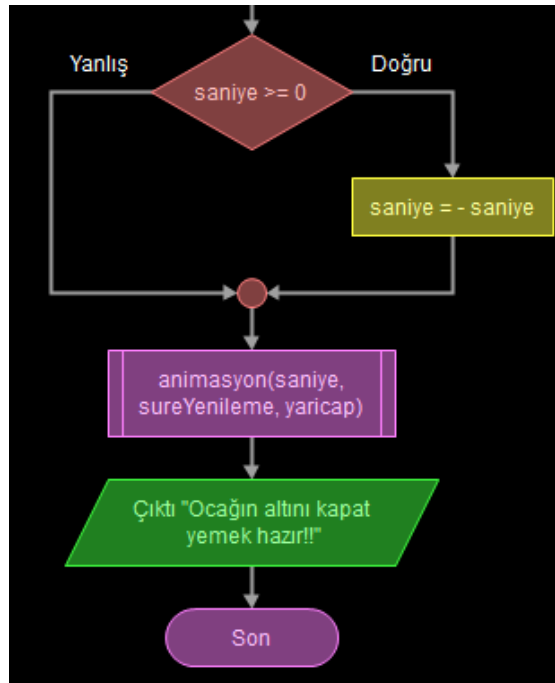
Saat yüzeyini çizdikten sonra, imleci tekrar ana fonksiyona (Şekil 8-43) yönlendiriyoruz.



Bu bir ocak saati uygulaması olduğundan, kullanıcıdan ocağın ne kadar süreyle açık kalması gerektiğini öğrenmemiz gerekiyor. Bu süreyi saat, dakika ve saniye olarak ayrı ayrı aldıktan sonra, en sonunda tüm süreyi saniye cinsine çevirerek işlemlerimize devam edeceğiz. Kullanıcıdan gerekli değerleri aldıktan sonra, dakika değerini 60 ile, saat değerini ise 3600 ile çarparak ve ardından çıkan sonuçlara girilen saniye değerini ekleyerek, toplam süreyi saniye cinsinden hesaplamış oluruz.

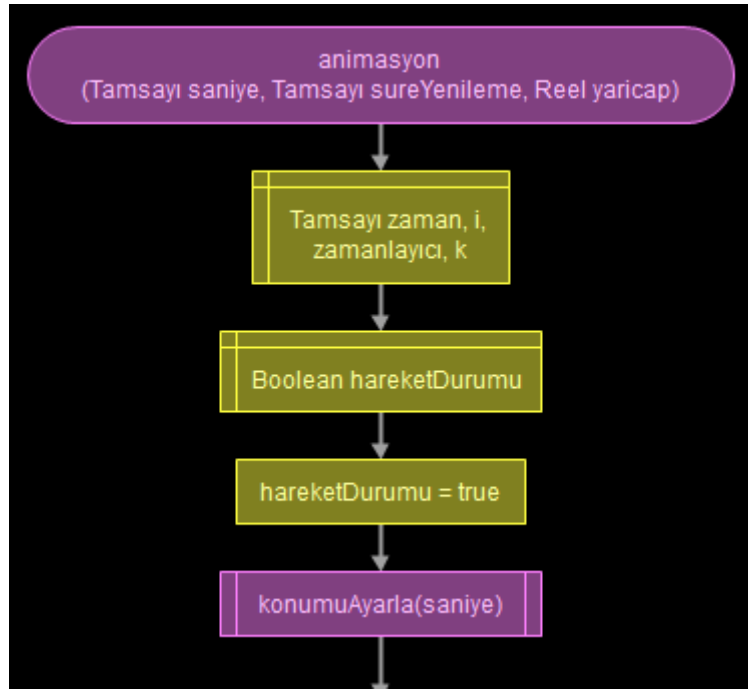
Bilgi:

- 1 saat = 3600 saniye
- 1 dakika = 60 saniye



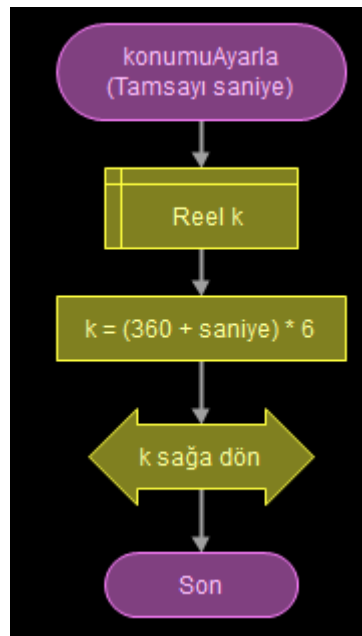
Kullanıcıdan gerekli bilgileri alıp süreyi saniye cinsinden hesapladıktan sonra, programın animasyonunun saniye = 0 olduğunda sona ermesini istiyoruz. Bir if kontrolü ile, saniye değeri 0'dan büyükse, değerini negatif olacak şekilde işaretledik. Örneğin, kullanıcı 60 saniye girerse, bu değeri -60 olacak şekilde değiştirdik. Artık animasyon(saniye, sureYenileme, yaricap) fonksiyonumuzu (Şekil 8-44) yazmaya hazırız.

Şekil 8-43 Ana fonksiyon 2. kısım



Şekil 8-44 saat animasyonunu gerçekleştiren algoritma 1. kısım

Gerekli değişkenleri tanımladıktan sonra, `konumuAyarla(saniye)` isimli fonksiyonumuzu çağırarak devam edelim. Bu fonksiyonu çağırmamızın nedeni, imlecin başlangıçta merkezde ve ileriye bakar konumda durmasıdır. İmleci, geçeceği çentiğin yönüne doğru döndürmek için `konumuAyarla(saniye)` fonksiyonunu kullanıyoruz. Şimdi, bu fonksiyonu (Şekil-45) tanımlayalım.



Şekil 8-45 Saat imlecinin konumunu ayarlayan algoritma

Fonksiyonumuzu oluşturmaya, bir `k` değişkeni tanımlayarak başlayalım. İmlecimiz, `k` değişkeninin değeri kadar sağa doğru dönecek. Örnekle açıklayacak olursak:

kullanıcı 15 saniyelik bir süre girmiş olsun. Bu durumda imlecimiz saat 9 yönüne bakmalıdır. Ana fonksiyonumuzda bu değeri -15 olarak ayarlamıştık.

Hesaplama süreci:

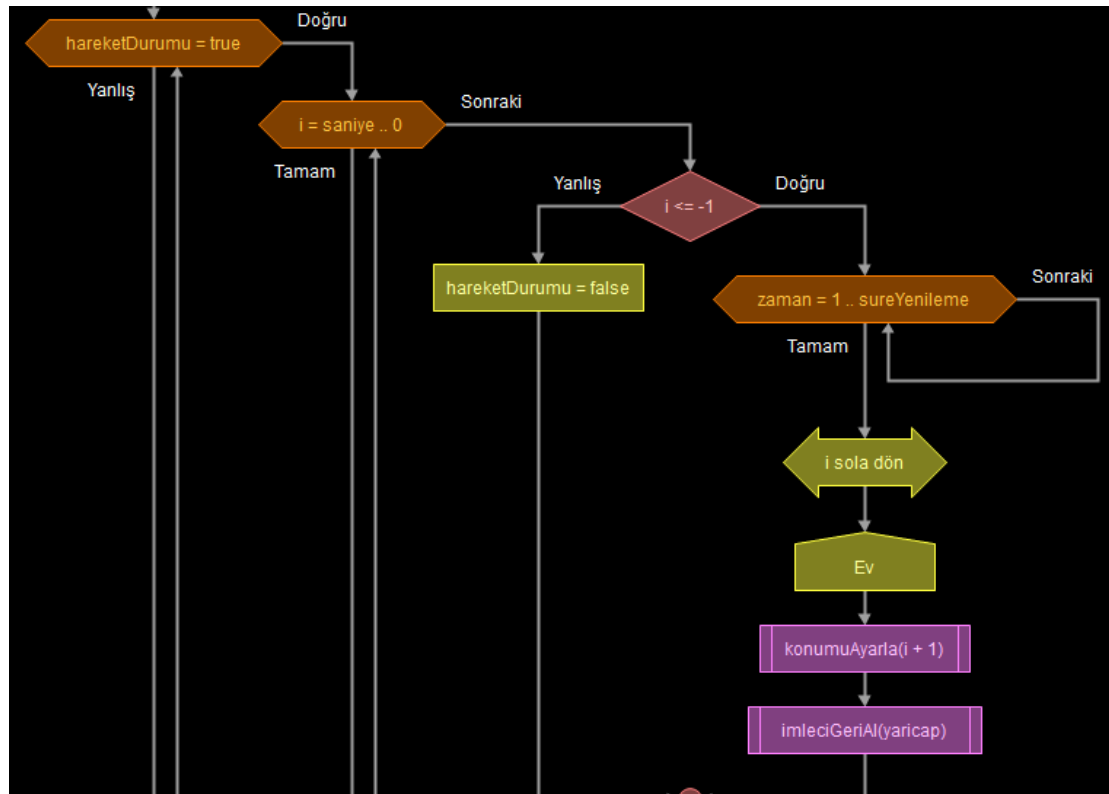
$$k = 360 + (-15) = 345$$

$$345 \times 6 = 2070$$

Bir tam tur 360 derece olduğuna göre, $2070 = 360 \times 5 + 270$ olarak yazılabilir.

Bu, imlecimizin kendi etrafında sağa doğru 5 tam tur döneceği ve 6. turunda 270 derecelik açı yaparak duracağı anlamına gelir. Sonuç olarak, imlecimiz tam olarak istediğimiz saat 9 yönüne bakmış olur.

İmlecimizin başlangıç yönünü ayarladığımıza göre, animasyonumuza devam edebiliriz.



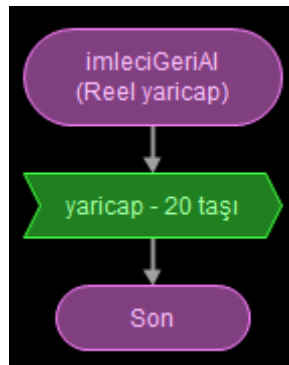
Şekil 8-46 saat animasyonunu gerçekleştiren algoritma 2. kısım

İlk olarak, while döngüsü tanımlayarak (Şekil 8-46) başladık. Bu döngü, boolean türündeki *hareketDurumu* değişkenimizin değeri true olduğu sürece çalışacaktır. While döngüsünün içine bir for döngüsü ekledik. Bu döngü, ana programdan gelen ve değerini negatif olarak aldığımız *saniye* değerinden başlayıp 0'a kadar devam edecektir.

For döngüsünün içine bir if kontrol fonksiyonu yerleştirdik. Eğer i değeri -1'e eşit veya küçükse, imlecimiz hareket etmeye devam edecektir. If fonksiyonu içindeki boş for döngüsü ise 1'den başlayarak ana fonksiyonumuzdan gelen *sureYenileme* değişkeninin değerine kadar çalışacaktır. *sureYenileme* değişkeninin değerini 12.800 olarak belirlemiştik. Bu sayede imlecimiz, iki çentik arasını yaklaşık bir saniyede kat etmektedir. Eğer çemberin yarıçapı 100'den büyük olursa, *sureYenileme* değişkeninin değerini 2.500 olarak ayarlamak gerekir.

İmlecin hareket animasyonunu gerçekleştirmek için imleci i değeri kadar sola döndürüyor ve tekrar başlangıç konumuna getiriyoruz. Ardından, *konumuAyarla* fonksiyonunda i değerine 1 ekleyerek imlecin bir birim ilerlemesini sağlıyoruz. Program bu şekilde çalıştırıldığında, imlecin çember üzerinde düzgün bir şekilde hareket ettiğini görürüz.

İmleci çember üzerinde değil, çemberin biraz daha içinde ve doğru çentiklerin altında hareket ettirmek için ise *imleciGeriAl* fonksiyonunu çağırıyoruz. Şimdi bu fonksiyonu tanımlayalım.

