

The HyperText Markup Language (HTML) is the standard markup language for documents designed to be displayed on a web browser. It can be assisted by styling technologies such as Cascading Style Sheets and scripting technologies such as JavaScript.

BÖLÜM 1

BİLGİSAYAR

ANAHTAR KAVRAMLAR

Donanım, Yazılım, İşlemci, Ana Bellek, İkili Sayı Sistemi, Bellek Ölçü Birimleri, Komut Çevremi

ÖĞRENME HEDEFLERİ

- 1 Temel bilgisayar mimarisini bilir
- 2 Verilerin bellekte nasıl saklandığını bilir
- 3 Bellek kapasite ölçü birimlerini bilir
- 4 Programların nasıl çalıştığını bilir

GİRİŞ

Bilgisayar alanındaki gelişmeler oldukça heyecan vericidir. Bilgisayarın kullanılmadığı alan neredeyse yoktur. Günümüzde bilgisayarlar verileri analiz etmek, etkili sunumlar hazırlamak, finansal işlemleri gerçekleştirmek, müşterilerle hızlı bir şekilde iletişim kurmak, otomasyon sistemlerini kontrol etmek ve daha birçok şey için kullanılmaktadır. İş hayatı yanı sıra ev kullanıcıları çevrimiçi ders içeriklerini takip etme, fatura ödeme, alışveriş yapma, arkadaşlarla anlık iletişim kurma, film izleme ve bilgisayar oyunları oynama gibi işler için kullanılmaktadır. Tabletler, cep telefonları, yol takip sistemleri, hesap makinesi gibi birçok cihazın da temel olarak bir bilgisayar olduğu unutulmamalıdır.

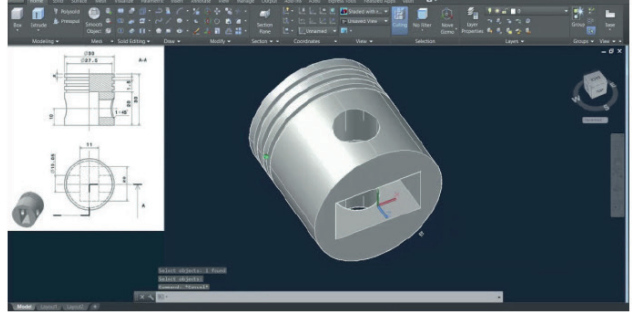
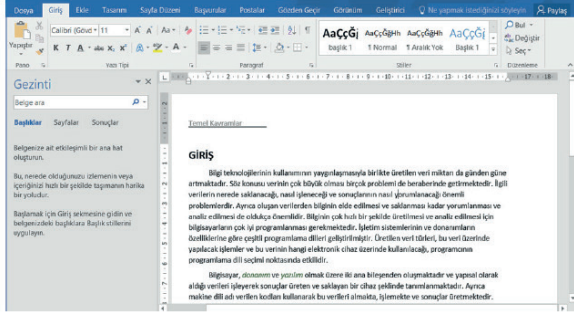
Bilgisayarların kullanımı günlük hayatımızda neredeyse sınırsızdır. Bilgisayarlar programlanabildikleri için birçok alanda kullanılmaktadır. Bu, bilgisayarların yalnızca bir işi yapmak için değil, programlarının onlara yapmalarını söylediği herhangi bir işi yapmak için tasarlandıkları anlamına gelmektedir. Program, bilgisayarın bir görevi gerçekleştirmek için izlediği bir dizi talimattır. Şekil 1.1'de yaygın olarak kullanılan iki program olan Microsoft Word ve AutoCAD'in ekran görüntüleri gösterilmektedir. Microsoft Word, bilgisayarınızla belgeler oluşturmanıza, düzenlemenize ve yazıcıdan çıktı almanıza yarayan bir kelime işlemci programıdır. AutoCAD ise bilgisayar destekli iki ve üç boyutlu nesne çizimleri, mimari çizimler yapmamızı ve makine parçaları tasarlamamızı sağlayan bir programdır. Programlar genellikle yazılım olarak adlandırılır. Yazılım bir bilgisayar için çok önemlidir çünkü bilgisayarın yaptığı her şeyi kontrol etmektedir. Bilgisayarlarımızı kullanışlı hale getirmek için kullandığımız tüm yazılımlar, programcı veya yazılım geliştirici olarak çalışan kişiler tarafından oluşturulmaktadır. Bir programcı veya yazılım geliştiricisi, bilgisayar programlarını tasarlamak, oluşturmak ve



test etmek için gerekli eğitim ve becerilere sahip kişidir. Finans, tıp, güvenlik, tarım, akademi, eğlence ve diğer birçok alanda programcılar çalışmaktadır.

Bu kitapta C++ dilini kullanarak bilgisayar programlamanın temel kavramlarını anlatıla-

caktır. Bu bölümde ise programlamaya başlamadan önce bilgisayarlar ve nasıl çalıştıkları hakkında birkaç temel konudan bahsedilecektir. Bilgisayarların temel fiziksel bileşenlerini, bilgisayarların verileri nasıl depoladığını ve programları nasıl çalıştırdığını inceleyeceğiz.



Şekil 1.1 Bir kelime işlem programı ve bir tasarım programı.

1. BİLGİSAYAR NEDİR?

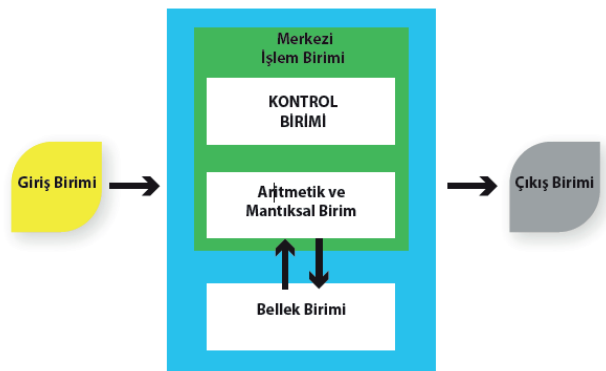
Bilgisayar; belirli bir talimat dizisine göre verileri okuyan, aritmetik ve mantıksal işlemler yapan, sonuçları kullanıcıya yorumlayabileceği şekilde sunan ve verileri hafızasında saklayan elektronik bir cihaz olarak tanımlanabilir. Bilgisayarlar, insanlardan çok daha hızlı hesaplama yapabilmekte ve mantıksal karar verebilmektedir. Kullandığımız kişisel bilgisayarlar saniyede milyarlarca işlem yapabilmektedir. 2022 yılı dünyanın en hızlı bilgisayarı, Japon Fujitsu firması tarafından geliştirilmiş Fugaku isimli süper bil-

gisayar ise saniyede 442 katrilyon işlem yapabilmektedir.

Bir bilgisayar, donanım (hardware) ve yazılımdan (software) oluşur. Donanım işlemci, bellek, sabit disk, ekran, klavye ve fare gibi fiziksel bileşenlerdir. Yazılım ise bilgisayarlara ne işlem yapacağını bildiren talimatlar bütünüdür. Bir bilgisayar sisteminin çalışması için hem donanım hem de yazılım gereklidir.

2. DONANIM

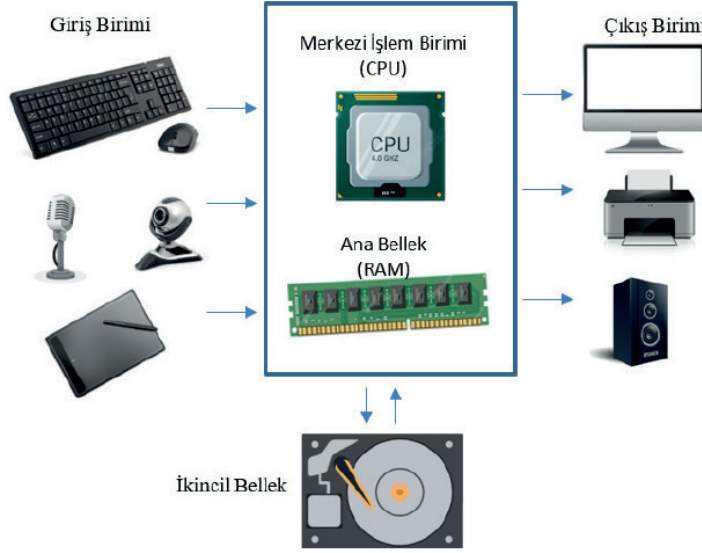
Donanım terimi, bir bilgisayarın yapıldığı tüm fiziksel aygıtları veya bileşenleri ifade etmektedir. Bilgisayar tek bir aygıt değil birlikte çalışan parçaların bütünüdür. Bilgisayarlar görünüşleri farklı olmasına rağmen belirli birimlere ayrılmaktadır. 1945'te John von Neumann tarafından geliştirilen mimari günümüz bilgisayarının temel çalışma prensiplerini teşkil etmektedir (Şekil 1.2). Bu mimaride bir işlemci, bir bellek ve giriş/çıkış birimleri yer almaktadır.



Şekil 1.2. John von Neumann mimarisi.

- Merkezi İşlem Birimi – MİB (Central Processing Unit - CPU)
- Bellek birimi (Memory)
- Giriş birimi (Input)
- Çıkış birimi (Output)

Bir bilgisayar satın aldığınızda merkezi işlem birimine, belleğe, disk sürücülerine, ekran kartına vb. aygıtlara sahip olursunuz. Tipik bir bilgisayar sistemi Şekil 1.3’de gösterilen ana bileşenlerden oluşmaktadır.



Şekil 1.3 Tipik bir bilgisayar sistemi.

2.1. Merkezi İşlem Birimi

Merkezi İşlem Birimi, bilgisayar sisteminin beyini olarak etkin bir işlev görmektedir. MİB, bilgisayar sistemindeki birimlerin işletilmesini, bu birimler arasındaki veri akışını kontrol etmeyi

ve veriler üzerinde işlemler gerçekleştirmeyi sağlayan ana donanım birimidir. Merkezi İşlem Birimi, ayrıca mikroişlemci veya işlemci olarak da adlandırılmaktadır.

2.2. Aritmetik ve Mantıksal Birim (Arithmetic Logic Unit-ALU)

Aritmetik ve Mantıksal Birim, işlemci içerisinde yer alan en önemli birimlerden biridir. Toplama, çıkarma, çarpma, bölme gibi aritmetik işlem-

ler ve AND-NAND, OR-NOR, XOR-XNOR gibi mantıksal işlemler bu birimin görevidir.

2.3. Kontrol Birimi

Kontrol birimi, bilgisayarın işlemlerini düzenler ve bütünleştirir. Ana bellekten uygun sırayla komutları seçer ve alır. Alınan bu komutlar, işlem-

leri yerine getirebilmek için diğer sistem öğelerini harekete geçirecek şekilde yorumlanır.

2.4. Bellek Birimi

Bilgisayar bellek birimi, kapasite, erişim hızı, fiziksel boyut ve bellek maliyeti açısından farklılık gösteren farklı bellek türlerinden oluşur. Bir bilgisayar sistemi, yazmaçlar (registers), ön bellekler (cache), ana bellek ve yardımcı bellekler-

den oluşmaktadır. İşlemci, işlemek için ihtiyaç duyduğu bilgilerin önbellekte saklanıp saklanmadığını kontrol eder. Varsa, önbellek ile verileri işlemciye döndürür. Bilgi önbellekte değilse onu ana bellekten alır. Eğer bilgi ana bellekte de

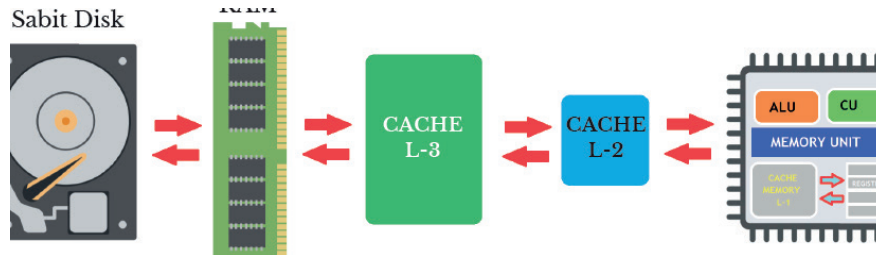
değilse yardımcı bellekten ana belleğe yüklenmesini sağlar. Belleklerin her biri, erişim hızına, boyutuna ve işlemciye yakınlığına bağlı olarak bilgisayar sisteminde en uygun şekilde kullanılarak işlemcinin performansı optimize edilmektedir. İşlemci çekirdeği içinde yer alan yazmaçlar, çok küçük bellek alanına sahip olup işlemciyle aynı hızda çalışmaktadır. Önbellekler ise işlem-

cinin daha hızlı işlem yapabilmesi için sık kullanılan talimatları ve verileri geçici olarak saklayan belleklerdir. Önbelleklerin sayısı ve bellek büyüklükleri işlemci mimarisine göre değişmektedir. Genellikle işlemcinin yarı hızında çalışmaktadır bu nedenle önbellek miktarının büyük olması işlemci performansını oldukça artırmaktadır.

2.5. Ana Bellek

Ana bellek, işlemcinin işleyeceği verileri ve sonuçlarını tutan depolama birimidir. İşlemciye doğrudan bağlı olan ana bellek Rastgele Erişimli Bellek (Random Access Memory-RAM), birincil bellek olarak da anılmaktadır. İşlemci herhangi bir bellek alanına eşit hızda erişebilmektedir. Ana bellek, sistemin çalışması için gerekli komutları işletim sisteminde saklamakta ve ikincil depolama birimlerinden okunan veriler ile çıkış birimlerine gönderilecek verileri tutmaktadır. Bir bilgisayar için desteklenen maksimum ana bellek boyutu (4/8/16 GB vb.), sistem yapılandırmasına ve işletim sisteminin türüne (32 veya 64 Bit) bağlı olarak değişebilmektedir. Ana bellekte

veriler elektrik enerjisi ile saklandığından sistem kapatıldığında üzerindeki veriler kaybolmaktadır. Bu yüzden bellek bilgisayarda kalıcı depolama alanı olarak kullanılamaz. Sistem kapatıldığında bile verilerin saklanması için çeşitli ikincil (yardımcı) depolama birimleri geliştirilmiştir. İşlemciler ve ana bellek farklı hızlarda çalışmaktadır. İşlemcilerin hızı ve işlem gücü arttıkça ana bellek ile işlemci arasındaki senkronizasyon farkı artmaktadır. Her bir işlem adımında, işlemcinin veri işlerken/iletirken performansını optimize etmek için bellek birimleri arasında önbellek adı verilen geçiş birimleri oluşturulmuştur (Şekil 1.4).



Şekil 1.4 Bilgisayar bellek hiyerarşisi.

2.6. İkincil Bellek Birimi

İkincil bellek, programların ve verilerin kullanılmak üzere tutulduğu, etkin olmayan programları ve verileri saklayan bir birimdir. Bu bellek birimindeki veri, bilgisayar kapatıldığında kaybol-

maz. DVD, CD gibi optik diskler, USB bellekler ve sabit diskler en yaygın kullanılan ikincil belleklerdir. İlgili bellek birimlerine veri yazmak ya da okumak ana belleğe göre daha yavaştır.

2.7. Giriş Birimi

Bilgisayarda işlenecek bilgiler (veri ve programlar) giriş birimi aygıtlarından alınarak ana belleğe ve merkezi işlem birimine aktarılmaktadır. Fare, klavye, dokunmatik ekran, mikrofon, kamera,

tarayıcı, barkod okuyucu, ikincil bellekler (USB bellekler, sabit diskler, optik diskler) ve sensörler giriş birimlerine örnek olarak gösterilebilir.

**DİKKAT EDELİM**

Dokunmatik ekranlar, optik diskler, USB bellekler, sabit diskler hem giriş hem de çıkış birimi olarak kullanılırlar.

2.8. Çıkış Birimi

Çıktı, bilgisayarın insanlar veya diğer cihazlar için ürettiği herhangi bir veridir. Bu, bir satış raporu, bir ürün listesi veya bir grafik olabilir. Çıkış birimleri, çıktıyı kendilerine gönderilen veriyi biçimlendirerek sunmaktadırlar. Bilgisayardan çıkan veri, ekran üzerinde gösterilebilir, kâğıda

yazdırılabilir, ses veya görüntü olarak oynatılabilir, ağ üzerinden iletebilir ya da başka cihazlara (bir makine ya da robotu bilgisayar ile kontrol etmek gibi) aktarılabilir. Çıkış aygıtlarına örnek olarak ekran, hoparlör, yazıcı, DVD ve disk sürücüler sayılabilir.

3. YAZILIM

Bir bilgisayarın çalışmasında yazılım isteğe bağlı değildir. Bilgisayarın açıldığı andan kapatılana kadar bilgisayarın yaptığı her şey yazılımın kontrolü altındadır. Yani yazılım olmadan bilgisayar hiçbir işlem yapamaz. Bilgisayar yazılımı, belirli görevleri gerçekleştirmek için tasarlanmış bir

dizi talimatlar bütünüdür. Bir bilgisayar programı C, C++, Java, Python vb. gibi programlama dillerinde yazılır. Sistem yazılımı ve uygulama yazılımı olmak üzere iki genel yazılım kategorisi vardır.

3.1. Sistem Yazılımı

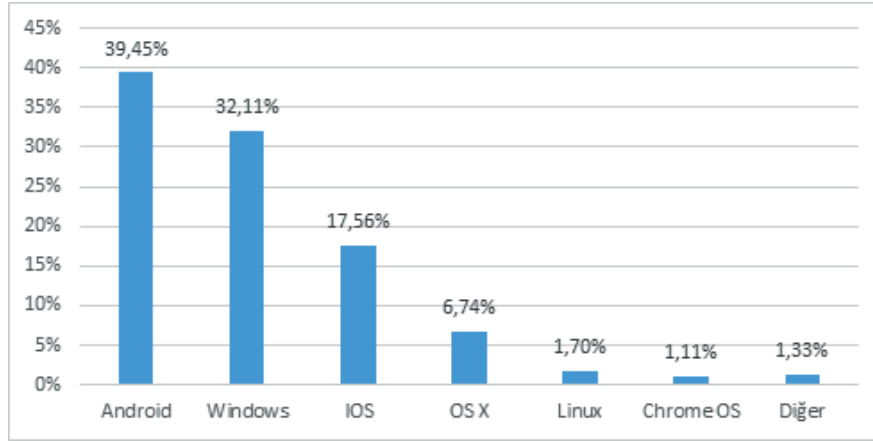
Sistem yazılımı, bilgisayarın dâhili işleyişini ve donanım bileşenlerini yönetmek ve kontrol etmek için kullanılır. Daha basit bir ifadeyle kulla-

nıcı ile donanım arasında ara bir katman olduğu söylenebilir. Sistem yazılımı tipik olarak aşağıdaki program türlerini içerir.

3.2. İşletim Sistemleri

Bir işletim sistemi, bir bilgisayardaki en temel program grubudur. İşletim sistemi, bilgisayar donanımı üzerindeki işlemleri kontrol eder, bilgisayara bağlı tüm cihazları yönetir, veriyi depolama cihazlarına kaydeder/okur ve bilgisayarda

diğer yazılımların çalışmasına olanak verir. Şekil 1.5'de, 2022 Ocak ayı verilerine göre dünyada genelinde işletim sistemlerinin kullanım oranları gösterilmektedir (Android, Windows, IOS, MacOS, Linux ve Chrome OS).



Şekil 1.5 En popüler işletim sistemleri (<https://gs.statcounter.com/os-market-share>).

3.3. Yardımcı Programlar

Bilgisayarın çalışmasını geliştiren veya veriyi koruyan özel bir görevi yerine getirir. Yardımcı programlara örnek olarak antivirüs, dosya sıkıştırma ve veri yedekleme programları verilebilir. Yazılım geliştirme araçları, programcıların yazı-

lım oluşturmak, değiştirmek ve test etmek için kullandıkları programlardır. Birleştiriciler, derleyiciler ve yorumlayıcılar bu kategoriye giren programlara örnektir.

3.4. Uygulama Yazılımı

Uygulama yazılımı kullanıcı tarafından bilgisayar üzerinde çeşitli görevleri gerçekleştirmek için geliştirilmiş programlardır. Bunlar, genellikle kullanıcıların zamanlarının çoğunu bilgisayarlarında çalıştırarak geçirdikleri programlardır. Web tarayıcıları, ofis uygulamaları, muhasebe

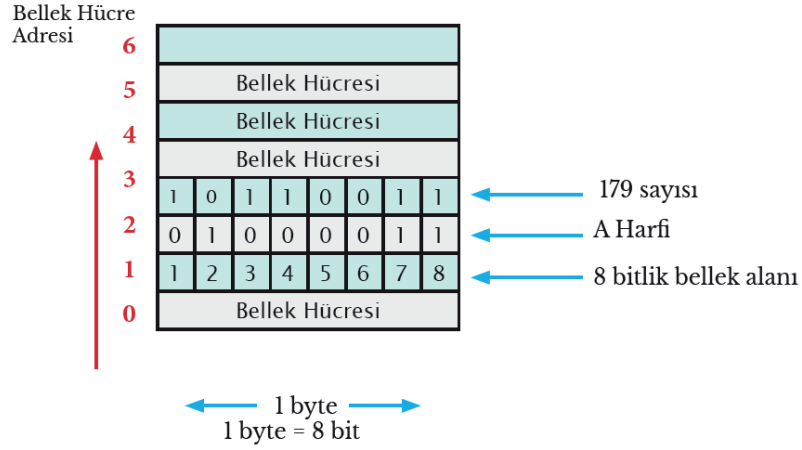
programları, müzik/film oynatıcıları, fotoğraf düzenleyiciler, oyun programları uygulama yazılımı çeşitlerine örnek olarak gösterilebilir. Uygulama yazılımları, işletim sistemine uyumlu bir şekilde yüklenerek kullanıcıya sunulmaktadır.

5. BİLGİSAYAR VERİLERİ NASIL SAKLAR?

Bilgisayar içerisindeki devreler verinin bir yerden bir yere aktarılmasından sorumludur. Elektronik devreler ile bir kabloda akımın olup olmadığı kontrol edilebilir. Böylelikle var-yok, açık-kapalı, evet-hayır, doğru-yanlış ya da 1-0 gibi iki durumlu herhangi bir şey temsil edilebilmektedir. Daha fazla durumu temsil edebilmek için daha fazla devre kullanılmalıdır. Dijital elektronik sistemlerinde mantık devreleri bilgi oluşturmak ve iletmek için 1'leri ve 0'ları kullanılmaktadır. 1 ve 0 ile gösterilen bu sembollere Binary Digit veya bu kelimelerden türetilmiş Bit denilmektedir. Bu devreler 20. yüzyılın en bü-

yük icadı olarak ifade edilen transistörlerle yapılmaktadır. Transistörler üzerinde yapılan gelişmeler sonucunda daha ucuz, daha küçük boyutlu ve daha hızlı bilgisayarların üretilmesi mümkün hale gelmiştir.

Bilgisayarın belleği, bayt (byte) olarak bilinen depolama hücrelerine bölünmüştür. Her bayt, 8 bitten oluşmaktadır. Bir bayt bellek alanında yalnızca bir harf veya küçük bir sayı saklanabilmektedir. Daha fazla sayı ve veri ile işlem yapabilmek için bir bilgisayarın çok sayıda bellek alanına sahip olması gerekir. Günümüzde çoğu bilgisayar milyarlarca bayt belleğe sahiptir.



Şekil 1.6 Temel bellek yapısı.

Bir veri parçası 1 baytlık alanda depolandığında, bilgisayar sekiz biti bu veriyi temsil edecek şekil ayarlar. Örneğin, Şekil 1.6'da "2" numaralı bellek hücresinde A harfinin bir bayt içinde nasıl saklanacağını, "3" numaralı bellek hücresinde ise

179 sayısının bir bayt bellek alanında nasıl saklanacağını gösterilmektedir. Bellek ölçü birimlerinin küçükten büyüğe doğru sıralaması aşağıdaki gibidir.

Tablo 1.1 Bellek ölçü birimleri.

Ölçü Birimi	Kısa Yazılışı	Boyutu	
1 Kilobyte	KB	1024 Byte	= 2^{10} Byte
1 Megabyte	MB	1024 KB	= 2^{20} Byte
1 Gigabyte	GB	1024 MB	= 2^{30} Byte
1 Terabyte	TB	1024 GB	= 2^{40} Byte
1 Petabyte	PB	1024 TB	= 2^{50} Byte
1 Exabyte	EB	1024 PB	= 2^{60} Byte
1 Zettabyte	ZB	1024 EB	= 2^{70} Byte
1 Yottabyte	YB	1024 ZB	= 2^{80} Byte

5.1. Sayıları Saklamak

Günlük yaşamda onlu sayı sistemi kullanılmaktadır. Ancak bilgisayarlar veriyi, elektrik akımının olup olmamasına bağlı olarak iki farklı değer

ile temsil edebilir. Bu, ikili sayı sistemine (binary) mükemmel bir şekilde uymaktadır.



EK BİLGİ

Sayı sistemleri için genel tanım
 $(abcde)_T = a.T^4 + b.T^3 + c.T^2 + d.T^1 + e.T^0$
 Burada T sayı sisteminin tabanı olarak adlandırılmaktadır. Rakamlar ise taban değerinden küçük ve 0 dâhil tam sayılardır.
 $0 \leq a, b, c, d, e < T$
 $(abcde)_T = a.T^4 + b.T^3 + c.T^2 + d.T^1 + e.T^0$
 En büyük değerli basamak
 En küçük değerli basamak

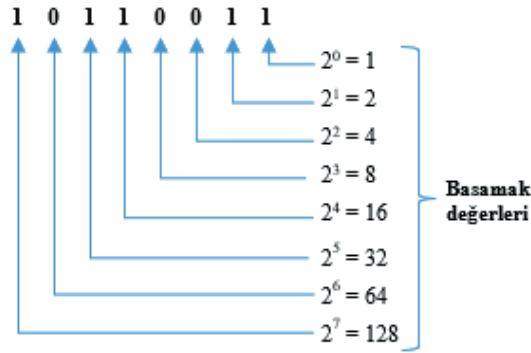
Onlu sayı sistemindeki sayıları ikili sayıya çevirirken bölme metodu kullanılır. Aşağıdaki örnekte görüleceği üzere sayı devamlı 2'ye bölünür ve kalan değerler tersten yazılarak ikili sayı elde edilir.

$$\begin{array}{r}
 38 \div 2 = 19 \text{ kalan } 0 \\
 19 \div 2 = 9 \text{ kalan } 1 \\
 9 \div 2 = 4 \text{ kalan } 1 \\
 4 \div 2 = 2 \text{ kalan } 0 \\
 2 \div 2 = 1 \text{ kalan } 0 \\
 1 \div 2 = 0 \text{ kalan } 1
 \end{array}
 \quad (38)_{10} = (100110)_2$$

İkili olarak yazılmış bir sayı örneği 10110011 şeklindedir. İkili bir sayıdaki her basamağın ko-

numu, kendisine atanmış bir değere sahiptir. Şekil 1.7'de gösterildiği gibi, en sağdaki basamaktan sola doğru basamak değerleri 20, 21, 22, 23, 24, 25, 26, 27 şeklindedir. Şekil 1.7, hesaplanan basamak değerleriyle aynı diyagramı göstermektedir. En sağdaki basamaktan sola doğru basamak değerleri 1, 2, 4, 8, 16, 32, 64, 128'dir.

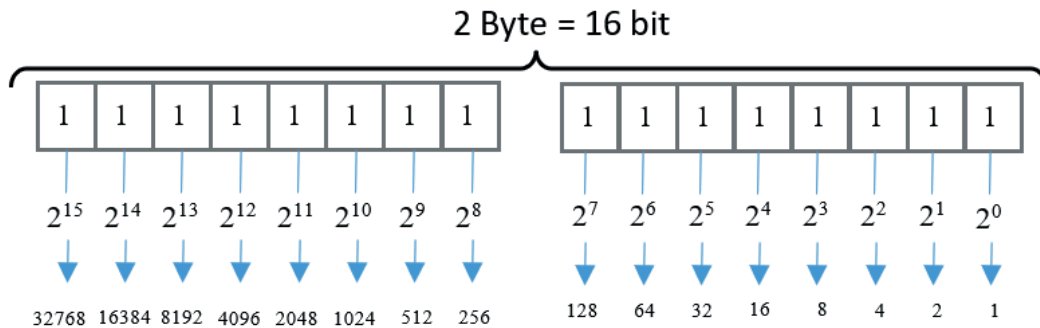
Bir ikili sayının değerini belirlemek için tüm 1'lerin konum değerlerini toplamak yeterlidir. Örneğin, 10110011 ikili sayısında 1'lerin konum değerleri 1, 2, 16, 32 ve 128'dir (Şekil 1.7). Tüm 1'lerin basamak değerleri toplamı 179'dur.



Şekil 1.7 İkili sayı sistemi basamak değerleri.

Bir baytlık bellek alanında tüm bitler 0 olduğunda sayı değeri 0 olurken, tüm bitler 1 olduğunda sayı değeri en büyük değer olan 255 ($1+2+4+8+16+32+64+128=255$) olacaktır. Bir bayt

sekiz bitten oluştuğu için böyle bir sınır vardır. 255'ten büyük bir sayının bellekte saklanması istendiğinde daha fazla bayt bellek alanı kullanmak gerekmektedir.



Şekil 1.8 İki baytlık bellek alanı.

Örnek olarak, iki baytlık bellek alanı bir araya getirdiğimizde bu bize 16 bitlik alan verecektir. Bu 16 bitin basamak değerleri Şekil 1.8'de gösterildiği gibi 20, 21, ..., 214, 215 olacaktır. Tüm basa-

mak değerleri toplandığında 2 baytta saklanabilecek maksimum değer 65.535 olacaktır. Bundan daha büyük bir sayı saklamak istendiğinde daha fazla bayt gerekecektir.

5.2. Karakterleri Saklamak

Bilgisayarın belleğinde saklanan herhangi bir veri ikili sayı biçiminde saklanmaktadır. Buna harfler ve karakterler de dâhildir. Bir karakter belleğe kaydedildiğinde, önce sayısal bir koda dönüştürülür. Sayısal kod daha sonra bellekte ikili sayı olarak saklanır. Yıllar içinde bilgisayar belleğindeki karakterleri temsil etmek için farklı kodlama sistemleri geliştirilmiştir. Bu kodlama şemalarından en önemlisi Bilgi Değişimi için Amerikan Standart Kodu (American Standard

Code for Information Interchange) anlamına gelen ASCII'dir. ASCII, İngilizce harfleri, çeşitli noktalama işaretlerini ve sembolleri içeren 128 farklı (33'ü yazdırılamayan kontrol karakteri ve 95'i yazdırılabilen karakter) sayısal koddur. Örnek olarak büyük "A" harfinin kodu 65 iken küçük "a" harfinin kodu 97'dir. Klavyeden büyük A harfi yazdığınızda, 65 sayısının ikili karşılığı olan 1000001 belleğe kaydedilmektedir. ASCII tablosu Tablo 1.2'de gösterilmektedir.

Tablo 1.2 ASCII tablosu.

İkili	Onluk	On altılık	Karakter	İkili	Onluk	On altılık	Karakter	İkili	Onluk	On altılık	Karakter	İkili	Onluk	On altılık	Karakter
000 0000	0	00	Null	010 0000	32	20	(boşluk)	100 0000	64	40	@	110 0000	96	60	·
000 0001	1	01	Start of header	010 0001	33	21	!	100 0001	65	41	A	110 0001	97	61	a
000 0010	2	02	Start of text	010 0010	34	22	"	100 0010	66	42	B	110 0010	98	62	b
000 0011	3	03	End of text	010 0011	35	23	#	100 0011	67	43	C	110 0011	99	63	c
000 0100	4	04	End of trans.	010 0100	36	24	\$	100 0100	68	44	D	110 0100	100	64	d
000 0101	5	05	Enquiry	010 0101	37	25	%	100 0101	69	45	E	110 0101	101	65	e
000 0110	6	06	Acknowledgment	010 0110	38	26	&	100 0110	70	46	F	110 0110	102	66	f
000 0111	7	07	Bell	010 0111	39	27	'	100 0111	71	47	G	110 0111	103	67	g
000 1000	8	08	Backspace	010 1000	40	28	(	100 1000	72	48	H	110 1000	104	68	h
000 1001	9	09	Tab	010 1001	41	29	)	100 1001	73	49	I	110 1001	105	69	i
000 1010	10	0A	Line feed	010 1010	42	2A	*	100 1010	74	4A	J	110 1010	106	6A	j
000 1011	11	0B	Vertical tab	010 1011	43	2B	+	100 1011	75	4B	K	110 1011	107	6B	k
000 1100	12	0C	Form feed	010 1100	44	2C	,	100 1100	76	4C	L	110 1100	108	6C	l
000 1101	13	0D	Carriage return	010 1101	45	2D	-	100 1101	77	4D	M	110 1101	109	6D	m
000 1110	14	0E	Shift out	010 1110	46	2E	.	100 1110	78	4E	N	110 1110	110	6E	n
000 1111	15	0F	Shift in	010 1111	47	2F	/	100 1111	79	4F	O	110 1111	111	6F	o
001 0000	16	10	Data link escape	011 0000	48	30	0	101 0000	80	50	P	111 0000	112	70	p
001 0001	17	11	Device cont. 1	011 0001	49	31	1	101 0001	81	51	Q	111 0001	113	71	q
001 0010	18	12	Device cont. 2	011 0010	50	32	2	101 0010	82	52	R	111 0010	114	72	r
001 0011	19	13	Device cont. 3	011 0011	51	33	3	101 0011	83	53	S	111 0011	115	73	s
001 0100	20	14	Device cont. 4	011 0100	52	34	4	101 0100	84	54	T	111 0100	116	74	t
001 0101	21	15	Negative acknow.	011 0101	53	35	5	101 0101	85	55	U	111 0101	117	75	u
001 0110	22	16	Synchronous idle	011 0110	54	36	6	101 0110	86	56	V	111 0110	118	76	v
001 0111	23	17	End of trans. Block	011 0111	55	37	7	101 0111	87	57	W	111 0111	119	77	w
001 1000	24	18	Cancel	011 1000	56	38	8	101 1000	88	58	X	111 1000	120	78	x
001 1001	25	19	End of medium	011 1001	57	39	9	101 1001	89	59	Y	111 1001	121	79	y
001 1010	26	1A	Substitute	011 1010	58	3A	:	101 1010	90	5A	Z	111 1010	122	7A	z
001 1011	27	1B	Escape	011 1011	59	3B	;	101 1011	91	5B	[	111 1011	123	7B	{
001 1100	28	1C	File separator	011 1100	60	3C	<	101 1100	92	5C	\	111 1100	124	7C	
001 1101	29	1D	Group separator	011 1101	61	3D	=	101 1101	93	5D	]	111 1101	125	7D	}
001 1110	30	1E	Request separator	011 1110	62	3E	>	101 1110	94	5E	^	111 1110	126	7E	~
001 1111	31	1F	Unit separator	011 1111	63	3F	?	101 1111	95	5F	_	111 1111	127	7F	DEL

ASCII karakter seti 1960'ların başında geliştirildikten sonra birçok bilgisayar üreticisi tarafından benimsenmiştir. Ancak ASCII yalnızca 128 karakter için tanımlandığı için oldukça sınırlı kalmıştır. 1981 yılında IBM tarafından 256 karakterlik genişletilmiş ASCII tanıtılmıştır. Genişletilmiş

ASCII, birçok Avrupa dilinde kullanılan karakterleri içermesine rağmen dünya dillerine ait karakterleri içermemektedir. Bu eksikliği gidermek için 1991 yılında Unicode karakter seti geliştirilmiştir. Unicode, ASCII ile uyumludur ve dünyadaki birçok dilde kullanılan karakterleri

de temsil edebilen kapsamlı bir kodlama şemasıdır. Bugün, Unicode bilgisayar endüstrisinde kullanılan standart karakter seti haline gelmiştir.

Unicode'un 2021 yılı sonunda tanıtılan son sürümü 144.697 farklı karakteri içermektedir.



DİKKAT EDELİM

Bir kaynaktan (dosya, veritabanı, web hizmeti) elde ettiğiniz metnin karakterleri düzgün görüntülenmiyorsa hatalı karakter kodlaması kullanılmış olabilir.