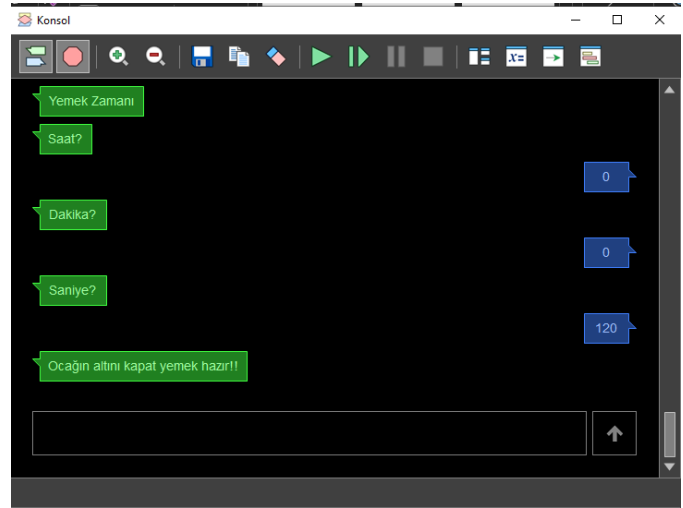
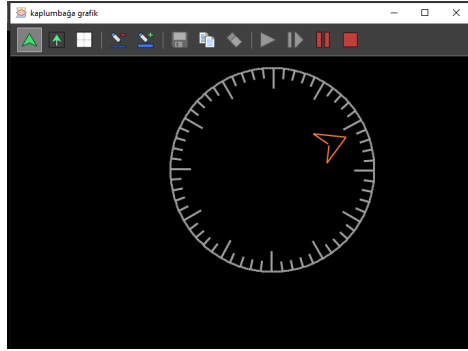


Bu fonksiyon Şekil (8-47) ile amacımız, imleci yarıçapın 20 birim eksiği kadar geri almaktır. *yaricap* değerinden 20 çıkararak fonksiyonumuzu tanımlamış olduk.

Şekil 8-47 İmleci merkeze taşıyan algoritma

Son olarak, program çalışıp süre tamamlandığında kullanıcıyı uyarmak amacıyla ana fonksiyonun sonuna “Ocağın altını kapat, yemek hazır” şeklinde bir çıktı ekledik ve böylece program tamamlanmış oldu.

Program çalıştığında, konsol ekranı ve grafik pencere aşağıdaki gibi (Şekil 8-48) görünmektedir.



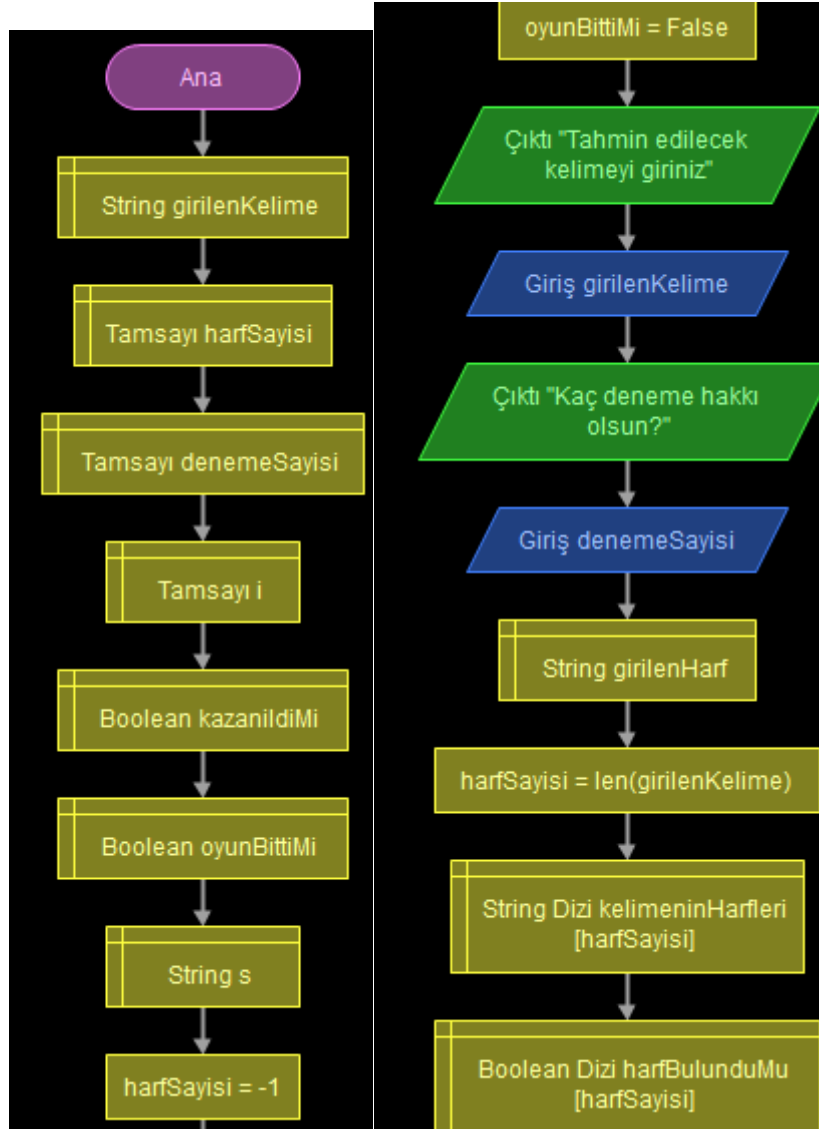
Şekil 8-48 Programın genel çıktısı



Flowgorithm dosyasına ulaşmak için
karekodu okutunuz

8.2.6 Adam Asmaca Oyunu

Okul yıllarında kâğıt kalemle oynadığımız o meşhur oyunu dijital dünyaya taşıma zamanı! Bu projede, gizli kelimeyi harf harf tahmin ederken hem çocukluğumuza dönecek hem de programlama kaslarımızı çalıştıracacağız. İhtiyacımız olan değişkenleri tanımlayarak (Şekil 8-49) oyunumuzu başlatıyoruz.



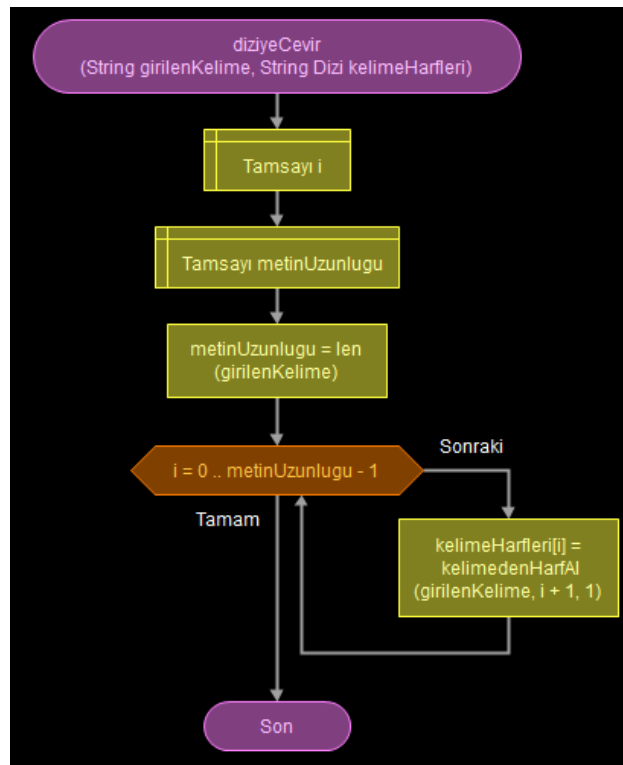
Şekil 8-49 Program için gerekli olan değişken isimleri

Oyunumuzda, seçilen kelimeyi ve kullanıcıya kaç kez yanlış yapma hakkı verileceğini belirlemek için girilenKelime ve denemeSayisi değişkenlerinin

değerlerini kullanıcıdan aldık. `harfSayisi` değişkenimizin değerini, `girilenKelime` değişkeninin uzunluğu yani karakter sayısı kadar olacak şekilde belirledik.

`kelimeninHarfleri` dizisinin boyutunu `harfSayisi` değişkeni belirleyecek şekilde ayarladık. Son olarak, boolean türündeki `harfBulunduMu` dizisinin boyutunu da yine `harfSayisi` değişkenine göre tanımladık.

Gerekli ayarlamaları yaptıktan sonra, kullanıcının girdiği kelimenin harflerini dizinin elemanlarına aktaran fonksiyonumuzu akış şemamıza ekliyor ve fonksiyonumuzu tanımlamaya Şekil 8-50 ile başlıyoruz.

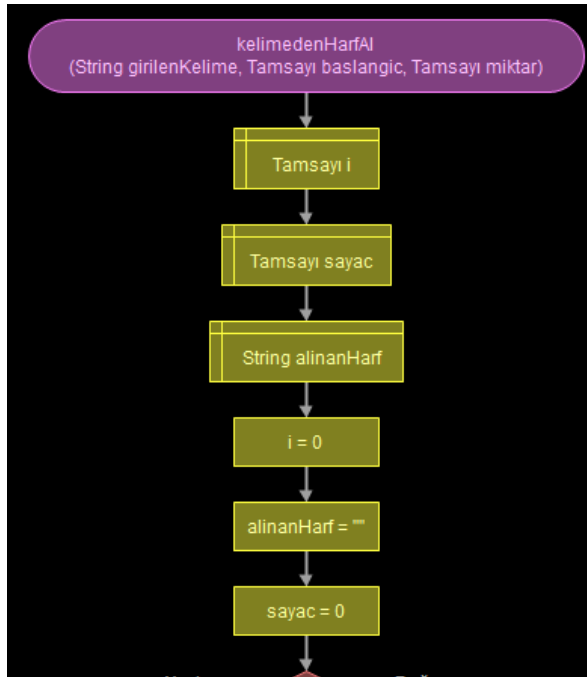


Şekil 8-50 Kullanıcının girdiği kelimedeki harfleri diziye toplayan fonksiyon

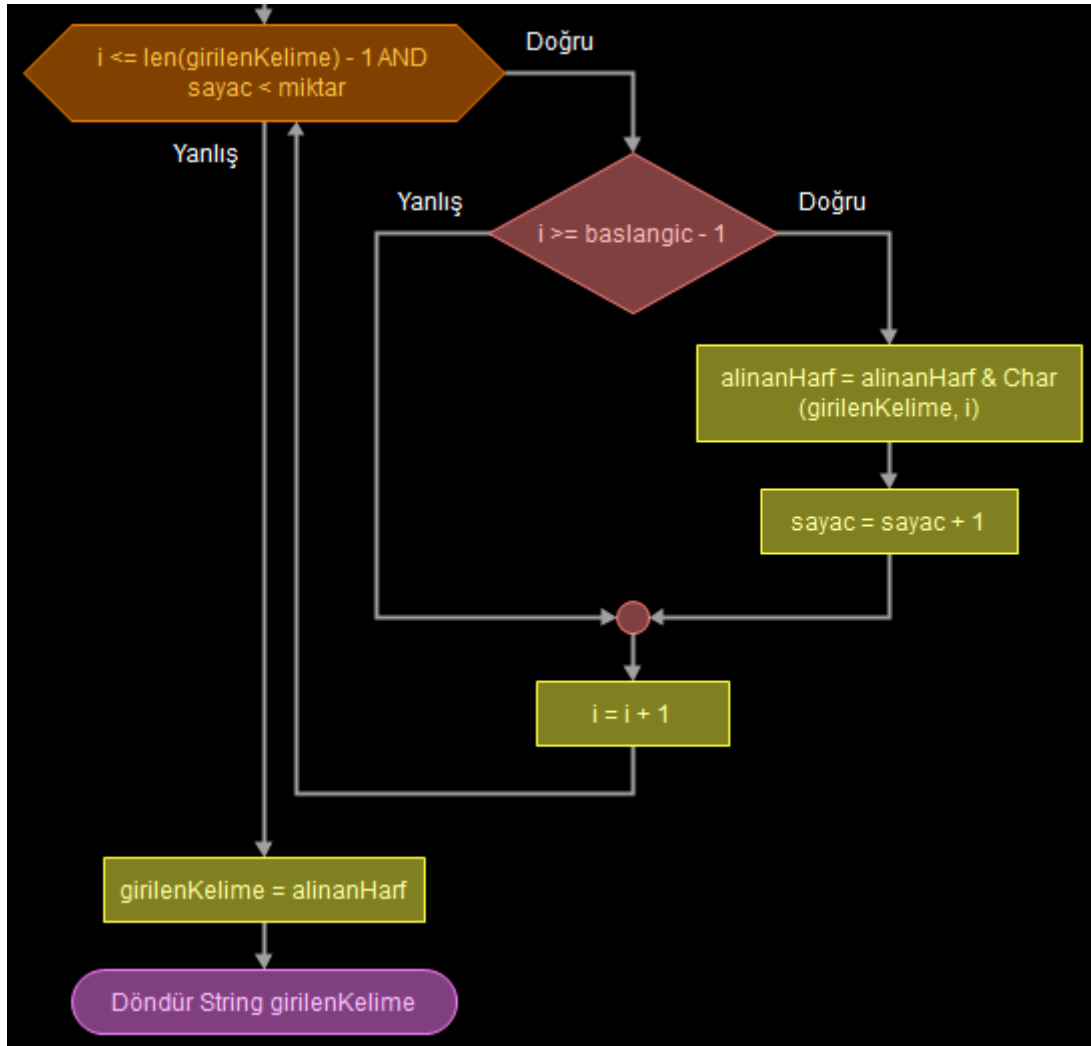
`diziyeCevir` fonksiyonunda ilk olarak `metinUzunlugu` isimli bir değişken tanımladık. Bu değişkenin değerini, `girilenKelime` değişkenindeki harf sayısına eşit olacak şekilde ayarladık.

Daha sonra bir for döngüsü başlattık. Bu döngü, 0'dan başlayarak `girilenKelime` uzunluğunun 1 eksiğine kadar devam edecektir. Ana fonksiyonumuzda tanımladığımız `kelimeHarfleri` dizisine, `girilenKelime` harflerini eleman olarak göndermek için bu döngüyü kullanacağız.

Girilen kelimenin harflerini almak amacıyla `kelimededenHarfAl` fonksiyonunu kullandık. Şimdi fonksiyonumuzu tanımlayarak devam edelim.



Şekil 8-51’de görüldüğü üzere, gerekli değişkenleri tanımlayarak fonksiyonu yazmaya başlıyoruz.



Şekil 8-51 kelimedeki harfleri dizinin elemanı yapan fonksiyon

Süreci, bir while döngüsü tanımlayarak başlatıyoruz. Bu döngü, girilenKelime değişkeninin harf sayısının bir eksiğine ulaşana kadar ve diziOlustur fonksiyonuna miktar değişkenini '1' olarak gönderdiğimiz için sayaç 1'den küçük olduğu sürece çalışmaya devam edecektir.

Döngü içerisine bir if kontrol bloğu ekledik. Eğer i değeri, bir önceki i değerinden büyükse; girilenKelime değişkeninin o anki sırasındaki elemanı, alınanHarf değişkenine atanacaktır.

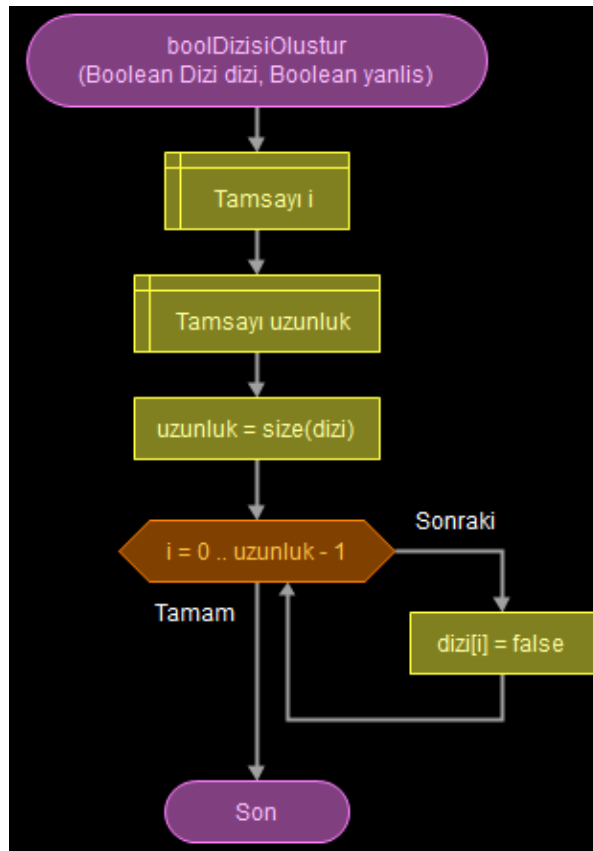
Bu işlemleri bir örnek üzerinden somutlaştıralım:

Kullanıcının girdiği kelimenin 'Selim' olduğunu varsayalım. Döngümüz \$i=0\$ ile başladığı için, kelimenin 0. indisindeki eleman, yani ilk harf olan 's' işleme alınır. Böylece alınanHarf değişkeni 's' değerini alır. Aynı zamanda diziyeCevir fonksiyonunda yaptığımız tanımlama sayesinde kelimeHarfleri[0] da 's' değerine eşitlenir.

'Selim' kelimesi 5 harfli olduğu için döngü 5 kez tekrar edecek; 0, 1, 2, 3 ve 4 indisli elemanları şu şekilde oluşturacaktır:

- kelimeHarfleri[0] = "s"
- kelimeHarfleri[1] = "e"
- kelimeHarfleri[2] = "l"
- kelimeHarfleri[3] = "i"
- kelimeHarfleri[4] = "m"

Böylece kullanıcının girdiği kelimenin harflerini tanımlamış olduk. Şimdi sırada, oyuncunun tahmin ettiği harfin doğruluğunu kontrol etmek var. Bu işlem için boolean türünde harfBulunduMu isminde bir dizi tanımlamıştık. Oyun başladığında bu dizinin tüm elemanlarına varsayılan olarak false değerini atamamız gerekiyor. Bunun için boolDizisiOlustur fonksiyonunu ana bloğumuzda çağırılım ve fonksiyonu oluşturmaya başlayalım.



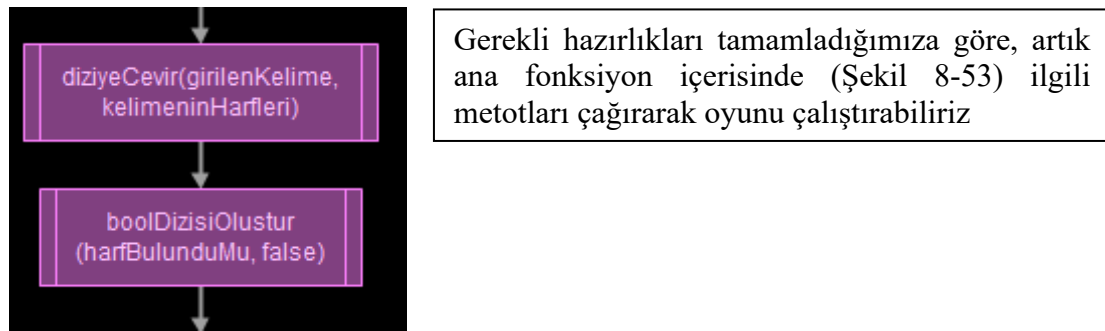
Şekil 8-52 Kullanıcının girdiği harfin doğruluğunu kontrol eden fonksiyon

Şekil 8-52’de görüldüğü üzere, bu fonksiyon parametre olarak bizden bir dizi ve bir boolean değer beklemektedir. Ana fonksiyonumuzda bu metodu çağırırken harfBulunduMu dizisini kullanmıştık. Hatırlayacağınız gibi, bu dizinin boyutu girilenKelime değişkeninin harf sayısına (uzunluğuna) eşitlenmişti.

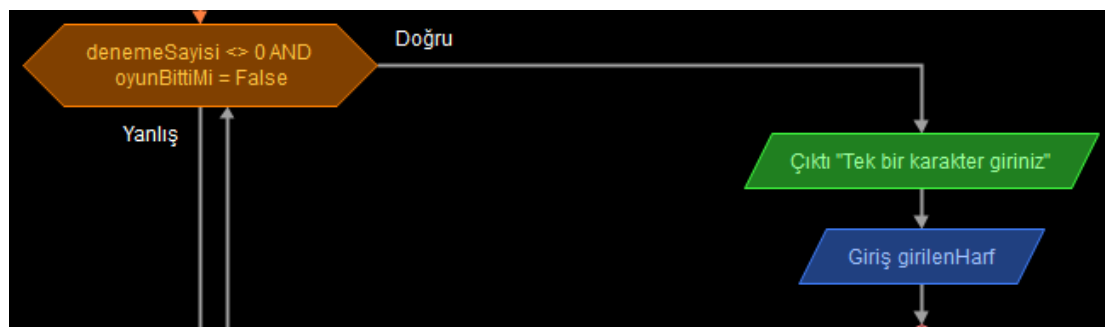
Kod bloğundaki for döngüsü, 0’dan başlayıp harf sayısının bir eksiğine kadar devam edecek ve dizimizin bütün indis değerlerini false olarak güncelleyecektir.

Örneğin; 4 harfli bir kelime girildiğinde, oluşan değerler aşağıdaki gibi olacaktır:

- harfBulunduMu[0] = false
- harfBulunduMu[1] = false
- harfBulunduMu[2] = false
- harfBulunduMu[3] = false



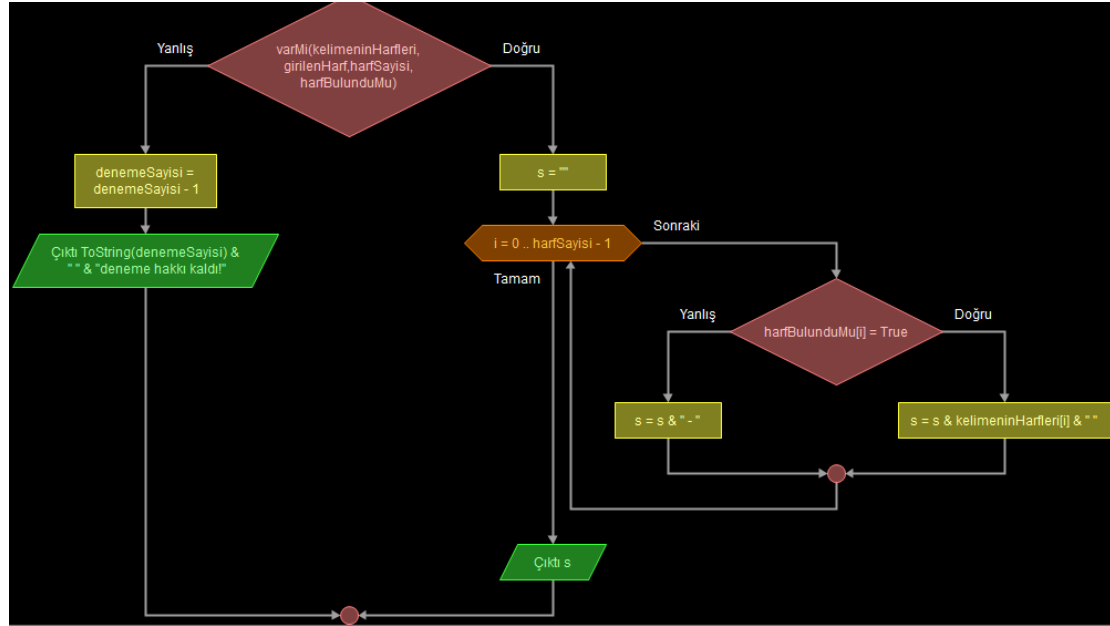
Şekil 8-53 Ana fonksiyon 2. kısım



Şekil 8-54 Ana fonksiyon 3. kısım

Süreci bir while döngüsü kurarak (Şekil 8-54) başlatıyoruz. Bu döngü, denemeSayisi 0’a düşmediği ve oyunBittiMi değişkeni henüz true olmadığı (yani oyun sonlanmadığı) sürece çalışmaya devam edecektir.