Unicode karakter seti 32 bit ile temsil edilmektedir. Bu, her bir karakter 4 bayt yer kaplayacak anlamına gelmektedir. İkilik karşılığı 32 bitten kısa sayıların başlarına sıfır eklenerek 32 bite tamamlama yöntemine göre geliştirilmiş UTF-8, UTF-16 ve UTF-32 (Unicode Transformation Format) Unicode dönüşüm biçimleri ile karakterlerin

daha az bellek alanı ile ifade edilmesi sağlanmıştır. Günümüzde yaygın olarak UTF-8 ve UTF-16 kodlama biçimleri kullanılmaktadır. Bir karakter Unicode ile temsil edilirken “U+” dan sonra karakterin onaltılık sayı ile gösterilen kod karşılığı kullanılmaktadır. Örneğin “A” harfinin Unicode karşılığı “U+0041” şeklindedir.



GÜNLÜK HAYATLA İLİŞKİLENDİRELİM

Unicode karakter seti, Unicode Konsorsiyumu tarafından yeni karakter eklenerek güncellenmektedir. Örneğin ülkemizde 1 Mart 2012’de kullanılmaya başlanan ₺ Türk para birimi simgesi Eylül 2012’de yayınlanan Unicode sürümüne eklenerek U+20BA koduyla tüm dünyada standart bir simge haline gelmiştir.

Tablo 1.3 Bazı karakterler ve Unicode onaltılı sayı karşılıkları.

Unicode	Karakter
U+0041	A
U+20AC	€
U+263E	☺
U+20BA	₺
U+5BB6	家

5.3. Diğer Sayı Tiplerini Saklamak

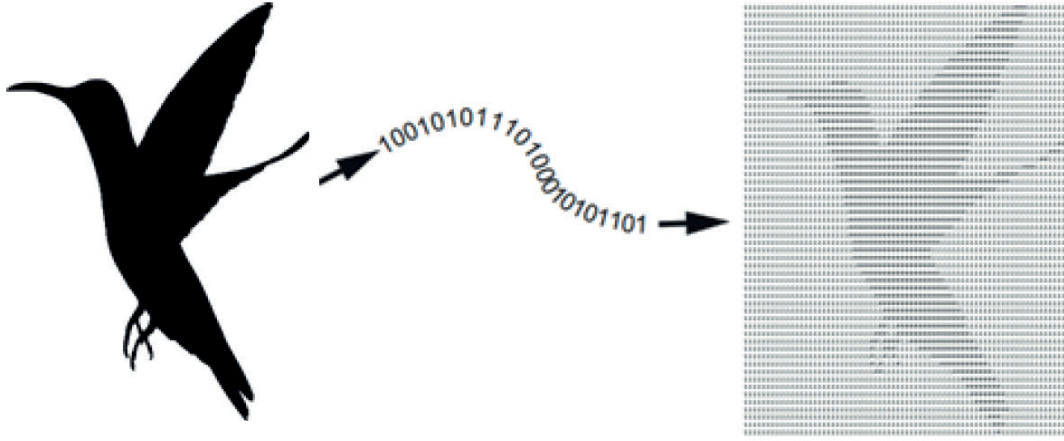
Yukarıda sayıların bellekte nasıl saklandıkları anlatılmıştır. Ancak negatif sayılar ve reel sayılar (3,14159 gibi) daha önce gösterilen basit ikili sayıya çevirme tekniği kullanılarak temsil edilemez. Bilgisayarlar, negatif sayıları ve reel sayıları bellekte saklayabilir ancak bunu yapmak için ikili sayı sistemi ile birlikte kodlama şemalarını

kullanırlar. Negatif sayılar, ikiye tümleyen olarak bilinen bir teknik kullanılmaktadır. Reel sayılar ise kayan nokta gösteriminde kodlanmaktadır. Bu kodlama şemalarının nasıl çalıştığına burada değinilmeyecektir. Negatif sayıların ve reel sayıların ikili sayı biçime dönüştürmek için bu kodlamaların kullanıldıklarını bilmek yeterlidir.

5.4. Diğer Veri Tiplerini Saklamak

Bilgisayarlar, sayı ve karakterler dışında birçok dijital veri türüyle de çalışmaktadır. Örneğin, dijital kamera ile çekilen fotoğraflarda görüntüler piksel olarak bilinen küçük renkli noktalardan oluşmaktadır. Bir görüntüdeki her piksel, pikselin rengini temsil eden sayısal bir koda dönüşür ve bu sayısal kod, bellekte ikili sayı olarak saklanmaktadır. Piksellerin renk kodlarının yanı sıra fotoğrafın her pikselinin kaç farklı renkle ifade edildiği, piksellerin boyutu ve fotoğrafın ebatları da saklanmaktadır. Fizikte kırmızı, yeşil, mavi (Red, Green, Blue-RGB) olmak üzere üç ana

renk vardır. Tüm renkler bu renklerin karışımı ile elde edilmektedir. Fotoğraflar 1, 4, 8, 16, 24, 32 bit renkli olabilmektedir. Günümüzde çoğu kamera ve tarayıcı 32 bit renkli görüntü işleyebilmektedir. Şekil 1.9'da bir bitlik (siyah-beyaz/1-0) görüntünün ikili sayı formatında bellekte nasıl kodlandığı gösterilmiştir. Renkli 24 bit bir fotoğrafta ise bir piksel için ilk 8 bit kırmızı, sonraki 8 bit yeşil ve son 8 bit ise mavi renk tonunu ifade etmek için kullanılmaktadır. Şekil 1.10'da 24 bit renk 50x50 piksel boyutlu bir görüntüde kullanılan piksel renk değerleri gösterilmektedir.



Şekil 1.9. Bir bitlik fotoğraf ve bellek görüntüsü.



Piksel rengi	Kırmızı (8 bit)	Yeşil (8 bit)	Mavi (8 bit)
	0 00000000	162 10100010	232 11101000
	240 11110000	255 11111111	9 00001001
	255 11111111	0 00000000	0 00000000
	0 00000000	0 00000000	0 00000000

Şekil 1.10. 24 bitlik fotoğraf ve piksel renk değerleri.

6. PROGRAMLAR NASIL ÇALIŞIR

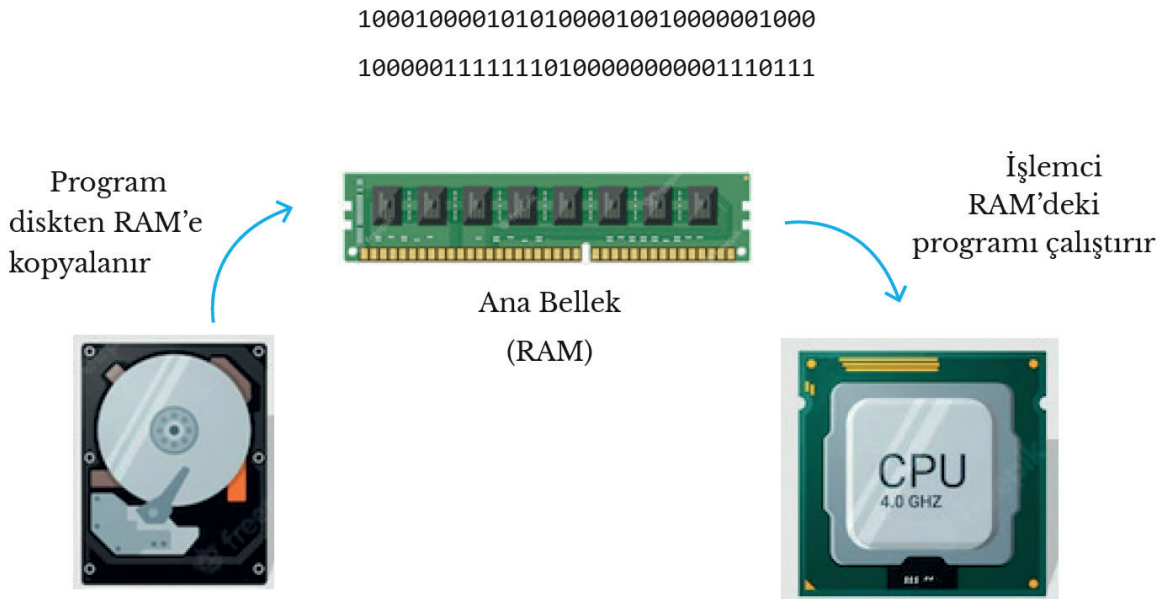
Bir bilgisayarın işlemcisi yalnızca makine dilinde yazılmış talimatları anlayabilmektedir. Makine dilinde talimat yazmak çok zor olduğundan çeşitli programlama dilleri geliştirilmiştir. İşlemci özellikle aşağıdaki gibi işlemleri gerçekleştirmek üzere tasarlanmış elektronik bir cihazdır:

- Ana bellekten bir veri parçasını okumak
- İki sayıyı toplamak
- Bir sayıyı diğer sayıdan çıkarmak
- İki sayıyı çarpmak
- Bir sayıyı başka bir sayıya bölmek
- İki sayının eşit olup olmadığını belirlemek
- Bir verinin bellekteki konumu değiştirmek

Bu listeden de görebileceğiniz gibi işlemci veriler üzerinde basit işlemler gerçekleştirmektedir. Ancak işlemci, kendi başına hiçbir şey yapmadığı için programlar ile ne yapması gerektiği ona söylenmelidir. Bir program, işlemcinin işlemleri gerçekleştirmesi sağlayan talimatlar bütünüdür. Bir programdaki her talimat, işlemciye belirli bir işlemi gerçekleştirmesini söyleyen bir

komuttur. Bir programda görünebilecek bir talimat örneği: 100010 şeklindedir. Bu, insanlar için sadece bir 0'lar ve 1'ler dizisiyken işlemci için “çıkarma” işlemi anlamına gelmektedir. İşlemcinin gerçekleştirebildiği her işlem için bir makine dili talimatı mevcuttur ve bunlara işlemci talimat seti denilmektedir.

Bilgisayarın anlamlı bir şey yapması için birden fazla talimat gerekmektedir. Bir program binlerce hatta milyonlarca makine dili talimatı içermektedir. Programlar genellikle sabit disk gibi ikincil bir depolama aygıtında saklanmaktadır. Programlar, ikincil bir depolama birimlerinde saklanmalarına rağmen işlemci tarafından her çalıştırıldığında ana belleğe/RAM'e kopyalanması gerekir. Örneğin, bilgisayarınızın diskinde bir kelime işlemci programınız olduğunu varsayalım. Programı yürütmek için fareyi kullanarak programın simgesine çift tıkladığınızda program diskten okunarak ana belleğe kopyalanır. Daha sonra işlemci, ana bellekte kopyası bulunan programı yürütür (Şekil 1.11).



Şekil 1.11 Program RAM'e yüklendikten sonra çalıştırılır.

İşlemci bir programdaki talimatları çalıştırdığında, getir, kod çöz ve çalıştır döngüsü (Fetch-Decode-Execute Cycle) olarak bilinen bir işlemle meşgul olur. Komut çevrimi adı verilen bu döngü, programdaki her komut için tekrarlanmaktadır.

- Getir: Program, uzun bir makine dili talimatları dizisidir. Döngünün bu ilk adımında, bir sonraki

talimat ana bellekten bulunur ve işlemci yazmaçlarına saklanır.

- Kod çöz: Bir makine dili talimatı, işlemciye bir işlem gerçekleştirmesini söyleyen komutu temsil eden bir ikili sayıdır. Bu adımda işlemci, hangi işlemi gerçekleştirmesi gerektiğini belirlemek için ana bellekten alınan talimatın kodunu çözer.

- Çalıştır: Döngünün son adımında, işlemci komutu çalıştırarak sonuçları yazmaçlara saklar. Gerekli işlemden sonra, bir sonraki komut çevrimi için işlemci hazırlanır.

BÖLÜM ÖZETİ

Bilgisayarlar, kendilerine verilen talimatlara göre verileri okuyarak, aritmetik ve mantıksal işlemler gerçekleştiren, bu işlemlerin sonuçlarını kullanıcıya anlamlı bir şekilde sunan ve verileri hafızasında saklayan programlanabilir elektronik cihazlardır. Bir bilgisayar, donanım ve yazılımdan oluşur. Donanım; işlemci, bellek, sabit disk, ekran, klavye ve fare gibi fiziksel bileşenlerden meydana gelirken, yazılım ise bilgisayara hangi işlemleri yapması gerektiğini bildiren talimatlar bütünüdür. Bilgisayarlar görünüşte farklılık gösterse de belirli birimlerden oluşur. Günümüz bilgisayarlarının temel çalışma prensipleri, 1945 yılında John von Neumann tarafından geliştirilen mimariye dayanır. Bu mimari, bir işlemci, bir bellek ve giriş-çıkış birimlerinden oluşan bir sistem öngörür.

Merkezi İşlem Birimi, bilgisayar sisteminin beyni olarak işlev görür. İşlemci, sistemdeki birimlerin çalışmasını denetler, veri akışını kontrol eder ve veriler üzerinde işlemler gerçekleştirir. Mikroişlemci veya işlemci olarak da adlandırılan bu birim, Aritmetik ve Mantıksal Birim ile Kontrol Birimi olmak üzere iki ana bölümden oluşur. Aritmetik ve Mantıksal Birim matematiksel ve mantıksal işlemleri yaparken, Kontrol Birimi tüm işlem süreçlerini yönetir. İşlemci, verileri bellekte depolar, işler ve sonuçları tekrar belleğe kaydeder. İşlemci performansını optimize etmek için yazmaçlar, ön bellekler, ana bellek (RAM) ve ikincil belleklerden oluşan bir bellek hiyerarşisi kullanılır. Yazmaçlar ve ön bellekler işlemciyle aynı hızda çalışırken, RAM daha yavaş bir hızda çalışır. İkincil bellekler (sabit disk, USB bellek, optik disk gibi) ise uzun süreli veri depolama için kullanılır ve elektrik kesildiğinde bile verileri korur.

Elektronik devrelerde elektrik akımının varlığı ya da yokluğu kolayca kontrol edilebilir. Bu durum, bilgisayarların verileri ikili sayı sistemine göre saklamasına olanak tanır. İkili sayı sisteminde yalnızca 0 ve 1 bulunur; 0 akımın olmadığını, 1 ise akımın var olduğunu ifade eder. Bilgisayarlar, tüm bilgileri (sayı, metin, fotoğraf, ses, video vb.) ikili sayı sistemine dönüştürerek depolar. Bellek birimi olarak adlandırılan bit, yalnızca 0 veya 1 değerini alabilir. Daha anlamlı veriler oluşturmak için 8 bitten oluşan baytlar kullanılır. Bir baytlık bellek hücresinde en fazla 255 farklı değer saklanabilir. Daha büyük sayılar ya da karakterler saklanmak istendiğinde daha fazla bellek hücresi gerekir.

Harfler ve sembollerin sayısal olarak ifade edilmesi için çeşitli kodlama standartları geliştirilmiştir. 1960'larda geliştirilen ASCII karakter seti, her karakter için bir sayısal kod tanımlar; örneğin "A" harfi için bu kod 65'tir. ASCII'nin yetersiz kaldığı durumlar için 1980'lerde genişletilmiş ASCII kullanıma sunulmuş ve karakter sayısı 127'den 255'e çıkarılmıştır. Tüm dünya dillerine ait karakterlerin kapsanabilmesi amacıyla 1991 yılında Unicode karakter seti oluşturulmuştur. Unicode, ASCII ile uyumludur ve 2021 yılı itibarıyla 144.697 farklı karakter içermektedir. Karakterlerin daha az bellek alanı kullanarak saklanabilmesi için UTF-8, UTF-16 ve UTF-32 gibi dönüşüm biçimleri geliştirilmiştir.

Bilgisayarlar, görüntüleri piksel adı verilen küçük renkli noktalar halinde işler. Her piksel, belirli bir renk koduyla temsil edilir ve bu kod ikili sayı sisteminde bellekte saklanır. Siyah-beyaz bir görüntü için her piksel 1 bit ile ifade edilirken, dijital bir fotoğraf için bu sayı genellikle 32 bit'e kadar çıkar.

Yazılım, bilgisayar sistemini yönetmek için gereken talimatlar bütünüdür ve program olarak da adlandırılır. Yazılımlar, sistem yazılımları (işletim sistemleri, programlama dilleri, antivirüs yazılımları vb.) ve uygulama yazılımları (kelime işlemciler, bilgisayar oyunları, web tarayıcıları vb.) olmak üzere iki ana gruba ayrılır. Sistem yazılımları, bilgisayar donanımını yönetirken, uygulama yazılımları belirli kullanıcı ihtiyaçlarını karşılar.

Bir bilgisayarın işlemcisi, verileri ana bellekten alarak işler. İkincil bellekte saklanan ve ikili sayı sistemine göre kodlanmış programlar, çalıştırılmak üzere ana belleğe aktarılır. Komut çevrimi adı verilen bu süreçte, işlemci bellekteki bir veri bloğunu alır, kodunu çözer ve işlemi gerçekleştirir. İşlemin sonucu yazmaçlara kaydedilir ve bir sonraki komut çevrimine geçilir. Bu döngü, bilgisayar kapatılana kadar devam eder.

GÖZDEN GEÇİRELİM

1. Bilgisayar sistemini yöneten donanım bileşeni aşağıdakilerden hangisidir?
 - a) Ana kart
 - b) Ana bellek
 - c) Merkezi işlem birimi
 - d) Sabit disk
 - e) RAM bellek
2. Aşağıdakilerden hangisi ikincil depolama aygıtlarından değildir?
 - a) SSD disk
 - b) DVD
 - c) Sabit disk
 - d) USB bellek
 - e) Ana bellek
3. Aşağıdakilerden hangisi bilgisayar sistemi kapalı olsa bile verileri uzun süre saklar?
 - a) İşlemci
 - b) Yazmaç
 - c) RAM
 - d) Sabit disk
 - e) Ön bellek
4. Aşağıdakilerden hangisi uygulama yazılımı türü değildir?
 - a) Kelime işlemci
 - b) Resim düzenleme programı
 - c) Web tarayıcısı
 - d) Bilgisayar oyunları
 - e) İşletim sistemi
5. Aşağıdakilerden hangisinde bellek ölçü birimleri doğru sıralanmıştır?
 - a) Kilobayt < Gigabayt < Terabayt < Megabayt
 - b) Byte < Megabayt < Kilobayt < Yottabayt
 - c) Megabayt < Zettabayt < Yottabayt < Exabayt
 - d) Terabayt < Petabayt < Exabayt < Zettabayt
 - e) Terabayt < Petabayt < Yottabyte < Zettabayt
6. Bellekte alfabenin bir harfini veya küçük bir sayıyı saklamak için ne kadar bellek yeterlidir?
 - a) 1 bit
 - b) 1 bayt
 - c) 2 bayt
 - d) 4 bayt
 - e) 1 kilobayt
7. Aşağıdakilerden hangisi ikili sayı sistemindeki (11101001)₂ sayısının, onlu sayı sistemindeki karşılığıdır?
 - a) 139
 - b) 115
 - c) 116
 - d) 228
 - e) 233

8. “Merhaba Dünya” ifadesi genişletilmiş ASCII kodlama sistemine göre hafızada ne kadar yer tutar?

- a) 12 bit
- b) 12 bayt
- c) 13 bayt
- d) 24 bayt
- e) 26 bayt

9. Unicode kodlama şeması ile ilgili aşağıdakilerden hangisi yanlıştır?

- a) Her karakter bellekte aynı boyutta alan tutar
- b) Daha önce geliştirilen kodlama sistemlerini destekler
- c) Büyük ve küçük harfler farklı kodla temsil edilir
- d) Mevcut karakterlere yeni karakterler eklenemez
- e) Dünya dillerinin çoğuna ait karakterleri kapsar

10. Aşağıdakilerden hangisi bir bilgisayar sistemi ana belleğinin (RAM) işlevlerinden değildir?

- a) Çalıştırılacak programların kodunu çözer
- b) İşletim sisteminin çalışması için gerekli bileşenleri saklar
- c) Sistemde çalışan programlara ait komutları ve verileri saklar
- d) Programların oluşturduğu sonuçları saklar
- e) İkincil depolama aygıtlarına ve çıktı birimlerine gönderilecek bilgileri saklar

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

Kaynakça

- Demir, S., Eryiğit, R., Ar, Y., Tuğrul, B., Gök, O., Kul, S., . . . Yağanoğlu, M. (2019). Programlama Temelleri. Erzurum: Atatürk Üniversitesi Açıköğretim Fakültesi.
- Gaddis, T. (2018). Starting Out with Python. Pearson.
- Kaleli, C., Bilge, A., Ergin, S., & Şora Günel, E. (2016). Bilgisayar ve Programlamaya Giriş. Eskişehir: Anadolu Üniversitesi.

Faydalı İnternet Kaynakları

- <https://acikders.ankara.edu.tr/course/view.php?id=5157>
- https://www.youtube.com/playlist?list=PLsH-EmuhVHnZxP0LKJq0ca6_LDU5NJku5
- <https://www.youtube.com/watch?v=5Mh3o886qpq>
- <http://www.khanacademy.org.tr/bilgisayar-dunyasi/bilgisayar-bilimi/24>

BÖLÜM 2

PROGRAMLAMA TEMELLERİ

ANAHTAR KAVRAMLAR

Komut (Command), Program, Kod (Code), Derleyici (Debugger), Yorumlayıcı (Interpreter), Düzenleme (Editing), Derleme (Compiling), Birleştirme (Linking), Yükleme (Loading), Çalıştırma (Executing), Hata Ayıklama (Debugging) Programlama Dilleri

ÖĞRENME HEDEFLERİ

-  1 Programlama hakkında gerekli teknik terminolojiyi bilme
-  2 Program yazma süreci ve ihtiyaçlarını bilme/ayırt etme
-  3 Program yazmak için kullanılan programlama dillerini tanıma

İKİNCİ BÖLÜM

GİRİŞ

Kitabımızın ikinci bölümünde programlamada geçen temel kavramlara ve programlama dillerine detaylı bir şekilde değinilmektedir. Programlama öğrenmek isteyen kişilerin diğer öğrenme yöntemlerinde olduğu gibi öncelikle terminolojiye ve program geliştireceği sisteme hem donanımsal hem de yazılımsal olarak hâkim olması gerekir. Birinci bölümünde program geliştireceğimiz bilgisayar sistemi, sayı sistemleri ve donanım üzerine bilgiler sizlere verilmiştir. Bu bölüme başlamadan önce bilgisayar hakkındaki bilgilerinizi gözden geçirmeniz gerekmektedir. Bölüm 2’de ise programlamaya giriş yapılacaktır. Bu sebeple, Komut (Command), Program, Kod (Code), Derleyici (Compiler), Yorumlayıcı (Interpreter), Düzenleme (Editing), Derleme (Compiling), Birleştirme (Linking), Yükleme (Loading), Çalıştırma (Executing), Hata Ayıklama (Debugging) ve Programlama Dilleri gibi kavramlar açıklanacaktır.



DİKKAT EDELİM

Bir program yalnızca bilgisayar sistemi üzerine geliştirilmez. Günlük hayatta sıklıkla kullandığımız tabletler, telefonlar hatta su ısıtıcı üzerinde de program geliştirilebilir.



1. TEMEL KAVRAMLAR

Temel kavramlar bölümü programlama ile uğraşan kişilerin gündelik hayatta kullandığı unsurları açıklamaktadır. Programlama işlemi bir dizi komutun belli bir sırayla donanımsal kaynaklar üzerinde yürütülmesi ile sağlanır. İlgili süreçte, program yazımında kullandığınız editörler (düzenleyici) ve donanımsal kaynaklar önemlidir. Bölüm 4’de C++ programlama dili kullanılarak program yazacağınız editörden bahsedilecektir. Kullandığımız işletim sisteminin sürümü ve işlemcinin türü dikkat edilmesi gereken bir husustur.

Bu sebeple, program yazan kişiler ürünlerini dağıtırken donanımsal ve yazılımsal kısıtlardan bahsederler. Ayrıca bazı programlar, kullandığımız programlama diline göre işletim sisteminin bağımsız olarak çalıştırılabilirler. Örneğin Java programlama dili ile bahsi geçen türde bir program yazılabilir. Programlama dili ile ilgili detaylı bilgilere ilerleyen kısımlarda yer verilmektedir. Öncelikle programlama için kullanılan temel terminolojik kavramlara yer verilmiştir.



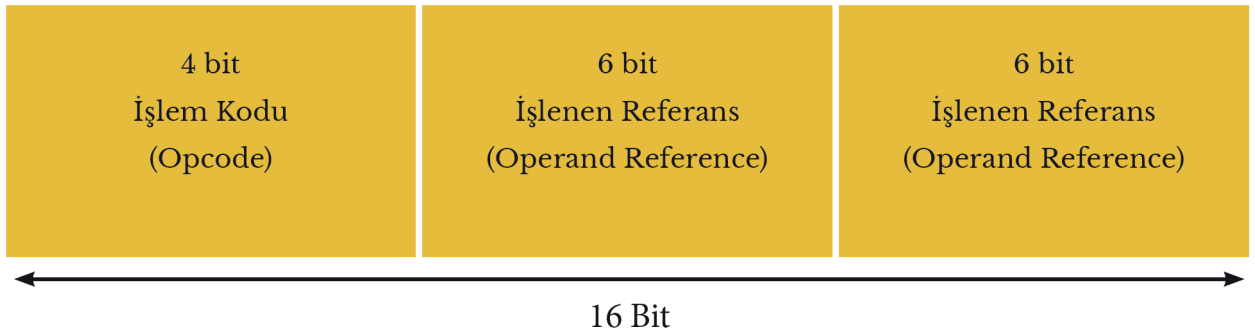
EK BİLGİ

Intel, AMD, Cyrix ve Motorola işlemcilerin mimarileri ve kullandıkları komut kümeleri aynı olma-yabilir. Bu farklılıktan dolayı yazılan programların çalıştırılacak sistemlerde değişik etkilere sahip olabileceğini unutmayınız!

Komut (Command)

İkili sayı sistemindeki elemanların (0 ve 1’ler) farklı şekillerde dizilişi ile komutlar oluşur. Komutlar, donanım birimlerinin tasarlanmasında ve donanımsal birimlerin birbirleri ile iletişim kurmasında kullanılır. Ayrıca her işlemci üreti-

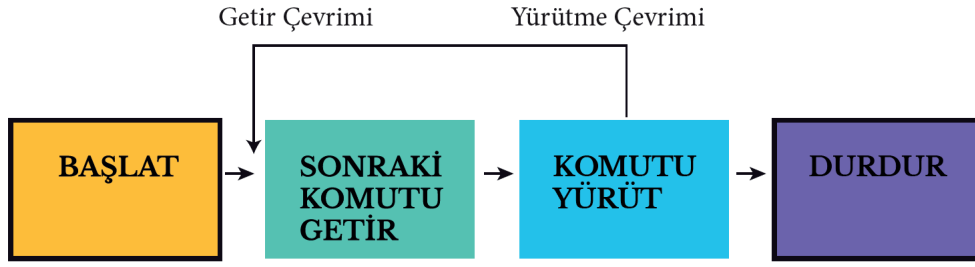
lirken kendisine ait özel komutlara sahip olur. Programlarda yapılması istenen işlemler işte bu komutlar aracılığıyla işlemciye iletilmektedir. İşlemci kendi üstüne gelen komutlara göre diğer donanım birimleri ile iletişime geçer. Şekil 2.1’de örnek bir komut çerçevesi gösterilmektedir.



Şekil 2.1. Basit Bir Komut Çerçevesi.

Şekil 2.1’de verilen komut çerçevesinde 16 bitlik bir komut verilmiş olup, burada ilk 4 bit komutun ne olduğunu belirlemede sonraki 6’lık bitler ise kaynak ve sonuç adresleri göstermede kullanılmaktadır. Böylece komutun ne iş yapacağı ve nerede çalışacağı işlemci tarafından anlaşılır.

Şekil 2.2’de ise işlemci üzerine gelen basit bir komut çevrimi örneği verilmektedir. Çevrimin (Getir-Yürüt-Kodunu Çöz Çevrimi) animasyonlu anlatımı için bölümün sonunda verilen Faydalı Kaynaklar ve İnternet Kaynakları kısmına bakılabilir.



Şekil 2.2. Basit Komut Çevrimi.

Şekil 2.2’de komut çevrimi için kullanılan komut, işlemcide bulunan ve komut kaydı (Instruction Register IR) olarak bilinen bir kayıt defterine yüklenmektedir. Komutlarda işlemcinin gerçekleştireceği eylemi belirten bitler bulunmaktadır.

Bu süreçte işlemciye iletilen komut yorumlanır ve gerekli eylem gerçekleştirilir. Genel olarak, bu eylemler dört kategoriye ayrılır (Tablo 2.1).

Tablo 2.1. Farklı Kategoride Yer Alan Komutlar.

Kategori	Açıklama
İşlemci-bellek	Veriler işlemciden belleğe veya bellekten işlemciye aktarılabilir.
İşlemci-G/Ç	Veriler, işlemci ve bir G/Ç modülü arasında aktarım yapılarak bir çevresel ağıta veya bu ağıttan işlemciye aktarılabilir.
Veri işleme	İşlemci, veriler üzerinde bazı aritmetik veya mantıksal işlemler gerçekleştirebilir.
Kontrol	Bir komut, yürütme sırasının değiştirilmesini belirtebilir.

Tablo 2.1’de kategorize edilen ve ikilik dizilerden oluşan komutlar yani makine kodu sadece donanım tarafından anlaşılabilir. Programı yazarların kolaylıkla anlayabilmesi adına bu dize-ler için semboller kullanılmaktadır. Söz konusu semboller Assembly kodu olarak anılmaktadır. İki bellek konumunda saklanan numaraları top-

layan bir işlem düşünelim. Matematiksel olarak, bu program $x = 2$, $y = 5$, $x + y = z$ formüllerini temsil eder. Burada x , y ve z değişkenleri sırasıyla 13, 14 ve 15 bellek konumlarına karşılık gelsin. Programın yönergeleri Tablo 2.2’de listelenmiştir.

Tablo 2.2. Makine ve Assembly Kodu Gösterimi ile Toplama İşlemi Yapan Program.

#	Makine Kodu	Assembly Kodu	Tanım
0	001 1 000010	LOAD #2	Akümülatöre 2 değerini yükleyin
1	010 0 001101	STORE 13	Akümülatörün değerini bellek konumu 13’te saklayın
2	001 1 000101	LOAD #5	Akümülatöre 5 değerini yükleyin
3	010 0 001110	STORE 14	Akümülatörün değerini bellek konumu 14’te saklayın
4	001 0 001101	LOAD 13	Bellek konumu 13’ün değerini Akümülatöre yükleyin
5	011 0 001110	ADD 14	Akümülatöre bellek konumu 14’ün değerini ekleyin
6	010 0 001111	STORE 15	Akümülatörün değerini bellek konumu 15’te saklayın
7	111 0 000000	HALT	Yürütmeyi durdur

Tablo 2.2’de komutların makine kodu, Assembly kodu ve tanımı verilmektedir. Anlaşılacağı üzere LOAD yükleme, STORE depolama ve ADD ekleme işlemi için kullanılmaktadır. Toplama işlemini gösteren bu programda olduğu gibi komutların Assembly kodu olarak ifade edilebilmesi ile programcı verilerin bellek adresleri ve sembolleri ile yeni programlar yazabilmektedir. Bu tarz bir program yazabilmek adına işlemcilerin komut kümelerini bilmemiz gerekir. Günümüzde bu dillerin öğrenilmesi zor olmakla birlikte donanımın yakın bir dile sahip olması nedeni ile sadece donanım üzerinde program yazanlar için avantaj haline gelmektedir. Üniversitelerde bu tarz diller mikroişlemci, sistem programlama gibi isimler ile okutulmaktadır. Programlama dillerinde seviyeler kısmında bu konuya tekrar değinilecektir.

Program

Belli bir işlemi bilgisayar veya uygun yapılarda yerine getirebilmek adına kullanılan komutlar dizisine program adı verilmektedir. Bir programı oluşturabilmek adına ilgili komutların belirlenmesi ve bunların istenilen biçimde kullanılmasına da programlama adı verilmektedir. Program yazan kişiler ise Programcı olarak ifade edilir. Programın nihai hedefi son kullanıcıdır (End-User). Son kullanıcının görmediği taraftaki mimariyi oluşturanlar arka-uç programcı (Back-End), grafiksel tasarım geliştirenler ise ön-uç programcı (Front-End) olarak isimlendirilir. Bilişim dünyasında programcıların Masaüstü, Web, Mobil ve Veritabanı programlama olarak geliştirme yaptığı uygulama türleri bulunmaktadır (Tablo 2.3).



ARAŞTIRALIM

Assembly kodlarının kullanılabilmesi adına sembollerle işlem yapan programlama dilleri vardır. Emu8086 Assembly dilini Intel8086 komut kümesi ile kullandığınız bir ortamdır

Tablo 2.3. Uygulama Türleri.

Uygulama Türü	Geliştirildiği Yer	Örnek
Masaüstü	İşletim sistemi (Windows, Linux, MAC vb.)	Winrar, GIMP vb.
Web	İnternet, Intranet	https://www.ankara.edu.tr/ vb.
Mobil	iOS, Android vb.	Whatsapp, PUBG vb.
Veritabanı	Veritabanı yazılımları (Oracle, MySQL vb.)	Masaüstü, Web, Mobil uygulamaların alt yapısı için ya da Fatura, Kayıt Formu gibi raporlar

Bir programcı Tablo 2.3’de verildiği türdeki gibi uygulamalar geliştirmektedir. Burada önemli olan ihtiyacı belirlemektir. Geliştirilecek programlama son kullanıcının gereksinimlerini karşılamalıdır. İhtiyaç analizi yapıldıktan sonra başlatılan programlama sürecinde bir ya da bir-

den fazla türde uygulama geliştirilebilir. Genelde analiz yapmak için Şelale (Waterfall), Scrum, Aşırı Programlama (Extreme Programming), Birleşik Rasyonel İşlem (Rational Unified Process) gibi Çevik (Agile) proje yönetim analizleri kullanılmaktadır.



GÜNLÜK HAYATLA İLİŞKİLENDİRELİM

Gündelik yaşantımızda ihtiyaç analizlerimizi belirlerken neler yaparız? Örneğin alış-verişe gitmeden önce; yeterince nakit ya da karşılığı olan diğer varlıklarımız (kredi kartı, çek, senet vb.) var mı?, nelere ihtiyaç var?, alış-verişte öncelik hangilerine verilmeli? şeklinde olabilir. Programlamaya başlamadan önce sizce nasıl analizler yapılır? Program yazma için gerekli olan analiz yöntemlerini inceleyelim.

Kaynak Kod (Source Code)

Kaynak kod bir programlama diliyle yazılan programların işlemcinin anlayabileceği makine diline dönüştürülerek işlenip yorumlanmasından önce programcının okuyabildiği ve üzerinde çalışabildiği kodlar olarak tanımlanır. Buradan da anlaşılacağı üzere kaynak kod programcı dostu bir yazıma sahiptir. Bu kodlar evrensel bir dilde tanımlanmalıdır. Böylece dünyanın herhangi bir yerindeki diğer programcılar tarafından incelenmesi mümkün olacaktır. Bu bağlamda, yeni bir özellik ekleme, sorun saptama ve farklı programlar geliştirmeye yardımcı olunmaktadır. Evrensel dilde anlaşmak programlama dillerinin doğmasına yol açmaktadır. Dersimizde kullanacağımız C++ böyle bir dil olup, işletim sistemlerinin yazılmasından, çeşitli amaçlara hizmet eden pek çok programa kadar geniş bir yelpazeye sahiptir.

Her kaynak kod bir dosyada muhafaza edilir. Örneğin;

- C++ için <dosya_adı>.cpp
- C# için <dosya_adı>.cs
- Java için <dosya_adı>.java
- Python için <dosya_adı>.py

Kaynak kodlar bir bilgisayar üzerinde direkt çalıştırılmazlar. Bunun için gerekli bir takım yazılımların yüklenmesi gerekmektedir. Bölüm 4'de detaylı bir şekilde anlatılacaktır.

Amaç Kod (Object Code)

Amaç kod ise işlemcinin komut kümeleri kullanılarak kaynak kodun makine dile çevrilmesidir. Böylece programcı hem kendi anlayabildiği bir dilde program yazmış olacak hem de derlenen program makinenin anladığı bir dile dönüştürülecektir. Bu tarz dosyalar genelde .obj uzantılı dosyalarda ya da obj isimli klasörlerde tutulmaktadır. Ancak düzenlenmesi için oluşturulan dosyalar değildir. Amaç kod yazılan kaynak koddan birleştirme işlemi ile çalıştırılabilir bir dosya oluşturmak adına kullanılır.