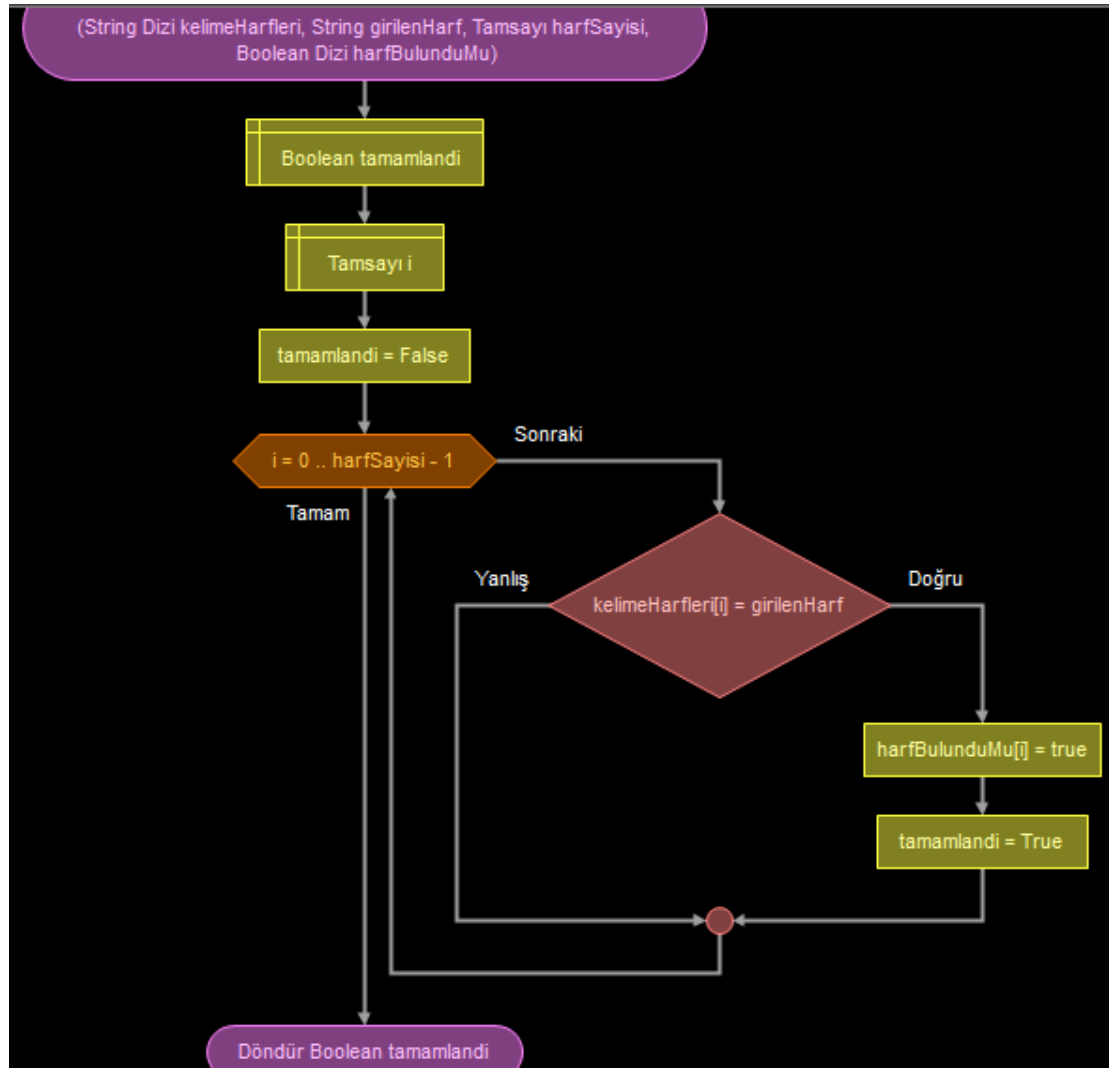
Döngü bloğunun içerisinde; kullanıcıya gerekli yönlendirmeyi yaparak, tahmin ettiği harfi girmesini sağlıyor ve bu girdiyi girilenHarf değişkenine atıyoruz.



Şekil 8-55 Ana fonksiyon 4. kısım

Kullanıcı tarafından girilen harfin, önceki aşamalarda oluşturduğumuz kelimeHarfleri dizisinde bulunup bulunmadığını kontrol etmek için bir if bloğu açıyoruz.

Kontrol şartı olarak varMi isimli fonksiyonumuzu (Şekil 8-55) çağıracağız. Bu fonksiyon; parametre olarak kelimeHarfleri, girilenHarf, harfSayisi ve harfBulunduMu değişkenlerine ihtiyaç duymaktadır. if kontrol yapısının detaylarına girmeden önce, gelin varMi fonksiyonunu Şekil 8-56’da oluşturalım.



Şekil 8-56 Kullanıcının kalan hakkını kontrol eden fonksiyon

Her zamanki gibi, fonksiyonumuzu gerekli değişkenleri tanımlayarak başlatıyoruz. Tanımlamaların ardından, 0'dan başlayıp harfSayisinin bir eksiğine kadar devam edecek bir for döngüsü kuruyoruz.

Döngü içerisinde bir if kontrol bloğu kullanarak şu sorguyu yapıyoruz: Eğer kullanıcının girdiği harf (girilenHarf), kelimeHarfleri dizisinin o anki sırasındaki (i) elemanına eşitse; harfBulunduMu dizisinin ilgili indisini true olarak güncelliyoruz.

Örnek: Kelimemiz 'Selim', kullanıcının girdiği harf ise 's' olsun.

kelimeHarfleri[0] değeri 's' olduğundan, eşleşme sağlanır ve harfBulunduMu[0] değeri true olur. Böylece varMi fonksiyonumuz, işlemin sonunda true değerini döndürür.

Şimdi ana fonksiyonumuza (Şekil 8-57) dönelim.

Harf Bulunamazsa: varMi fonksiyonu false dönerse, denemeSayisi 1 azaltılır ve kalan hak bilgisi konsola yazdırılır.

Harf Bulunursa: Fonksiyon true dönerse, başlangıçta _ _ _ _ _ şeklinde gizli olan kelimeyi güncellememiz gerekir.

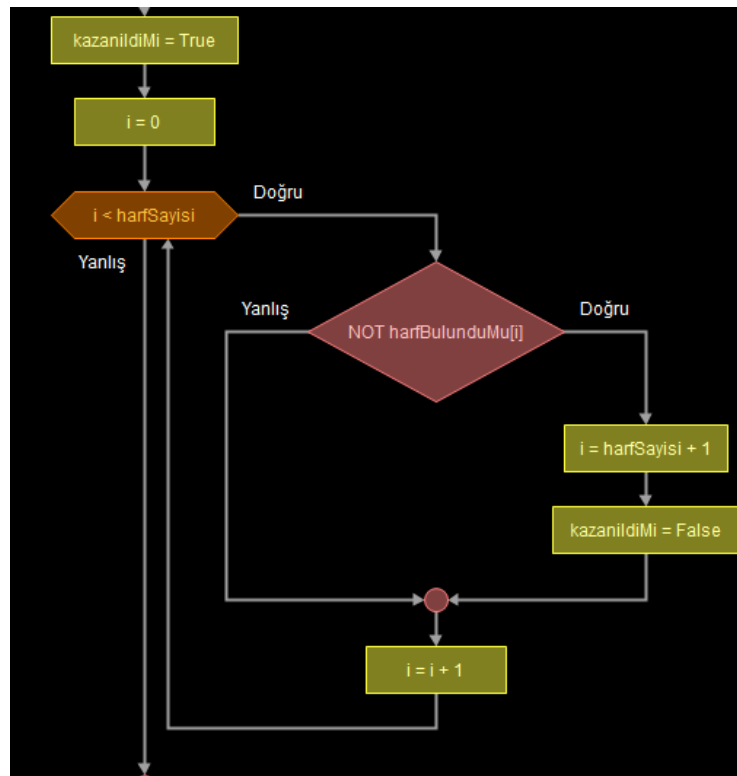
Bu güncelleme işlemi için String türündeki s değişkenimizi (görüntülenen kelimeyi) önce boş ("") olarak sıfırlıyoruz. Ardından bir for döngüsü ile tüm indisleri tarıyoruz:

Eğer harfBulunduMu[i] true ise (yani harf bilindiyse), s değişkenine gerçek harfi (kelimeHarfleri[i]) ekliyoruz.

Eğer false ise, s değişkenine tire (-) ekleyerek harfi gizli tutuyoruz.

Örneğimizde 's' harfi bulunduğu için 0. indis açılır, diğerleri kapalı kalır. Sonuçta konsol ekranına şu çıktı verilir:

S----



Şekil 8-57 Ana fonksiyon 5. kısım

Oyunun kazanılıp kazanılmadığını tespit etmek için boolean türünde kazanildiMi adında bir değişken tanımlamıştık. Kontrol mantığımız şu şekilde işleyecek:

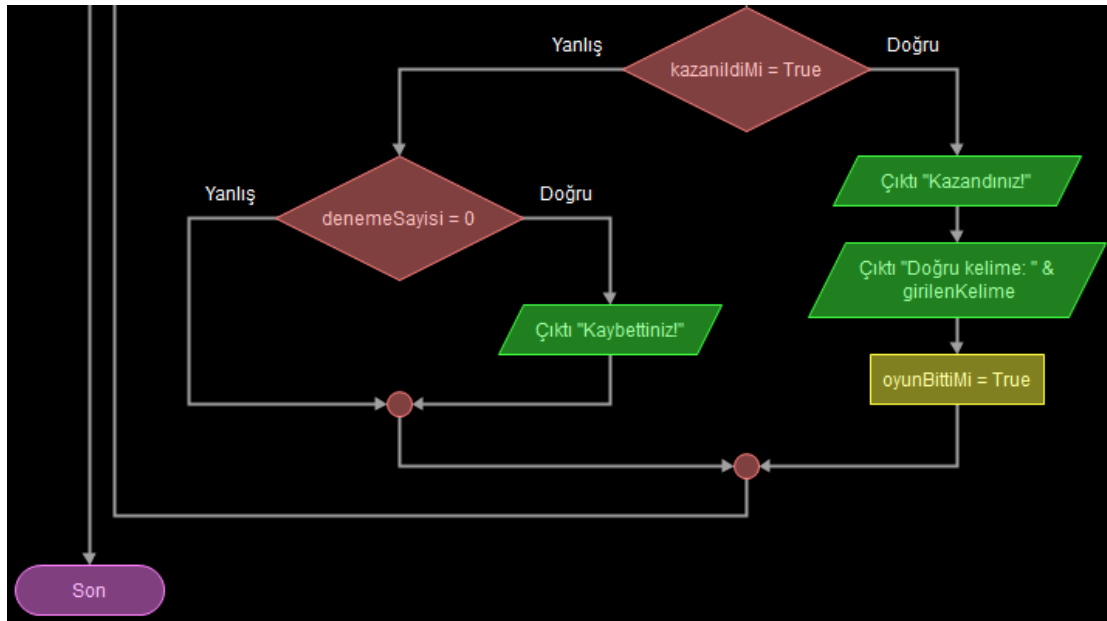
Oyuncunun oyunu kazandığını varsayarak başlayacağız, eğer tek bir harf bile eksikse bu varsayımı bozacağız.

Bu nedenle kazanildiMi değişkenine true, sayaç değişkenimiz olan i'ye ise 0 değerini atayarak ilerliyoruz.