Derleyiciler (Compiler)

Derleyiciler programlama ile uğraşan kişilerin en önemli aracıdır. Söz konusu süreçte kullanılan programlama dili hangi derleyicileri kullanacağımızı da belirlemektedir. Herhangi bir programlama diliyle yazılmış olan kaynak kodumuzun, makine diline yani amaç programa dönüştüren özel programlara derleyici adı verilmektedir. Burada yazılan kaynak kodun tamamı tek seferde derlenmektedir. Bu anlamda, hem çevrimiçi hem de çevrim dışı kullanabilen pek çok derleyici bulunmaktadır. Java, C, C++, COBOL, Fortran çevrim dışı yani işletim sistemine yüklenerek kullanabilecek programlama dilleridir. Yanı sıra https://www.onlinegdb.com/online_c++_compiler web adresinde olduğu gibi C++ ile yazılmış kodları derleyen çevrim içi editörler de bulunmaktadır. Çevrim içi editörlerde bilgisayarımıza bir yazılım yüklemeye gerek kalmadan da programlarımızı derleyebiliriz. Bu tarz



uygulamalarda sadece aktif internet bağlantısına ihtiyacımız bulunmaktadır.

Yorumlayıcılar (Interpreter)

Evrensel dil kullanılarak oluşturduğumuz kaynak kodun ilk satırdan başlayarak satır satır makine diline çevrilmesine ve yazılan kodun çalıştırılmasında etken olan araca yorumlayıcı adı verilmektedir. Genelde web uygulaması için yazılan programlarda yorumlayıcılar kullanılmaktadır. Yazılan kaynak kodun herhangi bir satırında oluşan hata, programın bütününün çalışmasında sorun yarattığı için yorumlayıcılar bazı küçük hataları görmemektedir. Çünkü hata olan satırdan sonraki yazılan kaynak kodlar çalıştırılmayacaktır. Bu anlamda yorumlayıcı kullanılarak yazılan programlarda çok dikkatli olunması gerekmektedir. Bilgisayarlarımızda kullanıldığımız tarayıcılar aynı zamanda birer yorumlayıcıdır. Örneğin;

- Mozilla Firefox
- Google Chrome
- Microsoft Edge/Microsoft Internet Explorer
- Opera
- Netscape

Bunun dışında Perl, PHP, Python, Ruby ve JavaScript, kullanılan diğer yorumlayıcılar arasındadır.

Derleyici ve Yorumlayıcı Arasındaki Farklar

Programcılar sadece kaynak kod için kullanılan kısımları anlayabilmekte ya da düzenlemektedir. Bilgisayar veya diğer donanımsal yapılar ise yalnızca ikilik dilde yazılmış kodları anlayabilir. Bu nedenle program geliştirilen yapılar için derleyici ya da yorumlayıcıya ihtiyaç bulunmaktadır. Programcıların kullandığı sistemler ya yorumlama ya da derleme üzerine çalışmaktadır. Kullanı-

lan sistem kaynak kodu makinenin anlayabileceği koda yani makine diline çevirmektedir.

- Yorumlayıcı programcının yazdığı her bir satırdaki kod ifadesini okur, doğrudan dönüştürür ve yürütür.

- Derleyici programcının yazdığı bütün satırları tek seferde işletim sistemi tarafından çalıştırılabilen yerel (derlenmiş) koda çevirir. Bunun için derleme işlemi ile birlikte yürütülebilir (.exe uzantılı) bir dosya meydana getirir.

- Derleyici yorumlayıcıya göre kaynak kodu satır satır işletmediği için daha hızlı çalışmaktadır. Çünkü yorumlayıcı derleyicinin aksine tüm programı taramak yerine her seferinde bir ifadeyi çevirmektedir.

- Derleyiciler belleği daha verimli kullanma eğiliminde olan yorumlayıcıların aksine bağlanmak için daha fazla bellek gerektiren ara nesne kodu üretmek zorundadır.

- Bir yorumlayıcı, kodu tek bir işlemde okuyup çalıştırdığından, komut dosyası oluşturmada ve diğer küçük programlar için çok kullanışlıdır.

- Yorumlayıcı yazılan kaynak kodu satır satır çalıştırdığı için hatalı olan kısımlardan sonrasını çalıştırmaz. Bundan dolayı programın diğer kısımlarında yazılmış doğru kodlar çalıştırılmayacaktır. Derleyici ise yazılan kaynak kodun tamamını tek seferde işlettiği için hatalı kısımlar doğru yazılan kaynak kodları etkilemeyecektir.

Düzenleme (Editing)

Bulduğumuz çağda kod yazmak oldukça önemli bir hale gelmiştir. Bugün pek çok disiplininde işleri kolaylaştırmak adına farklı görevleri yerine getiren masaüstü, mobil ya da web uygulamaları geliştirilmektedir. Günümüzde nesnelerin interneti (IoT) kavramı ile de program geliştirilen araçların sayısı artmıştır. Programcı programını yazmadan önce bir kod düzenleyi-

ci belirlemelidir. Tümüleşik geliştirme ortamları (Integrated Development Environment IDE) kodlama, derleme ve hataları tespit edebilme adına pek çok süreçte kolaylık sağlamaktadır. Hatta artık pek çok IDE ile birden fazla programlama diline destek verilmektedir.

Microsoft firmasının geliştirdiği Visual Studio, 36 farklı programlama dilini destekler. Yerleşik diller arasında C, C++ , C++/CLI, Visual Basic .NET, C# , F# , JavaScript , TypeScript , XML , XSLT , HTML ve CSS bulunur. Python, Ruby, Node.js gibi diğer diller için de destek sunulmaktadır. Bazı diller ise eklentiler aracılığıyla kullanılabilir. Visual Studio'nun en temel sürümü olan Topluluk (Community) sürümü ücretsizdir.

Popüler olan düzenleyiciler,

- Visual Studio Code
- Cygwin
- Notepad++
- Sublime Text
- UltraEdit
- Atom
- Brackets
- GNU Emacs
- BBEdit
- CodePen

Derleme (Compiling)

Derleme işleminde, kullanılan programlama dilinin (C++, C# vb.) ön işlemcisi tarafından oluşturulan kaynak kod derleyici tarafından programın yazıldığı makinedeki işlemcinin komut kümesine uygun makine diline ve ardından işletim sistemine uygun amaç koda dönüştürülür. Aynı süreçte birleştirme işlemi ile çalıştırılabilir bir dosya oluşur.

Birleştirme (Linking)

Amaç kodun, son kullanıcının kullanabileceği yani yürütülebilir bir program oluşturması adına gerekli destekleyici kodla birleştirilmesi gerekmektedir. Bu adımda genellikle gerekli olan kitaplıklara ekleme işlemi bulunmaktadır. Birleştirme işlemi derleme işlemi ile birlikte tetiklenerek, amaç kod ile fonksiyonların ait oldukları kütüphaneler birleştirmede kullanılmaktadır. İşlem sonunda çalıştırılacak bir program dosyası oluşur.

Yükleme (Loading)

Programcı tarafından yazılan kodun yürütülmesinde çalışılan sistemin birincil belleği kullanılmaktadır. Bu sebeple programın öncelikle buraya yüklenmesi gerekmektedir. İşletim sistemleri yükleyici adı verilen bir bileşene sahiptir ve çalıştırılmaya ihtiyaç duyulan program ve gerekli kütüphane dosyası depolama biriminden okunarak bilgisayarın birincil belleğine yüklenir.

Çalıştırma (Executing)

Çalıştırma işleminde, derlenen/yorumlanan kaynak koddan üretilen makine dili programı ya da makine dili komutu işlemci aracılığıyla bellekten alınarak yürütme işlemi yapılmaktadır.

Hata Ayıklama (Debugging)

Derlenen/yorumlanan kod çalıştırılmadan önce varsa hatalarından ayıklanması gerekmektedir. Kullandığımız IDE aracılığıyla bu tarz bir işlem yapabiliriz. Burada üç türde hata ile karşılaşırız.

- Sözdizimi hatası: Dilbilgisi hatası gibidir. Programcının kullandığı programlama diline göre belli yazım kuralları vardır. Bu kurallara göre bir noktalama işaretinin unutulması, kelimelerin yanlış ya da eksik yazılması sözdizimi hatasıdır. Söz konusu hatalar, program derlen-

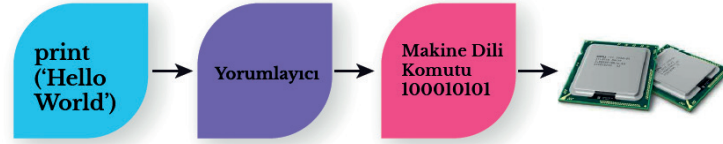
meden önce bilgisayar tarafından otomatik olarak algılanabilir.

- **Mantık hatası:** Bu hata türü sadece programcının tespit edebileceği türdendir. Yani temelde bilgisayarın çökmesine veya bir hata mesajı görüntülenmesine neden olmayan, ancak program çalıştığında istenen sonucu göremediğimizde oluşan hatalardır. Örneğin, matematiksel bir işlemde bir değeri 10 ile çarpmak yerine, yanlışlıkla 100 ile çarpmamız. Bilgisayar sonuçta işlemi yapacak ve hata vermeyecektir. Fakat sonuç yanlış hesaplanmış olacaktır.

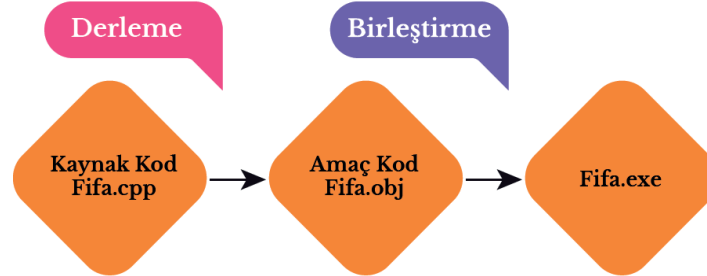
- **Çalışma zamanı hatası:** Program çalışırken bazı geçersiz komutlar yürütüldüğünde, progra-

mın çökmesi ve/veya bir hata ekranı açılması durumudur. Örneğin, haftanın her günü için 7 elemanlı bir verinin program ile belleğe aktarıldığını düşünelim. Program bu dizinin olmayan 8. konumundaki içeriğe erişmeye çalışırsa, komut geçersiz olduğundan ve bilgisayar talimatı nasıl işleyeceğini bilmediğinden bir çalışma zamanı hatası üretebilir. Bilgisayarların olası çalışma zamanı hatalarını önceden algılaması temelde imkânsızdır. Bunlar, programı test ederken tesadüfen keşfettiğiniz hata türleridir.

Şekil 2.3'de yorumlayıcı kullanan, Şekil 2.4'de ise derleyici kullanan programlama durumlarına diyagram kullanılarak örnek verilmiştir.



Şekil 2.4. Derleyici ile Programlama Süreci



Şekil 2.4. Derleyici ile Programlama Süreci.

2. PROGRAMLAMA DİLLERİ

Programlama dili sisteme girdi olarak verilen öğeleri çeşitli makine kodu çıktılarına dönüştüren ve kendine has kurallar dizisine sahip yazılımlardır. Programlama dilleri, bilgisayarın avantajlarını kullanarak arzu edilen işi hızlı ve verimli yerine getirmekte kullanılmaktadır. Ancak her dilin farklı kullanım amaçları olduğundan yapılmak istenen işe göre tercih edilmesi gerekir. Bu anlamda donanım programlamak için makine diline yakın diller ya da otomasyon geliştirmek adına kullanıcı dostu diller bulunmaktadır.

Programlama süreci 1800'li yıllara dayanmaktadır. Geçmişte çeşitli müzik enstrümanları ve dokuma tezgâhlarını programla fikri ile bir dizi komut kümesi kullanılmıştır. O zamandan süre gelen gelişimler programlama dillerini hemen hemen her türlü araç-gereçte kullanmaya imkân sağlamaktadır.

Günümüzde popüler olarak kullanılan programlama dilleri:

- **Python:** Programlamaya yeni başlayanlar için kullanılan ve sözdizimi basit bir dildir. Geniş

bir kitaplığı bulunmaktadır. C ve C++ gibi programlama dilleriyle kolay entegre olmaktadır. Yayıncılık, finansal hizmetler ve veri bilimi dâhil olmak üzere çok çeşitli uygulamalarda kullanılmaktadır. YouTube, Google ve iRobot makineleri dâhil olmak üzere Instagram ve Pinterest gibi sosyal medya siteleri Python üzerine kuruludur.

- **JavaScript:** Web ve mobil uygulama oluşturmak için tercih edilen bir programlama dilidir. Son kullanıcının bir uygulama indirmesini gerektirmeyen tarayıcı tabanlı uygulamaların geliştirilmesine olanak tanır. Son zamanlarda çeşitli eklentiler ile JavaScript'in işlevselliği artırılmıştır. Bu anlamda, Node.js, Vue.js, AngularJS, React gibi çeşitli çerçeveler (frameworks) oluşturulmuştur. Bu çerçeveler ile programcının uygulamayı bir cihaza duyarlı olarak kodlaması mümkündür. Bu yanıt verebilirlik, üst düzey bir makine dili kullanma konusunda oldukça popüler olmasının bir başka nedenidir.

- **C:** Genelde programlamayı yeni başlayanların tercih ettiği ve ilk geliştirilen programlama dillerinden birisidir. 1972 yılında geliştirilmiştir. Bugün pek çok üniversite programlamaya giriş dersinde C ya da C++ programlama dilini tercih etmektedir. Python, Ruby ve PHP gibi son zamanlarda popüler olan programlama dillerine temel teşkil etmektedir. Her tür cihazda çalışabildiğinden, genellikle otomobillerdeki gömülü cihazlar ve sağlık hizmetlerinde kullanılan tıbbi cihazlar gibi donanımları programlamak için kullanılır.

- **C++:** Genelde işletim sistemleri programlamak için tercih edilmektedir. Zengin bir kütüphanesi bulunmaktadır. Matematiksel simülasyonlarda ve bilgisayar oyunlarında da kullanılması sık rastlanan durumdur. Çoklu cihazlarda kullanımı ve çeşitli platformları desteklemesi üstünlükleri arasında sayılır.

- **Java:** 1991 yılında geliştirilmiş olup, bugün halen dünya çapında büyük işletmeler tarafından tercih edilmektedir. İstemci-sunucu mimarisine uygun bir programlama yapısı vardır. Bu

dilde yazılan uygulamalar Java'yı destekleyen herhangi bir platformda çalıştırılabilir. İş uygulamalarıyla birlikte Java, Android mobil işletim sisteminde de yaygın olarak kullanılmaktadır.

- **C#:** Microsoft tarafından 2002 yılında C'nin alt yapısı kullanılarak geliştirilmiştir. Windows, Android, Azure, iOS, Linux, macOS, tvOS ve Xbox platformlarına destek vermektedir. Bulut, Konsol, Masaüstü, Tarayıcı Uzantısı, Oyun, Nesnelerin İnterneti, Makine Öğrenmesi, Mobil vb. projelerde tercih edilmektedir. Microsoft .NET IDE'si ile tamamen entegredir. Büyük bir kod kitaplığı olup üzerinde çeşitli veri türleri barındırmaktadır. Söz konusu özelliği ile sistem kaynaklarını daha etkin kullanır. Son zamanlarda Mono adlı .NET Framework uzantısını kullanan mobil cihazlar ve video oyun konsolları için yazılan çeşitli C# uygulamaları göze çarpmaktadır.

- **GO:** Microsoft'tan sonra piyasaya hâkimliği ile tanınan Google firması tarafından geliştirilen bir dildir. Özellikle ağdaki cihazların haberleştirildiği dağıtık sistemlerde tercih edilmektedir. Söz dizimi basit olup, bellek güvenliği ve yönetimi konusunda etkindir. Bugün C/C++ gibi dillerle yarışır durumdadır. 2007 yılında geliştirilmiş olup açık kaynak bir programlama dilidir.

- **R:** İstatistikçilerin kullandığı bir numaralı programlama dili olma yolunda ilerlemektedir. Büyük küme veri analizleri, makine öğrenimi gibi çağımız uygulamalarında da sıklıkla tercih edilmektedir.

- **Swift:** Apple firmasının 2014 tarihinde başlattığı bir proje olup, iOS ve macOS uygulamaları geliştirmek için tercih edilmektedir. Nesneye yönelik bir programlama dilidir. Swift ile Apple'ın tüm ürünlerinde çalıştırılabilir hızlı ve etkin uygulamalar geliştirilebilmektedir. Swift'in dezavantajı, iOS 7 sonrası sürümleri desteklemesidir.

- **PHP:** 1995 yılında geliştirilmiştir. Facebook, WordPress ve Wikipedia gibi içerik odaklı web sitelerinin yazılımında kullanılan, dinamik bir programlama dilidir. Hata ayıklaması kolaydır.

Ancak son zamanlarda Python ve JavaScript gibi diller popüleritesini azaltmıştır.

Yukarıda bahsedilen programlama dillerinin yanı sıra Rust, Ruby, Perl ve MATLAB gibi daha e  çok dil programcılarının ilgisini  ekmektedir.

3. PROGRAMLAMA DİLLERİNİN HEDEFLERİ

Programlama dillerinin belirgin amacı, temelde bilgisayara talimat sağlamaktır. Bu nedenle, programlama dilleri, daha yüksek derecede kesinlik ve tamlık gerektirmeleri bakımından diğ er ifade bi imlerinden farklıdır. Bu bağlamda programcının programlamaya başlamadan  nce hedeflerini tam olarak belirlemesi gerekmektedir.

Basitlik ve Okunabilirlik

Yazılan programlar ortalama bir programcı tarafından anlaşılabilir ve okunması kolay olmalıdır. Dolayısıyla ideal bir programlama basit ve okunabilir kriterlere g re tasarlanmalıdır. Program geliştiren firmalar bu bağlamda bir standart belirlemektedirler. Değışken ve fonksiyon isimlerinin nasıl tanımlanacağını belirlenmesi, açıklayıcı yorum satırlarının eklenmesi, yazılan her mod l i in dok man hazırlanması  rnek olarak verilebilir. G n m zde  zellikle a ık kaynak kodlamanın h kim olması, yazılımın bu hedefi ile de  rt şmektedir.

G venirlik ve Destek

Yazılan programların bir test etme s reci bulunmaktadır. Test ařamasında yazılımın farklı platformlara uygulanıp uygulanamayacağı, aynı girdi verileri ile  alıřtırıldığında aynı sonuca ulařıp ulařmadığı yazılımın destek mekanizması ve g venirliğı a ısından  nemlidir.

tedir. Burada  nemli olan programlama yapmak istediğiniz g reve en uygun dili belirlemektir. Programcılar ayrıca dillerin seviyelerine g re de tercih de bulunmaktadır. İlerleyen kısımda programlama dili seviyelerine de değinilecektir.

S rd r lebilirlik

Yazılım geliřtirildikten sonra da programlama iřlemi bitmemektedir. Bu bağlamda, programların değış en kořullara uygun parametrik yapısının olması yazılımın s rd r lebilirliğı gereğidir. Geliřen biliřim d nyasında programların  zerinde  alıřtırıldığı iřletim sistemleri de zorunlu olarak değışmektedir. Dolayısıyla program yazılırken kaynak kodun o zamanki sisteme g re yeniden yapılandırılması gerekmektedir. Ayrıca yazılan programlar i in verilen eğitimler s rd r lebilirlik noktasında diğ er  nemli bir husustur.

Verimlilik

Programların sistem gereksinimleri ve  alıřma hızları verimliliğine etki eden unsurlardır. G n m zde pek  ok iřlemin     mleri yazılan programlarla saėlanmaktadır. Bu durum programlara d řen rol  artırmaktadır. Dolayısıyla y ksek verimliliğe sahip yazılımlara ihtiya  duyulmaktadır. Y ksek verimliliğın saėlanabilmesi adına da sistemin kaynakları optimal olarak kullanılmalıdır. Bellek gereksinimi, g venlik ve y netim programcının dikkat etmesi gereken bařlıca konulardır.

4. PROGRAMLAMA DİLLERİNDE SEVİYELER

Programlama dilleri geliřtirildiğı platformun diline yakınlığına g re seviyelere ayrılmaktadır. Kullanılan dil ne kadar kullanıcı dostu ise dilin seviyesi de artmaktadır. Ancak burada  nemli

olan yazılan programın amaca uygunluğudur. D ř k seviyede diller her ne kadar kullanıcı dostu olmasa da donanımın programlanmasında  nemlidirler.

Makine Dilleri

Makine dilleri, programı yazdığımız bilgisayarın işlemcisi tarafından doğrudan yürütülebilen dillerdir. Temelde bir bit deseni olarak ifade edilmektedir. Bu desenler sekizli ya da onaltılı sayı sistemleri ile formüle edilir. Oluşturulan her bit deseni işlemciye ait temel işlemlerden birisine karşılık gelmektedir. Dolayısıyla makine dilleri bir işlemci ailesine özgüdür. Bu bağlamda,

bir işlemci ailesine ait olan bir bit deseni diğer işlemci ailelerinde kullanılamaz ya da uyumluluk modu içeren ek donanıma sahip işlemciler tarafından kullanılabilecektir. Makine dilleri işlemciyi üreten firma tarafından tanımlanmaktadır. Yaygın olarak kullanılan makine kodları komut setleri Tablo 2.4’de verilmiştir. Şekil 2.5’de ise Fibonacci sayılarını hesaplamak için yazılmış örnek bir makine dili programı bulunmaktadır.



EK BİLGİ

Fibonacci sayıları 1, 1, 2, 3, 5, 8, 13 ve 21 şeklinde giden ve altın oran olarak bilinen bir dizidir. Burada her sayı kendisinden önce gelen sayı ile toplanarak büyümektedir. 1202 yılında, İtalyan matematikçi Leonardo Fibonacci tarafından bir probleme çözüm bulmak için geliştirilmiştir.

Tablo 2.4. Makine Kodları Komut Kümeleri.

Komut Kümesi Ailesi	Versiyonlar
ARM	16, 32, 64 bit
DEC	PDP, LINC, DECSYSTEM, VAX, Alpha
Intel 8008, 8080 ve 8085	Zilog Z80
x86	Intel 8086, 8088, 80186, 80286, 80386, AMD
IBM	
MIPS	
Motorola 6800	
MOS Technology 65xx	6502, 6510, 65816/65802
National Semiconductor NS320xx	
POWER	PowerPC, Power ISA
Oracle SPARC	
UNIVAC	490, 492, 494, 1230, 1101, 1103, 1105, 1100/2200 series

```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C3538B 01000000
B9010000 008D0419 83FA0376 078BD989
C14AEBF1 5BC3
```

Şekil 2.5. Fibonacci Sayısını Hesaplayan 32-Bit x86 Makine Kodunun Onaltılık Gösterimi.

Düşük Seviyeli/Çevirici Diller

Makine dillerini çok yakın olan bir dil grubudur. İşlemcinin komut kümesine yapısal olarak benzeyen semboller ile kullanılmaktadır. Bu sebeple, düşük seviyeli dillerin kullanımında derleyici/yorumlayıcıya ihtiyaç olmaksızın donanımın ihtiyacı olan makine koduna dönüştürme işlemi yapılabilir. Çevirici (Assembler) sayesinde

elde edilen kod, işlemci üzerinde doğrudan çalışmaktadır. Düşük seviyeli dillerde bellek kullanımı düşük olduğu için hızlıdır. Kullanıcı dostu olmadıklarından öğrenilmesi zordur. Ancak donanım programlamada önemli bir yere sahiptir. Şekil 2.6'da çevirici dil ile yazılmış örnek bir program bulunmaktadır.

```
_fib: movl
      $ 1 , %eax
      xorl %ebx , %ebx
.fib_loop:
      cmpl $1 , %edi
      jbe .fib_done
      movl %eax , %ecx
      addl %ebx , %eax
      movl %ecx , %ebx
      subl $1 , %edi
      jmp .fib_
.fib_done:
      ret
```

Şekil 2.6. Fibonacci Sayısını Hesaplayan AT&T Sözdizimi.

Yüksek Seviyeli Diller

Makine dili ve çevirici dil dışında kalan ve kullanıcı dostu olarak ifade edilen dillerdir. Söz konusu programlama dilleri yüksek seviyeli olarak geçmektedir. Ancak bu programlama dilleri her ne kadar kullanıcıya yakın olsa da işin işlemci tarafından yürütülmesinden dolayı yazılan programların donanımın anlayacağı dile dönüştürülmesi gerekmektedir. Bu bağlamda devreye derleyici ve yorumlayıcılar girmektedir. Şekil 2.7'de yüksek seviyeli dile örnek bir program verilmiştir.

İlk yaygın olarak kullanılan yüksek seviyeli dil Fortran'dır. Yüksek seviyeli dillerde diğer dil ailesinden farklı olarak bellek adresleri ve çağrılar ile uğraşmak yerine değişken, dizi, nesne, aritmetik ve mantıksal ifadeler, fonksiyonlar ve döngüler gibi işlemlere geçilmiştir. Bölüm 3'de sözde kod yazılarak bu işlemlere giriş yapılacaktır. Kalan diğer bölümlerde ise C++ söz dizimi kullanılarak programlama öğretilacaktır.

```
unsigned int fib(unsigned int n) {
    if (!n)
        return 0;
    else if (n <= 2)
        return 1;
    else {
        unsigned int a, c;
        for (a = c = 1; ; --n) {
            c += a;
            if (n <= 2) return c;
            a = c - a;
        }
    }
}
```

Şekil 2.7. Fibonacci Sayısını Hesaplayan C Programı.

BÖLÜM ÖZETİ

Bu bölümde programlamayı öğrenmek isteyen bireylere temel kavramları kazandırmak ve farklı programlama dillerinin sunduğu olanakları göstermeyi amaçlamaktadır. Genel olarak, geliştirilmek istenen uygulamanın türü ve kullanılacak programlama dili, ihtiyaç analizi sürecinden sonra belirlenmektedir. Bu süreç tamamlandıktan sonra programlama dilinin seviyesine ve belirlenen seviye için hangi dilin tercih edilmesi gerektiğine karar verilir. Bu karar verme sürecinde verimlilik, sürdürülebilirlik ve sağlanabilecek destek gibi faktörler önemli rol oynamaktadır.

Programlama alanında masaüstü, web, mobil ve veritabanı uygulamaları geliştirme konuları sıklıkla ele alınmaktadır. Web uygulamaları geliştirirken Python, JavaScript ve PHP; masaüstü yazılımlarında C# veya Java; veri analizi ve bilimsel hesaplamalarda ise R ve MATLAB öne çıkan diller arasında yer almaktadır. Gömülü sistem uygulamalarında C, C++ ve Rust sıklıkla tercih edilmektedir. Bulut tabanlı uygulamalarda Go ve Scala gibi diller dikkat çekerken, mobil cihazlar için uygulama geliştirme süreçlerinde Swift ve Kotlin gibi diller yaygın olarak kullanılmaktadır.

Programlama dili seçiminde, uygulama türü ve ihtiyaç duyulan performans kriterleri belirleyicidir. Farklı dillerin sunduğu özellikler ve kullanım alanları göz önünde bulundurularak, hedeflere en uygun dil veya diller tercih edilmelidir.

GÖZDEN GEÇİRELİM

1. Aşağıda verilenlerden hangisi ikilik sayı sistemindeki elemanların farklı dizilişleri ile oluşturulmaktadır?
 - a. Komut
 - b. İşlemci
 - c. Bellek
 - d. G/Ç Modülü
 - e. Scrum
2. Derleyici ve yorumlayıcı için verilen bilgilerden hangisi yanlıştır?
 - a. Yorumlayıcı satır satır işler.
 - b. Derleyici tüm kaynak kodu tek seferde işler.
 - c. Derleyici, belleği yorumlayıcıdan daha verimli kullanır.
 - d. Yorumlayıcıda hatalı satırdan sonrası işletilmez.
 - e. Derleyici hatalı satırlar olsa da diğer satırları çalıştırır.
3. Microsoft Visual Studio'da yerleşik olarak hangi programlama dili bulunmaz?
 - a. C
 - b. Visual Basic
 - c. F#
 - d. C#
 - e. Linux
4. Aşağıdakilerden hangisi programlama dili için kullanılan bir düzenleyici (editör) değildir?
 - a. Cygwin
 - b. Linux
 - c. UltraEdit
 - d. Atom
 - e. Notepad++
5. Akümülatöre değer yükleyen Assembly kodu aşağıdakilerden hangisidir?
 - a. Load
 - b. Store
 - c. Add
 - d. Halt
 - e. Mov
6. Bir programlama diliyle yazılan programların işlemcinin anlayabileceği makine diline dönüştürülerek işlenip yorumlanmasından önce programcının okuyabildiği ve üzerinde çalışabildiği kısımlar aşağıdakilerden hangisidir?
 - a. Yorumlayıcı
 - b. Derleyici
 - c. Amaç Kod
 - d. Kaynak Kod
 - e. Birleştirme

7. İstatistik için kullanılan programlama dili hangisidir?

- a. Go
- b. R
- c. Scala
- d. Kotlin
- e. C++

9. İlk yaygın olarak kullanılan yüksek seviyeli dil hangisidir?

- a. JavaScript
- b. PHP
- c. Java
- d. C#
- e. Fortran

8. Aşağıda verilenlerden hangisi makine kodları için kullanılan komut kümesi ailelerinden değildir?

- a. Motorola 6800
- b. ARM
- c. Java
- d. IBM
- e. Intel 8008

10. Aşağıda verilen programlama dili hedeflerinden hangisi sürdürülebilirlik ilkesine uygundur?

- a. Programların değişen koşullara uygun parametrik yapısının olması
- b. Programın farklı platformlara uygun olması
- c. Programın basit ve anlaşılır olması
- d. Programın sistem gereksinimi ve çalışma hızı
- e. Hiçbiri

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

Kaynakça

William, S. (2016). Computer organization and architecture: designing for performance, Pearson.

Demir, S., Eryiğit, R., Ar, Y., Tuğrul, B., Gök, O., Kul, S., . . .

Yağanoğlu, M. (2019). Programlama Temelleri. Erzurum: Atatürk Üniversitesi Açıköğretim Fakültesi.

Kaleli, C., Bilge, A., Ergin, S., & Şora Günel, E. (2016). Bilgisayar ve Programlamaya Giriş. Eskişehir: Anadolu Üniversitesi.

En.wikipedia.org. 2022. Low-level programming language - Wikipedia. [online] <https://en.wikipedia.org/wiki/Low-level_programming_language> [Erişim Tarihi 30 Ocak 2022].

Faydalı İnternet Kaynakları

Getir-Yürüt-Kodunu Çöz Çevrimi (<https://www.youtube.com/watch?v=04UGopESS6A>)

Assembly Dili (<https://tr.wikipedia.org/wiki/Assembly>)

BÖLÜM 3

ALGORİTMA VE AKIŞ DİYAGRAMLARI

ANAHTAR KAVRAMLAR

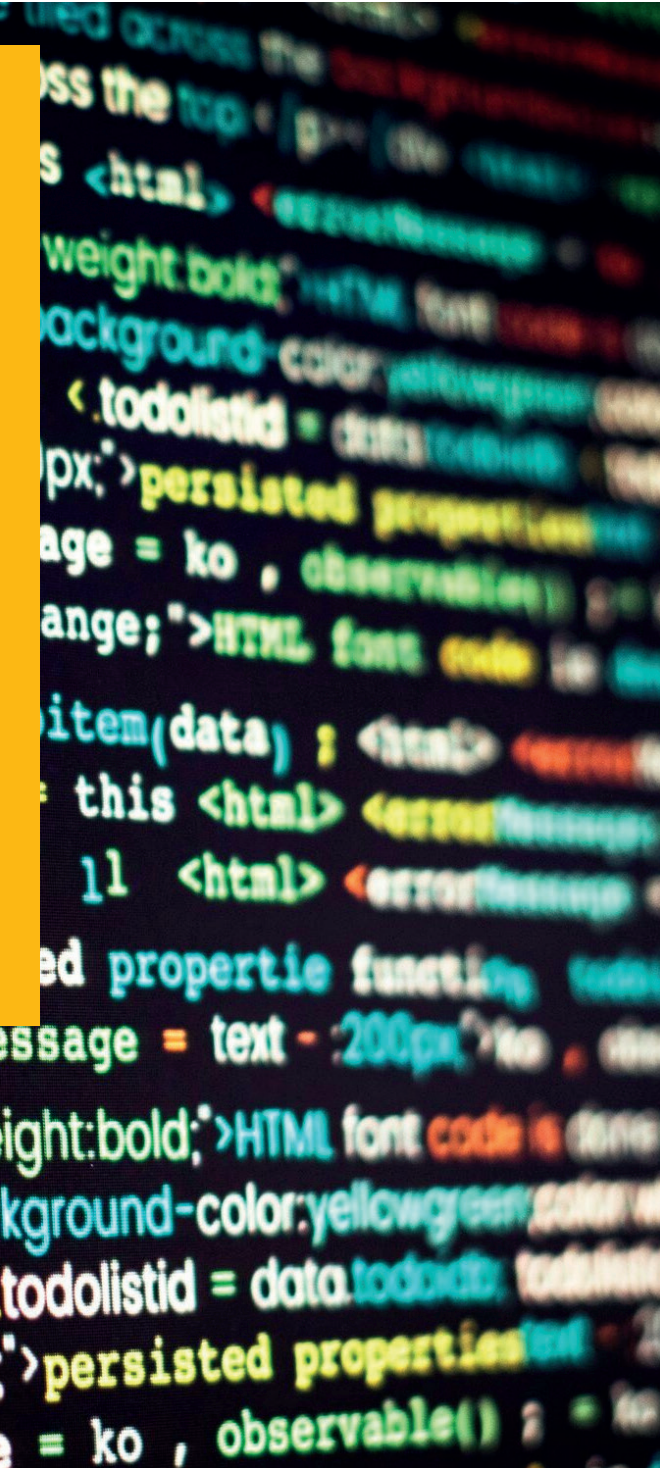
Algoritma, Sözde Kod, Akış Diyagramı, Yazılım Geliştirme Süreci, Problem Çözme Yapıları

ÖĞRENME HEDEFLERİ

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

GİRİŞ

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



1. TEMEL KAVRAMLAR

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

- Sistem Yazılımları (Microsoft Windows, Google ChromeOS, Linux vb.)
- Programlama Yazılımları (C++, C#, Python, JavaScript, R, Swift vb.)

- Uygulama Yazılımları (Open Office (Kelime İşlemci/Elektronik Tablo/Sunu ofis uygulamaları), Adobe Creative Cloud, Autodesk CAD/CAM/CAE ve PCB (Çizim/Tasarım Uygulamaları), Medya (Ses/Video) Yürütücüler vb.)

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



EK BİLGİ

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

Matematiksel İşlemler

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



ARAŞTIRILIM

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

Tablo 3.1. Matematiksel İfadelerin Programlamadaki Gösterimleri.

İşlem	Matematiksel İfade	Kod Gösterimi
Toplama	$A+B$	$A+B$
Çıkarma	$A-B$	$A-B$
Çarpma	$A \times B$	$A*B$
Bölme	$A:B$	A/B
Üs Alma	A^B	A^B veya $A**B$ veya $\text{pow}(A,B)$
Modüler İşlem	$A \bmod B$	$A\%B$

Kaynak kodda yazacağımız matematiksel ifadelerde iç içe yazım türünün derleyici tarafından çözümlenmesi zordur. Yapılacak hesaplamalarda matematiksel işlemlerin operatör önceliklerine

göre ayrıştırılması, ardından bunların sıralanması ve ilgili dizilime göre işlem yapılması gerekmektedir. Derleyiciler işlemleri kolaylaştırmak için önek ya da sonek notasyonunu kullanır.

Tablo 3.2. Matematiksel İşlemlerde Öncelik.

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

Bölme ve mod alma işlemlerinde mantık hatasına düşmemek programcının dikkat etmesi gereken bir unsurdur. Çünkü kullanılan sistemler tam sayılı işlemlerde yuvarlama yapmaktadır.

Mod alma işlemlerinde ise sayının pozitif veya negatif olmasına göre sonuç değişmektedir. Aşağıda duruma uygun örnekler verilmiştir.



DİKKAT EDELİM

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

Mod alma örnekleri:

$$16 \% 3 = 1$$

$$-16 \% 3 = -1$$

$$16 \% -3 = 1$$

$$-16 \% -3 = -1$$

Bölme örnekleri:

$$9/4 = 2$$

$$-9/4 = -3$$

Bazı hesaplamalar karmaşık yapıya sahip olabilir. Bu tarz ifadelerin kaynak koda yazılabilir

hale dönüştürülmesi gerekir. Aşağıda duruma uygun bir örnek gösterilmektedir:

$$a \times (-b \pm ((b^2 - (4 \cdot a \cdot c))^{(1/2)}) / 2 \cdot a$$

Karşılaştırma (Karar) İşlemleri

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

Tablo 3.3. Karşılaştırma (Karar) İşlemleri.

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

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

Mantıksal ve Bitsel İşlemler

Mantıksal ve Bitsel (Bitwise) işlemler problem içerisinde koşul gerektiren durumlarda karşımıza çıkmaktadır. Söz konusu durum bir

veya birden fazla olabilir. Bir koşulun olmamasını istediğimizde değilse anlamında “Mantıksal DEĞİL” işlemi; iki koşulun birden sağlanmasını istediğimizde “Mantıksal VE” işlemi; iki koşuldandan en az birisinin doğru olmasını beklediğimizde “Mantıksal VEYA” işlemi; iki koşuldandan sadece birinin doğru olmasını beklediğimiz durumlarda da “Bitsel XOR” işlemi yaparız. Tablo 3.4 “Mantıksal DEĞİL”, Tablo 3.5 “Mantıksal VE”, Tablo 3.6 “Mantıksal VEYA” ve Tablo 3.7 “Bitsel XOR” doğruluk tablosunu göstermektedir. Sırası ile bu işlemler “!”, “&&”, “||”, “^” sembolleri ile kullanılmaktadır.

Tablo 3.4. Mantıksal DEĞİL Doğruluk Tablosu.

A	!A
0	1
1	0

Tablo 3.5. Mantıksal VE Doğruluk Tablosu.

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

Tablo 3.5. Mantıksal VEYA Doğruluk Tablosu.

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

Tablo 3.5. Bitisel XOR Doğruluk Tablosu.

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

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

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

Bitsel VE işlemi,

$$10101010 \& 01010101 = 00000000$$

Bitsel VEYA işlemi,

$$10101010 | 01010101 = 11111111$$

Bitsel XOR işlemi,

$$10101010 \wedge 01010101 = 11111111$$

Bitsel DEĞİL işlemi,

$$10101011 = 01010100$$

Mantıksal VE işlemi,

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

2. ALGORİTMA

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

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

haline getirilmektedir. Temel kavramlar ünitesinde bu konuya detaylı olarak yer verilmiştir.

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

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

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

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

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



EK BİLGİ

Suyun buhar basıncı atmosfer basıncına eşit olduğunda su kaynar. Su dağın tepesinde ve deniz seviyesinde farklı kaynama noktasına sahiptir. Deniz seviyesinde 100°C 'de kaynar.

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

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

Problemin belirlenmesi çözüme giden yoldaki en önemli adımlardan birisidir. Problem belirlendikten sonra deneysel yöntemler ya da yukarıda anlatıldığı gibi algoritmik metot tercih edilir. Bu bağlamda sıra olarak problemin tanımlanması, probleme etken olan girdi ve çıktıların belirlenmesi, çözüm yollarının aranması, çözümün test edilmesi ve test sonuçlarına göre

iyileştirilmesi süreçleri başlar. Bu süreç istenilen şekilde tamamlandıktan sonra algoritma tasarımına geçilip, hazırlanan algoritmanın test ve geri dönüşler ile iyileştirilmesi gerekmektedir.



GÜNLÜK HAYATLA İLİŞKİLENDİRELİM

Kendimize gündelik yaşamdan algoritmasını yazacağımız bir problem belirleyelim. Ardından adımları keskin, sıralı ve sonlu olarak oluşturalım.

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

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

ise kullanılacak donanım, işletim sistemi, ihtiyaç duyulan bellek ya da belleğin sistemi yormayacak şekilde optimize edilmesi algoritmaya etki edecek bileşenler olarak karşımıza çıkar. Tüm bu durumlar analiz edildikten sonra algoritma yazma aşamasına geçilir. Burada da algoritmanın hangi programlama dili ile gerçekleştirileceği, algoritmanın sınanıp sınanmadığı, hataların

giderilip giderilmediği ve performans durumuna bakmak gerekir. Algoritmanın yazılıma dönüşmeden önceki geçirdiği süreç bu şekildedir. Ancak algoritmanın yazılıma dönüştürülmesinde her program geliştiren ekip farklı yöntemler izlemektedir. Bu yöntemler ve geliştirme süreci yazılım geliştirme süreci kısmında detaylı olarak verilmektedir.

3. YAZILIM GELİŞTİRME SÜRECİ

Bir problemin çözülmesi için geliştirilen algoritmalar ve akış diyagramları bir yazılıma dönüştürüldüğünde de belli bir süreci takip etmektedir. Bu sürece yazılım geliştirme yaşam döngüsü

denmektedir. Döngü; planlama, analiz, tasarım, gerçekleştirme ve son olarak teslim ve bakım olarak gerçekleşir (Şekil 3.1).



Şekil 3.1. Yazılım Geliştirme Yaşam Döngüsü.

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

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

Tasarım; mantıksal ve fiziksel olarak gerçekleşir. Programdan beklenen istemler, mantıksal kısımda sistemin yapısını oluşturmada ve fizik-

sel kısımda ise yazılımın bileşenlerini detaylı bir şekilde göstermede kullanılır. Ayrıca hangi programlama dilinin kullanıldığı, teknoloji, mimari tasarım bu süreçte ele alınmaktadır.

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

nen sonucu üretip üretmediği ile ilgili testler de burada gerçekleştirilir.

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

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

Yazılım süreç modelleri,

- Kodla ve Düzelt Model Yaşam Döngü Modelleri
- Çağlayan Yaşam Döngü Modeli
- V Süreç Modeli
- Evrimsel Geliştirme Süreç Modeli

- Prototipleme Yaşam Döngü Modeli
- Spiral Model
- Formal Sistem Geliştirme Modeli
- Yeniden Kullanıma Yönelik Geliştirme Modeli
- Artırımsal Geliştirme Süreç Modeli
- Çevik Yazılım Geliştirme Metodolojileri (En popüler yaklaşımdır)
 - o Aşırı Programlama (Extreme Programming - XP)
 - o Birleşik Rasyonel İşlem (Rational Unified Process)
 - o Özellik Odaklı Geliştirme (Feature-Driven Development - FDD)
 - o Test Odaklı Geliştirme (Test-Driven Development - TDD)
 - o LEAN Geliştirme (LEAN Development)
 - o Dinamik Sistem Geliştirme Metodolojisi (Dynamic System Development Methodology - DSDM)
 - o Microsoft Çözüm Çerçevesi (Microsoft Solution Framework - MSF)
 - o SCRUM

3.1. Algoritmanın Yazılım Geliştirme Sürecindeki Yeri

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

betik (script) programlar gibi durumlarda algoritma yapmadan da programlamaya geçilebilmektedir.

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

fazla çözüm yolunu kodlamadan sözlü olarak geliştirebileceğimiz en rahat ortamlardır. Kod-Şekil 3.2’de örnek bir algoritma yazımı verilmiştir:

lama yapılarak başlanan projelerde birden fazla çözüm belirlemenin zor olacağı aşîkârdır.

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

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

3.2. Algoritmaların Sınıflandırılması

Algoritmalar problemin türlerine göre basit (lineer), mantıksal ve döngüsel olarak ele alınmaktadır. Basit algoritmalar, bölümün temel kavramlar kısmında belirtilen mantıksal ve bitisel işlemlerin yapılmadığı ve adımların düz bir hatta sahip ol-

duğu yani dallanmaların yapılmadığı türlerdir. Mantıksal işlemler yer almadığı için de herhangi bir koşul/karar yapısı bulunmamaktadır. Şekil 3.3’de örnek bir basit algoritma yazılmıştır:

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

Şekil 3.3. Dairenin Alanını Hesaplayan Basit Algoritma.

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

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

Şekil 3.4. Ders Çalışma için Mantıksal Algoritma.

Döngüsel algoritmalar ise hem karar/koşul yapısını barındıran hem de bir adımın birden

fazla tekrar ettiği algoritmalardır. Örnek bir algoritma Şekil 3.5’de gösterilmektedir.

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

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

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

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



ARAŞTIRALIM

Şekil 3.2’de verilen algoritma hangi algoritma sınıflandırmasına girmektedir?

3.3. Algoritma Geliştirilmesi

Algoritma geliştirilirken tespit edilen problemin bulduğumuz çözüm adımları metin halinde cümlelere dönüştürülmelidir. Her bir cümlede adım sırası önemlidir. Ardından akış diyagramları (flow-chart) ile yazılan adımlar standart şekillere çevrilir. Son olarak, belirlenen bu adımlar kod benzeri metinlerle veya kısaltmalarla sözde

koda (pseudo-code) dönüştürülür. Tüm bu aşamalar algoritma geliştirilmesi için gereklidir. Tablo 3.6’da örnek bir algoritma geliştirme süreci gösterilmektedir. Burada verilen sözde kod ve akış diyagramı işlemi için ilerleyen bölümde daha detaylı bilgi verilecektir.

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

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

Algoritma tasarımında nümerik ve alfanümerik olmak üzere çeşitli veriler kullanılmaktadır. Söz konusu veriler için “değişken” adı verilen ve bellek üzerinde bir adrese etiketlenen alanlar kullanılmaktadır. Tablo 3.6’da verilen örnekte Sayı1, Sayı2 ve Ortalama bir değişkendir ve içerisinde reel sayılar tutulmaktadır. Değişkenlerin değeri program içerisinde değişebilir. Pi sayısı, ışık hızı ve bunun gibi değeri değişmeyen alanlar için ise “sabit” tanımı yapılmaktadır. Değişken ve sabitlerin yanı sıra alt yordam da tanımlayıcı olarak algoritmada kullanılan diğer bir kavramdır. Alt yordam, algoritma içerisinde tekrar eden işleri topladığımız kısımdır. Örneğin bir görselin birden fazla görsel ile karşılaştırıldığı bir durum düşünelim. Karşılaştırma işlemi her bir görsel için

tekrar edeceği için her seferinde karşılaştırma işlemi için kod yazmak yerine bir tane karşılaştırma alt yordamı yazılır. Karşılaştırılan değerler ise alt yordama parametre olarak verilir. Böylece karşılaştırma için gerekli algoritma tekrar tekrar yazılmamaktadır.

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

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

Şekil 3.6. Algoritma ile Sayaç Kullanımı.

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

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

Algoritmalarda bir de döngü adını verdiğimiz bir kullanım bulunmaktadır. Algoritmanın belli

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

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

Şekil 3.7. Algoritma ile Döngü Kullanımı.

3.4. Algoritma Yazma Kuralları

Algoritma yazımında satırlar 1'den başlayarak numaralandırılır. Adımlar sonlu sayıda ve özellikle sıralı olmalıdır. Tekrar eden işlemlerden kaçınılmalı ya da bu kısım bir alt yordam algoritması olarak tasarlanmalıdır. Gereksiz ve etkisiz kullanım bellekte boş yere işgal edeceğinden ilgili adımlar tespit edilip algoritmadan çıkarıl-

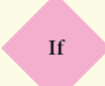
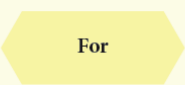
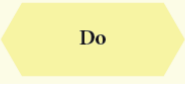
malıdır. Algoritma içerisindeki dallanmalarda adım numarası belirtilmelidir. Adımların herkes tarafından anlaşılabilir bir keskinlikte olmasına dikkat edilmelidir. Her algoritmada bir girdi ve çıktı bulunmalı ve algoritma Dur/Son ile bitirilmelidir.

4. AKIŞ DİYAGRAMI

Akış diyagramı, algorithmada yazılan adımların belirli bir standarda göre sembolize edilmesidir. Akış diyagramları için IBM ve Spesifikasyon ve Açıklama Dili (Specification and Description

Language – SDL) gibi pek çok farklı standart bulunmaktadır. Akış diyagramı sembolleri ve açıklamaları Tablo 3.7’de verilmektedir.

Tablo 3.7. Akış Diyagramı Sembol ve Açıklamaları.

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

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

Flowgorithm ile akış diyagramlarını kullanarak, tipik bir programlama dilinin tüm nüansları yerine programlama kavramlarına odaklanabilirsiniz. Programlarınız doğrudan Flowgorithm’de de çalıştırılabilmektedir. Ayrıca Flowgo-

rithm ile hazırlanan akış diyagramlarınızı tek tıklama ile sözde koda dönüştürebilirsiniz. Programlama mantığını akış diyagramları ile anladıktan sonra, ana dillerden birini öğrenmeniz çok kolay olmaktadır. Flowgorithm ile akış çizelgenizi etkileşimli olarak 18’den fazla programlama diline dönüştürme imkânı da bulunmaktadır. Bunlardan bir kısmı C#, C++, Java, JavaScript, Lua, Perl, Python, Ruby, Swift, Visual Basic .NET ve VBA’dır. İlâveten, programın pek çok dil desteği olup, Türkçe bunlardan birisidir. Dersteki uygulamaları ve sizlerin geliştirmek istediğiniz

diğer algoritmalar için uygulamayı <http://www.flowgorithm.org/download/index.html> adresinden indirebilirsiniz. Ancak kullandığınız Win-

dows işletim sisteminin 32/64 bit sürümleri için uygun olanını kullanmanız gerekmektedir.



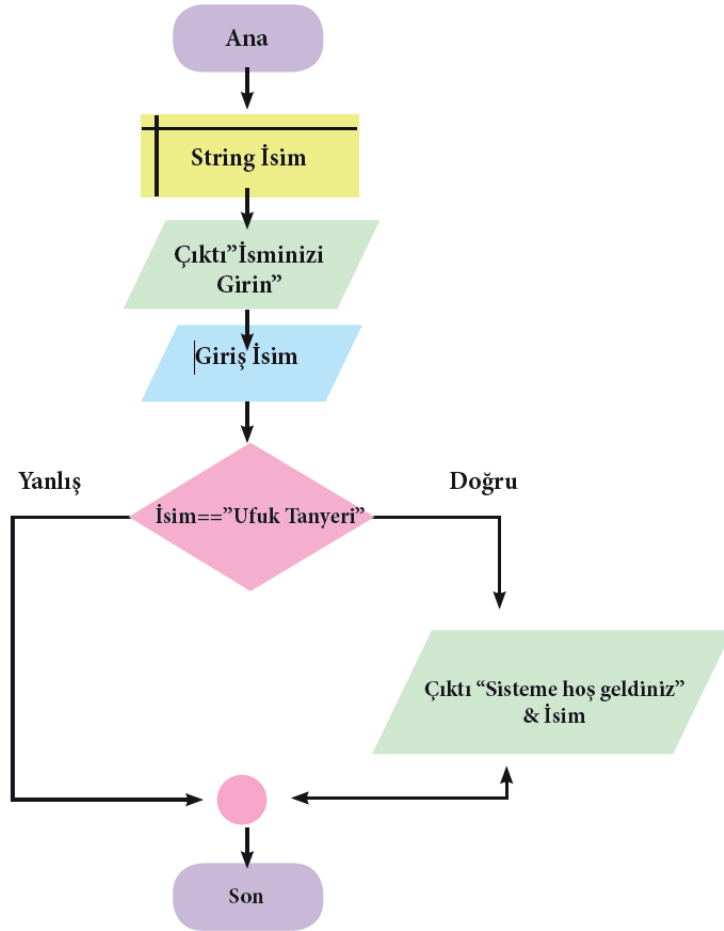
EK BİLGİ

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

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

Algoritmada adımların akışını değiştiren ifadeler karar kontrol ifadeleridir. Şekil 3.8’de bu

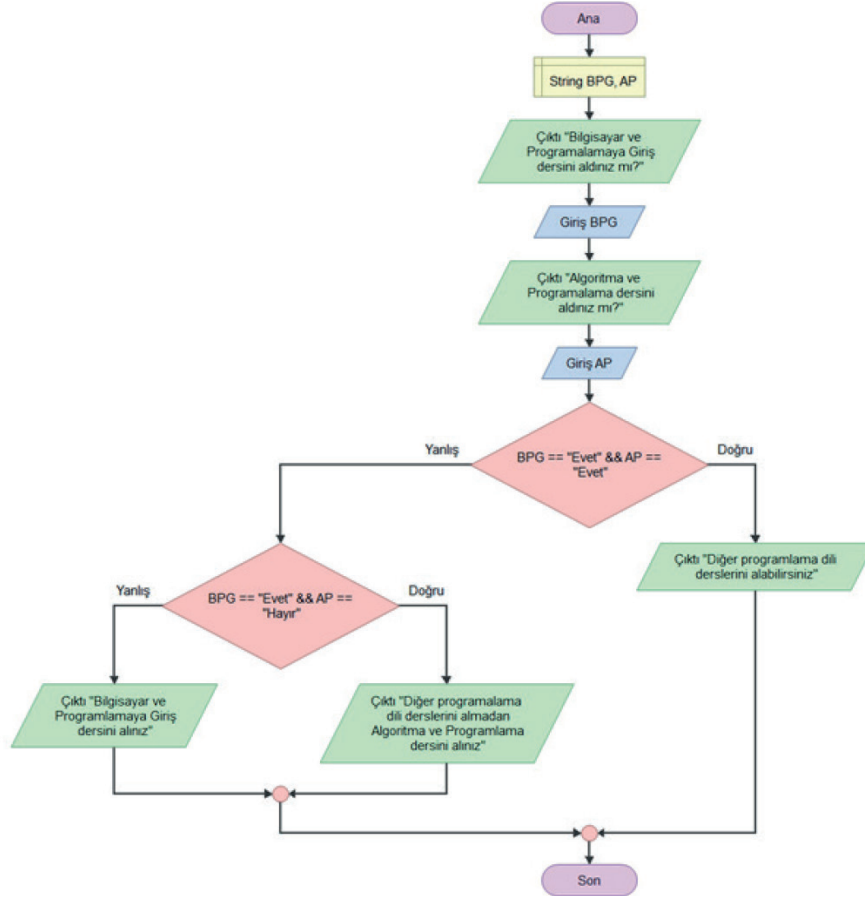
duruma örnek bir akış diyagramı verilmektedir.



Şekil 3.8. Eğer İfadesi Akış Diyagramı.

Şekil 3.8’de klavyeden okutulan değer “İsim” adlı bir değişkene aktarılmaktadır. Eğer değer “Ufuk Tanyeri” ise “Sisteme hoş geldiniz” yazı-

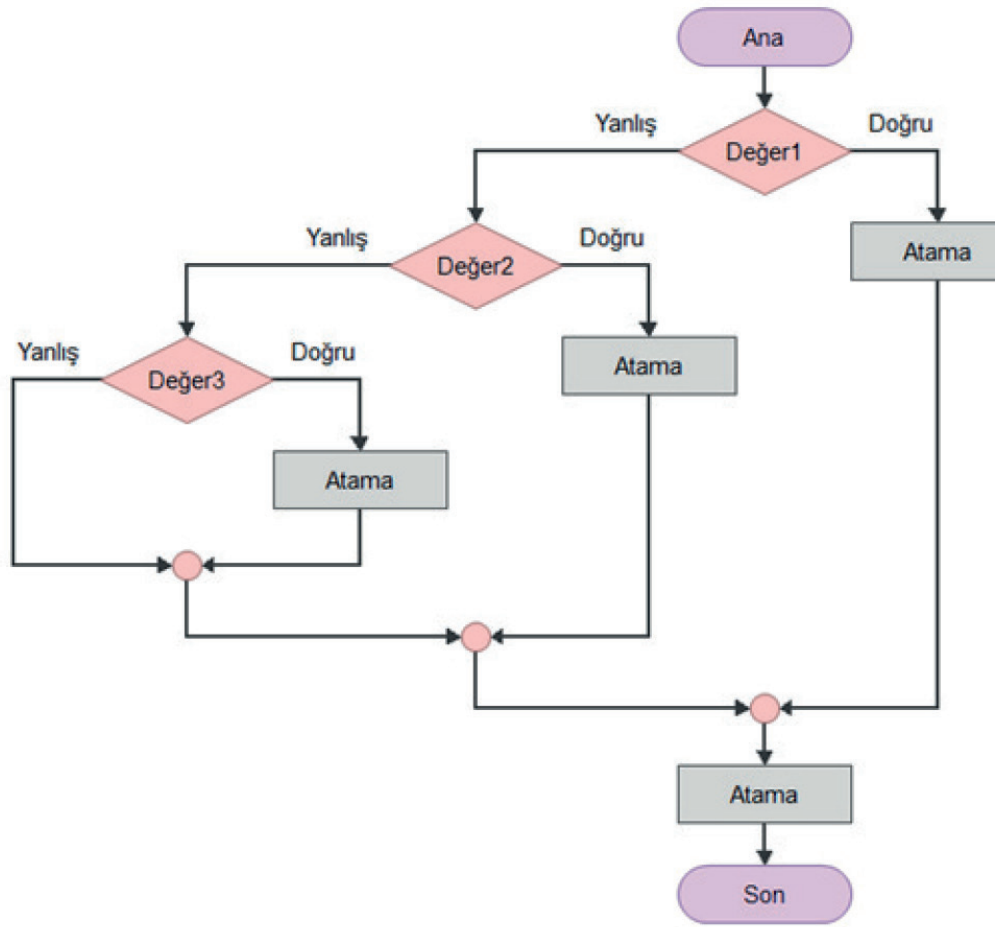
sı ekrana yazılıp program sonlanacaktır. Değilse sistem sonlanacaktır. Daha karmaşık bir eğer ifadesi Şekil 3.9’da verilmektedir.



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

Şekil 3.9’daki yapılar daha karmaşık eğer yapısına örnektir. Burada iç içe eğer ifadesi kullanılmaktadır. Programda Bilgisayar ve Programlamaya Giriş dersi BPG, Algoritma ve Programlama dersi AP isimli karakter katırı (string)

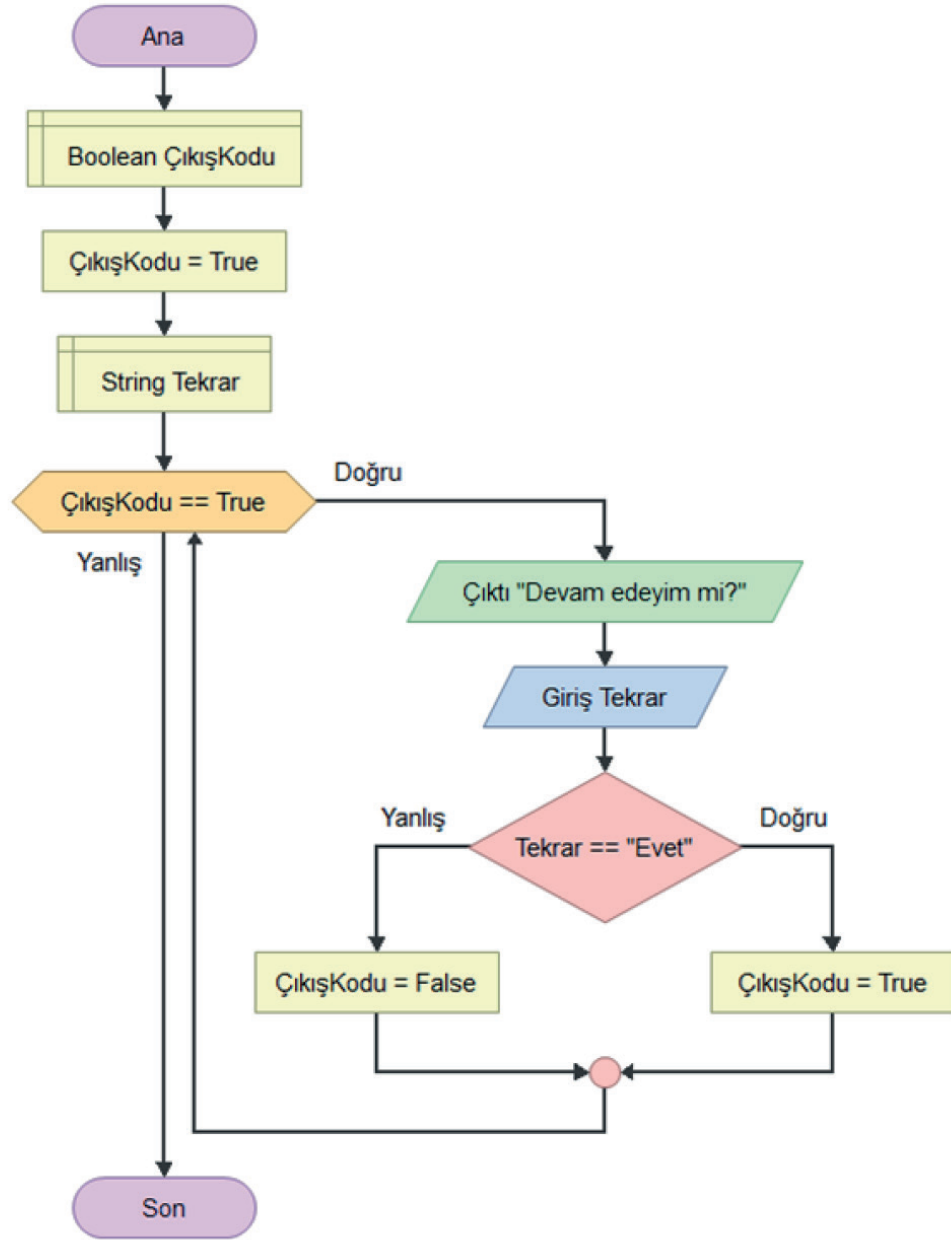
değişkenler olarak tanımlanmıştır. Algoritma kişinin bahsi geçen dersleri alma durumuna verdiği “Evet” veya “Hayır” cevabına göre öneride bulunmaktadır. Çok-yönlü karar yapıları için de Şekil 3.10 örnek olarak verilmektedir.



Şekil 3.10. Çok-yönlü Karar İfadesi Akış Diyagramı.

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

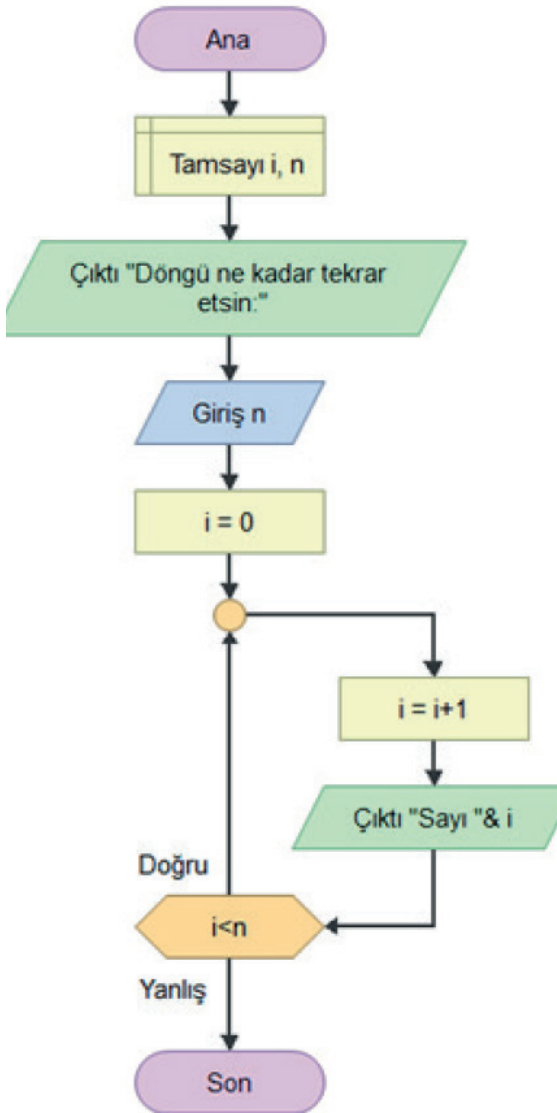
- While döngüsü
- Do/Do-While döngüsü
- For döngüsü



Şekil 3.11. While Döngüsü Akış Diyagramı.

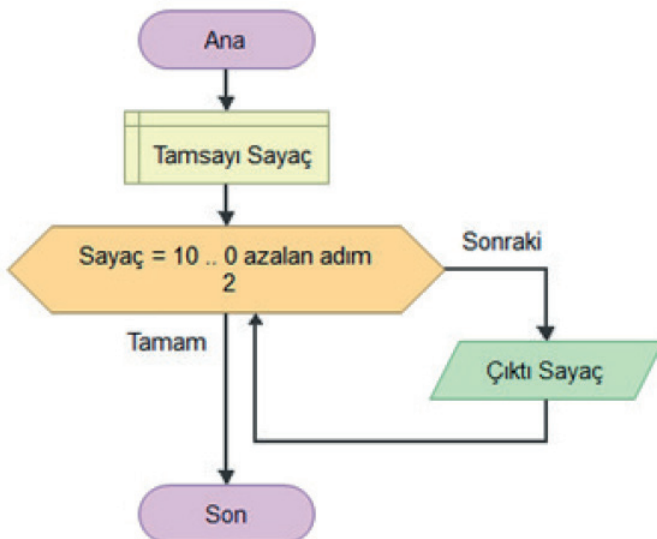
Şekil 3.11’de verilen ÇıkışKodu, Doğru/Yanlış (True/False) değerini tutan mantıksal bir değişkendir ve başlangıçta değeri True’dur. ÇıkışKodu değeri True olduğu sürece program “Devam edeyim mi?” sorusunu soracaktır. Cevap evet

olduğu sürece de döngü kendini tekrar edecektir. Başlangıçta ÇıkışKodu değişkeni False olursa döngüye hiç girilmeyecektir. Şekil 3.12, Do While döngüsü için bir örnektir.



Şekil 3.12. Do While Döngüsü Akış Diyagramı.

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



Şekil 3.13. For Döngüsü Akış Diyagramı.

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

5. SÖZDE KOD

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

Şekil 3.8’de verilen Basit bir karar yapısı örneği olan “Eğer İfadesi” nin sözde kod hali:

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

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

Şekil 3.9’da verilen “Eğer- Değilse Eğer- Değilse İfadesi” nin sözde kod hali:

```
Declare String bPG, aP

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

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

If Değer1 Then

 // Set Ad = Açıklama

Else

 If Değer2 Then

 // Set Ad = Açıklama

Else

 If Değer3 Then

 // Set Ad = Açıklama

 End If

End If

End If

 // Set Ad = Açıklama

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

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

Declare Boolean çıkışKodu

Set çıkışKodu = True

Declare String tekrar

While çıkışKodu == True

 Display “Devam edeyim mi?”

 Input tekrar

 If tekrar == “Evet” Then

 Set çıkışKodu = True

 Else

 Set çıkışKodu = False

 End If

End While

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

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

Declare Integer i, n

Display "Döngü ne kadar tekrar etsin:"

Input n

Set i = 0

Do

Set i = i + 1

Display "Sayı ", i

While i < n

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

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

Declare Integer sayaç

For sayaç = 10 To 0 Step -2

Display sayaç

End For

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

Gaddis sözde kodda kullanılan matematiksel operatörler:

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

İlişkisel operatörler:

- $>$ (Büyüktür)
- $<$ (Küçüktür)
- $>=$ (Büyük eşit)
- $<=$ (Küçük eşit)
- $=$ (Eşit)
- $\neq$ (Eşit değil)

Mantıksal operatörler:

- AND (Ve)
- OR (Veya)
- NOT (Değil)

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

BÖLÜM ÖZETİ

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

GÖZDEN GEÇİRELİM

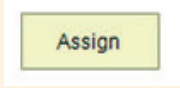
1. Aşağıda verilen yazılım türlerinden hangisi bir sistem yazılımıdır?
 - a. ChromeOS
 - b. Python
 - c. Open Office
 - d. JavaScript
 - e. Adobe Creative Cloud
2. A'nın B'ye göre modunu al matematiksel işleminin kod gösterimi aşağıdakilerden hangisidir?
 - A) A*B
 - B) A%B
 - C) A+B
 - D) A/B
 - E) A**B
3. Baytlar ile işlem yapan ve iki koşulun birden sağlanması istediğimiz mantıksal işlem aşağıdakilerden hangisidir?
 - A) Mantıksal DEĞİL
 - B) Bitset XOR
 - C) Mantıksal VEYA
 - D) Mantıksal VE
 - E) Bitset VE
4. Bir problemin çözümünü sonlu ve sözel adımlarla anlatma işi nedir?
 - a. Program
 - B) Yazılım
 - C) Sözde Kod
 - D) Akış Diyagramı
 - E) Algoritma
5. Aşağıdakilerden hangisi akış diyagramı için kullanılan araçlardan biri değildir?
 - a. Algobuild
 - B) Gaddis
 - C) PSeInt (İspanyolca)
 - D) Raptor
 - E) LARP

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

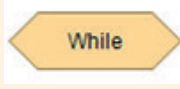
a.



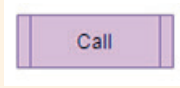
b.



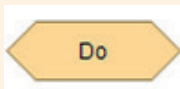
c.



d.



e.



7. Display “Bilgisayar ve Programlamaya Giriş dersini aldınız mı?” ile verilen sözde kod komut satırı ne işe yaramaktadır?

- Ekrana tırnak içerisinde verilen bilgiyi yazar
- Klavyeden veri girişi yapar
- Tırnak içerisinde verilen koşulu arar
- Yorum satırıdır
- Bir değişkene tırnak içerisindeki bilgiyi aktarır

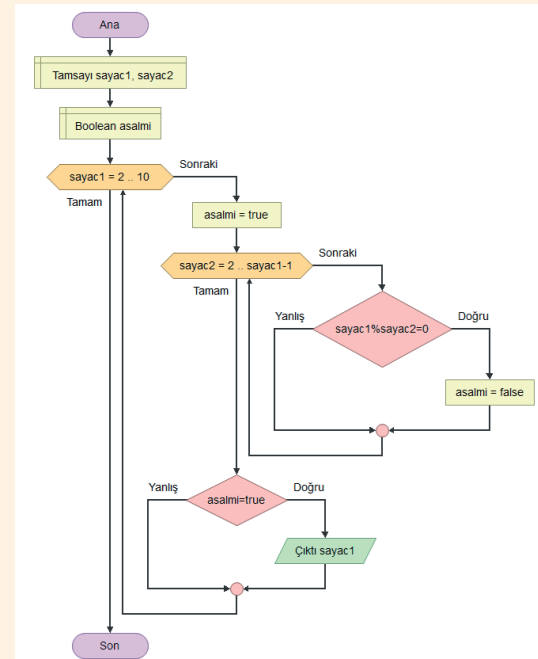
8. Gaddis standardına göre mod alma işlemi nasıl yapılır?

- /
- ^
- **
- %
- MOD

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

- Scratch
- Alice
- Gaddis
- Microsoft Small Basic
- Hiçbiri

10. Aşağıdaki akış diyagramının ekran çıktısı aşağıdakilerden hangisidir?



- 2 4 6
- 1 3 5 7
- 3 5 7 9
- 2 3 5 7
- 2 4 6 8

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

Kaynakça

- Algoritma: Uygulamalı Algoritma Kılavuzu, 5. Baskı, Kadir Çamoğlu, KODLAB, 2011
- Algoritma Geliştirme ve Programlamaya Giriş, 13. Baskı, Fahri Vatansever, Seçkin Yayıncılık, 2017
- Algoritma ve Programlamaya Giriş, 6. Baskı, Ebubekir Yaşar, Ekin Basım Yayın, 2016
- Gaddis, T. (2016). Starting Out with Programming Logic and Design, 4th Edition. Pearson. ISBN 9780133998160

Faydalı İnternet Kaynakları

- Kaçar, B. (2021, Aralık 8). YAZILIM GELİŞTİRME VE SÜREÇ MODELLERİ - Berfin Kaçar. Medium.
<https://medium.com/@brfn.kcr26/yazilim-geli%C3%9Cme-ve-s%C3%A7-modelleri%C3%A7-2131ea5f09b2>
- Self learning algorithm example. (n.d.). 11 Şubat 2022'de erişildi. <https://www.jolinaforuno.top/products.aspx?cname=self%2Blearning%2Balgorithm%2Bexample&cid=7>
- Flowgorithm - Flowchart Programming Language. (2014, Ağustos 6). Flowgorithm. 11 Şubat 2022'de erişildi. <http://www.flowgorithm.org/>

BÖLÜM 4

C++ YAZILIM GELİŞTİRME ORTAMLARI

ANAHTAR KAVRAMLAR

Derleyici (Compiler), Hata Denetimi,
Çalıştırılabilir (Executable) Dosyalar



ÖĞRENME HEDEFLERİ

-  Öğrenci C++ programlarını nasıl ve nerede geliştireceğini bilir
-  Öğrenci yazmış olduğu programları nasıl derleyeceğini ve çalıştırmaya hazır hale getireceğini öğrenir
-  Öğrenci yazmış olduğu programın hata denetimini nasıl yapacağını bilir
-  Öğrenci yazmış olduğu programı çalıştırmasını sonuçları yorumlayı bilir

GİRİŞ

BC++ genel amaçlı bir programlama dilidir. C++, C programlama dilinin bir çok özelliğini içermekle birlikte, C programla dilinden farklı olarak kullanıcıya nesne yönelimli (object-oriented) programlama özellikleri sunar.

Bu bölümde C++ programlarının nasıl ve nerede yazılacağı anlatılmaktadır. Yazılan programın, kullanılan derleyici üzerinde derlenip, hata denetiminin yapıldıktan sonra nasıl çalıştırılacağı gösterilmektedir.



1. C++ YAZILIM GELİŞTİRME ORTAMLARI

C++ genel amaçlı bir programlama dilidir ve Windows, Linux, Unix, Mac vb. birçok platformda çalışır. C++ genel amaçlı programlama özelliklerinin yanısıra nesne yönelimli (object-oriented) programlama özelliklerine de sahiptir.

C++ ile programlamaya başlamadan önce, yazdığımız programları derlemek (compile) ve çalıştırmak (run) için, kullanmış olduğumuz bilgisayarın ve işletimin sisteminin özelliklerine uygun bir yazılım geliştirme platformunu (IDE)

veya derleyicinin bilgisayarlarımıza kurulması gerekmektedir. Eğer yerel (bireysel) bilgisayarınıza bir C++ yazılım geliştirme platformu kurmak istemiyorsanız çevrimiçi olarak yazmış olduğunuz kodları derleyen ve çalıştırmanıza izin veren ortamlarda mevcuttur.

Bu bölümde bir C++ programını geliştirmek, derlemek ve çalıştırmak için ihtiyaç duyulan ortamlar tanıtılacak ve ilk C++ programı yazılarak izlenmesi gereken adımlar anlatılacaktır.

2. C++ YAZILIM GELİŞTİRME ORTAMININ KİŞİSEL BİLGİSAYARINIZA KURULMASI

Bu bölümde öncelikle Linux ortamında yazılım geliştirme ortamının düzenlenmesi gösterilecek ardından Windows üzerinde kullanılacak olan Cygwin derleyicisinin kurulumu gösterilecektir.

Linux işletim sistemi için C++ programlarını geliştirebilecekleri ortamın oluşturulması aşağıdaki gibi yapılabilir.

2.1. Linux Ortamında GCC Kurulumu

Linux işletim sistemi kullanıcıları aşağıdaki komutları kullanarak GCC (GNU Compiler Collection) derleyicisini bilgisayarlarına kurabilirler.

Kurulum işlemi için öncelikle bilgisayarınızdan terminal ekranını açınız ve sırası ile aşağıdaki komutları giriniz.