Bir while döngüsü başlatıyoruz. Bu döngü, \$\$ değeri kelimenin harf sayısından küçük olduğu sürece çalışacak. Döngü içerisinde bir if bloğu ile harfBulunduMu dizisinin o anki elemanını (harfBulunduMu[i]) kontrol ediyoruz.

Eğer dizide 'false' değerine sahip bir elemana rastlarsak (yani henüz bulunmamış bir harf varsa), oyun henüz bitmemiş demektir. Bu durumda kazanildiMi değişkenini false olarak güncelliyoruz.

Döngünün sürekliliğini sağlamak ve bir sonraki harfi kontrol etmek için, if bloğunun dışında i değerini 1 artırarak ($i = i + 1$) işlemlerimize devam ediyoruz.



Şekil 8-58 Ana fonksiyon 6. kısım

Artık kodlamamızın son ve en heyecanlı kısmına geldik. Bir if kontrol bloğu (Şekil 8-58) açarak oyunun sonucunu değerlendiriyoruz.

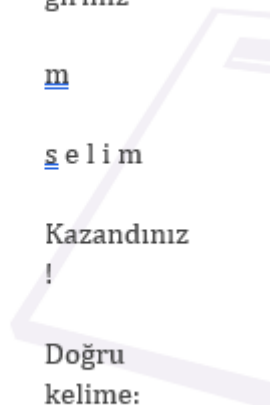
Eğer kazanildiMi değişkeninin değeri true ise; kullanıcıya 'Kazandınız' mesajını ve doğru kelimeyi gösteriyoruz. Ardından oyunBittiMi değişkenini true yaparak, ana programda kurduğumuz ilk while döngüsünü sonlandırıyor ve oyunu başarıyla tamamlıyoruz.

Diğer senaryoda; eğer kazanildiMi değeri false ise ve denemeSayisi 0'a düşmüşse, kullanıcıya 'Kaybettiniz' mesajını iletiyoruz. Bu durumda denemeSayisi artık 0

olduğu için, ana döngümüzün '*deneme sayısı 0 olmadığı sürece*' şartı bozuluyor ve döngü otomatik olarak sonlanıyor.

Böylece oyunumuzun kodlamasını tamamlamış oluyoruz.

Programın çalışması tamamlandığında, konsol ekranı aşağıdaki gibi görünecektir.



Tahmin edilecek kelimeyi giriniz	Tek bir karakter giriniz	Tek bir karakter giriniz	Tek bir karakter giriniz
<u>selim</u>	<u>s</u>	<u>l</u>	<u>m</u>
	<u>s</u> - - -	<u>s</u> e l - -	<u>s</u> e l i m
Kaç deneme hakkı olsun?	Tek bir karakter giriniz	Tek bir karakter giriniz	Kazandınız !
3	<u>e</u>	<u>i</u>	Doğru kelime: selim
	<u>s</u> e - - -	<u>s</u> e l i -	

8.2.7 Programın Python Çıktısı

```
def boolDizisiOlustur(dizi, yanlis):
```

```
    uzunluk = len(dizi)
```

```
    for i in range(0, uzunluk - 1 + 1, 1):
```

```
        dizi[i] = False
```

```
def diziyeCevir(girilenKelime, kelimeHarfleri):
```

```
    metinUzunlugu = len(girilenKelime)
```

```
    for i in range(0, metinUzunlugu - 1 + 1, 1):
```

```
        kelimeHarfleri[i] = kelimededenHarfAl(girilenKelime, i + 1, 1)
```

```
def kelimedendenHarfAl(girilenKelime, baslangic, miktar):
```

```
    i = 0
```

```
    alinanHarf = ""
```

```
    sayac = 0
```

```
    while i <= len(girilenKelime) - 1 and sayac < miktar:
```

```
        if i >= baslangic - 1:
```

```
            alinanHarf = alinanHarf + girilenKelime[i]
```

```
            sayac = sayac + 1
```

```
        i = i + 1
```

```
    girilenKelime = alinanHarf
```

```
    return girilenKelime
```

```
def varMi(kelimeHarfleri, girilenHarf, harfSayisi, harfBulunduMu):
```

```
    tamamlandi = False
```

```
    for i in range(0, harfSayisi - 1 + 1, 1):
```

```
        if kelimeHarfleri[i] == girilenHarf:
```

```
            harfBulunduMu[i] = True
```

```
            tamamlandi = True
```

```
    return tamamlandi
```

```
# Main
```

```
harfSayisi = -1
```

```
oyunBittiMi = False
```

```
print("Tahmin edilecek kelimeyi giriniz")

girilenKelime = input()

print("Kaç deneme hakkı olsun?")

denemeSayisi = int(input())

harfSayisi = len(girilenKelime)

kelimeninHarfleri = [""] * (harfSayisi)

harfBulunduMu = [False] * (harfSayisi)

diziyeCevir(girilenKelime, kelimeninHarfleri)

boolDizisiOlustur(harfBulunduMu, False)

while denemeSayisi != 0 and oyunBittiMi == False:

    print("Tek bir karakter giriniz")

    girilenHarf = input()

    if varMi(kelimeninHarfleri, girilenHarf, harfSayisi, harfBulunduMu):

        s = ""

        for i in range(0, harfSayisi - 1 + 1, 1):

            if harfBulunduMu[i] == True:

                s = s + kelimeninHarfleri[i] + " "

            else:

                s = s + " - "

        print(s)

    else:

        denemeSayisi = denemeSayisi - 1

        print(str(denemeSayisi) + " " + "deneme hakkı kaldı!")
```

```
kazanildiMi = True

i = 0

while i < harfSayisi:

    if not harfBulunduMu[i]:

        i = harfSayisi + 1

        kazanildiMi = False

    i = i + 1

if

kazanildiMi == True:

    print("Kazandınız!")

    print("Doğru kelime: " + girilenKelime)

    oyunBittiMi = True

else:

    if denemeSayisi == 0:

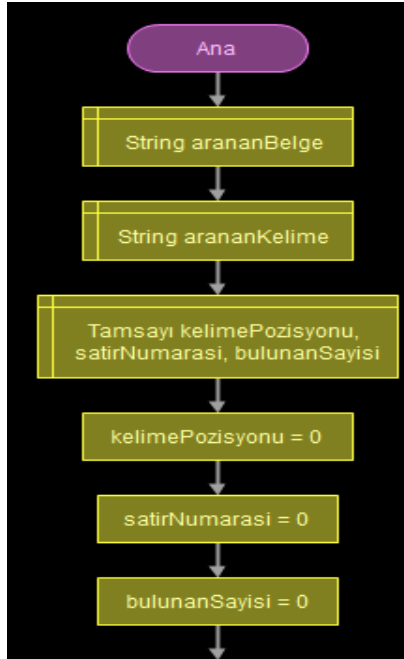
        print("Kaybettiniz!")
```



Flowgorithm dosyasına ulaşmak için
karekodu okutunuz.

8.2.8 Bir Belgede Kelime Arama Uygulaması

Günlük hayatta kullandığımız 'Ctrl+F' (Bul) fonksiyonunun arkasında aslında nasıl bir mekanizma çalışıyor? Bu uygulamada, devasa metin yığınları içerisinde istediğimiz anahtar kelimeyi saniyeler içinde bulup çıkaran kendi mini arama motorumuzu tasarlayacağız.



Programımıza, ana fonksiyon içerisinde ihtiyaç duyduğumuz değişkenleri tanımlayarak (Şekil 8-59) başlıyoruz. arananBelge adlı değişkene, üzerinde çalışacağımız belgedeki verileri aktaracağız

Şekil 8-59 Ana fonksiyon 1. kısım

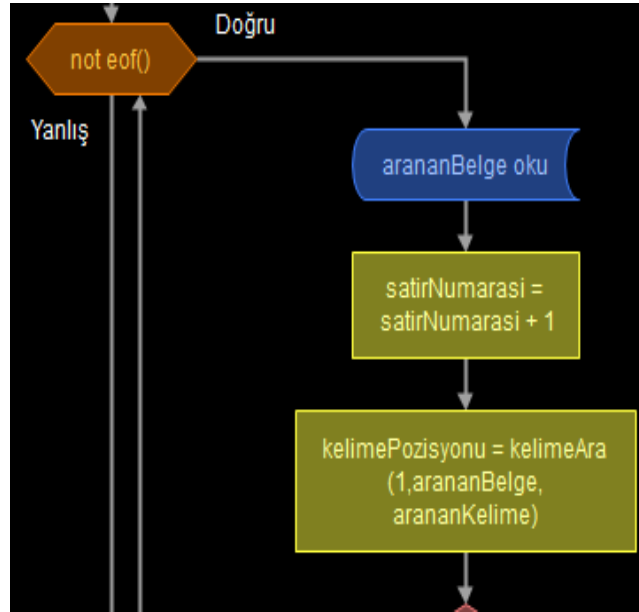
Programı yazmaya başlamadan önce, hazırlık aşaması olarak meyveler.txt adında yeni bir metin belgesi oluşturalım. Belgenin içerisine, alt alta gelecek şekilde birkaç meyve ismi yazıp dosyayı kaydedelim.



Gerekli değişken atamalarının hemen ardından meyveler.txt dosyasını okuma modunda Şekil 8-60'ta görüldüğü gibi açıyoruz.

Önemli Not: Programın hatasız çalışabilmesi için oluşturduğumuz metin belgesi ile Flowgorithm proje dosyasının bilgisayarda aynı klasör içerisinde (aynı konumda) bulunması gerekmektedir. Dosya bağlantısını sağladıktan sonra, kullanıcıdan aramak istediği veriyi girmesini istiyor ve bu değeri arananKelime değişkenine atıyoruz.

Şekil 8-60 Ana fonksiyon 2. kısım



Şekil 8-61 Ana fonksiyon 3. kısım

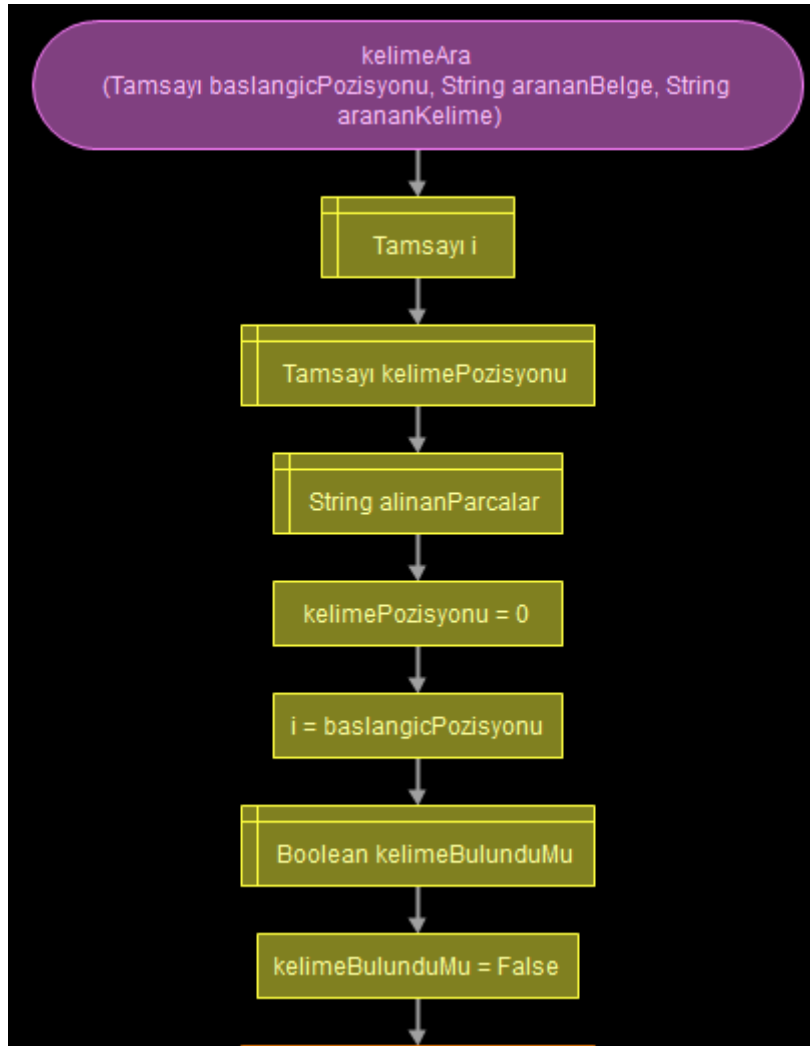
Programın akışında bir while döngüsü (Şekil 8-61) başlatıyor ve döngü koşulu olarak NOT EOF() (Dosya Sonu Değil) ifadesini kullanıyoruz. Bu dahili fonksiyon, dosyanın sonuna ulaşılmadığı sürece döngünün çalışmaya devam etmesini sağlar.