```
sudo apt-get update
sudo apt-get install gcc
sudo apt-get install g++
```

Yukarıda verilen komutların sırayla çalıştırılması bilgisayarınıza GCC derleyicisini kurulumunu sağlayacaktır.

Sisteminizden kaynaklı olarak bazı durumlarda ek olarak aşağıdaki komutun kullanılması gerekebilir.

```
sudo apt-get install build-essential
```

Bu komut bir C++ programının derlenmesi ve çalıştırılması için gerekli kütüphanelerin yüklenmesini sağlayacaktır¹.

Yukarıdaki işlemlerin tamamlanmasının ardından bir metin (text) dosyasına C++ kodları yazılır ve aşağıdaki örnekte gösterildiği gibi bu dosyanın uzantısı `cpp` olarak belirlenir.

ilkprogram.cpp

Daha sonra yazılan programın derlenip çalıştırılması için aşağıdaki adımlar izlenir

- Bir Linux terminali açarak programını oluşturulduğu klasör içerisine gidilir.
- Terminal ekranında **g++ ilkprogram.cpp** yazılarak programın derlenmesi sağlanır. Bu komut **.cpp** uzantılı dosyanız ile aynı isimde **.exe** uzantılı çalıştırılabilir bir dosya oluşturur. Çalıştırılabilir dosyanıza başka bir isim vermek isteniyorsa ko-

¹ Kurulum esnasında herhangi bir hata oluşmadığınızdan emin olunuz. Bunun için `g++ --version` komutunu kullanabilirsiniz.

mut **g++ ilkprogram.cpp -o yeniisim** şeklinde değiştirilebilir. Bu durumda **yeniisim.exe** isimli bir dosya oluşturulur.

- Bir önceki işlem adımından sonra herhangi bir hata alınmaz ise klasör içerisinde **ilkprogram.exe** ya da **yeniisim.exe** adında yeni bir dosya oluşturulur. Bu dosya çalıştırılabilir bir dosyadır.

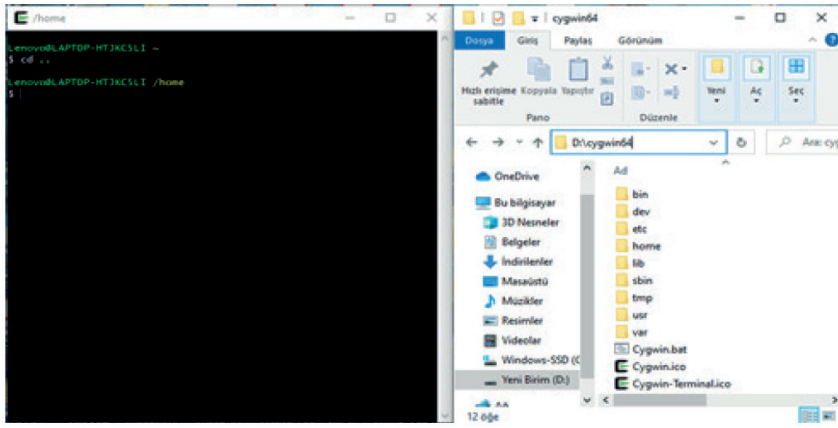
- Yazılan programı exe uzantılı dosyanın ismini kullanarak **./ilkprogram** veya **./yeniisim** şeklinde bir komut kullanarak çalıştırabilir ve sonucu yine terminal ekranında izlenebilir.

Not: Yukarıdaki yönergelerin detaylı anlatımı video kaydında bulunabilir.

2.2. Windows Ortamında C++ Derleyicisinin Kurulumu

Windows ortamında C++ programlarını yazmak, derlemek ve çalıştırmak için bir-çok farklı IDE veya derleyici kullanılması mümkündür. C++ programlama dilinin evrensel yapısı göz önünde bulundurularak bu kitapta Cygwin derleyicisi kullanılacaktır.

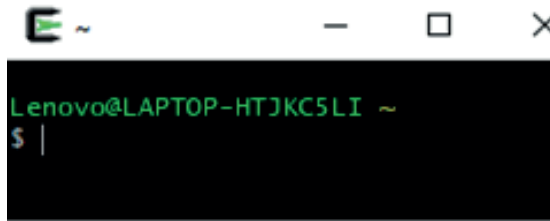
Cygwin, Windows'ta Linux dağıtımına benzer işlevsellik sağlayan bir GNU ve açık kaynak araçlarının kullanımına olanak sağlayan bir ortam olarak ifade edilebilir (Cygwin, 2022)².



Şekil 4.1. Cygwin terminal ekranı ve klasör yapısı

Kurulum işlemi sonrasında terminal ekranını ve kurulum klasörü açıldığında Şekil.4.1'dekine benzer bir yapı görülür. Terminal ekranında klasörler arasında geçiş işlemleri için temel Linux/DoS komutları kullanılacaktır.

Terminal ekranı ilk açıldığında kurulum kök klasörünü gösteren Şekil.4.2'dekine benzer bir adres ekranda gözükecektir.



Şekil 4.2. Terminal ekranı ilk açılış görüntüsü

² Cygwin kurulum dosyalarına ve yönergelerine <https://www.cygwin.com/> adresinden ulaşılabilir. Kurulum ile ilgili önerilen videoya ise <https://www.youtube.com/watch?v=hh-V6el8Oxk> adresinden ulaşılabilir.

Terminal ekranına `cd ..` komutu yazılarak kurulum klasörünün içerisinde bulunan home klasörünün içerisine geçilebilir.

Not: `cd` komutu klasörler arasında geçiş için kullanılır. İngilizce change directory kelimelerinin baş harflerinden oluşturulmuş bir komuttur.

```

/home
Lenovo@LAPTOP-HTJK5LI ~
$ cd ..
Lenovo@LAPTOP-HTJK5LI /home
$

```

Şekil 4.3. Cygwin terminali home klasörüne geçiş

Bu kitap örnekleri için home klasörünün içerisinde ders adında bir alt klasör daha oluşturulmuş ve kod dosyaları bu klasör içerisinde oluşturulmuştur.

Terminal ekranından bu klasöre erişim için home klasörü içerisinde iken `cd ders` yazılmalıdır.

```

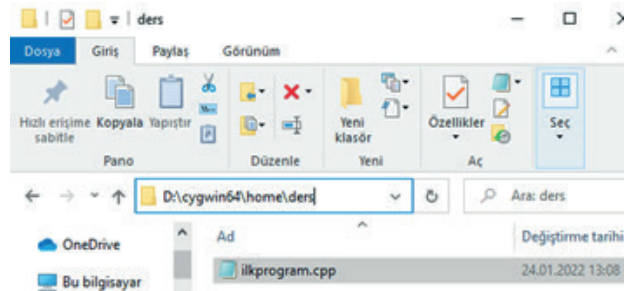
/home/ders
Lenovo@LAPTOP-HTJK5LI ~
$ cd ..
Lenovo@LAPTOP-HTJK5LI /home
$ cd ders
Lenovo@LAPTOP-HTJK5LI /home/ders
$

```

Şekil 4.4. Cygwin terminali üzerinden klasörler arası geçişler

Bu klasör içerisinde oluşturulacak C++ programlarını derlemek ve çalıştırmak için gerekli dosyalar tutulacaktır. Windows ekranından aşağıda gösterildiği gibi `....\home\ders\` klasörünün

içerisine geçerek **ilk program** için `ilkprogram` adında bir metin dosyası oluşturun ve dosyanın uzantısını **cpp (c plus plus)** olarak belirleyin.



Şekil 4.5. İlk program için cpp uzantılı dosyanın oluşturulması

Oluşturulan dosya bir metin dosyasıdır. Dosyayı açıp Örnek.4.1 kodlarını dosya içerisine yazınız.

Örnek.1. `ilkprogram.cpp`

```

#include<iostream>
using namespace std;
int main() {
    cout<<"Ankara Üniversitesine Hoş Geldiniz";
    return 0;
}

```

Kodların Analizi:

- `#include`: Yazılan programa standart veya kullanıcı tanımlı bir dosya eklemek için kullanılan komuttur ve çoğunlukla bir C++ programının başında kullanılır.

- `<iostream>`: C++ programları için temel girdi-çıkı (input-output) servislerini aktif hale getirir.

`#include<iostream>` komutunun birlikte kullanılması, yazılım içerisindeki input-output servislerinin kullanılması için gerekli header (kütüphane) dosyasını programa dahil eder. input-output işlemleri dışında başka header dosyalarına ihtiyaç duyulursa bunlarında include komutu ile birlikte programın başında tanımlamasının yapılması gerekmektedir. Örneğin yazılan programda karekök alma, logaritma hesaplama veya yuvarlama işlemlerinin yapılması isteniyorsa bir matematik kütüphanesini programa eklenmesi gerekmektedir.

Ör. `#include<cmath>`

- `using namespace std`: Bu komut ile std adında bir alan adının (namespace) programda kullanılacağı belirtilmektedir. Std standart kelimesinin kısaltılmasıdır. Eğer programda `cout`, `cin` gibi input-output işlemleri için gerekli fonksiyonların kullanılması gerekiyorsa, programın bu işlemlerin nasıl yapılacağını bilmesi gerekmektedir. Std alan adı bu fonksiyonların tanımlandığı bir alandır. Eğer başlangıçta bu tanımlama yapılmazsa, kodlar yazılırken `std::cout`, `std::cin`, `std::endl` gibi her bir fonksiyon öncesi std komutunun :: ile birlikte kullanılması gerekir.

- `int main()`: Derleyicinin ilk bağlantı kurduğu bölümdür. Her programın bir main fonksiyonu olmak zorundadır. Bu örnekte integer (tamsayı) türünde değer döndüren bir main fonksiyonu tanımlaması verilmiştir. Void gibi türlerin tanımlanması da mümkündür (void main). Main fonksiyon adından sonra açılıp kapatılan parantezler

'(' , ') ' içerisinde fonksiyonun aldığı parametreler tanımlanır. Fonksiyonlar bölümünde bu konu daha detaylı olarak ele alınacaktır.

- `{ }` : main fonksiyonundan hemen sonra açılan ve programın sonunda kapatılan küme parantezleri ise fonksiyonun etkili olduğu alanı (scope) tanımlar. Küme parantezleri içerisinde yazılan kodlar sadece ilgili fonksiyonun kullanımı içindir.

- `cout<<`: `cout` fonksiyonu ekrana bir şey yazdırmak için kullanılır. Yazdırılan ifade değişkenin içerisinde tutulan değer ya da bir string ifade olabilir. `cout` komutundan hemen sonra kullanılan `<<` operatörleri, aşırı yüklenmiş (over-loaded) stream (akış) operatörleri olarak ifade edilir. Programa girdi olarak bir değer alınacaksa `cin>>` şeklinde kullanımı vardır. Bu operatörler programlamada bitwise shift operatörleri olarak bilinir. leftshift (sola kaydır, `<<`) ve rightshift (sağa kaydır, `>>`) olarak tanımlanmıştır. Ancak input-output işlemleri için overload edilmiş bu fonksiyonlar `cout` ve `cin` komutları ile birlikte kullanılırken `std::ostream` (output akışı), `std::istream` (input akışı) işlemleri için ayrı bir görev ile yeniden tanımlanmışlardır (Bkz. Operatör overloading).

- `return 0`: Normalde bütün fonksiyonlar geriye bir değer döndürür. Program `return 0` komut satırını gördüğünde program sorunsuz bir şekilde tamamlanmış demektir. C++ da `return 0` komutunun kullanılması isteğe bağlı olmakla birlikte C programlama dilinde kullanımı bir zorunluluktur. Profesyonel yazılımcılar `return` tipi olarak farklı değerler kullanabilirler. Bu konu yine fonksiyonlar bölümünde detaylı olarak ele alınacaktır.

- `;` (Noktalı Virgül) : Programda bazı satırların sonunda `;` karakterinin kullanıldığına dikkat ediniz. C++ da komut satırları `;` karakteri ile sonlandırılmak zorundadır. Aksi halde derleme zamanında hata alınır.

- “ ”: cout komutunu takip eden “Ankara Üniversitesine Hoş Geldiniz” ifadesinin “...” işaretleri içerisinde yazıldığına dikkat ediniz. Eğer bir metni (string ifade) ekrana yazdırmak istiyorsanız bu tırnak işaretlerini kullanmak zorundasınız. Şimdi yazdığımız programı derleyelim ve çalıştıralım. Kullanılan Cygwin derleyicisi bizlere

Linux benzeri bir ortam sunduğu için Linux ortamı başlığı altında bahsedilen program derleme ve çalıştırma adımları bu bölüm içinde geçerlidir.

Öncelikle daha önce bahsedildiği gibi Cygwin terminal ekranını açınız ve programı yazdığınız klasörü işaret ediniz (Ör. .../cd ...).

```

/home/ders
Lenovo@LAPTOP-HTJKC5LI ~
$ cd ..
Lenovo@LAPTOP-HTJKC5LI /home
$ cd ders
Lenovo@LAPTOP-HTJKC5LI /home/ders
$ ls
ilkprogram.cpp
Lenovo@LAPTOP-HTJKC5LI /home/ders
$

```

Şekil 4.6. Terminal ekranından klasör içerisine göz atma

Şekil 4.6’nın 3. satırında kullanılan ls (list files) komutu bulunduğunuz klasör altındaki dosyaları görüntülemek için kullanılır. Burada ders klasörünün altında metin dosyası içerisine yazılmış olan ilkprogram.cpp uzantılı dosya görüntülenmektedir. Bu dosyanın çalıştırılabilir (executable) bir dosya haline getirmek için derlenmesi gerekmektedir. Şimdi aşağıdaki adımları terminal ekranında sırası ile yazarak programınızı derleyiniz ve çalıştırınız.

- g++ ilkprogram.cpp //Bu komut yazdığınız programı derleyecek ve herhangi bir hata alınmaması durumunda çalıştığınız klasör altında a.exe uzantılı yeni bir dosya oluşturacaktır. Dosyanın oluşup oluşmadığını terminal ekranından ls komutunu kullanarak veya Windows ekranlarından mevcut klasörün altına giderek kontrol edebilirsiniz.

- Eğer herhangi bir hata almadan exe uzantılı dosya oluşturuldu ise bu sefer terminal ekranına ./a.exe komutunu yazarak programınızı çalıştırabilirsiniz.

Derleme esnasında bir sözdizimi (syntax) hatası alınmadıysa ve program yazılırken mantıksal

(logical) bir hata yapılmadıysa ./a.exe komutunun kullanımı ile yazılan program çalıştırılır ve cout komutu ile ekrana yazılması istenilen ifade ekranda görüntülenir.

Çalıştırılabilir dosyanın adı derleyici tarafından varsayılan (default) olarak a.exe olarak belirlenir. Aynı klasör içerisinde tutulan farklı programlar derlendiğinde a.exe uzantılı dosya en son derlenen programa ait olacaktır. Programları tekrar tekrar derlemek yerine varsayılan isim yerine özel bir isim kullanılarak programlar kendilerine verilen özel isimler ile oluşturulan exe uzantılı dosyalar üzerinden çalıştırılabilir. Derlenecek olan exe uzantılı dosya için özel bir isim kullanılmak istenirse derleme komutu:

- g++ ilkprogram.cpp -o yeniisim

şeklinde yazılabilir. Bu komut ilkprogram.cpp dosyasından yeniisim.exe adında çalıştırılabilir bir dosya oluşturur (Sizler istediğiniz herhangi bir ismi verebilirsiniz).

Yukarıda bahsedilen tüm işlem adımları aşağıdaki Şekil.4.7’de verilmiştir.

```

/home/ders
Lenovo@LAPTOP-HTJKCSLI ~
$ cd ..
Lenovo@LAPTOP-HTJKCSLI /home
$ cd ders
Lenovo@LAPTOP-HTJKCSLI /home/ders
$ ls
ilkprogram.cpp
Lenovo@LAPTOP-HTJKCSLI /home/ders
$ g++ ilkprogram.cpp
Lenovo@LAPTOP-HTJKCSLI /home/ders
$ ls
a.exe ilkprogram.cpp
Lenovo@LAPTOP-HTJKCSLI /home/ders
$ ./a.exe
Ankara Üniversitesine Hoşgeldiniz
Lenovo@LAPTOP-HTJKCSLI /home/ders
$ g++ ilkprogram.cpp -o yeniisim
Lenovo@LAPTOP-HTJKCSLI /home/ders
$ ls
a.exe ilkprogram.cpp yeniisim.exe
Lenovo@LAPTOP-HTJKCSLI /home/ders
$ ./yeniisim.exe
Ankara Üniversitesine Hoşgeldiniz
Lenovo@LAPTOP-HTJKCSLI /home/ders
$ |

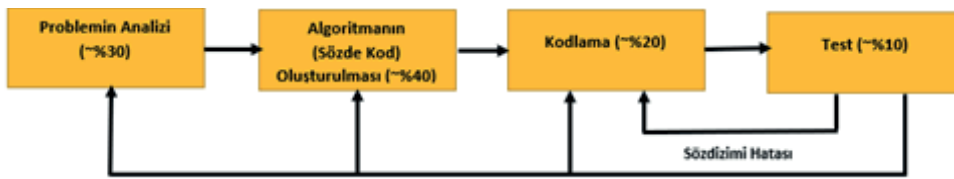
```

Şekil 4.7. Terminal ekranından bir programın derlenmesi ve çalıştırılması

3. SÖZDİZİMİ HATA DENETİMİ

Yazılan programın derlenmesi esnasında karşılaşılabilecek sözdizimi (syntax) hataları bu bölümde ele alınmıştır. Program geliştirme esnasında sözdizimi hatalarının yanı sıra mantıksal hataların yapılması da olasıdır. Bir örnek oluşturması

açısından problem analizinde ya da algoritma geliştirirken toplama işlemi yapılması gereken bir yerde çıkarma işleminin yapılması mantıksal bir hata meydana getirecektir.



Şekil 4.8. Program geliştirme adımları

Programda yapılacak mantıksal hatalar, hatanın yapıldığı bölüme bağlı olarak problem analizi kısmına kadar yapılan işleri gözden geçirmeyi gerektirebilir. Sözdizimi hataları ise kod yazımı esnasında yapılan hataları ifade eder. Bu hatalar kodlama ekranında düzeltilip program yeniden derlenip çalıştırılabilir.

Bir sözdizimi hatası için Örnek.4.2.a'daki kodları inceleyin:

Örnek.4.2.a. program2.cpp

```

#include<iostream>
using namespace std;
int main() {
    cout<<"Ankara Üniversitesine Hoş Geldiniz"
    return 0;
}

```

Programın 4. satırında cout ile verilen ifadenin sonunda “;” özellikle kaldırılmıştır. Program

bu şekilde kaydedilip derlendiğinde aşağıdaki ekran görüntüsündeki gibi bir hata ile karşılaşılır.

```

/home/ders
Lenovo@LAPTOP-HTJK5LI ~
$ cd ..

Lenovo@LAPTOP-HTJK5LI /home
$ cd ders

Lenovo@LAPTOP-HTJK5LI /home/ders
$ ls
a.exe  ilkprogram.cpp  program2.cpp  yeniisim.exe

Lenovo@LAPTOP-HTJK5LI /home/ders
$ g++ program2.cpp -o program2
program2.cpp: 'int main()' işlevinde:
program2.cpp:4:45: hata: expected ';' before 'return'
   4 | cout<<"Ankara Üniversitesine Hoş Geldiniz"
     |                                           ^
   5 | return 0;
     | ~~~~~
Lenovo@LAPTOP-HTJK5LI /home/ders
$

```

Şekil 4.9. Sözdizimi (syntax) hatası

Şekil 4.9'da görüldüğü gibi program2.cpp dosyası program2.exe dosyasını oluşturacak şekilde derlenmeye çalışılmıştır. Ancak derleme esnasında programın 4. satırında return komutundan hemen önce olası bir “;” ifadesinin kullanımının unutulduğu ile ilgili bir uyarı ekranda görüntülenmiştir. Aynı zamanda bu bölüm bir ok işareti ile gösterilmiştir. Sözdizimi hataları, derleyici tarafından kolaylıkla tespit edilebilir ve programcayı sözdizimi hataları konusunda uyarır.

Şimdi Örnek.4.2.b'de cout kullanımında “<<” (stream out) operatörünü “>>” (stream in) operatörü ile değiştirin.

Örnek.4.2.b. program2.cpp

```
#include<iostream>
```

```
using namespace std;
```

```
int main() {
```

```
cout>>"Ankara Üniversitesine Hoş Geldiniz";
```

```
return 0;
```

```
}
```

```

/home/ders
Lenovo@LAPTOP-HTJK5LI /home/ders
$ ls
a.exe  ilkprogram.cpp  program2.cpp  yeniisim.exe

Lenovo@LAPTOP-HTJK5LI /home/ders
$ g++ program2.cpp -o program2
program2.cpp:1:1: error: no match for 'operator<>' (operand types are 'std::ostream' (aka 'std::basic_ostream<char>') and 'const char [17]')
1 | cout>>"Ankara Üniversitesine Hoş Geldiniz";
  | ~~~~~^~~~~~
  |
  | 1 | const char [17]
  | 2 |
  | 3 | std::ostream (aka std::basic_ostream<char>)
  |
  | In file included from /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/string:58:
  |   from /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/bits/locale_classes.h:60:
  |   from /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/bits/locale_base.h:41:
  |   from /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/ios:42:
  |   from /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/ostream:38:
  |   from program2.cpp:1:
  | /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/bits/basic_string.tcc:1476:5: note: candidate: 'template<class _CharT, class _Traits, class _Alloc> std::basic_ostream<_CharT, _Traits, _Alloc> operator<>(std::basic_ostream<_CharT, _Traits, _Alloc>&, const _CharT*)'
  | /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/bits/basic_string.tcc:1476:5: note:   template argument deduction/substitution failed:
  | program2.cpp:6:17: note:   'std::ostream' (aka 'std::basic_ostream<char>') is not derived from 'std::basic_ostream<_CharT, _Traits, _Alloc>'
  | 6 | cout>>"Ankara Üniversitesi Hoş Geldiniz";
  |
  | In file included from /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/ostream:40:
  |   from /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/bits/ostream.tcc:111:5: note: candidate: 'template<class _CharT, class _Traits> std::basic_ostream<_CharT, _Traits> operator<>(std::basic_ostream<_CharT, _Traits>&, const _CharT*)'
  | /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/bits/ostream.tcc:111:5: note:   template argument deduction/substitution failed:
  | program2.cpp:6:17: note:   'std::ostream' (aka 'std::basic_ostream<char>') is not derived from 'std::basic_ostream<_CharT, _Traits>'
  | 6 | cout>>"Ankara Üniversitesi Hoş Geldiniz";
  |
  | In file included from /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/ostream:40:
  |   from /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/bits/ostream.tcc:111:5: note: candidate: 'template<class _Traits> std::basic_ostream<_Traits> operator<>(std::basic_ostream<_Traits>&, const _Traits*)'
  | /usr/lib/gcc/x86_64-linux-gnu/10/include/c++/bits/ostream.tcc:111:5: note:   template argument deduction/substitution failed:
  | program2.cpp:6:17: note:   'std::ostream' (aka 'std::basic_ostream<char>') is not derived from 'std::basic_ostream<_Traits>'
  | 6 | cout>>"Ankara Üniversitesi Hoş Geldiniz";
  |
  |

```

Şekil 4.10. Derleme zamanı hatası

Programın derlenmesi sırasında, terminal ekranında Şekil 4.10'dan görüldüğü gibi oldukça uzun bir hata mesajı ile karşılaşılır. Daha önce belirtildiği gibi, "<<" ve ">>" operatörleri aşırı yüklenmiş operatörlerdir. Bu operatörler cout ya da cin gibi komutlar ile kullanılırken kendi görevleri dışında (bitshift) stream akışını (out ya da in) kontrol etmek üzere programlanmışlardır (aşırı yüklenmişlerdir, overload edilmişlerdir). Hata ekranında iki bölümde gösterildiği gibi (normal ekranda daha fazlası gözükecektir) std::ostream (stream out) sınıfından ">>" operatörünün türetilemediğine dair bir hata mesajı çıkmaktadır. Bu-

radaki hata, bir operatörün görevi dışında başka bir amaçla kullanılmasından kaynaklanmaktadır.

Şimdi, iostream header dosyasının (kütüphane) programa dahil edilmediği Örnek.4.2.c'yi inceleyiniz. Örnekte iostream dışında kalan bölümler düzgün olarak verilmiştir.

```
Örnek.4.2.c. program2.cpp
using namespace std;
int main() {
    cout<<"Ankara Üniversitesine Hoş Geldiniz";
    return 0;
}
```

```

/home/ders
Lenovo@LAPTOP-HTJKCSLI /home/ders
$ g++ program2.cpp -o program2
program2.cpp: 'int main()' işlevinde:
program2.cpp:4:1: hata: 'cout' bu etki alanında bildirilmemiş
    4 | cout<<"Ankara Üniversitesine Hoş Geldiniz";
      | ~~~~~
program2.cpp:1:1: bilgi: 'std::cout' is defined in header '<iostream>'; did you forget to
#include '<iostream>'?
+++ |+#include <iostream>
    1 |
Lenovo@LAPTOP-HTJKCSLI /home/ders
$ |

```

Şekil 4.11. Gerekli header dosyası yüklenmediğinde karşılaşılan hata

Terminal ekranından görüleceği üzere, gerekli header dosyası programa eklenmediğinde yine derleme zamanı hatası ile karşılaşılır. Hata mesajında main fonksiyonu içerisinde kullanılan cout ifadesinin bildirilmediği ve cout komutu-

nun iostream header dosyası içerisinde olduğu ifade edilmektedir. Sonuç olarak iostream header dosyasının programa dahil edilmediği için bir derleme zamanı hatası oluştuğu belirtilmektedir.

4. TERMINAL EKRANINDAN VERİ GİRİŞİ

Daha önceki örneklerde cout komutunu kullanarak ekrana string bir ifadenin nasıl yazdırılacağı gösterildi. Programa terminal ekranından kullanıcının veri girişini nasıl yapacağı bu bölümde gösterilecektir. İlgili klasör içerisinde oluşturulacak yeni program içerisine aşağıdaki kodları yazınız.

Örnek.4.3. program3.cpp

//Bu satır açıklama satırıdır derleyici tarafından derlenmez

```
#include<iostream>
using namespace std;
```

```
int main() {
    int a;
    int b, c;
    cout<<"İlk Sayıyı Giriniz:";
    cin>>a;
    cout<<"İkinci Sayıyı Giriniz:\n";
    cin>>b;
    c= a+b;
    cout<<"Toplam="<<c<<endl;
    return 0;
}
```


Kodların Analizi:

- Programın 1. Satırında verilen “//” ifadesi program yazılırken açıklama satırları eklenmek istenirse kullanılır. Tek satırlık açıklamalar için kullanılır. Eğer birden fazla satırı açıklama satırı haline getirmek istiyorsanız “/**/” şeklinde kullanılır. Program tarafından işlenmez.

- Programın 5. Satırında integer (tamsayı) türünde a adında bir değişken tanımlaması yapılmıştır. Değişken türleri ile ilgili daha detaylı bilgi için 5. bölümü inceleyiniz.

- Programın 6. satırında yine integer türünde b ve c adında iki değişken tanımlaması yapılmıştır. 5. Satırdan farklı olarak burada bir satırda 2 değişken birlikte tanımlanmıştır. 5. Satırda olduğu gibi bir değişken tanımlamasından sonra “;” kullanımı ile o satır sonlandırılabilirdiği gibi, değişkenler arasına “,” işareti konularak birden fazla değişken tanımlanması mümkündür.

- 7. satırda programı kullanan kişiye bir sayı girmesi gerektiğini söyleyen bir mesaj oluşturulmuştur. 9. satırda yazılan mesajın sonunda “\n” ifadesinin kullanıldığına dikkat ediniz. “\n” ifadesi programlamada kullanılan escape (kaçış) ifadelerinden bir tanesidir ve yeni satır (new line) anlamına gelir. Bu karakter kullanıldığında mesaj ekrana yazıldıktan sonra sanki enter tuşuna basılmış gibi terminal ekranındaki imleç yeni

ekranda görüntülenir. 7 ve 9. Satırlarda kullanılan iki kullanımda doğrudur ve sadece kullanım farklılıklarını göstermek için farklı şekillerde gösterilmiştir.

- 8 ve 10. satırda verilen cin komutu kullanıcının terminal ekranından programa bir değer girmesine olanak tanır. istream (input stream) komutu için “>>” aşırı yüklenmiş operatörleri ile birlikte kullanılır. Program çalıştırılırken sıra bu satıra gelince denetim terminal ekranına geçer ve kullanıcının bir değer girmesi beklenir. Sonrasında klavyeden enter tuşuna basılarak denetim yeniden derleyiciye bırakılır, bu esnada kullanıcı tarafından girilen değerler a ve b değişkenleri içerisine atanır.

- 11. satırda kullanıcı tarafından a ve b değişkenleri için girilen iki sayı toplanır ve sonuç c değişkenine atanır.

- 12. satırda öncelikle ekrana “Toplam=” ifadesi yazdırılır daha sonra bu ifadeye birleşik olacak şekilde c değişkeni içerisindeki değer ekrana yazdırılır ve son olarak endl (end line) komutu yazma işlemi sonrası programın yeni bir satıra geçmesi sağlanır. endl komutu “\n” komutu ile aynı işi yapar. Bu satırda, cout komutunun “<<” operatörlerinin tekrarlı kullanımı ile birden fazla işlemi aynı anda gerçekleştirildiğine dikkat ediniz.

```

/home/ders
Lenovo@LAPTOP-HTJKC5LI /home/ders
$ ls
a.exe  ilkprogram.cpp  program2.cpp  program3.cpp  yeniisim.exe

Lenovo@LAPTOP-HTJKC5LI /home/ders
$ g++ program3.cpp -o program3

Lenovo@LAPTOP-HTJKC5LI /home/ders
$ ./program3.exe
İlk Sayıyı Giriniz:5
İkinci Sayıyı Giriniz:
6
Toplam=11

Lenovo@LAPTOP-HTJKC5LI /home/ders
$ |

```

Şekil 4 12. Terminal ekranından veri giriş örneği

BÖLÜM ÖZETİ

Bu bölümde temel veri türleri, değişkenler, atama ve aritmetik işleçlerden bahsedilmiştir. Bir bilgisayar programı hafızada sakladığı çeşitli verileri işleyerek birtakım sonuçlar üretir. Birçok farklı alanda programlara ihtiyacımız olduğu düşünülünce çok farklı veri türlerine gereksinim duyulmaktadır. C++ programlama dilinde tanımlı üç farklı temel veri türü vardır. Bu türler sayısal, boolean ve karakter veri türleridir. Sayısal veri türleri kendi içinde tam ve kesirli sayılar olmak üzere ikiye ayrılmaktadır. Bir bilgisayar programı programcının tanımladığı değişkenleri işleyerek istenilen sonuca ulaşır. Değişkenlere uygun isim vermek yazılan kodun okunabilirliğini artırır. Çünkü yazılım yaşam döngüsünde bulunan süreçlerin gereği olarak bir programcının yazmış olduğu kodu başka programcılarında kullanması veya test etmesi gerekmektedir. Ayrıca C++ programlama dilinde bir değişkene isim vermek için bazı kurallar vardır. Değişkenlere bu kurallara uygun isimler vermek derleme hatalarının önüne geçecektir. Tanımlı değişkenler aritmetik, ilişkisel veya mantıksal işleçler içerisinde kullanılarak daha karmaşık ifadelerin yazılması mümkün hale gelmektedir. İşleçler konusu oldukça geniş bir kavramdır. Bu bölümde atama ve aritmetik işleçlerden bahsedildi. Atama işleci bir değişkene bir değer atamak için kullanılır. Buna ek olarak toplama, çıkarma, bölme, çarpma ve kalan gibi aritmetik işleçler anlatılmıştır. Bir ifade içerisinde birden fazla işleç kullanarak daha karmaşık ifadelerin elde edilmesi mümkündür. Bu durumda hangi işlecin önce hesaplanması gerektiği işlem öncelik kurallarına göre belirlenir. Bu kuralları bilmek program içerisinde yapılabilecek hataları en aza indirgeyecektir.

GÖZDEN GEÇİRELİM

1. Aşağıdakilerden hangisi C++ programları için dosya uzantısı olarak kullanılır?
 - a) c
 - b) cpp
 - c) c++
 - d) ls
 - e) cd
2. Bir program yazılırken komut satırının sonuna ";" eklenmesinin unutulması nasıl bir hata oluşmasına neden olur?
 - a) Sözdizim (Syntax) Hatası
 - b) Mantıksal (Logical) Hata
 - c) Kütüphane hatası
 - d) Alan adı hatası
 - e) Hata oluşmaz
3. Ornek.cpp isimli bir dosyanın Ornek.exe şeklinde derlenmesi için gerekli komut aşağıdakilerden hangisidir?
 - a) g++ Ornek.cpp -o
 - b) g++ Ornek.cpp
 - c) g++ Ornek.cpp -o Ornek
 - d) Ornek.exe -o g++ Ornek.cpp
 - e) g++ Ornek.exe
4. Ornek.exe uzantılı dosyayı çalıştırmak için aşağıdaki komutlardan hangisi kullanılmalıdır?
 - a) ./Ornek.cpp
 - b) Ornek.cpp
 - c) Ornek.exe
 - d) -Ornek.exe
 - e) ./Ornek.exe
5. Bir C++ programına iostream header'ı eklenmemişse aşağıdaki sorunlardan hangisi ile karşılaşılır?
 - a) Değişken tanımlaması yapılamaz
 - b) Main fonksiyonu hata verir
 - c) Aşırı yükleme hatası alınır
 - d) input-output işlemlerinde hata alınır
 - e) Mantık hatası verir
6. Ornek.cpp isimli dosya g++ Ornek.cpp yazarak derlenirse oluşturulacak çalıştırılabilir dosyanın adı ve uzantısı aşağıdakilerden hangisi olur?
 - a) Ornek.cpp
 - b) Ornek.exe
 - c) a.exe
 - d) a.cpp
 - e) Hata verir

7. **cout** komutu ile ilgili aşağıdakilerden hangisi(leri) söylenebilir?

I. Çıkış akışını (output-stream) kontrol eder

II. Giriş akışını (input-stream) kontrol eder

III. Sağa kaydır operatörüdür

- a) Yalnız I
- b) Yalnız II
- c) Yalnız III
- d) II ve II
- e) Hiçbiri

8. **cd** (Change Directory) komutu ne için kullanılır?

- a) Klasör ismini değiştirmek için
- b) Klasörler arası geçiş için
- c) Klasör silmek için
- d) Klasörleri yeniden isimlendirmek için
- e) Dosya oluşturmak için

9. **ls** (List Files) komutu ne için kullanılır?

- A) Dosya isimlerini değiştirmek için
- B) Dosya oluşturmak için
- C) Klasör isimlerini değiştirmek için
- D) Klasörleri silmek için
- E) Bulunulan klasör içerisindeki dosyaları ve diğer klasörleri listelemek için

10. **endl** komutu ile ilgili olarak aşağıdakilerden hangisi(leri) söylenebilir?

I. Program bu komutu görünce bir alt satırdan devam eder

II. **/n** komutu ile benzer görevi gerçekleştirir

III. Programı sonlandırır

IV. Hata denetimi için kullanılır

- a) Yalnız I
- b) Yalnız II
- c) Yalnız III
- d) I ve II
- e) III ve IV

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

Sözlük

Header file

Program başında kullanılması gereken, program içerisindeki komutların bulunduğu alan adı.

Compiler (Derleyici):

Yazılan programların derlenip denetiminin yapıldığı ortam

Exe (Executable)

Çalıştırılabilir dosyaların uzantısı

cpp: C++ programlarının yazıldığı dosyanın uzantısı

syntax (Sözdizimi)

Program geliştirilirken kullanılan dile ait özellikler (Örneğin, ,, küme parantezleri vb)

overload (aşırı yükleme)

Bir operatörün normal kullanım amacı dışında farklı görevler içinde yüklenmesi

cd (change directory)

Terminal ekranında klasörler arası geçiş komutu

ls (list-file)

Terminal ekranından bir klasörün içerisinde bulunan alt klasör ve dosyaları listeleme komutu

g++

Program derleme komutu

Kaynakça

(2022, Ocak 22). Cygwin: <https://www.cygwin.com/> adresinden alındı

BÖLÜM 5

VERİ TÜRLERİ VE ARİTMETİK İŞLEÇLER

ANAHTAR KAVRAMLAR

Tam Sayı, Kesirli Sayı, Karakter, Değişken, Atama İşleci, Aritmetik İşleçler, İşleç Önceliği

ÖĞRENME HEDEFLERİ

1

Veri türleri arasındaki farkları bilir

2

Değişken tanımlama ve isimlendirme kurallarını bilir

3

Atama işlecini nasıl kullanacağını bilir

4

Aritmetik işleçleri nasıl kullanacağını bilir

5

Aritmetik işleçlerin öncelik sıralamasını bilir

BEŞİNCİ BÖLÜM

GİRİŞ

Bir veri türü, bir türün alabileceği değer aralıkları ve üzerinde tanımlanmış iş-

lemler ile tanımlanır. Bir bilgisayar programı sakladığı verileri işleyerek sonuçlar üretir. Bazı programlama dilleri bir değişken tanımlanırken veri türünü belirtmeyi şart koşarken bazıları daha esnek olmaları sebebiyle böyle bir şart koşturmazlar. Veri türünün önceden bilinmesiyle o değişken için hafızada ne kadar alan gerektiği derleyici tarafından önceden hesap edilebilir. C, C++ ve Java gibi programlama dillerinde değişken oluştururken veri türünün belirtilmesi gerekir. Python ve Ruby gibi diller de böyle bir şart yoktur. Bu tür dillerin avantajı programcının veri türleriyle uğraşmasını ortadan kaldırır. Aynı isme sahip farklı veri türlerinde değişken oluşturmak mümkün hale gelmektedir. Böylece bellek kapasitesi daha verimli bir şekilde yönetilir.



1. TEMEL VERİ TÜRLERİ

Temel veri türleri diğer türler cinsinden tanımlanmayan veri türleri olarak tanımlanır. C++ programlama dilinde tanımlı üç adet temel veri tipi vardır:

- Sayısal
- Boolean
- Karakter

Tablo 5.1 C++ Programlama Dilinde Tanımlı Temel Bazı Veri Türleri ve Değer Aralıkları

Veri Türü	Boyut	Değer Aralığı
unsigned short	2 bayt	0 – 65.535
short	2 bayt	-32.768 – 32.767
unsigned long	4 bayt	0 – 4.294.967.295
long	4 bayt	-2.147.483.648 – 2.147.483.647
int	4 bayt	-2.147.483.648 – 2.147.483.647
unsigned int	4 bayt	0 – 4.294.967.295
char	1 bayt	256 karakter
float	4 bayt	1,2e-38 – 3,4e38
double	8 bayt	2,2e-308 – 1,8e308

Tablo 5.1’de C++ programlama dilinde tanımlı bazı veri türleri ve tutabilecekleri değer aralıkları verilmiştir. Tablodaki değerler kullanılan işletim sistemine ve derleyiciye bağlı olarak farklılıklar

gösterebilmektedir. Bir değişkenin hafızada ne kadar alana gereksinim duyduğunu öğrenmek için “sizeof()” fonksiyonu kullanılır.

```
cout << sizeof(short)<<endl;
cout << sizeof(int)<<endl;
cout << sizeof(long)<<endl;
cout << sizeof(float)<<endl;
cout << sizeof(double)<<endl;
cout << sizeof(char)<<endl;
cout << sizeof(bool)<<endl;
```



DİKKAT EDELİM

Her bir veri türünün tutabileceği değer aralıkları belirlidir. Değişken tanımlaması yaparken saklanacak veriye uygun veri türleri seçilmelidir.



ARAŞTIRALIM

Tablo 5.1'de olmayan diğer temel veri türleri <https://en.cppreference.com/w/cpp/language/types> adresinden incelenebilir.

2. SAYISAL VERİ TÜRLERİ

Bilimsel hesaplamalar genellikle sayısal veriler kullanılarak yapılır. Birçok bilim dalının ihtiyacını karşılamak için farklı sayısal veri türleri ta-

nımlanmıştır. C++ programlama dilinde tanımlı sayısal veri türleri tam sayılar ve kesirli sayılar olmak üzere ikiye ayrılır.

2.1. Tam Sayı Veri Türleri

Tam sayı veri türlerinin kesirli parçaları yoktur. Bir sınıftaki öğrenci sayısı ve otoparktaki araç sayısı bu gruba girer. En genel haliyle “int” anahtar kelimesi kullanarak tam sayı veri türünden değişkenler tanımlanabilir. Kullanıcının ihtiyacına göre daha az alan gerektiren “short” veya daha çok alan gereksinimi olan “long” anahtar kelimeleri ile tam sayı veri türleri tanımlamak mümkündür. Bu anahtar kelimelerin önüne “unsigned” eklenmesiyle sadece pozitif değer aralığında tam sayılar elde edilebilir.

Örnek:

```
int fiyat = 50000;
short mesafe = 40;
long dünya_nufusu=8000000000;
cout<<"Fiyat="<<fiyat<<endl;
cout<<"Mesafe="<<mesafe<<endl;
cout<<"Dünya nüfusu="<<dünya_nufusu<<endl;
```

Çıktı:

```
Fiyat=50000
Mesafe=40
Dünya nüfusu=8000000000
```

2.2. Kesirli Sayı Veri Türleri

Kesirli sayı veri türleri ondalık değerlere sahip verileri temsil etmek için tanımlanmıştır. Bir kişinin ağırlık ve boy bilgisi bu grupta değerlendirilir. En genel haliyle “float” veya “double” anahtar kelimeleri kullanarak tanımlanır. Bir değişken “double” olarak tanımlanmışsa daha geniş değer aralığına sahip olmaktadır ama hafızada daha fazla alan gereksinimi olacaktır. Benzer şekilde önlerine “unsigned” kelimesi eklenerek pozitif değerler aralığında olması sağlanır. Bu türden değerler bilgisayar hafızasında tam olarak saklanamadığı için dikkatli bir şekilde kullanılması gerekmektedir. Özellikle finans uygulamalarında gerçek değer ile saklanan veri türleri tercih edilmelidir.

Örnek:

```
float boy=150.35;
cout<<"Boy= "<<boy<<endl;
```

Çıktı:

```
Boy= 150.35
```



GÜNLÜK HAYATLA İLİŞKİLENDİRELİM

Günlük hayatta kullandığımız bilgilerin hangileri tam sayı hangileri kesirli sayı veri türündedir.

2.3. Boolean Veri Türü

Bir boolean veri türü sadece “true” veya “false” olmak üzere iki farklı değer alır. Karar alma veya ilişkisel işlemlerde kodun okunabilirliğini artırmak için tercih edilir. Bir boolean değişkeni hafızada 1 baytlık alanda saklanır. Bir boolean değişkeni bool anahtar kelimesi ile tanımlanır. bool aktif=true; Eğer boolean veri türündeki bir değişkenin çıktısının alınması durumunda true için 1 false için 0 değeri yazdırılır.

Örnek:

```
bool aktif = true;
cout <<aktif <<endl;
aktif = false;
cout <<aktif <<endl;
```

Çıktı:

```
1
0
```

2.4. Karakter Veri Türü

Karakter veri türü standart ASCII karakter setindeki 256 karakter veya sembollerden birini temsil eder ve hafızada bir baytlık alanda saklanır. Bir değişkene atama yapılırken tek tırnak içerisinde alınması gerekir.

Örnek:

```
char harf = 'a';
cout << harf <<endl;
harf = 'A';
cout <<harf <<endl;
harf = '5';
cout <<harf <<endl;
harf = '?';
cout <<harf <<endl;
```

Çıktı:

```
a
A
5
?
```

Tablo 5.2 Kaçış Karakterlerinin Kullanımı.

\n	Sonraki satıra geç
\t	Klavyedeki tab tuşuna karşılık gelen karakterdir
\'	Tek tırnak
\\	Ters eğik çizgi (backslash)
\0	Boş (null)

Karakter veri türü için <cctype> kütüphanesinde tanımlanmış fonksiyonlarının bazıları Tablo 5.3’de verilmiştir.

Tablo 5.3 Karakter Veri Türü İçin Tanımlanmış Bazı Fonksiyonlar.

int isdigit(int c)	Eğer c bir rakam ise 1, değilse 0 döndürür.
int isalpha(int c)	Eğer c bir harf ise 1, değilse 0 döndürür.
int islower(int c)	Eğer c küçük harf ise 1, değilse 0 döndürür.
int isupper(int c)	Eğer c büyük harf ise 1, değilse 0 döndürür.
int tolower(int c)	Eğer c büyük harf ise küçük harfe dönüştürür, değilse değişiklik yapmaz.
int toupper(int c)	Eğer c küçük harf ise büyük harfe dönüştürür, değilse değişiklik yapmaz.

2.5. Numaralandırma (enum) Veri Türü

Numaralandırma veri türü bir sabit listesi oluşturmak için kullanılır. Örneğin araba markalarından oluşan bir enum veri türü aşağıdaki gibi tanımlanabilir.

```
enum araba {Honda, Toyota, Mazda, Subaru};
```

Eğer liste içerisindeki değerlere programcı tarafından bir değer ataması yapılmaz ise listedeki ilk değere 0 sonraki değerler birer artırılarak değer ataması yapılır. Bu durumda yukarıda verilen örnekteki gibi bir tanımlama yapılırsa Honda "0", Toyota "1", Mazda "2" ve Subaru "3" değerlerini alırlar. Programcı isterse listedeki her bir değere aşağıda verilen örnekte olduğu gibi uygun bir atama da yapabilir.

```
enum araba {Honda=10, Toyota=20, Mazda=30, Subaru=40};
```

Örnek:

```
enum araba {Honda=10, Toyota=20, Mazda=30, Subaru=40};
```

```
araba mycar=Toyota;
```

```
cout<<mycar;
```

Çıktı:

```
20
```

Değişkenler

Değişken bilgisayar belleğinde verinin saklandığı ve üzerinde işlemler yapıldığı bir depolama konumudur. Bir değişken adı, sakladığı içeriğe göre veri türü ve üzerinde çağrılabilceği işlemler ile tanımlanır.

Değişken Bildirimi

Bir değişken saklayacağı veri türü ve değişken adıyla birlikte tanımlanır. Örneğin kişinin yaş bilgisini tutacak değişkenin bildirimi aşağıdaki gibi olacaktır.

```
int yas;
```

Bir değişkenin bildirimi yapıldıktan sonra kullanmak için bir değer ataması yapılması gerekir. Örneğin kişinin yaşının 40 olduğu durumda değer atama işlemi için aşağıdaki ifade yazılmıştır.

```
yas = 40;
```

C++ programlama dili bir değişkenin hem bildiriminin hem değer atamasının aynı satırda yapılabilmesini mümkün kılmaktadır. Dolayısıyla yukarıda iki ayrı satırda gerçekleştirilen bildirim ve atama işlemini tek satırda yapmak için yazılması gereken ifade aşağıdaki gibi olmalıdır.

```
int yas = 40;
```


Eğer daha sonra yas değişkenine tekrar ihtiyaç duyulursa veri türü bilgisi olmadan kullanılması gerekir. Örneğin yas değerini 10 artırmak istiyorsak aşağıdaki gibi aritmetik ifade yazılmalıdır.

```
yas = yas + 10;
```

Gerektiği durumlarda birden fazla aynı türe ait değişkenin aynı satırda bildirimi gerçekleştirilir. Örneğin tam sayı olarak tanımlanacak yas ve maas değişkenleri için aşağıdaki ifade uygun olacaktır.

```
int yas, maas;
```

Birden fazla değişkenin bildirimi yapılırken aynı zamanda da değer ataması yapılabilir.

```
int yas=40, maas=1000;
```

Eğer değişkenler farklı türden iseler ayrı satırlarda bildirim yapılması gerekmektedir.

```
int yas, maas;
```

```
float kilo, boy;
```

Değişkenlerin İsimlendirilmesi

Bir değişkene uygun ve anlamlı bir isim vermek önemlidir. Çünkü programcının yazacağı kodu sadece kendisi değil aynı zamanda aynı projede çalışan yazılım ekibindeki diğer programcılar da kullanabilecektir. Ayrıca bir yazılımın yaşam döngüsü içerisinde bulunan test aşamasında genellikle kodu yazan kişiden farklı biri test sürecini yürütebilmektedir. Belki ilerleyen zamanlarda yazılıma bakım yapılması gerekebilecek ve büyük ihtimalle yine kodu yazan tarafından değil başka programcılar tarafından bakım süreci yürütülecektir. Örneğin bir kişinin yaş bilgisini saklayacak bir değişkene yas ismini vermek uygundur. Ama x veya y gibi bir isim vermek uygun olmayacaktır.

Bir değişkene uygun bir isim verilebilmesi için bazı kurallara uymak gerekmektedir. Bu kurallar aşağıda belirtilmiştir.

- Değişken isimleri rakam ile başlayamaz.
- Değişkenlerin isimleri için “!, ?, {, }” gibi özel karakterler kullanılamaz.
- C++ programlama dili büyük ve küçük harf duyarlıdır. Örneğin var, VAR ve Var üç farklı değişken olarak algılanır.
- Değişken isimleri İngiliz alfabesinde bulunan büyük (A-Z) veya küçük (a-z) karakterler ya da rakamlar (0-9) kullanılarak oluşturulmalıdır.
- Değişken isminin uzunluğu en fazla 255 karakter olabilir ama ilk 32 karakter anlamlıdır. Yani ilk 32 karakteri aynı olan 32. karakterden sonraki herhangi bir karakteri farklı olan değişkenler aynı değişken olarak kabul edilir.
- Değişken ismi birden fazladan kelimeden oluşuyor ise kelimeler arasında boşluk bırakarak değişken ismi oluşturulamaz. (Örneğin “en büyük değer”). Bu durumda ya kelimeler boşluk olmadan birleştirilebilir (“enbuyukdeger”) ya da kelimeler arasında alt çizgi kullanılabilir (en_buyuk_deger). Genellikle kod okunabilirliği artırmak için deve (camel) gösterimi gibi bir standart tercih edilmektedir (“enBuyukDeger”). Deve gösteriminde değişken isminin ilk kelimesinin ilk harfi küçük sonraki kelimelerin ilk harfleri büyük kullanılması tercih edilir.

- C++ programlama dilinde anahtar kelime olarak tanımlanan kelimeler değişken ismi olarak kullanılamaz (Örneğin: for, while, case gibi kelimeler bir değişkene isim olarak tanımlanamazlar). C++ dilindeki tanımlı anahtar kelimeler Tablo 5.5’de verilmiştir.

Tablo 5.4’de değişken isimlendirme kurallarına göre doğru ve yanlış örnekler verilmiştir.

Tablo 5.4 Değişken İsimleri için Doğru ve Yanlış Örnekler.

Doğru	Yanlış
adSoyad	ad soyad
abc9	9abc
buyuk	büyük
en_buyuk	[enbuyuk]

Tablo 5.5 C++ Programlama Dilinde Tanımlı Anahtar Kelimeler.

asm	auto	break	case	char
const	continue	default	do	double
else	enum	extern	float	for
goto	if	inline	int	long
register	return	short	signed	sizeof
static	struct	switch	typedef	union
unsigned	void	volatile	while	
and	and_eq	bitand	bitor	bool
catch	class	compl	const_cast	delete
dynamic_cast	explicit	export	false	friend
mutable	namespace	new	not	not_eq
operator	or	or_eq	private	protected
public	reinterpret_cast	static_cast	template	this
throw	true	try	typeid	typename
using	virtual	wchar_t	xor	xor_eq
alignas	alignof	char16_t	char32_t	constexpr
decltype	noexcept	nullptr	static_assert	thread_local

3. İŞLEÇLER

İşleçler bir programın sonuç üretebilmesi için gerekli matematiksel veya mantıksal hesaplamaların yapılmasını sağlar. İşleçlerin bir sonuç üretebilmesi için bir veya daha fazla işlenen kullanılarak ifadelerin oluşturulması gerekir. Bir işleç, eğer sonuç üretebilmesi için tek bir işlenen yeterli ise birli (unary), iki işlenene gereksinim duyuyor ise ikili (binary) ve üç işlenen gerekiyor ise üçlü (ternary) olarak ifade edilir. Örneğin artırma ve eksiltme işleci birli bir işleçtir. Bir değişkene ait değerin 1 artırılması veya eksiltmesi amacıyla kullanır. Yani işlecin bir sonuç üretmesi için bir değişken yeterlidir. Diğer taraftan toplama çıkarma gibi aritmetik işleçler iki işlenen kullanarak sonuç üretirler. Şartlı işleç (? :) ise üç işlenene gereksinim duyar. İşleçleri kullanım amaçlarına göre sınıflandırılması aşağıdaki gibidir:

- Atama işleçleri
- Aritmetik işleçler
- İlişkisel işleçler
- Mantıksal işleçler
- Bit Tabanlı İşleçler

Bu bölümde atama ve aritmetik işleçler üzerinde durulacaktır. Diğer işleç türleri sonraki bölümlerde açıklanacaktır.

Atama İşleçleri

Bir değişkene bir değer atamak için kullanılır. Atama işleci "=" sembolü kullanılarak gerçekleştirilir. Atama işlecinin sağındaki değer işlecin solundaki değişkene atanır. İşlecin sağındaki değer sabit bir sayı olabileceği gibi aritmetik, mantıksal veya ilişkisel bir ifade de olabilir. Örneğin 50 yaşında olan bir kişinin yaşını tanımlı yas değişkenine atanması isteniyorsa aşağıdaki gibi bir ifade yazılmalıdır:

```
yas = 50;
```

Aritmetik İşleçler

Matematiksel ifadelerin hesaplanabilmesi için bir veya daha fazla işleç ve işlenen kullanarak bir programın sonuç üretebilmesi sağlanır. C++ programlama dilinde toplama (+), çıkarma (-), çarpma (*), bölme (/) gibi sıklıkla kullanılan aritmetik işleçler tanımlıdır. Bunlara ek olarak kalan (%), artırma (++) ve azaltma (--) gibi işleçler de mevcuttur.

Toplama ve Çıkarma İşleçleri

Toplama işleci iki işlenen değerini toplamak için kullanılır. Örneğin sayi1 ve sayi2 değişkenlerinin toplamının elde edilmesi ve sonucun sonuc değişkenine atanması isteniyorsa yazılması gereken aritmetik ifade aşağıdaki gibi olmalıdır:

```
sonuc = sayi1 + sayi2;
```

Benzer şekilde iki değişkenin farkının hesaplanması gerektiğinde yazılması gereken ifade aşağıdaki gibi olmalıdır:

```
sonuc = sayi1 - sayi2;
```

Çarpma ve Bölme İşleçleri

İki değişkene ait değerlerin çarpılması isteniyorsa yazılması gereken ifade aşağıdaki gibi olmalıdır:

```
sonuc = sayi1 * sayi2;
```

Bir değişkene ait değerin diğer bir değişken değerine bölünmesi durumunda elde edilen bölüm değerini başka bir değişkene atamak için aşağıdaki ifade yazılmalıdır:

```
sonuc = sayi1 / sayi2;
```

**DİKKAT EDELİM**

Bir tam sayı olarak tanımlanmış değişkenin yine bir tam sayı değişkenine bölümü sonucunda bir tam sayı değeri elde edilir. Kesirli değer elde edilmek isteniyorsa tür dönüşümü yapılması gerekir.

Örnek:

```
int a = 5;
cout << a << endl;
cout << --a << endl;
cout << a << endl;
```

Çıktı:

```
Fiyat / Kilo = 33
Fiyat / Kilo = 33.3333
```

Kalan işleci

Bazı durumlarda bir bölme işlemi sonucu ortaya çıkan kalan değerinin elde edilmesi istenebilir. Kalan işleci bir bölme işlemi sonunda ortaya çıkan değeri elde etmek için kullanılır. Bu işleç “%” sembolü ile kullanılır. Örneğin sayi1 değişkeninin sayi2 değişkenine bölünmesi durumunda elde edilecek kalan değerini sonuc değişkenine atanması gerekiyorsa aşağıdaki ifade yazılmalıdır:

```
sonuc = sayi1 % sayi2;
```

Eğer sayi1 değişkeninin değeri 12 ve sayi2 değişkeninin değeri 5 olması durumunda kalan 2 olacağı için sonuc değişkenine 2 değeri atanır.

Artırma ve Azaltma İşleçleri

Özellikle döngüler oluşturulurken kullanılan ifadelerde bir döngünün kaç defa çalışmasını kontrol eden sayaç değişkeninin problemin çözümüne bağlı olarak ya 1 artırılması ya da 1 azaltılması söz konusudur.

Dolayısıyla toplama veya çıkarma işleçlerini kullanarak istenen sonuçlar elde edilebilir. Bu durumda yazılması gereken ifade aşağıdaki gibi olmalıdır.

```
sayac = sayac +1
```

veya

```
sayac = sayac - 1
```

Bu ifadeler yazılımcılar tarafından sıklıkla kullanıldıkları için C++ programlama diline 1 artırma (++) ve 1 çıkarma (--) işleçleri eklenmiştir. Böylece bir değişkenin değeri 1 artırılmak isteniyorsa sayac++ veya 1 azaltılması isteniyorsa sayac-- ifadeleri tercih edilebilir.

Her iki işleç iki farklı şekilde kullanılmaktadır. Eğer işleç işlenenden önce kullanılırsa (++sayac veya --sayac şeklinde) önce artır veya azalt, işlenenden sonra kullanılırsa (sayac++ veya sayac-- şeklinde) sonra artır veya azalt olarak tanımlanır. Her iki kullanıma bağlı olarak aritmetik ifadelerin sonuçları farklılık gösterilmektedir. Bu yüzden her iki işleç dikkatli bir şekilde kullanılmalıdır. Kullanıma bağlı olarak farklı sonuç üretebilecek durumlar aşağıdaki örnek kod üzerinden anlatılacaktır.

Örnek:

```
int a = 5;
cout << a << endl;
cout << --a << endl;
cout << a << endl;
```

Çıktı:

```
5
4
4
```



İlk olarak işleç uygulanmadan önce, değişkenin ilk atanan değeri 5 yazdırılır. İşlecin uygulanmasıyla değişkenin değeri önce 1 azaltılır sonra ekrana yazdırılır. Bu yüzden ekrana 4 değeri yazdırılır. İşlecin kullanılmasında sonra değişken değeri tekrar kontrol edildiğinde 4 olduğu gözlemlenir.

Örnek:

```
int a = 5;
cout << a << endl;
cout << a-- << endl;
cout << a << endl;
```

Çıktı

5
5
4

Yukarıdaki kod bloğunda sonra azalt işleci kullanılmıştır. İlk olarak değişkenin ilk değeri olarak 5 yazdırılır. Sonrasında değişkenin değeri herhangi azaltma yapılmadan yazdırılır. Bu durumda ekrana 5 değeri yazdırılır. İşlecin kullanılmasında sonra değişken değeri tekrar kontrol edildiğinde 4 olduğu gözlemlenir.

Artırma ve azaltma işleçleri diğer aritmetik işleçlerle birlikte matematiksel ifadeler içerisinde beraber kullanılabilir.

Örnek:

```
int a=5, b=10;
cout<<(a++)*(--b)<<endl;
```

Çıktı:

45



DİKKAT EDELİM

Hatalı sonuçlar üretmemek veya kod okunabilirliğini artırmak için aritmetik ifadelerde uygun yerlerde parantezler kullanılmalıdır.

Bileşik Atama İşleçleri

Programlarda genellikle aynı değişkenin değeri sabit veya başka değişkenler kullanılarak güncellenmesi gerekir. Örneğin tam sayı tanımlı var değişkenin değeri 5 olsun ve değişkenin değerini 3 artırmak için yazılması gereken ifade “var = var + 3;” olacaktır. Görüldüğü gibi ifade

içerisinde iki defa değişken adının tekrar edilmesi söz konusudur. Bu ve buna benzer ifadeleri kısaltmak için bileşik atama işleçleri tanımlanmıştır. Aşağıda verilen tabloda görüleceği üzere toplama işlecine ek olarak çıkarma, çarpma, bölme ve kalan bileşik atama işleçleri de tanımlıdır.

Tablo 5.6 Bileşik Atama İşleçleri ve Kullanımları.

İşleç	Örnek	Eşdeğeri
+=	var +=3;	var = var + 3;
-=	var -=3;	var = var - 3;
*=	var *=3;	var = var * 3;
/=	var /=3;	var = var / 3;
%=	var %=3;	var = var % 3;

Aritmetik İşleçlerin Öncelik Sıraları

Bir matematiksel ifade içerisinde birden fazla aritmetik işleç kullanılarak daha karmaşık ifadelerin hesaplanması sağlanabilir. Böyle ifadelerin doğru şekilde sonuç üretebilmesi için aritmetik işleçlerin öncelik sırası bilinmelidir. Değilse hatalı sonuç almak mümkündür. Örneğin, $a + b * 3 - (c / d)$ ifadesinde toplama, çarpma, çıkarma ve bölme işleçleri aynı ifade içerisinde kullanılmıştır. İlk hangi işlecin hesaplanması gerektiği, aşağıda verilen kurallara göre belirlenmektedir.

Aritmetik İşleçlerin Öncelik Kuralları

- Eğer ifadede parantez varsa öncelikli olarak parantez içerisindeki işleçler hesaplanır. Parantezin içerisinde de başka parantezler var ise en içteki parantezden başlayarak hesaplamalar yapılır.
- Daha sonra çarpma, bölme ve kalan işleçleri hesaplanır. Eğer bir ifadede birden fazla çarpma,

bölme veya kalan işleci var ise hesaplama soldan başlanarak sağa doğru devam edilir. Çarpma, bölme ve kalan işleçleri aynı seviye önceliği sahiptir.

•En son olarak toplama ve çıkarma işleçleri uygulanır. Eğer ifade içerisinde birden fazla toplama veya çıkarma işleci var ise, ifadenin hesaplanmasına soldan başlanacaktır. Toplama ve çıkarma işleçleri aynı seviye önceliğe sahiptir.

Örnek: $y = 1 * 2 + (3 + 4) - 5$ ifadesinin hesaplanmasında sırasıyla şu işleçler uygulanacaktır. Öncelikle parantez içi hesap edilerek ifade $y = 1 * 2 + 7 - 5$ halini alacaktır. Daha sonra kalan işleçler arasında çarpma işleci öncelikli olduğu için çarpma işlemi uygulanacaktır. İfade $y = 2 + 7 - 5$ halini alacaktır. Kalan toplama ve çıkarma işleçleri aynı seviyede oldukları için soldan başlanarak hesaplama devam edilecektir. Kalan ifade $y = 9 - 5$ olacaktır. En son olarak çıkarma işlemi uygulanarak sonuç $y = 4$ olarak bulunacaktır.



DİKKAT EDELİM

Doğru sonuçlar elde edebilmek için işleçlerin öncelik kuralları bilinmelidir.

BÖLÜM ÖZETİ

Bu bölümde temel veri türleri, değişkenler, atama ve aritmetik işlemlerden bahsedilmiştir. Bir bilgisayar programı hafızada sakladığı çeşitli verileri işleyerek birtakım sonuçlar üretir. Birçok farklı alanda programlara ihtiyacımız olduğu düşünülünce çok farklı veri türlerine gereksinim duyulmaktadır. C++ programlama dilinde tanımlı üç farklı temel veri türü vardır. Bu türler sayısal, boolean ve karakter veri türleridir. Sayısal veri türleri kendi içinde tam ve kesirli sayılar olmak üzere ikiye ayrılmaktadır. Bir bilgisayar programı programcının tanımladığı değişkenleri işleyerek istenilen sonuca ulaşır. Değişkenlere uygun isim vermek yazılan kodun okunabilirliğini artırır. Çünkü yazılım yaşam döngüsünde bulunan süreçlerin gereği olarak bir programcının yazmış olduğu kodu başka programcılarında kullanması veya test etmesi gerekmektedir. Ayrıca C++ programlama dilinde bir değişkene isim vermek için bazı kurallar vardır. Değişkenlere bu kurallara uygun isimler vermek derleme hatalarının önüne geçecektir. Tanımlı değişkenler aritmetik, ilişkisel veya mantıksal işlemler içerisinde kullanılarak daha karmaşık ifadelerin yazılması mümkün hale gelmektedir. İşlemler konusu oldukça geniş bir kavramdır. Bu bölümde atama ve aritmetik işlemlerden bahsedildi. Atama işlemi bir değişkene bir değer atamak için kullanılır. Buna ek olarak toplama, çıkarma, bölme, çarpma ve kalan gibi aritmetik işlemler anlatılmıştır. Bir ifade içerisinde birden fazla işlem kullanarak daha karmaşık ifadelerin elde edilmesi mümkündür. Bu durumda hangi işlemin önce hesaplanması gerektiği işlem öncelik kurallarına göre belirlenir. Bu kuralları bilmek program içerisinde yapılabilecek hataları en aza indirgeyecektir.

GÖZDEN GEÇİRELİM

1. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
int a = 2 * 5 - 3 + 4 % 2;
```

```
cout << a << endl;
```

- a. 3
- b. 4
- c. 5
- d. 6
- e. 7

4. Aşağıdakilerden hangisi C++ programlama dilinde tanımlı bir tam sayı veri türüdür?

- a. int
- b. float
- c. double
- d. char
- e. string

2. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
int a = 4, b = 4;
```

```
cout << (++a) * (b--) << endl;
```

- a. 12
- b. 15
- c. 16
- d. 20
- e. 24

5. Aşağıdakilerden hangisi C++ programlama dilinde tanımlı bir kesirli sayı veri türüdür?

- a. short
- b. float
- c. int
- d. long
- e. char

3. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
int a= 4, b = 4;
```

```
++a;
```

```
b++;
```

```
int c = a * b;
```

```
cout << c << endl;
```

- a. 16
- b. 20
- c. 24
- d. 25
- e. 30

6. Aşağıdakilerden hangisi C++ programlama dilinde atama işleci olarak kullanılır?

- a. =
- b. *
- c. %
- d. /
- e. ==

7. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
int a = 10;
a += 2;
cout << a << endl;
```

a. 5
b. 8
c. 10
d. 12
e. 20

9. Aşağıdakilerden hangisi C++ programlama dilinde tanımlanabilecek bir değişken ismi değildir?

- a. boy
b. kilo
c. ad
d. soyad
e. enKüçük

8. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
int sayi1=13, sayi2=3;
int sonuc = sayi1 % sayi2;
cout<<sonuc<<endl;
```

a. 0
b. 1
c. 2
d. 3
e. 4

10. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
enum renkler {siyah, beyaz, gri};
renkler araba = beyaz;
cout<<araba<<endl;
```

a. 0
b. 1
c. 2
d. 5
e. 10

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

Kaynakça

- Deitel, P., Deitel, H. (2017). C++ How to Program. 10. Baskı. Pearson. Essex, İngiltere.
- Dikici, M. (2017). C++ Programlama Dili: Yapısal Programlama, Nesnelerle Programlama ve Jenerik Programlama. Seçkin Yayıncılık. İstanbul, Türkiye.
- Çobanoğlu, B. (2021). C/C++ Programlama Eğitim Kitabı. Kodlab. İstanbul, Türkiye.
- Demir, S., Eryiğit, R., Ar, Y., Tuğrul, B., Gök, O., Kul, S., ... Yağanoğlu, M. (2019). Programlama Temelleri. Erzurum: Atatürk Üniversitesi Açıköğretim Fakültesi.
- Kaleli, C., Bilge, A., Ergin, S., & Şora Günel, E. (2016). Bilgisayar ve Programlamaya Giriş. Eskişehir: Anadolu Üniversitesi.

Faydalı Kaynaklar ve İnternet Kaynakları

<https://en.cppreference.com/w/>
<https://www.w3schools.com/cpp/default.asp>

BÖLÜM 6

İLİŞKİSEL VE MANTIKSAL İŞLEÇLER VE SEÇMELİ KONTROL YAPILARI

ANAHTAR KAVRAMLAR

İlişkisel İşleçler, Mantıksal İşleçler, Kontrol Yapıları, If Kontrol Yapısı, If...else Kontrol Yapısı, Switch Kontrol Yapısı, Break Anahtar Kelimesi, Koşullu İşleç

ÖĞRENME HEDEFLERİ

- 1 İlişkisel işleçleri nasıl kullanacağını bilir
- 2 Mantıksal işleçleri nasıl kullanacağını bilir
- 3 İlişkisel ve mantıksal işleçlerin öncelik sıralamasını bilir
- 4 Program akışının if, if...else gibi yapılarla nasıl kontrol edilebileceğini bilir
- 5 Aritmetik işleçlerin öncelik sıralamasını bilir

GİRİŞ

C++ programlama dilinde ifadeler, farklı bir biçimde yönlendirilmediği sürece ardışık olarak çalıştırılmaktadır. Diğer bir ifadeyle, program önce kod bloğundaki ilk ifadeyi, sonrasında sonuncaya kadar sırasıyla diğer ifadeleri çalıştırmaktadır. Bununla birlikte C++ programlama dilinde akış yalnızca ardışık çalışma diye adlandırılan bu yöntemle sınırlandırılmamıştır. Çalıştırılması esnasında program, akışı kontrol etmeye imkan veren farklı kontrol yapıları yardımıyla çatalanabilir, belirli kod bloklarını tekrar uygulayabilir ya da farklı seçeneklerden birini çalıştırabilir. C++ programlama dilinin bu esnek yapısı etkin ve kullanışlı kod blokları oluşturulmasına imkan vermektedir. C++ programlama dilinde ardışık, seçmeli ve yinelemeli olmak üzere 3 farklı kontrol yapısı bulunmaktadır. Bu bölümde seçmeli kontrol yapılarından bahsedilecektir. Öncesinde seçmeli kontrol yapılarındaki koşulları oluşturabilmemiz için gerekli olan ilişkisel ve mantıksal işleçlerden bahsedilecektir.



1. İLİŞKİSEL İŞLEÇLER

İlişkisel işleçler, C++ programlama dilinde iki değişken ya da iki değer arasındaki ilişkiyi tanımlayan ya da koşullu ifadelerin bir parçası olarak bu ilişkiyi sınavan yapılardır. İlişkisel işleçler üzerinden oluşturulmuş ilişkisel ifadelerin sonucu “true” (doğru) ya da “false” (yanlış) değeri alan boolean veri tipinde olmaktadır: eğer ifadedeki değerler ya da değişkenler arasında ilişkisel işlecin tanımladığı gibi bir ilişki varsa ifade true boolean değeri almakta, yoksa false boolean değeri almaktadır.