Döngü içerisine bir 'Oku' (Read) bloğu ekleyerek, meyveler.txt dosyasındaki verileri satır satır okuyor ve bu verileri arananBelge değişkenine aktarıyoruz.

Döngünün her adımında, işlem yapılan satırın sırasını takip edebilmek adına satirNumarasi değişkeninin değerini 1 artırıyoruz.

Kullanıcının aradığı kelimeyi tespit etmek amacıyla kelimeAra fonksiyonunu çağırıyor ve fonksiyondan dönen sonucu kelimePozisyonu değişkenine atıyoruz.

Şimdi, programın akışına kelimeAra fonksiyonunu (Şekil 8-62) oluşturarak devam edelim.



Şekil 8-62 Kelime arama fonksiyonu 1. kısım

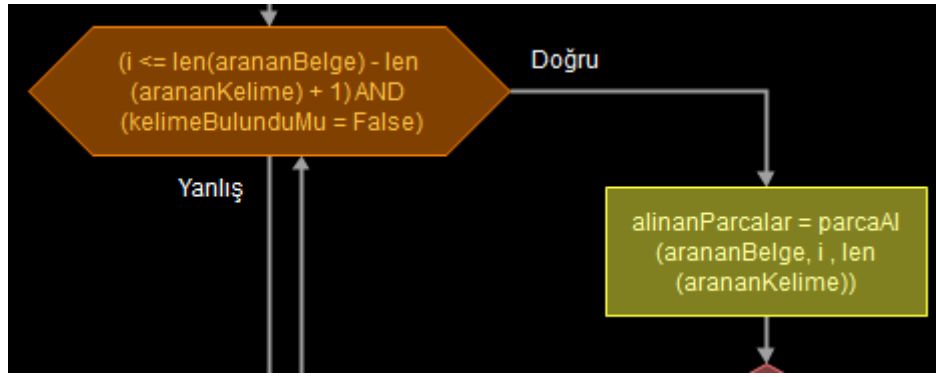
Fonksiyonumuz için gerekli değişkenleri tanımlayarak başlıyoruz. Bu fonksiyon, işlem yapabilmek için parametre olarak bizden üç farklı değer beklemektedir: baslangicPozisyonu, arananBelge ve arananKelime.

Ana fonksiyonda (Main) bu fonksiyonu çağırırken, parametreleri sırasıyla şu şekilde gönderiyoruz:

Başlangıç Pozisyonu: Aramaya baştan başlamak için.

Belge İçeriği: Ana fonksiyondaki arananBelge değişkeninin değeri.

Aranan Kelime: Kullanıcının girdiği arananKelime değişkeninin değeri.



Şekil 8-63 Kelime arama fonksiyonu 2. kısım

Fonksiyonun devamında bir while döngüsü (Şekil 8-63) başlatıyoruz. Bu döngünün çalışma şartı; i değerinin, `arananBelge` uzunluğu ile `arananKelime` uzunluğu arasındaki farkın 1 fazlasından küçük olmasıdır.

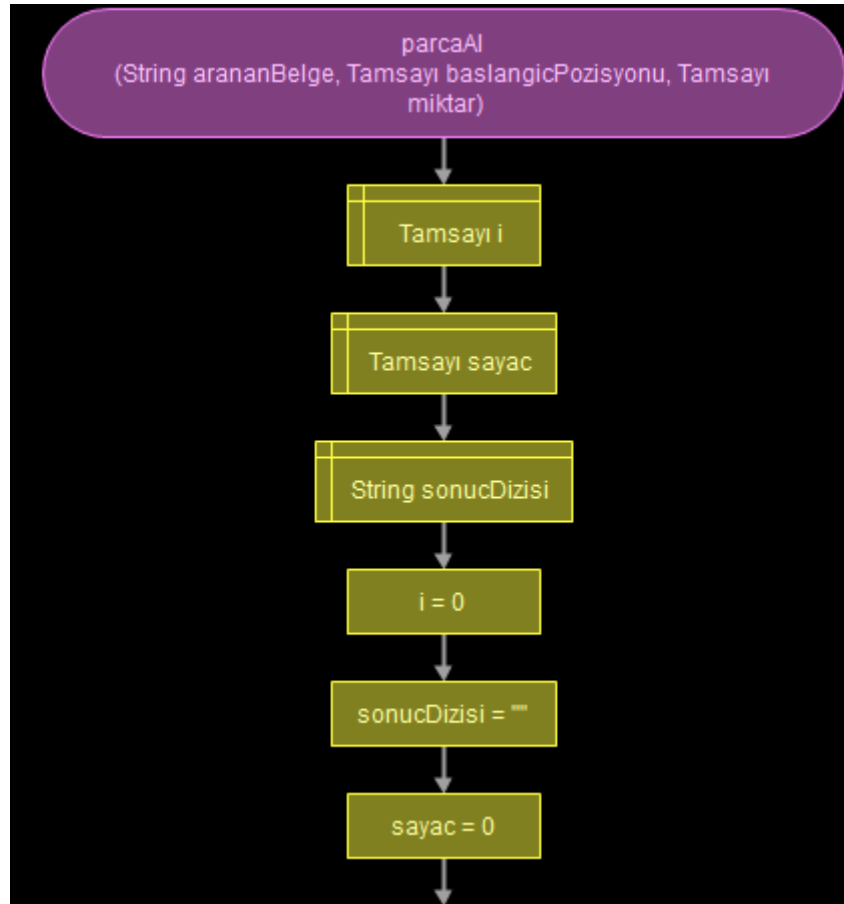
Peki, neden böyle bir şart koyduk? Amacımız, kullanıcının girdiği kelimenin harf sayısını baz alarak belgemizi parçalara ayırmak ve taramaktır. Eğer oluşturulan bu parçalar taranıp biterse döngüden çıkılmasını istiyoruz.

Örnekle Açıklayalım: Belgemizde sadece 'elma' (4 harfli) yazdığını, kullanıcının ise 'bu' (2 harfli) kelimesini aradığını varsayalım. Döngü sınırimız: $4 - 2 + 1 = 3$ olacaktır. Yani 'elma' kelimesi 2'şerli gruplara ayrıldığında şu 3 parça ortaya çıkar:

1. 'el'
2. 'lm'
3. 'ma'

Bu durumda döngünün 3 kez çalışması tüm ihtimalleri kontrol etmek için yeterlidir.

Şimdi bahsettiğimiz bu parçaları kod üzerinde oluşturalım. Parçalama işlemi için `parcaAl` fonksiyonunu (Şekil 8-64) çağırıyor ve dönen değeri `alinanParcalar` değişkenine atıyoruz.



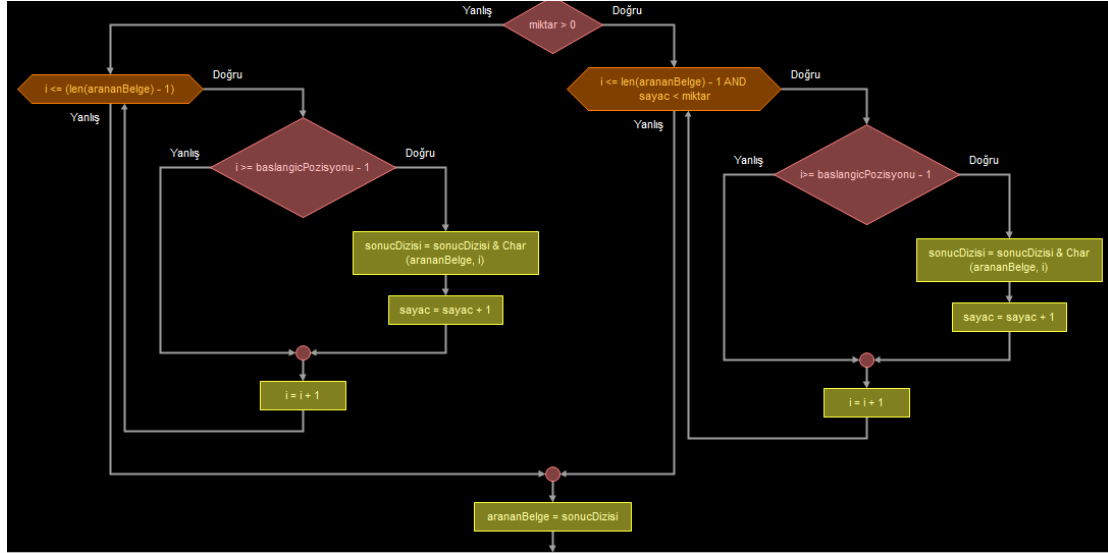
Şekil 8-64 Parça al fonksiyonu 1. kısım

Bu fonksiyon, işlem yapabilmek için bizden parametre olarak 3 farklı değer beklemektedir: **arananBelge**, **baslangicPozisyonu** ve **miktar**.

Fonksiyonu çağırırken gönderdiğimiz değerler ise sırasıyla şöyledir:

1. **arananBelge:** meyveler.txt dosyasından okuyup değişkene aktardığımız veriler.
2. **baslangicPozisyonu:** kelimeAra fonksiyonundaki döngü sayacımız olan *i* değeri.
3. **miktar:** arananKelime değişkeninin harf sayısı (uzunluğu).

Gerekli değişken tanımlamalarını ve eşleşmeleri yaptıktan sonra fonksiyonun kodlarını yazmaya Şekil 8-65 ile devam ediyoruz.

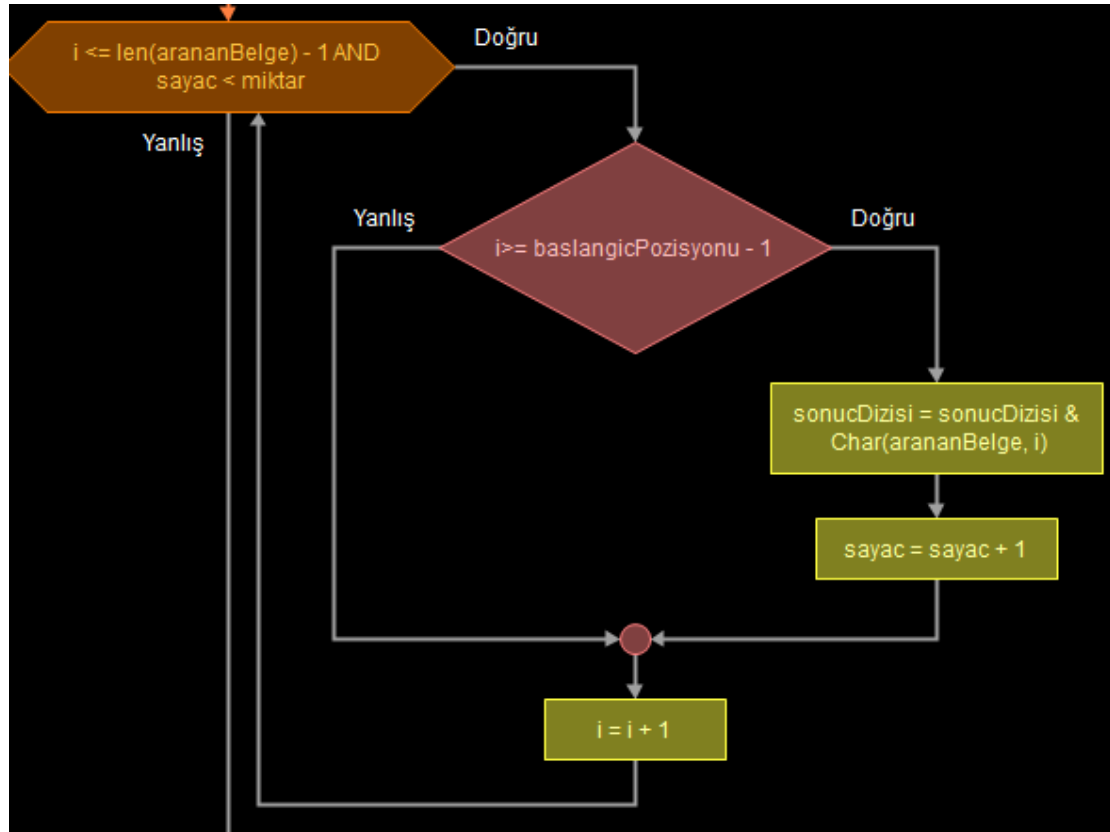


Şekil 8-65 Parça al fonksiyonu 2. kısım

Hatırlayacağınız üzere, girilenKelime değişkeninin karakter sayısını (uzunluğunu) miktar değişkenine atamıştık. Şimdi bu değer üzerinden bir kontrol sağlıyoruz.