```
int sayi_1 = 2, sayi_2 = 3;
```

```
bool buyuktur = (sayi_1 > sayi_2);
```

Örneğin yukarıdaki kod bloğunda sayi_1 ve sayi_2 birer tam sayı değişkeni olarak tanımlanmış ve atama işleci kullanılarak bu değişkenlere sırasıyla 2 ve 3 değerleri atanmıştır. Kod bloğunda ‘büyüktür’ ilişkisel işleci kullanılarak oluşturulmuş ‘sayi_1 > sayi_2’ ilişkisel ifadesi yoluyla sayi_1 ve sayi_2 değişkenlerine atanmış değerler kıyaslanmakta ve değerler arasında ifadede belirtildiği gibi bir ilişki olmadığı için, diğer bir ifadeyle sayi_1’e atanmış 2 değeri sayi_2’ye atanmış 3 değerinden büyük olmadığı için false boolean sonuç değeri buyuktur değişkenine atanmaktadır.

Yukarıdaki kullanımının yanı sıra, ilişkisel işleçler aşağıdaki kod bloğunda olduğu gibi koşullu ifadelerin bir parçası olarak da kullanılmaktadır. Kod bloğunda yine yukarıdakine benzer olarak sayi_1 ve sayi_2 birer tam sayı değişkeni olarak tanımlanmış ve atama işleci kullanılarak bu değişkenlerin ilkinde 2 değeri ve ikincisine 3 değeri atanmıştır. Burada ‘eşit değildir’ ilişkisel işleci if koşullu ifadesinin bir parçası olarak sayi_1 ve sayi_2 değişkenlerinin eşit olup olmadığını sınamaktadır. sayi_1 ve sayi_2 değişkenlerine atanmış değerler farklı olduğu için ‘sayi_1 != sayi_2’ ilişkisel ifadesi true değer alacak ve kod çalıştırıldığında ekrana “x ve y esit degildir” ifadesi yazılacaktır.

```
int sayi_1 = 2, sayi_2 = 3;
```

```
if (sayi_1 != sayi_2){
```

```
    cout << “sayi_1 ve sayi_2 esit degildir” <<
```

```
endl;
```

```
}
```

Aşağıdaki tabloda da belirtildiği gibi; eşittir, eşit değildir, küçüktür, küçük eşittir, büyüktür ve büyük eşittir olmak üzere 6 adet ilişkisel işleç tanımlanmaktadır.

Tablo 6.1 C++ Programlama Dilinde Tanımlı İlişkisel İşleçler.

İlişkisel İşleç	Sembolü	İlişkisel İfade	İlişkisel İfadenin Anlamı
eşittir	==	x == y	x y’ye eşittir
eşit değildir	!=	x != y	x y’ye eşit değildir
büyüktür	>	x > y	x y’den büyüktür
büyük eşittir	>=	x >= y	x y’den büyük eşittir
küçüktür	<	x < y	x y’den küçüktür
küçük eşittir	<=	x <= y	x y’den küçük eşittir

Diğer birçok programlama dilinde olduğu gibi C++ programlama dilinde de iki değerin birbirine eşit olup olmadığını belirlemek için kullanılan eşittir (==) işleci ile tanımlı değişkenlere değer atanırken kullanılan atama işleci (=) sık sık karıştırılmaktadır. Program tasarlanırken bu durum dikkate alınmalı ve ilgili işleçler doğru yerde kullanılmalıdır.



DİKKAT EDELİM

Eşittir işleci ile atama işleci birbirine karıştırılmamalı ve doğru işlecin doğru yerde kullanımına dikkat edilmelidir.

Örneğin aşağıdaki kod bloğunda `sayi_1`, `sayi_2` ve `sayi_3` birer tam sayı değişkeni olarak tanımlanmış ve atama işleci kullanılarak bu değişkenlere sırasıyla 3, 4 ve 5 değerleri atanmıştır. İlk koşullu ifadede 'eşittir' ilişkisel işleci `sayi_1` ve `sayi_3` değişkenlerinin eşit olup olmadığını sorgulamaktadır. `sayi_1` ve `sayi_3` değişkenlerine atanmış değerler farklı olduğu için '`sayi_1 == sayi_3`' ilişkisel ifadesi false boolean değeri alacak ve ilgili koşullu ifadenin gövdesindeki kod çalıştırılmayacaktır. İkinci koşullu ifadede ise 'küçüktür' ilişkisel işleci ifadedeki ilk değişkenin ikinci değişkenden küçük olup olmadığını sorgulamaktadır. `sayi_1` değişkenine atanmış 3 değeri `sayi_2` değişkenine atanmış 4 değerinden küçük olduğu için '`sayi_1 < sayi_2`' ilişkisel ifadesi true boolean değeri alacak ve ilgili koşullu ifadenin gövdesindeki kod çalıştırılarak ekrana "`sayi_1 sayi_2'den kucuktur`" cümlesi yazılacaktır. Son olarak üçüncü koşullu ifadede 'büyük eşittir' ilişkisel işleci ifadedeki ilk değişkenin ikinci değişkenden büyük eşit olup olmadığını sorgulamaktadır. `sayi_3` değişkenine atanmış 5 değeri `sayi_2` değişkenine atanmış 4 değerinden büyük eşit olduğu için '`sayi_3 >= sayi_2`' ilişkisel ifadesi true değer alacak ve ilgili koşullu ifadenin gövdesindeki kod çalıştırılarak ekrana "`sayi_3 sayi_1'den buyuktur`" cümlesi yazılacaktır.

```
int sayi_1 = 3, sayi_2 = 4, sayi_3 = 5;
if (sayi_1 == sayi_3){
    cout << "sayi_1 ve sayi_3 esittir" <<endl;
}
if (sayi_1 < sayi_2){
    cout << "sayi_1 sayi_2'den kucuktur" <<endl;
}
if (sayi_3 >= sayi_2){
    cout << "sayi_3 sayi_1'den buyuktur" <<endl;
}
```

Daha önceki bölümlerde de belirtildiği gibi, C++ programlama dilinde tasarlanan kodun düzenli ve doğru çalışabilmesi için kullanılan işleçler için farklı öncelik sıraları belirlenmiştir. Aritmetik işleçler ilişkisel işleçlere göre daha yüksek önceliğe sahipken, atama işleci ilişkisel işleçlere nispeten daha düşük önceliğe sahiptir. Birden fazla ilişkisel işlecin kullanıldığı ilişkisel ifadelerde işleçlerin kendi içeri-

sinde öncelik sıraları soldan sağa doğru olmaktadır. Doğru ve etkili çalışan bir kod tasarlayabilmek için işleçlerin öncelik sıralarının bilinmesi gerekmektedir. Örneğin aşağıdaki kod bloğunda `sayi_1` tamsayı değişkenine atanmış 4 değeri `sayi_2` tamsayı değişkenine atanmış 5 değerinden küçük olmasına rağmen, diğer bir ifadeyle '`sayi_1 > sayi_2`' ilişkisel ifadesi doğru olmamasına rağmen; aritmetik işleçlerin işlem önceliği olduğu için, '`2*sayi_1 > sayi_2 + 2`' ilişkisel ifadesinde önce aritmetik işlemler uygulanacak, ve aritmetik işlemlerden sonra 'büyüktür' işlecinin solundaki değer ($2*4 = 8$) sağındaki değerden ($5 + 2 = 7$) büyük olacağından '`2*sayi_1 > sayi_2 + 2`' ilişkisel ifadesi true değer alacaktır. Sonuç olarak ilgili koşullu ifadenin gövdesindeki kod çalıştırılarak ekrana "parantez kullanımı ifadelerin daha anlaşılır olmasını sağlar" cümlesi yazılacaktır. Her ne kadar aritmetik işleçlerin ilişkisel işleçlerden daha yüksek işlem önceliğine sahip olduğu bilinse de, bunun parantez yoluyla vurgulanması; yani '`2*sayi_1 > sayi_2 + 2`' ilişkisel ifadesinin parantezler yardımıyla '`(2*sayi_1) > (sayi_2 + 2)`' şeklinde yazılması, kodun daha anlaşılır ve kolay takip edilmesini sağlayacaktır.

```
int sayi_1 = 4, sayi_2 = 5;
if (2*sayi_1 > sayi_2 + 2){
    cout << "parantez kullanımı ifadelerin daha anlaşılır olmasını sağlar" << endl;
}
```



DİKKAT EDELİM

Doğru şekilde çalışan kod blokları oluşturabilmek için işleçlerin öncelik sıraları bilinmelidir. Bu noktada parantez kullanımı hangi işleçlerin önce uygulanması gerektiğini vurgulayacak, daha anlaşılır ve takip edilmesi kolay kod blokları oluşturabilmemize imkan sağlayacaktır.

2. MANTIKSAL İŞLEÇLER

İlişkisel işleçler '`not > = 90`' veya '`sıcaklık < 21`' gibi tek bir durumun sınındığı koşullu ifadeler oluşturmak için yeterlidir. Ancak "sıcaklık 30'dan küçük ve nem %80'den fazlaysa" ya da "vize notu 90 ve final notu 80 ise" gibi birden fazla durumun sınındığı daha karmaşık koşullu ifadelerin oluşturulabilmesi mantıksal işleçlerin kullanımıyla mümkün olmaktadır. Aşağıdaki tabloda da belirtildiği gibi C++ programlama dilinde DEĞİL, VE ve VEYA olmak üzere 3 adet mantıksal işleç tanımlanmaktadır.

Tablo 6.2 C ++ Programlama Dilinde Tanımlı Mantıksal İşleçler.

İşleç	Sembolü	Mantıksal İfade	Mantıksal İfadenin Anlamı
DEĞİL	!	$!(x < y)$	x y'den küçük değildir
VE	&&	$(x == y) \&\& (y < z)$	x y'ye eşit ve y z'den küçüktür
VEYA		$(x > y) (y >= z)$	x y'den büyük veya y z'den büyük eşittir

```
int sayi_1 = 12, sayi_2;
if (!(sayi_1 < 10)){
    sayi_2 = 10;
}
```

Mantıksal değil işleci ile oluşturulmuş ifadenin boolean değeri, ifadedeki durum doğru olduğunda false, yanlış olduğunda true olacaktır. Örneğin yukarıdaki kod bloğunda sayi_1 ve sayi_2 birer tam sayı değişkeni olarak tanımlanmış ve atama işleci kullanılarak bu değişkenlerden ilkinde 12 değeri atanmıştır. Kod bloğunda ‘küçüktür’ ilişkisel işleci kullanılarak oluşturulmuş ‘sayi_1 < 10’ ilişkisel ifadesi sayi_1’in değeri 10’dan küçük olmadığı için false boolean değeri alacaktır. ‘sayi_1 < 10’ ilişkisel ifadesi false boolean değeri aldığı için, mantıksal DEĞİL işleci ile oluşturulmuş ‘!(sayi_1 < 10)’ ifadesi true boolean değeri alacaktır ve sonuç olarak koşullu ifadenin gövdesindeki kod çalıştırılarak sayi_2 değişkenine 10 değeri atanacaktır. Aşağıdaki tabloda herhangi bir durumun alacağı farklı boolean değerlerine karşılık mantıksal DEĞİL işlecinin alacağı boolean değerleri verilmiştir.

Tablo 6.3 Mantıksal DEĞİL İşlecinin Boolean Değerleri Tablosu.

Durum	! Durum
true	false
false	true

Mantıksal VE işleci ile oluşturulmuş ifadenin boolean değeri, ifadedeki durumlardan her ikisi de doğru olduğunda true, diğer bütün durumlarda false olmaktadır. Örneğin aşağıdaki kod bloğunda sayi_1, sayi_2 ve sayi_3 birer tam sayı değişkeni olarak tanımlanmış ve atama işleci kullanılarak bu değişkenlere sırasıyla 1, 2 ve 3 değerleri atanmıştır. Değişkenlerin aldığı 1, 2 ve 3 değerleri göz önünde bulundurulduğunda, kod bloğundaki ilk koşullu ifadede sırasıyla ‘büyüktür’ ve ‘büyük eşittir’ ilişkisel işleçleri kullanılarak oluşturulmuş ‘sayi_3 > sayi_2’ ve ‘sayi_2 > = sayi_1’ ilişkisel ifadelerinin her ikisi de true boolean değeri alacaktır. Mantıksal VE işlecinin yukarıda verdiğimiz tanımı düşünüldüğünde, ifadedeki durumların her ikisi de true boolean değeri aldığından ‘(sayi_3 > sayi_2) && (sayi_2 > = sayi_1)’ ifadesinin kendisi de true değer alacaktır. Ve sonuç olarak koşullu ifadenin gövdesindeki kod çalıştırılarak ekrana “sayi_3 sayi_1’den buyuktur” cümlesi yazılacaktır.

```
int sayi_1 = 1, sayi_2 = 2, sayi_3 = 3;
if ((sayi_3 > sayi_2) && (sayi_2 > = sayi_1)){
    cout << "sayi_3 sayi_1'den buyuktur" <<endl;
}
if ((sayi_3 > sayi_2) && (sayi_3 < sayi_1)){
    cout << "sayi_1 sayi_3'den buyuktur" <<endl;
}
```

Diğer yandan yukarıdaki kod bloğundaki ikinci koşullu ifadede; ‘büyüktür’ ilişkisel işleci kullanılarak oluşturulmuş ‘sayi_3 > sayi_2’ ifadesi true ama ‘küçüktür’ ilişkisel işleci kullanılarak oluşturulmuş ‘sayi_3 < sayi_1’ ifadesi false değer alacağından, ‘(sayi_3 > sayi_2) && (sayi_3 < sayi_1)’ mantıksal ifadesi de false boolean değeri alacak ve dolayısıyla ilgili koşullu ifadenin gövdesindeki kod çalıştırılmayacaktır. Aşağıdaki tabloda herhangi iki durumun alacağı farklı bool değerlerine karşılık mantıksal ve işlecinin alacağı boolean değerleri verilmiştir.

Tablo 6.3 Mantıksal VE İşlecinin Boolean Değerleri Tablosu.

Durum_1	Durum_2	Durum_1 Durum_2
true	true	true
true	false	true
false	true	true
false	false	false

Mantıksal VEYA işleci ile oluşturulmuş ifadenin boolean değeri, ifadedeki durumlardan her ikisi de yanlış olduğunda false, diğer bütün durumlarda true olmaktadır. Örneğin aşağıdaki kod bloğunda sayi_1, sayi_2 ve sayi_3 birer tam sayı değişkeni olarak tanımlanmış ve atama işleci kullanılarak bu değişkenlere sırasıyla 1, 2 ve 3 değerleri atanmıştır. Değişkenlerin aldığı 1, 2 ve 3 değerleri göz önünde bulundurulduğunda, kod bloğundaki ilk koşullu ifadedeki sırasıyla ‘eşittir’ ve ‘büyüktür’ ilişkisel işleçleri kullanılarak oluşturulmuş ‘sayi_3 == sayi_2’ ve ‘sayi_2 > sayi_1’ ilişkisel ifadelerinin ilki false ve ikincisi true boolean değeri alacaktır. Mantıksal VEYA işlecinin yukarıda verdiğimiz tanımı düşünüldüğünde, ifadedeki durumların biri true değer aldığından ‘(sayi_3 == sayi_2) (sayi_2 > sayi_1)’ ifadesi de true değer alacaktır. Ve sonuç olarak koşullu ifadenin gövdesindeki kod çalıştırılarak ekrana “sayi_3 sayi_1’den buyuktur” cümlesi yazılacaktır.

```
int sayi_1 = 1, sayi_2 = 2, sayi_3 = 3;
if ((sayi_3 == sayi_2) || (sayi_2 > sayi_1)){
    cout << "sayi_3 sayi_1'den buyuktur" <<endl;
}
if ((sayi_3 < sayi_2) || (sayi_3 == sayi_1)){
    cout << "sayi_1 sayi_3'den buyuktur" <<endl;
}
```

Diğer yandan yukarıdaki kod bloğundaki ikinci koşullu ifadede; ‘küçüktür’ ilişkisel işleci kullanılarak oluşturulmuş ‘sayi_3 < sayi_2’ ifadesi ve ‘eşittir’ ilişkisel işleci kullanılarak oluşturulmuş ‘sayi_3 == sayi_1’ ifadesi false değer alacağından, ‘(sayi_3 < sayi_2) || (sayi_3 == sayi_1)’ mantıksal ifadesi de

false boolean değeri alacak ve dolayısıyla ilgili koşullu ifadenin gövdesindeki kod çalıştırılmayacaktır. Aşağıdaki tabloda herhangi iki durumun alacağı farklı bool değerlerine karşılık mantıksal veya işlecinin alacağı boolean değerleri verilmiştir.

Tablo 6.4 Mantıksal VEYA İşlecinin Boolean Değerleri Tablosu.

Durum_1	Durum_2	Durum_1 && Durum_2
true	true	true
true	false	false
false	true	false
false	false	false

İlişkisel işleçler mantıksal işleçlere göre daha yüksek önceliğe sahipken, atama işleci mantıksal işleçlere nispeten daha düşük önceliğe sahiptir. Birden fazla mantıksal işlecin kullanıldığı ilişkisel ifadelerde işleçlerin kendi içerisinde öncelik sıraları soldan sağa doğru olmaktadır. Örneğin aşağıdaki kod bloğunda koşullu ifadede ilişkisel işleçlerin mantıksal işleçlere nispeten işlem önceliği olduğu için, program önce iki ilişkisel ifadenin boolean değerini hesaplayacak ve sonrasında elde edilen boolean değerlerine mantıksal VE işlecinin uygulayacaktır. İlk ilişkisel ifadede; aritmetik işleçlerin işlem önceliği olduğu için önce $+$ işlemi uygulanacak, ve sonrasında ‘eşittir’ işlecinin solundaki değer ($2 + 1 = 3$) sağındaki değere ($\text{sayi_2} = 3$) eşit olduğu için ‘ $\text{sayi_1} + 1 == \text{sayi_2}$ ’ ilişkisel ifadesi true değer alacaktır. İkinci ilişkisel ifadede de benzer şekilde; ‘küçüktür’ işlecinin solundaki değer ($\text{sayi_2} = 3$) sağındaki değerden ($\text{sayi_3} = 5$) küçük olduğu için ‘ $\text{sayi_2} < \text{sayi_3}$ ’ ilişkisel ifadesi true boolean değer alacaktır. Koşullu ifadedeki her iki ilişkisel ifade de true değer aldığından mantıksal VE işlecinin sonucu da true boolean değeri alacak ve ilgili koşullu ifadenin gövdesindeki kod çalıştırılarak ekrana “aritmetik işlemler mantıksal işlemlere nispeten daha yüksek önceliğe sahiptir” cümlesi yazılacaktır.

```
int sayi_1 = 2, sayi_2 = 3, sayi_3 = 5;
if ((sayi_1 + 1 == sayi_2) && (sayi_2 < sayi_3)){
    cout << "aritmetik işlemler mantıksal işlemlere nispeten daha yüksek önceliğe sahiptir" << endl;
}
```



DİKKAT EDELİM

C++ programlama dilinde mantıksal ifadelerde kısa devre değerlendirme yöntemi uygulandığı unutulmamalı, ifadenin ikinci kısmına kod için kritik sayılabilecek işlemler konulmamalıdır.

Mantıksal VE işlecinin doğruluk değer tablosu incelendiğinde durumlardan ilki false olduğunda ikinci durum ne olursa olsun mantıksal ifadenin boolean değerinin false olduğu gözlemlenecektir. Yine benzer şekilde mantıksal VEYA işlecinin doğruluk değer tablosu incelendiğinde durumlardan ilki true olduğunda ikinci durum ne olursa olsun mantıksal ifadenin boolean değerinin true olduğu

gözlemlenecektir. Bu gerçekten hareketle C++ programlama dilinde mantıksal VE ve VEYA işleçleriyle oluşturulmuş mantıksal ifadelerde ilk durum ifadenin doğruluk değeri için yeterli ise program kısa devre değerlendirme adı verilen bir yöntemle ikinci durumu kontrol etmeden direk sonucu dönecektir. Diğer bir ifadeyle, mantıksal VE işleci ile oluşturulmuş ifadelerde, ilk durum false olduğunda program ikinci kısmı kontrol etmeksizin false boolean değeri; benzer şekilde mantıksal VEYA işleci ile oluşturulmuş ifadelerde, ilk durum true olduğunda program ikinci kısmı kontrol etmeksizin true boolean değeri dönecektir.



DİKKAT EDELİM

Mantıksal işleçlerin yukarıda verilen tanımları ışığında
!(true && false) || (false && true) ifadesinin alacağı boolean değerini belirleyiniz.

3. SEÇMELİ KONTROL YAPILARI

Seçmeli kontrol yapıları programın akışını belli koşullar üzerinden çatallandırmaya imkan veren yapılar. C++ programlama dilinde if (tek seçim), if...else (ikili seçim) ve switch (çoklu seçim) olmak üzere üç farklı seçmeli kontrol yapısı tanımlanmıştır.

4. TEK SEÇİM KONTROL YAPISI

If kontrol yapısı aşağıdaki formda kullanılmaktadır. Eğer koşul kısmındaki ifade doğru ise gövdedeki kod çalıştırılmakta, aksi durumda gövdedeki kod çalıştırılmamaktadır.

```
if (koşul) {
    kod satırı_1
    kod satırı_2
    ...
}
```

Aşağıda da gösterildiği gibi tek bir kod satırından oluşuyorsa gövde kısmı parantez kullanılmaksızın da oluşturulabilmektedir.

```
if (koşul)
    kod satırı
```

Örneğin aşağıdaki kod bloğunda sayi_1 tamsayı değişkeni pozitif bir değer almışsa, if kontrol yapısının koşul kısmındaki 'sayi_1 > 0' ilişkisel ifadesi true değer alacak ve gövde kısmındaki kod çalıştırılarak ekrana "sayi_1 pozitif bir tamsayıdır" yazdırılacaktır. Aksi durumda, yani sayi_1 tamsayı değişkeni pozitif bir değer almamışsa, 'sayi_1 > 0' ilişkisel ifadesi false değer alacağından koşul sağlanmayacak ve gövde kısmındaki kod çalıştırılmayacaktır.

```
int sayi_1;
if (sayi_1 > 0){
    cout << "sayi_1 pozitif bir tamsayıdır" <<endl;
}
```

5. İKİLİ SEÇİM KONTROL YAPISI

If kontrol yapısında gövdedeki kod ancak koşul doğru olduğunda çalıştırılır. Koşul yanlış olduğunda ise program yapının gövdesindeki kodu pas geçerek sonraki kod bloğuyla yoluna devam eder. If yapısından farklı olarak if...else kontrol yapısı ise koşul yanlış olduğunda da belirli bir kod bloğunun çalıştırılabilmesine imkan vermektedir. If...else kontrol yapısı aşağıdaki formda kullanılmaktadır: koşul kısmındaki ifade doğru ise kod_parçası_1 çalıştırılmakta, doğru değilse kod_parçası_2 çalıştırılmaktadır.

```
if (koşul) {
    kod_parçası_1
}
else {
    kod_parçası_2
}
```

Örneğin aşağıdaki kod bloğunda eğer basari_notu tamsayı değişkeninin aldığı değer 60'a eşit veya daha büyükse if...else kontrol yapısının koşul kısmındaki 'basari_notu > = 60' ilişkisel ifadesi true boolean değeri alacak ve gövde kısmındaki kod çalıştırılarak ekrana "dersi gectin" yazdırılacaktır. Aksi durumda, yani başarı_notu tamsayı değişkeni 60'tan daha küçük değer almışsa, 'başarı_notu > = 60' ilişkisel ifadesi false boolean değeri alacağından koşul sağlanmayacak ve alternatif gövdedeki kod çalıştırılarak ekrana "derste basarisiz oldun" ve "bir dahaki sefere daha cok calismalisin" cümleleri yazdırılacaktır.

```
int basari_notu;
if (basari_notu > = 60){
    cout << "dersi gectin" << endl;
}
else {
    cout << "derste basarisiz oldun" << endl;
    cout << "bir dahaki sefere daha cok calismalisin" << endl;
}
```

C++ programlama dilinde koşullu işleç adı verilen ve if...else kontrol yapısı ile aynı işlevi gören daha basit bir işleç de tanımlanmıştır. Koşullu işleç genel haliyle aşağıdaki formda kullanılmaktadır:

(koşullu ifade ? kod_1 : kod_2)

koşullu ifade doğru ise kod_1, yanlış ise kod_2 çalıştırılmaktadır. If...else yapısı ile oluşturulan yukarıdaki kod bloğunun koşullu işleç ile basit şekilde yazımı aşağıdaki gibi olmaktadır:

```
cout << (basari_notu > = 60 ? "dersi gectin" : "derste basarisiz oldun");
```

6. ÇOKLU SEÇİM KONTROL YAPISI

Yukarıda da ifade edildiği gibi if...else kontrol yapısı tek bir koşulu kontrol etmektedir. Eğer koşul doğruysa program yapının gövdesindeki kodu, yoksa alternatif kodu çalıştırmaktadır. Bazı durumlarda birden fazla koşulun test edilmesi ve duruma göre alternatif kod satırlarından birinin seçilmesi gerekebilir. If...else yapısının daha kapsamlısı olarak değerlendirilebilecek if...else if kontrol yapısı birden çok

koşulun test edilebilmesine imkan vermektedir. Yukarıdaki örnekte if...else yapısı kullanılarak öğrencinin dönem sonu başarı notuna göre dersi geçip geçmediğini belirleyen bir kod bloğu incelenmişti. Şimdi dönem sonu başarı notuna göre öğrencinin dersten aldığı harf notunu belirleyen bir program tasarlamaya çalışalım. Harf notu belirlenirken birden fazla koşulun test edilmesi gerektiği için if...else if yapısını kullanmak yerinde olacaktır.

```
int basari_notu;

if (basari_notu >= 90){
    cout << "başarı notuna karşılık gelen harf notu A'dir" << endl;
}
else if(basari_notu >= 80){
    cout << "başarı notuna karşılık gelen harf notu B'dir" << endl;
}
else if(basari_notu >= 70){
    cout << "başarı notuna karşılık gelen harf notu C'dir" << endl;
}
else if(basari_notu >= 60){
    cout << "başarı notuna karşılık gelen harf notu D'dir" << endl;
}
else{
    cout << "başarı notuna karşılık gelen harf notu F'dir" << endl;
}
```

Örneğin öğrencinin dönem sonu başarı notu 92 ise, yukarıdaki kod bloğundaki ilk koşul sağlandığından, yani 'basari_notu >= 90' ilişkisel ifadesi true boolean değeri alacağından, koşulun gövdesindeki kod çalıştırılacak ve ekrana "başarı notuna karşılık gelen harf notu A'dir" ifadesi yazdırılacaktır. Diğer yandan eğer öğrencinin dönem sonu başarı notu 75 ise, yukarıdaki kod bloğundaki ilk koşul sağlanmayacak ve program ilk koşulun gövdesindeki kodu pas geçerek ikinci koşulu test edecektir. Verilen başarı notu için ikinci koşul da sağlanmadığından, diğer bir ifadeyle 'basari_notu >= 80' ilişkisel ifadesi false boolean değeri alacağından, program bir önceki koşula benzer biçimde bu koşulun gövdesindeki kodu da pas geçecek ve bir sonraki koşulu test edecektir. Verilen başarı notu için üçüncü koşul sağlanacağından, yani 'basari_notu >= 70' ilişkisel ifadesi true boolean değeri alacağından program bu koşulun gövdesindeki kodu çalıştıracak ve ekrana "başarı notuna karşılık gelen harf notu C'dir" cümlesi yazdıracaktır.

Şimdi de taş-kağıt-makas oyunu tasarlamak için kullanılabilecek bir kod bloğu üzerinden if...else if kavramını pekiştirmeye çalışalım. Bilindiği gibi taş-kağıt-makas oyununda oyuncular aynı anda ve rastgele bir biçimde taş, kağıt ve makas alternatiflerinden birini seçer ve seçtiği alternatifi ilgili el işaretiyle karşı oyuncuya bildirirler. Oyunda önceden belirlenmiş kurallara göre (taş makası kırar, kağıt taş kaplar ve makas kağıdı keser) oyunculardan biri oyunu kazanmış sayılır. Aşağıdaki kod bloğu, birinci oyuncunun makası seçtiği bilindiğinde ikinci oyuncunun seçimine göre kazananı belirlemektedir. Kod bloğunda oyuncu_1 ve oyuncu_2 birer string değişkeni olarak tanımlanmış ve atama işlemi kullanılarak oyuncu_1 değişkenine "makas" kelimesi atanmıştır. Eğer oyuncu_2 string değişkeni "tas" değerini al-

mıysa ilk koşul sağlanmış olacak ve koşulun gövdesindeki kod çalıştırılarak ekrana “oyunu ikinci oyuncu kazandı” cümlesi yazdırılacaktır. Eğer oyuncu_2 değişkeninin alacağı değer “makas” ise, if...else if yapısındaki ne ilk ne de ikinci koşul sağlanacağından program ilk iki koşulun gövdesindeki kodu pas geçecek ve alternatif gövdedeki kodu çalıştırarak ekrana “oyuncular berabere kaldılar” cümlesini yazdıracaktır.

```
string oyuncu_1, oyuncu_2;
oyuncu_1 = "makas";
if (oyuncu_2 == "tas"){
    cout << "oyunu ikinci oyuncu kazandı" << endl;
}
else if(oyuncu_2 == "kagit"){
    cout << "oyunu birinci oyuncu kazandı" << endl;
}
else{
    cout << "oyuncular berabere kaldılar" << endl;
}
```

C++ programlama dilinde if...else if gibi çoklu seçime imkan bir diğer yapı da switch kontrol yapısıdır. Switch, bir değişkenin ya da ifadenin alabileceği farklı değerlere göre alternatif durumlardan birini seçmemize, diğer bir ifadeyle farklı kod satırlarını çalıştırabilmemize imkan veren seçmeli bir kontrol yapısıdır. Switch kontrol yapısı aşağıda belirtildiği gibi case, break ve default anahtar kelimelerinden oluşmaktadır. Bu kontrol yapısında, kontrol olarak adlandırılan değişken ya da ifade deger_1, deger_2 gibi değerlerden biri ile eşleşiyorsa, ilgili kod parçası break anahtar kelimesine kadar çalıştırılacak; eğer kontrol değişkeni hiç bir değerle eşleşmemişse, default anahtar kelimesi ile belirlenmiş kod parçası çalıştırılacaktır. Break ve default anahtar kelimeleri zorunlu olmamakla birlikte, daha kullanışlı ve etkili switch yapısı oluşturabilmek için genellikle tercih edilmektedirler. Break kontrol değişkeninin kalan değerlerle eşleşip eşleşmediğini test etmeden, yani daha fazla kod çalıştırmadan switch yapısını sonlandırmamıza imkan sağlarken; default kontrol değişkeninin tanımlanan değerlerden biriyle eşleşmediği durumlar için bile uygulanabilecek alternatif kod parçası oluşturmamıza imkan vermektedir.

```
switch(kontrol){
    case deger_1: {
        kod_parcasi_1
        break;
    }
    case deger_2: {
        kod_parcasi_2
        break;
    }
    ...
    default: {
        kod_parcasi*
        break;
    }
}
```

Switch yapısı kullanılarak geliştirilen aşağıdaki kod bloğu, yukarıdakine benzer olarak birinci oyuncunun makası seçtiği bilindiğinde ikinci oyuncunun seçimine göre kazananı belirlemektedir. Yalnız switch kontrol yapısı sadece tamsayı ve karakter veri türleri ile çalıştığı için, oyuncular seçimleri birer string değişkeni olarak değil de char değişkeni olarak tanımlanacaktır. Yani “tas”, “kagit” ve “makas” kelimeleri yerine sırasıyla ‘T’, ‘K’ ve ‘M’ karakterleri kullanılacaktır. Örneğin ikinci oyuncu kağıdı seçtiğinde, yani oyuncu_2 değişkeni ‘K’ değeri aldığı anda; değişkenin aldığı değer ikinci durumla eşleştiği için, program ikinci gövdedeki kod parçasını break anahtar kelimesine kadar çalıştıracak ve ekrana “oyunu birinci oyuncu kazandı” cümlesini yazdıracaktır. Kod bloğunda break anahtar kelimesi kullanılsaydı, program oyuncu_2 değişkeninin aldığı değer ikinci durumla eşleştiği için, ikinci gövdeden itibaren yapının sonuna kadar bütün kodları çalıştıracak ve ekrana sırasıyla “oyunu birinci oyuncu kazandı”, “oyunucular berabere kaldılar” ve “gecerli deger girilmedi” cümleleri yazdırılacaktı.

```
char oyuncu_1, oyuncu_2;
oyuncu_1 = 'M';
switch(oyuncu_2){
    case 'T': {
        cout << "oyunu ikinci oyuncu kazandı" << endl;
        break;
    }
    case 'K': {
        cout << "oyunu birinci oyuncu kazandı" << endl;
        break;
    }
    case 'M': {
        cout << "oyunucular berabere kaldılar" << endl;
        break;
    }
    default: {
        cout << "gecerli deger girilmedi" << endl;
        break;
    }
}
```



ARAŞTIRALIM

If...else if yapısı kullanılarak oluşturulan ve dönem sonu başarı notuna göre öğrencinin dersten aldığı harf notunu belirleyen kod bloğunu switch kontrol yapısını kullanarak yeniden tasarlayın.

7. İÇ İÇE IF...ELSE IF ÇOKLU SEÇİM KONTROL YAPISI

C++ dilinde bazı durumlarda bir biri içerisinde if...else if kontrol yapıları kullanarak çoklu durumları kontrol etmek ve alternatif kodlardan birini seçmek gerekebilir. Şimdi girilen 2 boolean değerine mantıksal VEYA işlecini uygulayan ve sonucu ekrana yazdıran kod bloğu üzerinden iç içe if...else if yapısının nasıl çalıştığı anlatılacaktır. Aşağıdaki kod bloğunda deger_1 ve deger_2 şeklinde iki boolean değişkeni tanımlanmış ve kullanıcıların klavye aracılığıyla girdikleri boolean değerleri sırasıyla bu değişkenlere atanmıştır. Sonrasında mantıksal VEYA işlecinin bölümde verilen tanımına uygun olarak bu boolean değerlerinin mantıksal VEYA işlecinin sonucu belirlenmiş ve ekrana yazdırılmıştır. Örneğin kullanıcı deger_1 ve deger_2 değişkenleri için sırasıyla false ve true değerlerini girmiş olsun. Kod bloğundaki ilk koşul doğru olacağından, bu koşulun gövdesindeki kod çalıştırılacaktır. Bununla birlikte içteki if...else kontrol yapısındaki koşul doğru olmadığından, program koşulun gövdesindeki kodu pas geçecek ve alternatif gövdedeki kodu çalıştırarak ekrana “deger_1 || deger_2 ifadesinin boolean degeri DOGRU dur” cümlesini yazdıracaktır.

```
bool deger_1, deger_2;
cout << "İki bool degeri giriniz: ";
cin >> deger_1 >> deger_2;
if (deger_1 == 0) {
    if (deger_2 == 0) {
        cout << "deger_1 || deger_2 ifadesinin bool degeri YANLIS tir" << endl;
    }
    else {
        cout << "deger_1 || deger_2 ifadesinin bool degeri DOGRU dur" << endl;
    }
} else {
    cout << "deger_1 || deger_2 ifadesinin bool degeri DOGRU dur" << endl;
}
```



ARAŞTIRALIM

Yukarıdaki örneğe benzer olarak, iç içe if...else if yapısı kullanarak girilen iki adet boolean değerine mantıksal ve işlecini uygulayıp sonucu ekrana yazdıran bir kod geliştirin.

Şimdi de taş-kağıt-makas oyunu tasarlamak için kullanılabilecek bir kod bloğu üzerinden iç içe if...else kavramı pekiştirilecektir. Aşağıdaki kod bloğu seçimleri bilindiğinde yukarıda da aktarılan kurala göre oyunun kazananını belirlemektedir. Örneğin ilk oyuncu taşı ve ikinci oyuncu makası seçtiğinde, diğer bir ifadeyle oyuncu_1 ve oyuncu_2 string değişkenlerine atanan değerler sırasıyla “taş” ve “makas” olduğunda; kod bloğundaki ilk koşul doğru olacağından, bu koşulun gövdesindeki kod çalıştırılacaktır. Bununla birlikte içteki if...else kontrol yapısındaki ilk koşul doğru olmadığından, içteki kontrol yapısının ikinci koşulu test edilecektir. İçteki kontrol yapısının ikinci koşulu doğru olacağından bu koşulun gövdesindeki kod çalıştırılacak ve ekrana “birinci oyuncu kazandı” cümlesi yazdırılacaktır.