Eğer miktar 0 ise, yani kullanıcı herhangi bir veri girmemişse akış şemanın sol tarafındaki döngüye yönlenecektir. Eğer en az 1 harf girilmişse, akış sağ taraftaki döngü üzerinden devam edecektir.

Gelin, öncelikle sol taraftaki (girdi olmayan) durumu inceleyelim.



Şekil 8-66 Kelime arama fonksiyonu 3. kısım

İşlemi gerçekleştirmek için bir while döngüsü (Şekil 8-66) tanımlıyoruz. Bu döngünün çalışmaya devam etmesi için iki temel şartımız var:

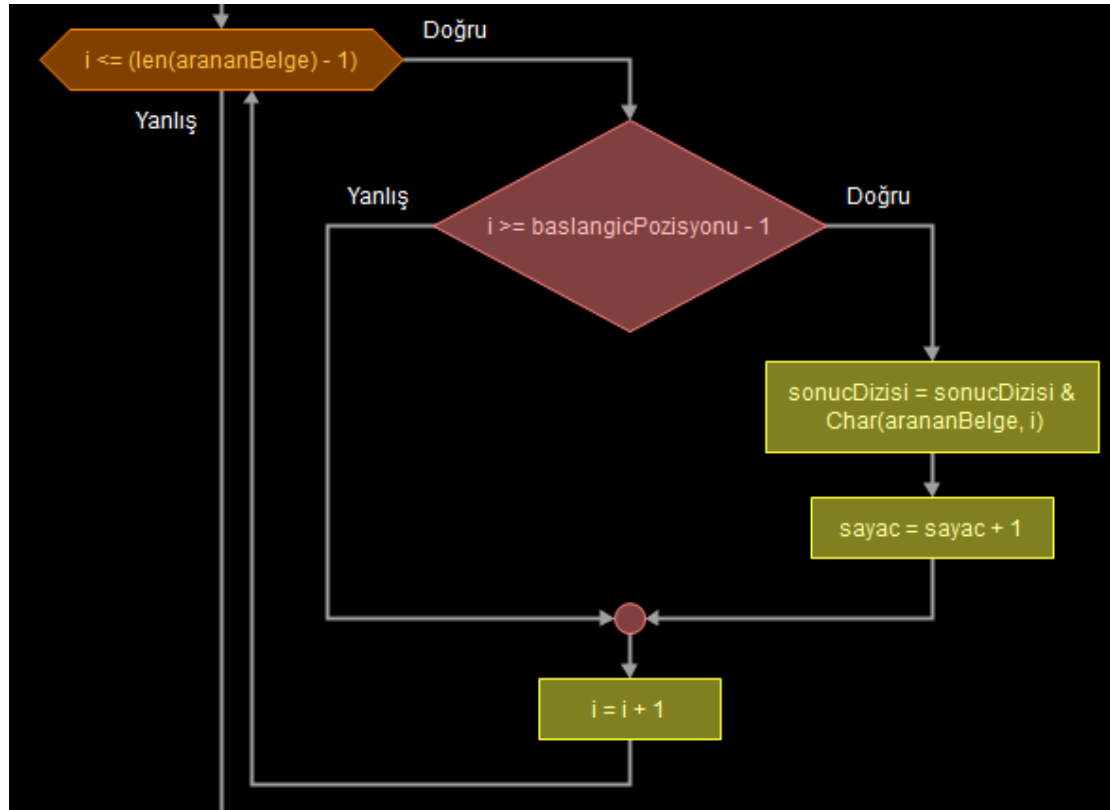
1. i değişkeninin, belgedeki toplam karakter sayısının 1 eksiğinden küçük olması,
2. sayac değişkeninin, aranan kelimenin harf sayısından (uzunluğundan) küçük olması.

Döngü içerisine bir if kontrol bloğu yerleştiriyoruz. Eğer i değeri $baslangicPozisyonu$ değerinden büyükse (veya eşitse); $sonucDegiskene$ ne, $arananBelge$ içerisindeki i . sıradaki karakteri ekleyerek birleştirme yapıyoruz.

Konuyu bir örnekle somutlaştıralım: Metin belgemizin içerisinde sadece 'Alper' yazdığını varsayalım. Kullanıcı ise aranan kelime olarak 'Alp' (3 harfli) girmiş olsun.

Program, 'Alper' kelimesini 3'erli gruplar halinde tarayacak ve $sonucDegiskene$ (veya sonuç dizisi) sırasıyla şu parçaları oluşturacaktır:

- 1. Parça: 'Alp'
- 2. Parça: 'lpe'
- 3. Parça: 'per'

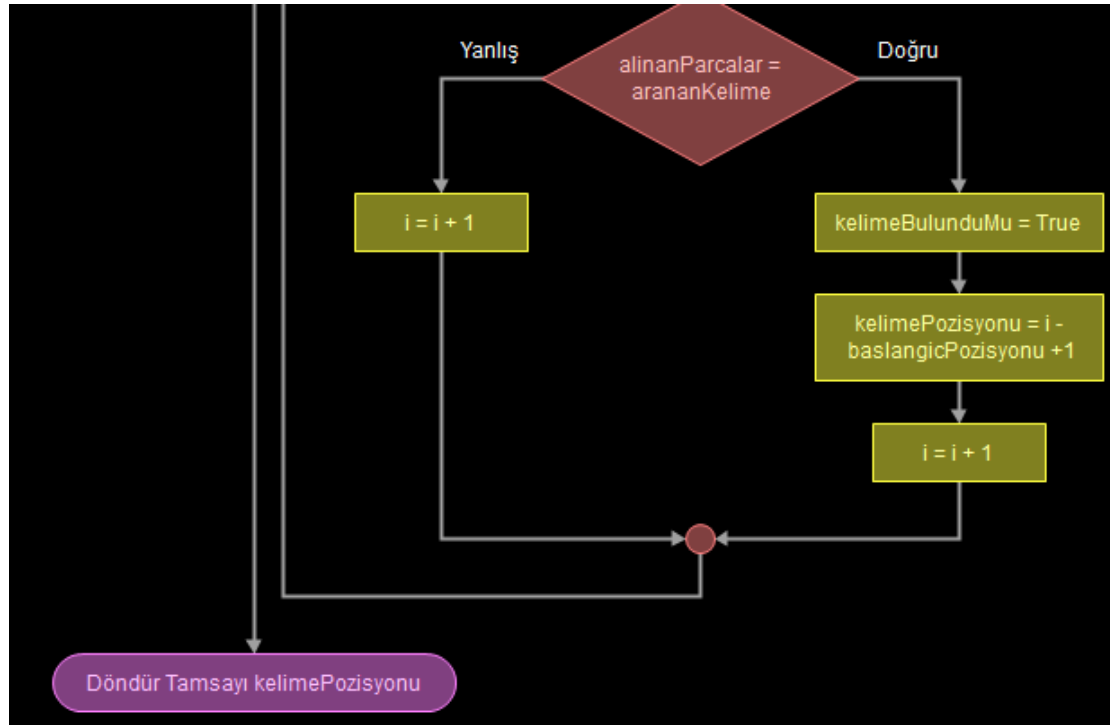


Şekil 8-67 Parça al fonksiyonu 3. kısım

Akış şemasının sol kolunda yer alan Şekil 8-67’de görüldüğü gibi while döngüsü ise; metin belgemizdeki boşluk karakterlerini tespit ederek, bu değerleri sonucDizisi değişkenine atayacaktır.

Hatırlayacağınız üzere parcaAl fonksiyonu, tamamlandığında arananBelge ismindeki değişkeni geri döndürecek (return) şekilde ayarlanmıştır. Bu nedenle, döngü sonunda elde ettiğimiz sonucDizisi değerini, dönüş değişkenimiz olan arananBelgeye eşitleyerek fonksiyonu sonlandırıyoruz.

parcaAl fonksiyonundaki işlemlerimizi tamamladık. Şimdi kelimeAra fonksiyonuna geri dönüyor ve akışa, fonksiyonu çağırdığımız satırın hemen ardından Şekil 8-68 ile devam ediyoruz.



Şekil 8-68 Parça al fonksiyonu 4. kısım

Döngü içerisine bir if kontrol bloğu ekleyerek asıl sorgumuzu yapıyoruz: Elde ettiğimiz alınanParcalar değeri, kullanıcının girdiği arananKelime değerine eşit mi?

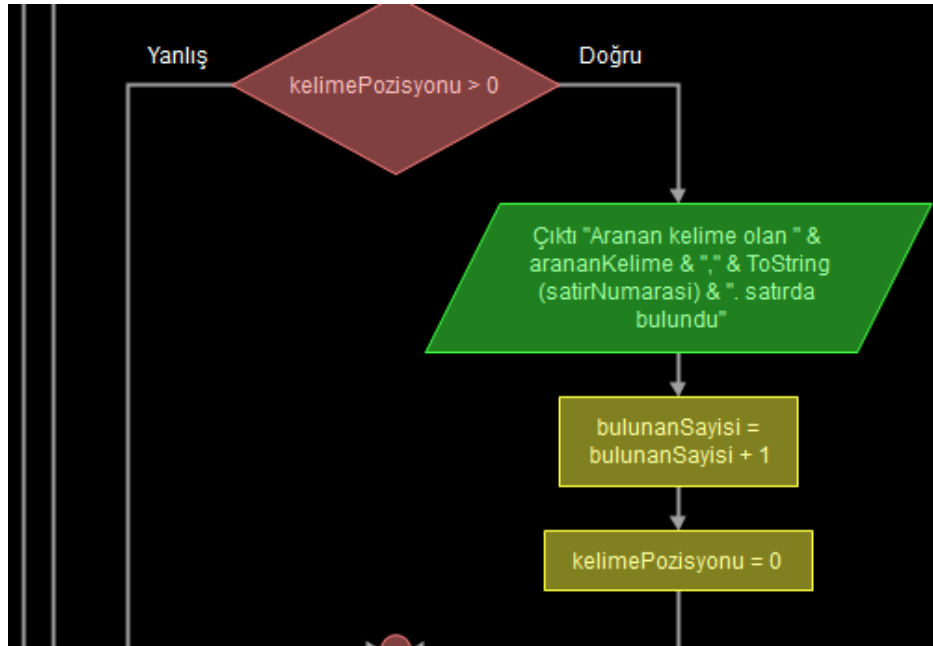
Eğer bu şart sağlanırsa (yani kelime bulunduysa);

1. kelimeBulunduMu değişkenini true yapıyoruz.
2. Kelimenin yerini tespit etmek için kelimePozisyonu değişkenini şu formülle güncelliyoruz:

$$i - \text{baslangicPozisyonu} + 1$$

Sorgu bitiminde, döngünün sürekliliğini sağlamak için i değerini 1 artırıyoruz ($i = i + 1$). Sonuç olarak; kelimeyi hangi pozisyonda bulduysak, o değer fonksiyonumuzun dönüş değeri (Return Value) olacaktır.

Böylece kelimeAra fonksiyonunu tamamlamış olduk. Şimdi ana fonksiyonumuza (Main) geri dönelim ve kaldığımız yerden, yani fonksiyonu çağırdığımız satırın altından devam ederek programı nihayete erdirelim.



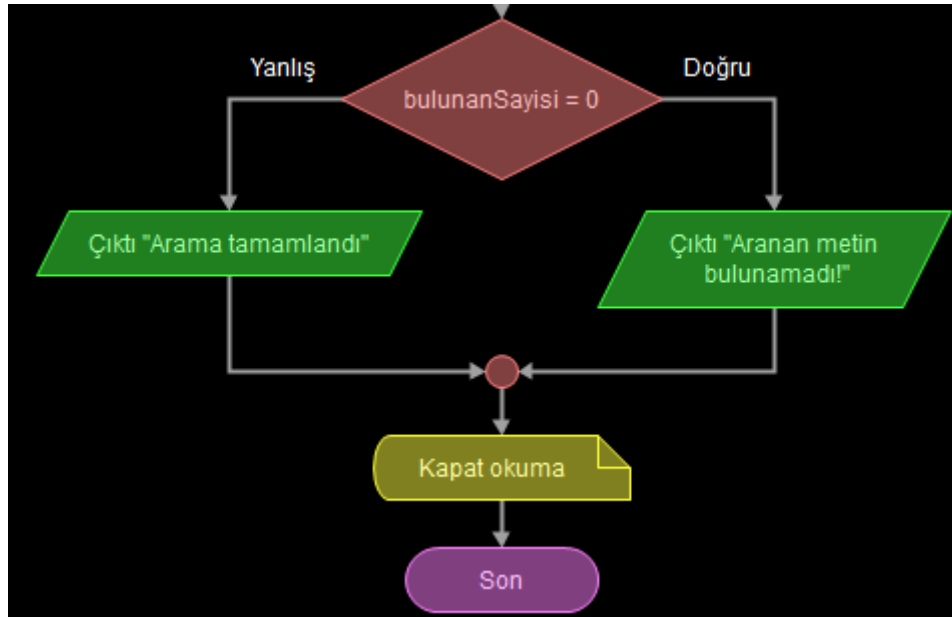
Şekil 8-69 Ana fonksiyon 4. kısım

Arama işleminin sonucunu değerlendirmek için bir if kontrol bloğu (Şekil 8-69) açıyor ve şart olarak kelimePozisyonu > 0 ifadesini ekliyoruz.

Mantığımız şu: Fonksiyonumuz kelimeyi bulduğunda 0'dan büyük bir değer döndürmektedir. Dolayısıyla kelimePozisyonunun 0'dan büyük olması, aranan kelimenin o satırda bulunduğu anlamına gelir.

Şartın sağlandığı (kelimenin bulunduğu) durumda şu işlemleri yapıyoruz:

1. Kullanıcıya, aranan kelimenin ne olduğu ve hangi satırda bulunduğu bilgisini ekrana yazdırıyoruz.
2. Toplam bulunan kelime sayısını tutmak için bulunanSayisi değişkenini 1 artırıyoruz.
3. Son olarak; bir sonraki satırda hatalı işlem yapmamak adına kelimePozisyonu değerini tekrar 0 olarak sıfırlıyoruz.



Şekil 8-70 Ana fonksiyon 5. kısım

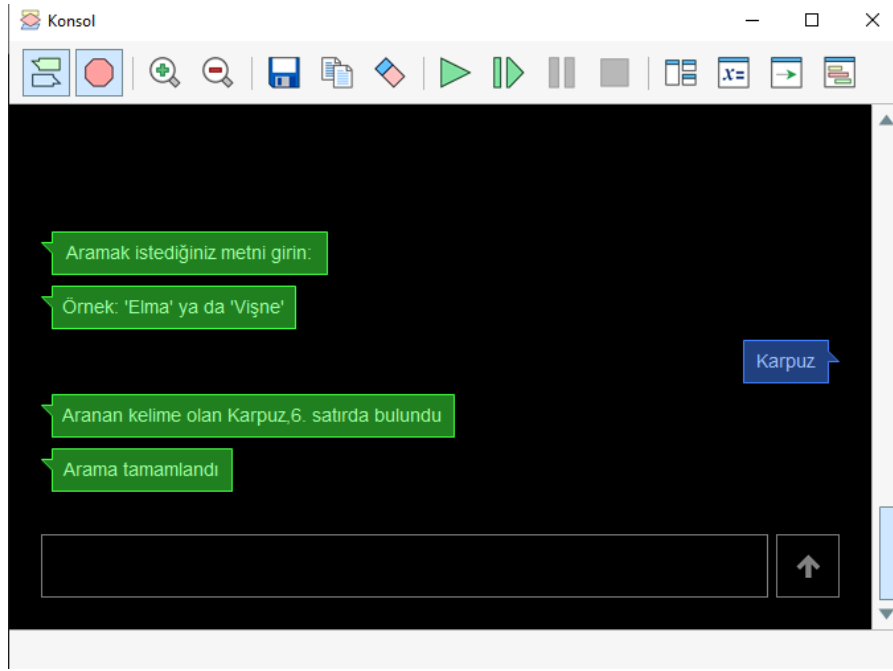
Döngü sona erdiğinde, bulunanSayisi değişkeninin değerini Şekil 8-70’de görüldüğü üzere kontrol etmemiz gerekir. Eğer bu değer hâlâ 0 ise, aranan kelime belgemizde hiç geçmiyor demektir.

Bu durumu yönetmek için son bir if bloğu ekliyoruz:

- Eğer bulunanSayisi == 0 ise: Kullanıcıya, aranan kelimenin belgede bulunamadığına dair bir mesaj veriyoruz.
- Eğer 0'dan farklı ise: (Yani kelime en az bir kez bulunduysa), kullanıcıya 'Arama işlemi tamamlandı' mesajını iletiyoruz.

Bu son adımla birlikte programımız görevini başarıyla tamamlamış oluyor.

Program çalıştırıldığında, konsol ekranı Şekil 8-71 gibi görünecektir.



Şekil 8-71 Programın çıktısı

8.2.9 Verilen programın Python Çıktısı

```
def kelimeAra(baslangicPozisyonu, arananBelge, arananKelime):
    kelimePozisyonu = 0
    i = baslangicPozisyonu
    kelimeBulunduMu = False
    while i <= len(arananBelge) - len(arananKelime) + 1 and kelimeBulunduMu == False:
        alinanParcalar = parcaAl(arananBelge, i, len(arananKelime))
        if alinanParcalar == arananKelime:
            kelimeBulunduMu = True
            kelimePozisyonu = i - baslangicPozisyonu + 1
            i = i + 1
        else:
            i = i + 1

    return kelimePozisyonu

def parcaAl(arananBelge, baslangicPozisyonu, miktar):
    i = 0
    sonucDizisi = ""
    sayac = 0
    if miktar > 0:
        while i <= len(arananBelge) - 1 and sayac < miktar:
            if i >= baslangicPozisyonu - 1:
                sonucDizisi = sonucDizisi + arananBelge[i]
                sayac = sayac + 1
            i = i + 1
```

```
else:
    while i <= len(arananBelge) - 1:
        if i >= baslangicPozisyonu - 1:
            sonucDizisi = sonucDizisi + arananBelge[i]
            sayac = sayac + 1
            i = i + 1
        arananBelge = sonucDizisi

    return arananBelge

# Main
kelimePozisyonu = 0
satirNumarasi = 0
bulunanSayisi = 0
infile = open("C:/Users/admin/Desktop/meyveler.txt")
nextLine = infile.readline()
print("Aramak istediğiniz metni girin:")
print("Örnek: 'Elma' ya da 'Vişne'")
arananKelime = input()
while not nextLine == "":
    arananBelge = nextLine
    nextLine = infile.readline()
    satirNumarasi = satirNumarasi + 1
    kelimePozisyonu = kelimeAra(1, arananBelge, arananKelime)
    if kelimePozisyonu > 0:
        print("Aranan kelime olan " + arananKelime + "," + str(satirNumarasi) + ". satırda bulundu")
        bulunanSayisi = bulunanSayisi + 1
        kelimePozisyonu = 0
if bulunanSayisi == 0:
    print("Aranan metin bulunamadı!")
else:
    print("Arama tamamlandı")
infile.close()
```



Flowgorithm dosyasına ulaşmak için
karekodu okutunuz.

9 Kaynakça

- Wing, J. (2006). Computational Thinking. *Communications of the ACM*, 49(3), 33-35. <https://doi.org/10.1145/1118178.1118215>
- Akgün, F. (2020). Öğretmen Adaylarının Bilgi ve İletişim Teknolojileri Yeterlilikleri ve Bilgi İşlemsel Düşünme Becerilerinin Çeşitli Değişkenler Açısından Değerlendirilmesi. *Trakya Üniversitesi Sosyal Bilimler Dergisi*, s. 629-654.
- Attila, A. Ş. (2025). *Uygulamalı Örneklerle Yapay Zekâ Algoritmaları ve Programlama*. 2. Baskı. Ankara: Seçkin Yayıncılık.
- Atzori, R. (2022). *Flowchart Shapes*. Flowgorithm: <https://www.flowgorithm.altervista.org/HTML%20ENG/index.html?computer.htmladresinden%20alindi>
- Aydoğdu, Ş. (2020). *Algoritma ve Programlama*. Pegem Akademi Yayıncılık.
- Balay, M., Timuçin, E. T., Çağlar, E., & Şentürk, A. (2006). *Bilgi teknolojileri 2*. Bursa: Ekin Yayınevi.
- Eryılmaz, S. (2019). *Algoritma tasarımı ve geliştirme: Java ve C# programlama dili örnekleri*. Ankara: Nobel Akademik Yayıncılık.
- Levitin, A. (2003). *Introduction to the Design and Analysis of Algorithms*. Addison-Wesley.
- Papert, S. (1980). *Mindstorms: Children, Computers, And Powerful Ideas*. New York: Basic Books, Inc., Publishers.
- Sarıkaya, D. A. (2019, 8 2). Programlama Öğretiminin Bilgi İşlemsel Düşünme Becerisine Etkisi. *Türkiye Sosyal Araştırma Dergisi*, s. 575-590.
- Şefik, Ö., & Urhan, S. (2023). *Matematik Eğitiminde Algoritma Tasarımı ve Python Programlamaya Giriş*. Pegem Akademi Yayıncılık.
- Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, 148.
- Tungut, H. B. (2019). *Algoritma ve programlama mantığı*. İstanbul: Kodlab Yayınları.
- Üzümcü, Ö. (2018). Eğitimde Yeni 21. Yüzyıl Becerisi: Bilgi İşlemsel Düşünme. *Uluslararası Türk Kültür Coğrafyasında Sosyal Bilimler Dergisi*, 3(2), 1-16.
- Vatansever, F. (2017). *Algoritma geliştirme ve programlamaya giriş*. Ankara: Seçkin Yayıncılık.

Kısa Özgeçmiş



Ömer Uysal, Bursa Uludağ Üniversitesi Eğitim Fakültesi Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümünde görev yapan bir akademisyendir. Son yıllarda özellikle yapay zekâ, proje tabanlı öğrenme, dijital oyun tabanlı öğrenme, etkileşimli materyal tasarımı konusunda akademik araştırmalar yapmakta; algoritma tasarımı ve geliştirme dersini vermektedir.



Gürkan Güreşci, Bursa Uludağ Üniversitesi Eğitim Fakültesi Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümünde tezli yüksek lisans yapmakta ve Sakarya Karasu Gazi Ortaokulunda matematik öğretmeni olarak görev yapmaktadır. Algoritma geliştirme, dijital oyun tabanlı öğrenme üzerine araştırmalar yapmaktadır.