```

string oyuncu_1, oyuncu_2;
if (oyuncu_1 == "tas"){
    if (oyuncu_2 == "kagit"){
        cout << "ikinci oyuncu kazandi" << endl;
    }
    else if (oyuncu_2 == "makas"){
        cout << "birinci oyuncu kazandi" << endl;
    }
    else {
        cout << "oyuncular berabere kaldilar" << endl;
    }
}
else if (oyuncu_1 == "kagit"){
    if (oyuncu_2 == "makas"){
        cout << "ikinci oyuncu kazandi" << endl;
    }
    else if (oyuncu_2 == "tas"){
        cout << "birinci oyuncu kazandi" << endl;
    }
    else {
        cout << "oyuncular berabere kaldilar" << endl;
    }
}
else {
    if (oyuncu_2 == "tas"){
        cout << "ikinci oyuncu kazandi" << endl;
    }
    else if (oyuncu_2 == "kagit"){
        cout << "birinci oyuncu kazandi" << endl;
    }
    else {
        cout << "oyuncular berabere kaldilar" << endl;
    }
}
}

```

BÖLÜM ÖZETİ

Bu bölümde programın akışını kontrol etmeye imkan veren yapı türlerinden biri olan seçmeli kontrol yapılarından, ve bunun öncesinde seçmeli kontrol yapılarındaki koşulları oluşturabilmemiz için gerekli olan ilişkisel ve mantıksal işleçlerden bahsedilmiştir. İlişkisel işleçler, C++ programlama dilinde iki değişken ya da iki değer arasındaki ilişkiyi tanımlayan ya da koşullu ifadelerin bir parçası olarak bu ilişkiyi sınavan yapılardır. C++ programlama dilinde eşittir, eşit değildir, küçüktür, küçük eşittir, büyüktür ve büyük eşittir olmak üzere 6 adet ilişkisel işleç tanımlanmaktadır. İlişkisel işleçler üzerinden oluşturulmuş ilişkisel ifadelerin sonucu, eğer ifadedeki değerler ya da değişkenler arasında ilişkisel işlecin tanımladığı gibi bir ilişki varsa ifade true boolean değeri almakta, yoksa false boolean değeri almaktadır. Birden fazla durumun sınıandığı daha karmaşık koşullu ifadelerin oluşturulabilmesi mantıksal işleçlerin kullanımıyla mümkün olmaktadır. C++ programlama dilinde DEĞİL, VE ve VEYA olmak üzere 3 adet mantıksal işleç tanımlanmaktadır. Bölüm içerisinde işlenenlerin farklı boolean değerleri için mantıksal işleçlerin aldığı boolean değerleri tablolar halinde gösterilmiştir. Seçmeli kontrol yapıları programın akışını belli koşullar üzerinden çatallandırmaya imkan veren yapılardır. C++ programlama dilinde if (tek seçim), if...else (ikili seçim) ve switch (çoklu seçim) olmak üzere üç farklı seçmeli kontrol yapısı tanımlanmıştır. If kontrol yapıları tek bir koşulun sınıandığı yapılardır. Eğer koşul kısımdaki ifade doğru ise gövdedeki kod çalıştırılmakta, aksi durumda gövdedeki kod pas geçilmektedir. If yapısından farklı olarak if...else kontrol yapısı ise koşul yanlış olduğunda da belirli bir kod bloğunun çalıştırılabilmesine imkan vermektedir. If...else kontrol yapısında koşul sağlandığında gövdedeki kod, aksi durumda ise alternatif gövdedeki kod çalıştırılmaktadır. Birden fazla koşulun test edilmesi gerektiğinde ise if ve if...else kontrol yapıları yetersiz kalmaktadır. Bu gibi durumlarda çoklu seçime imkan veren if...else if veya switch kontrol yapıları tercih edilmelidir.

GÖZDEN GEÇİRELİM

1. Aşağıdakilerden hangisi C++ programlama dilinde kullanılan ilişkisel işleçlerden biri değildir?

- a. ==
- b. >=
- c. <=
- d. =
- e. >

4. Aşağıdakilerden hangisi C++ programlama dilinde kullanılan mantıksal VE işlecidir?

- a. >=
- b. =>
- c. ||
- d. !&
- e. &&

2. Aşağıdakilerden hangisi C++ programlama dilinde kullanılan mantıksal işleçlerden birisidir?

- a. >=
- b. ==
- c. ||
- d. !=
- e. <

5. Aşağıda verilen kod bloğu çalıştırıldığında ekrana yazılacak değer aşağıdakilerden hangisi olacaktır?

```
int sayi_1 = 2, sayi_2 = 3;
cout << ( (3*sayi_1 == sayi_2*2) ? sayi_2
- sayi_1
```

```
: sayi_2 + sayi_1) << endl;
```

- a. 1
- b. 2
- c. 3
- d. 4
- e. 5

3. Aşağıda verilen kod bloğu çalıştırıldığında ekrana yazılacak değer aşağıdakilerden hangisi olacaktır?

```
int sayi_1 = 1, sayi_2 = 2, sayi_3 = 3;
if ((3*sayi_1 == sayi_2 + 1) && (sayi_3 <
sayi_2
+ 2)){
```

```
cout << sayi_1 + sayi_3 << endl;
}
```

```
else {
```

```
cout << sayi_2 + sayi_3 << endl;
}
```

- a. 3
- b. 4
- c. 5
- d. 6
- e. 7

6. Aşağıdaki anahtar kelimelerden hangisi switch kontrol yapısının bir parçası olarak kullanılmaktadır?

- I. if
- II. case
- III. break
- IV. while
- V. default

- a. I ve III
- b. I ve V
- c. II ve IV
- d. II, III ve V
- e. II, IV ve V

7. Aşağıda verilen kod bloğu çalıştırıldığında ekrana yazılacak değer aşağıdakilerden hangisi olacaktır?

```
int sayi_1 = 8, sayi_2 = 6, sayi_3 = 7;
if (sayi_1 < sayi_2){
if (sayi_1 < sayi_3){
cout << sayi_1 << endl;
}
else {
cout << sayi_3 << endl;
}
}
else {
if (sayi_2 < sayi_3){
cout << sayi_2 << endl;
}
else {
cout << sayi_3 << endl;
}
}
}
a. 5
b. 6
c. 7
d. 8
e. 9
```

8. C++ programlama dilinde kullanılan ilişkisel küçük eşittir işleci aşağıdakilerden hangisidir?

a. <
b. = <
c. < =
d. < <
e. < = <

9. C++ programlama dilinde temp tamsayısı değişkeninin 10'dan küçük pozitif bir tamsayı olup olmadığını kontrol eden mantıksal ifade aşağıdakilerden hangisinde doğru olarak verilmiştir?

a. temp >= 0 && temp <= 10
b. temp >= 0 && temp > 10
c. temp > 0 && temp < 10
d. temp < 0 && temp > 10
e. temp < 0 && temp < 10

10. Aşağıda verilen kod bloğu çalıştırıldığında ekrana yazılacak değer aşağıdakilerden hangisi olacaktır?

```
int sayi_1 = 4, sayi_2 = 5;
switch(sayi_1){
case 1: {
cout << sayi_2 - sayi_1 << endl;
break;
}
case 2: {
cout << 2*sayi_2 - sayi_1 << endl;
break;
}
case 3: {
cout << 3*sayi_2 - sayi_1 << endl;
break;
}
default: {
cout << 4*sayi_2 - sayi_1 << endl;
break;
}
}
a. 1
b. 6
c. 11
d. 16
e. 21
```

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

BÖLÜM 7

DÖNGÜLER

ANAHTAR KAVRAMLAR

While, For, Do-While

ÖĞRENME HEDEFLERİ

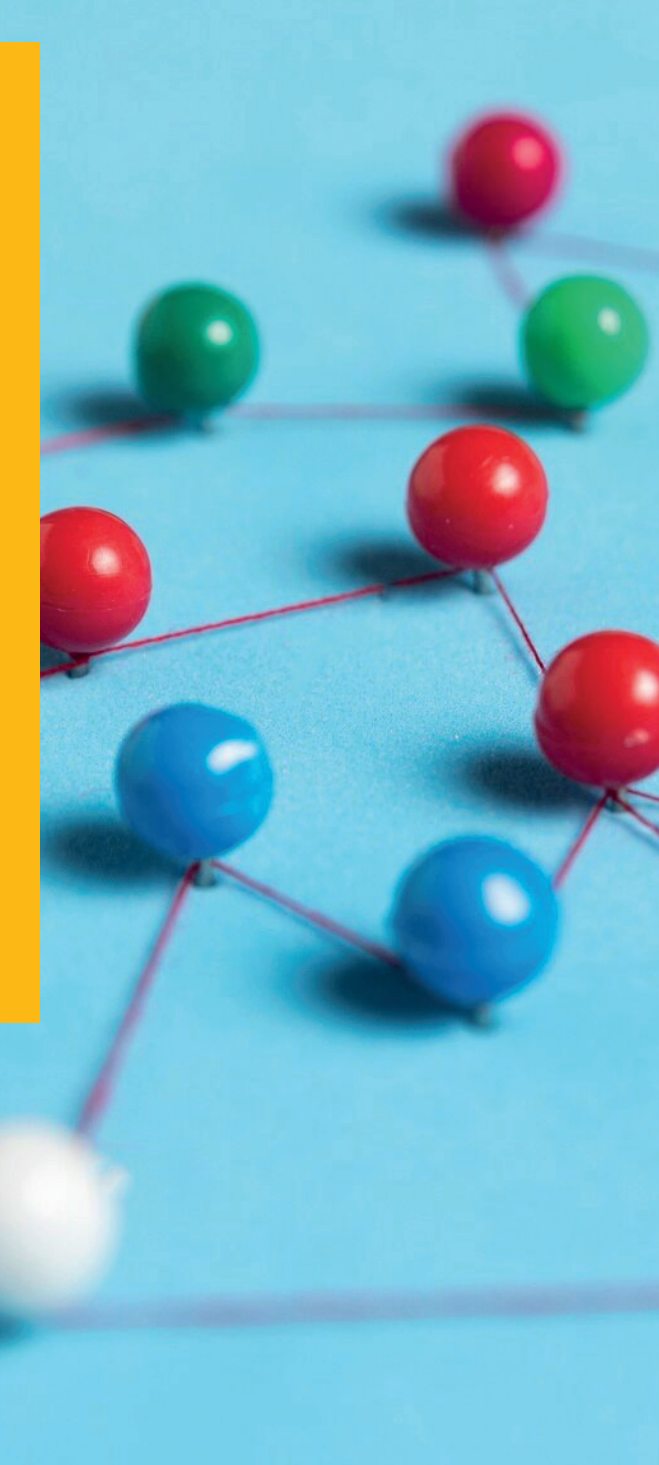
- 1 Öğrenci bölümlere ayırmaya ilişkin temel kavramları bilir
- 2 Öğrenci bölümlere ayırma ilkelelerini, sistemlerini ve önemini bilir
- 3 Öğrenci bölümlere ayırma sistemlerinin üstünlük ve sakıncalarını bilir
- 4 Öğrenci işletmelerde bölümlere ayırma sürecinde karşılaşılan temel sorunları algılar ve çözüm üretir
- 5 Aritmetik işlemlerin öncelik sıralamasını bilir

GİRİŞ

Döngüler, programlamının temelini oluşturan mantıksal yapılardan biridir.

Döngüler, bilgisayarların belirli görevleri yinelemeli (tekrar tekrar) gerçekleştirmesini sağlar. Yineleme işlemi, programcının başlangıçta belirttiği adım sayısı kadar veya belirli bir koşul gerçekleşene kadar devam eder. Bazı durumlarda kullanıcıdan bir değer girmesi istenerek programın devamına ya da sonlandırılmasına karar verilebilir. Programlamada kullanılan çeşitli döngü yapıları vardır. Bunlar;

- for döngüsü (for-loop)
- while döngüsü (while-loop)
- do-while döngüsü (do-while loop)
- İç-içe döngüler (Nested loops) şeklinde isimlendirilir.



1. DÖNGÜLER

Bilgisayarın belirli görevleri belirli bir koşul sağlanana kadar veya başlangıçta belirtilen adım sayısı kadar yapılması döngüler vasıtasıyla sağlanır. Örneğin bir yazılım geliştiricisi bir sınıftaki her bir öğrencinin not ortalamasını bulan bir program yazmak isterse, bunu her bir öğrenci için ayrı ayrı yapmak yerine belirli kurallar dahilinde oluşturduğu formülü döngü içerisinde kullanarak bu işlemi gerçekleştirir. Böylece bilgisayar tüm öğrenciler için not ortalaması hesaplanana kadar aynı işlemi yinelemeli olarak gerçekleştirir. Döngüler programlamada zaman ve maliyetin düşürülmesi konusunda oldukça etkilidir. Bu bölümde C++ programları geliştirilirken, programları daha etkili hale getirmek için döngülerden nasıl faydalanılacağı ve hangi döngü yapısının hangi problemlerde kullanılabileceği gösterilecektir.

C++ programlama dilinde kullanılabilecek 4 temel döngü yapısı bulunmaktadır. Bunlar;

- while Döngüsü
- do-while Döngüsü
- for Döngüsü
- İç-içe Döngüler (Nested Loops)

olarak ifade edilebilirler.

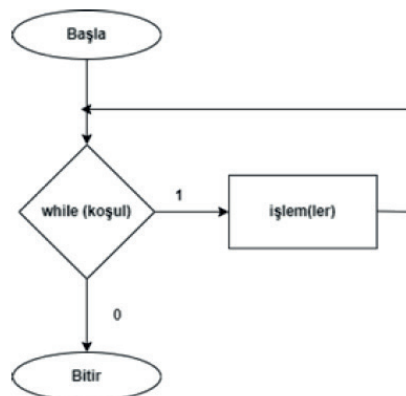
Döngüler aynı zamanda sabit döngüler ve koşullu döngüler olarak sınıflandırılabilirler. For döngüsü sabit döngü sınıfında değerlendirilirken, while ve do-while döngüleri koşullu döngülere örnek olarak verilebilir.

2. while DÖNGÜSÜ

Koşullu döngüler sınıfında değerlendirilen while döngüsünün başlatılması ve devamı belirli bir koşul doğru olduğu sürece geçerlidir. Koşul sağlanmadığında döngü başlatılmaz veya döngünün çalışması esnasında koşul bozulursa döngü sona erer.

```
while (koşul) {
    işlem(ler)
}
```

Yukarıdaki örnek kod ile while döngüsünün sözdizimi (syntax) yapısı verilmiştir. Bu yapının akış diyagramı (flowchart) ise aşağıdaki şekilde gösterilebilir.



Şekil.7 1. while döngüsü akış diyagramı.

Akış diyagramında 1 ve 0 şeklinde gösterilen ifadeler mantıksal ifadelerdir. Algoritma ve akış diyagramları oluşturulurken, evet anlamına gelen mantıksal ifadeler 1, hayır anlamına gelen mantıksal ifadeler ise 0 ile gösterilir. Bazı kaynaklarda 1 yerine true, 0 yerine ise false ifadeleri kullanılabilir. Yukarıdaki örnekte koşulun doğru (1) olması durumunda program döngü içerisinde tanımlanan işlem ya da işlemleri yapacak şekilde döngü içerisine girer. Eğer koşul başlangıçta ya da programın çalışması esnasında bozulursa program mantıksal hayır (0) sonucunu oluşturur ve program bu yönde ilerler.

Örnek.7.1. Ekranı 5 defa “Ankara Üniversitesi” ifadesini yazan C++ programını while döngüsü kullanılarak yazınız.

```
ÖRNEK 7.1. C++ Kodlar
#include<iostream>
using namespace std;
int main () {
    int a=0;
    while (a<5)
    {
        cout<<"Ankara Üniversitesi\n";
        a=a+1;
    }
}
```

Kodların Açıklaması:

- Main fonksiyonu içerisinde tanımlanan `int a=0` ifadesi döngü koşul ifadesi olarak oluşturulmuştur. Başlangıçta tanımlanan `a` değişkeninin değeri 0 (sıfır) olarak ayarlanmıştır.

- `while (a<5)` ifadesi döngünün başlatılması ve devamını kontrol edecek yapı olarak kurulmuştur. Başka bir ifade ile `a` değişkeninin değeri 5’den küçük olduğu sürece döngü başlatılacak ve bu koşul sağlandığı sürece döngü içerisindeki işlem(ler) gerçekleştirilecektir. Örnekte `a` değişkeninin başlangıç değeri 0’dır ve 0 (sıfır) 5’den küçük olduğu için program döngü içerisine girecektir.

- `cout<<"Ankara Üniversitesi\n"` döngü denetiminden sonra ilk çalıştırılacak kod satırıdır ve ekrana “Ankara Üniversitesi” ifadesini yazdıracaktır.

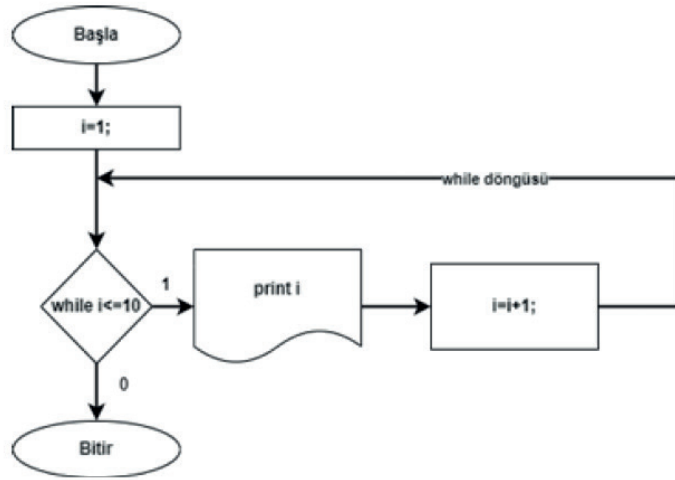
- `a=a+1` ifadesi döngü denetimini yapmak için kullanılan ifadedir. Programlamada sayaç (counter) olarak bilinir. Program bu satıra geldiği anda `a` değişkeninin değeri 1 artırılabacaktır. Böylece `a` değişkeni değerinin sürekli 5 değerinden küçük olmasının önüne geçilecektir. Bu denetimin yapılmadığı durumlarda program sonsuz (infinite loop) döngüye girecektir ve asla döngü dışına çıkamayacaktır. Döngü değişkeni `a`’nın değeri 1 artırıldıktan sonra program tarafından `a`’nın değerinin tekrar 5 değerinden küçük olup olmadığı kontrol edilecek, eğer küçük ise program yeniden döngü içerisindeki işlem(ler)i gerçekleştirecektir. Döngü değişkeni `a`’nın değeri 5’e ulaştığında bu denetim sonucu hayır (0) mantıksal cevabını oluşturacak ve program döngü yapısı içerisinde çıkacaktır.

Yukarıda açıklaması verilen kodların çalışma mantığı tablo görünümünde aşağıdaki şekilde ifade edilebilir.

Yineleme (Iteration)	Değişken Değeri	Koşul (a<5)	İşlem(ler)
1.	a=0	Evet (1)	Ankara Üniversitesi
2.	a=1	Evet (1)	Ankara Üniversitesi
3.	a=2	Evet (1)	Ankara Üniversitesi
4.	a=3	Evet (1)	Ankara Üniversitesi
5.	a=4	Evet (1)	Ankara Üniversitesi
6.	a=5	Hayır (0)	Döngü Dışı

Örnek.7.2. 1'den 10'a kadar olan sayıları ekrana yan yana yazdıran C++ kodlarını while döngü yapısı ile yazdırınız.

İstenilen programın akış diyagramı Şekil.7.2'de verilmiştir. Bu örnekte döngü değişkeni olarak i seçilmiştir. Örnekte 1'den 10'a kadar olan sayıların ekrana yan yana yazdırılması istenildiği için i değişkeninin başlangıç değeri 1 olarak belirlenmiştir. Program döngü denetimi noktasına geldiğinde öncelikle i değişkeninin değerinin küçük-eşit ($\leq$) 10 olup olmadığı kontrol edilecek, eğer değer bu koşula uyuyorsa program döngü içerisine girecektir. Döngü içerisinde öncelikle i değişkeninin değeri ekrana yazdırılacak arkasından değişkenin değeri 1 artırılabacaktır. Program bu noktadan sonra i değişkeninin yeni değeri ile yeniden döngü denetimin yapıldığı satıra dönecek ve değişken değerinin 10'a eşit ya da küçük olma durumu yeniden kontrol edilecektir. Değer küçük-eşit 10'olduğu sürece program tekrar-tekrar döngü içerisine girecek ve buradaki işlemleri gerçekleştirmeye devam edecektir.



Şekil.7 2 while döngüsü (devam).

```

ÖRNEK 7.2. C++ Kodlar
#include<iostream>
using namespace std;
int main () {
    int i=1;
    while (i<=10)
    {
        cout<<i<<" ";
        i=i+1;
    }
}
  
```

Çalışma Sorusu: Örnek 7.2 için 1’den 10’a kadar olan sayıları alt-alta yazdırmak için kodda nasıl bir değişiklik yapılmalıdır?

break Komutu: Bazı durumlarda döngü koşulu bozulmadan istenilen bir sonuca ulaşıldığında döngünün sonlandırılması gerekebilir. C++’da break komutu döngüyü sonlandırmak için kullanılan bir kontrol ifadesidir. Döngü içerisinde herhangi bir noktada program break komutu ile karşılaşırsa döngüyü o noktada sonlandırır ve kontrolü döngüden sonraki ilk ifadeye bırakır. Bu komut while, do-while, for, iç-içe döngüler ve sonsuz döngülerde kullanılabilir. Bu durum Örnek 7.3 üzerinden incelenebilir.

Örnek 7.3. Klavyeden girilen sayıların karekökünü hesaplayan C++ programını while döngüsü ile oluşturunuz. Eğer klavyeden negatif bir sayı girilirse program sonlandırılmalıdır. Bu örnek için sonsuz döngü (infinite loop) yapısı oluşturulabilir. Program kullanıcının klavyeden girdiği sayının karekökünü hesaplayacak ve sonucu ekrana yazacaktır. Bu durum kullanıcı negatif bir sayı girene kadar devam edecektir. Örnekte karekök hesaplaması yaptırılacağı için sqrt () fonksiyonun içerisinde bulunduğu <cmath> header dosyasının programa eklenmesi gerekmektedir.

```
ÖRNEK 7.3. C++ Kodlar
#include<iostream>
#include<cmath>
using namespace std;
int main () {
    double sayi, sonuc;
    while (true) {
        cout<<"Karekökü hesaplanacak sayıyı giriniz:";
        cin>>sayi;
        if (sayi<0)
            break;
        sonuc=sqrt(sayi);
        cout<<sayi<<" sayısının karekökü:"<<sonuc<<endl;
    }
    cout<<"Negatif bir sayı girildiği için döngü sonlandırılmıştır!!!";
}
```

- Program sonsuz döngü yapısı ile oluşturulurken döngü devam koşulu için “true” mantıksal değeri kullanılmıştır. Böylelikle while döngüsü başlangıç koşul sorusunun cevabı varsayılan olarak evet (true) şeklinde kabul edilmiştir. Bu örnekte döngü denetimi break komutu kullanımı ile gerçekleştirilmiştir.

- Program içerisinde kullanılan sayi ve sonuc isimli değişkenler double veri türünde oluşturulmuştur. Böylelikle ondalıklı sayı türlerinde işlem yapılabilmesi sağlanmıştır.

continue Komutu: Döngü yinelemesi (iteration) esnasında program continue komutu ile karşılaşırsa döngünün bu komuttan sonra gelen ifadeleri değerlendirilmez ve döngü bir sonraki yineleme adımı ile yeniden başlatılır. Komut while, do-while, for ve iç-içe döngü yapılarında kullanılabilir. Bu durum aşağıdaki örnek üzerinden incelenebilir.

Örnek 7.4. Bir ders için öğrenci ortalamasını hesaplayan C++ programını while döngüsü yapısı ile oluşturunuz. Eğer girilen not vize ya da final sınavı için 100’den büyük ya da 0’dan küçükse kullanıcıdan yeni değer girmesi istenilmelidir. Programdan çıkış için C değerinin girilmesi gerekmektedir.

ÖRNEK 7.4. C++ Kodlar

```
#include<iostream>
using namespace std;
int main () {
    int vize, final;
    double ortalama;
    char cikis;
    while (true) {
        cout<<"Öğrencinin Vize Notunu Giriniz:";
        cin>>vize;
        if (vize>100 || vize<0){
            cout<<"Geçerli Bir Değer Giriniz (0-100 arası);";
            continue;
        }
        cout<<"Öğrencinin Final Notunu Giriniz:";
        cin>>final;
        if(final>100 || final <0){
            cout<<"Geçerli Bir Değer Giriniz (0-100 arası);";
            continue;
        }
        ortalama=vize*0.3+final*0.8;
        cout<<"Öğrencinin Ortalaması="<<ortalama<<endl;
        cout<<"Programdan Çıkış İçin C, Devam İçin D Giriniz:";
        cin>>cikis;
        if (cikis=='C')
            break;
    }
    cout<<"Program kullanıcıni isteği ile sonlandırılmıştır";
}
```

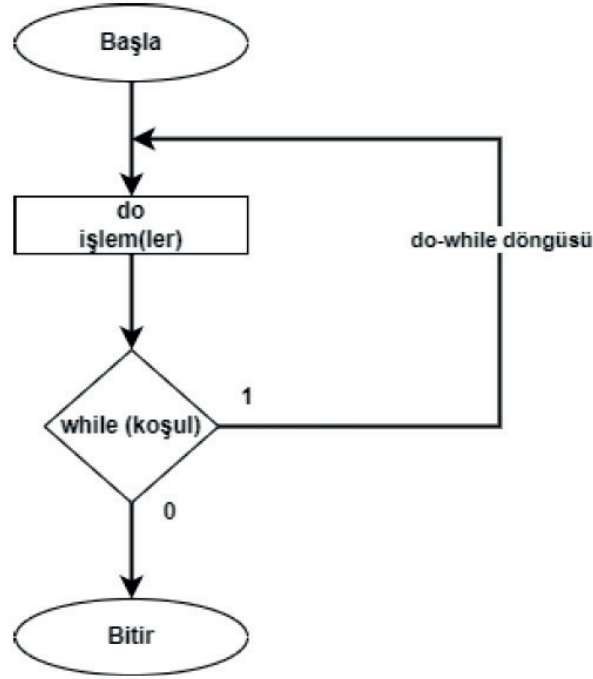
Önemli Not: C++ büyük-küçük harf duyarlı (Case Sensitive) bir programlama dilidir. Dolayısıyla programdan çıkış yapabilmek için büyük harf C'nin girilmesi gerekmektedir. Örnek için öğrencinin ortalaması vize notunun %30'u final sınavının %80'i alınarak hesaplatılacaktır. Program continue komutu ile karşılaşınca döngü başından itibaren yeniden çalıştırılırken break komutu ile karşılaştığında döngü sonlandırılmaktadır. Bu örnek içinde sonsuz döngü yapısı kullanılmıştır.

3. do-while DÖNGÜSÜ

do-while döngüsü while döngü yapısına oldukça benzemektedir. Bu yapıda döngü denetimi döngü başında değil döngü sonunda gerçekleştirilir. Bu özelliği yüzünden döngü içerisindeki ifadeler en az bir kez çalıştırılır. Bu döngünün syntax yapısı aşağıdaki şekilde verilir.

```
do {
    işlem(ler)
} while (koşul);
```

Bu yapı için akış diyagramı ise aşağıdaki şekilde oluşturulabilir.



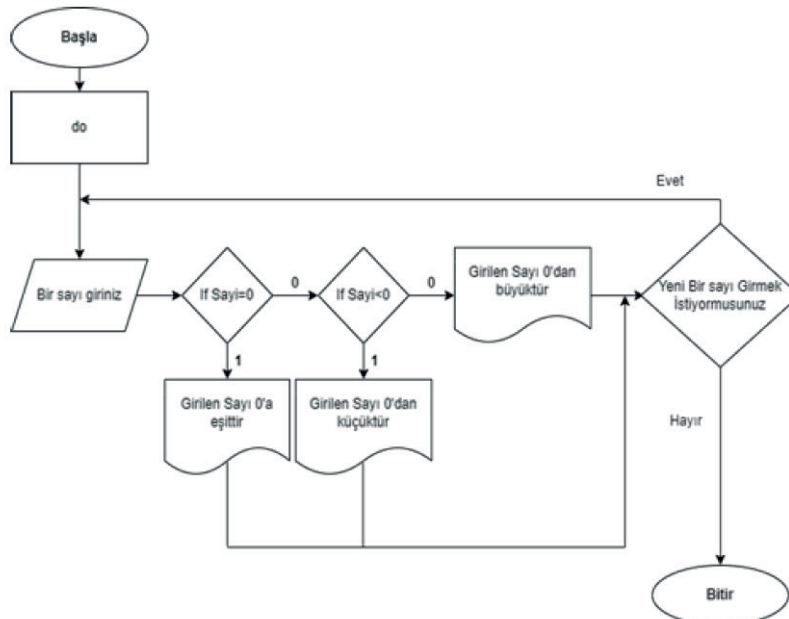
Şekil.7 3. do-while döngüsü akış diyagramı.

Akış diyagramından da rahatlıkla görülebileceği üzere döngü içerisindeki işlem(ler) en az bir kez çalıştırılır ve döngü denetimi bu işlem sonrasında gerçekleştirilir.

Örnek 7.5 üzerinden do-while ile while döngü yapısının arasındaki farklılıkları inceleyiniz.

Örnek.7.5. Klavyeden girilen bir sayının 0'dan küçük, 0'dan büyük ya da 0'a eşit olup olmadığını bulan C++ kodlarını do-while döngü yapısı yazınız.

Not: Program kullanıcıdan gelecek cevaba göre (Evet (E), Hayır (H)) çalışmaya devam edecek ya da sonlandırılacaktır.



Şekil.7 4. Örnek7.5 için akış diyagramı.


```

ÖRNEK 7.5. C++ Kodlar
#include<iostream>
#include<string>
using namespace std;
int main () {
    int sayi;
    char devam;
    do {
        cout<<"Bir Sayı Giriniz:";
        cin>>sayi;
        if (sayi==0)
            cout<<"Girilen Sayı 0'a eşittir\n";
        if (sayi<0)
            cout<<"Girilen Sayı 0'dan küçüktür\n";
        if (sayi>0)
            cout<<"Girilen Sayı 0'dan büyüktür\n";
        cout<<"Yeni Bir Sayı Girmek İstiyor musunuz (Evet için E, Hayır için H'ye Basınız):";
        cin>>devam;
    } while(devam=='E');
}

```

4. for Döngüsü

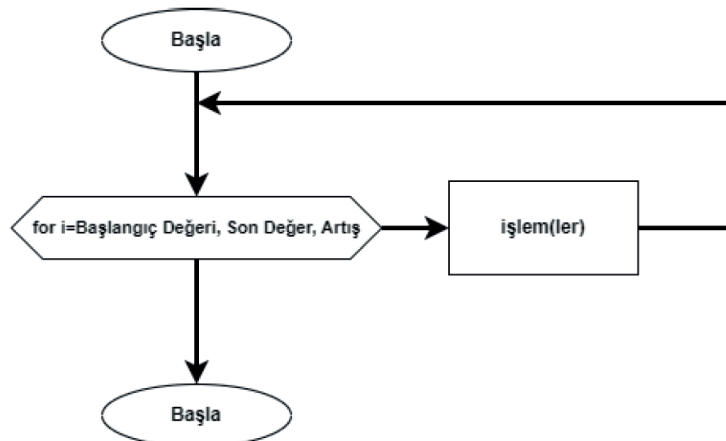
Programlamada kullanılan bir başka döngü yapısı for döngüsüdür. Sabit döngü türlerine örnek olarak gösterilebilir. Şartlı (koşullu) döngülerden farkı döngünün başlangıcı ve devamı için belirli bir koşulun gerçekleşmesine ihtiyaç duymamasıdır. Standart for döngüsü kullanımında döngü başlangıç değeri (initial value), son değer (final value) ve artış miktarı (increment) tanımlamaları ile döngünün yinelenmeli işlemleri kaç defa yapacağı döngü başında belirtilir. Döngünün syntax yapısı aşağıdaki şekilde verilebilir.

```

for (Başlangıç Değeri; Son Değer; Artış) {
    işlem(ler)
}

```

Döngünün başlangıç değeri artış miktarı ve son değeri arasında “;” karakterinin kullanılması gerekmektedir. Bu döngü yapısında döngü değişkeni ve değişken türü döngü başlangıç kısmında tanımlanır. Akış diyagramlarında while ve do-while döngüsünden farklı olarak aşağıdaki gibi bir sembol ile ifade edilirler.



Şekil.7 5. For döngüsü için akış diyagramı

Örnek 7.6. Kullanıcı tarafından klavyeden girilen sayı değerine kadar bütün sayıları toplayıp (1'den n'e kadar) sonucu ekrana yazan C++ programını for döngü yapısını kullanarak yazınız.

```
ÖRNEK 7.6. C++ Kodlar
#include<iostream>
using namespace std;
int main () {
    int sayi, toplam;
    toplam=0;
    cout<<"1'den Büyük Bir Sayı Giriniz:";
    cin>>sayi;
    for (int i=1; i<=sayi; i++){
        toplam +=i;
    }
    cout<<"1 ile "<<sayi<<" arasındaki sayıların toplamı="<<toplam<<endl;
}
```

Örnekte öncelikle kullanıcının gireceği sayı ve sonuc için tamsayı (integer) türünde değişken tanımlamaları yapılmıştır. Örnekte döngü değişkeni i'nin tanımlanması (int i=1) döngü içerisinde gerçekleştirilmiştir. Bu şekilde bir uygulama geliştirildiğinde döngü değişkeninin etki alanının sadece for döngüsü içerisinde olduğu unutulmamalıdır. Başka bir ifade ile döngü dışında i değişkeni değerine erişilemez. Örneğin döngü değişkeni olarak kullanılacak değişken main fonksiyonu etki alanında tanımlanmış olsaydı main fonksiyonun herhangi bir noktasından erişilebilir olurdu. Ancak bu kullanımın başka problemlere yol açabileceği unutulmamalıdır.

Örnek 7.7. Klavyeden girilen bir sayının asal olup olmadığını bulan C++ programını for döngüsü kullanarak yazınız.

```
ÖRNEK 7.7. C++ Kodlar
#include<iostream>
using namespace std;
int main () {
    int sayi;
    int sayac=0;
    cout<<"Bir Sayı Giriniz:";
    cin>>sayi;
    for (int i=1; i<=sayi; i++) {
        if(sayi % i ==0)
            sayac=sayac+1;
    }
    if (sayac==2)
        cout<<"Girilen Sayı Asaldır";
    else
        cout<<"Girilen Sayı Asal Değildir";
}
```

Ders video kaydında bu durum detaylı bir şekilde ele alınacaktır

Program yazılırken asal sayı tanımından yola çıkılmıştır. Sadece 1'e ve kendisine tam olarak bölünebilen sayılar asal sayılardır. Başka bir ifade ile asal sayıların 1 ve kendisi olmak üzere 2 adet tam böleni vardır. Döngü içerisinde tanımlanan `if (sa-yi%i==0)` ifadesi ile klavyeden girilen sayının 1'den başlayıp kendisine kadar bütün sayılarla tam olarak bölünüp bölünmediğinin kontrolü yapılmaktadır. Burada kullanılan % aritmetiksel operatörü aynı zamanda kalan (modulo) operatörü olarak bilinir. Bir sayının diğer sayıya bölümünden kalanını verir. Örnekte bu kalanın 0'a eşit olup olmadığı kontrol edilmektedir.

Her tam olarak bölünme işlemi sonrasında sayaç değişkeni değeri bir artırılmaktadır. Döngü dışında yeni bir kontrol mekanizması oluşturularak eğer sayaç değişkeninin değeri 2 ise (sadece 2 sayı ile tam olarak bölünmüştür) bu sayı için "Girilen Sayı Asaldır" ifadesi ekrana yazdırılmıştır. Eğer sayaç değişkeninin değeri 2'den farklı ise klavyeden girilen sayıyı tam olarak bölen 2'den fazla sayı olacaktır ve bu durum asal sayı tanımına uymayacaktır.

For döngüsünün farklı bir kullanımı için Örnek 7.8'i inceleyiniz.

```
ÖRNEK 7.8. C++ Kodlar
#include<iostream>
using namespace std;
int main () {
    for (;;) {
        cout<<"Ankara Üniversitesi\n";
    }
}
```

Bu örnekte, döngü başlangıç değeri, son değer ve artış miktarı belirtilmemiştir. Döngü bu haliyle sonsuz bir döngü formuna dönüşmüştür. Başlangıçta program döngü içerisine girer ve manuel olarak sonlandırılmadığı sürece (Ctrl+C) "Ankara Üniversitesi" ifadesini alt-alta yazar.

5. İç-içe Döngüler (Nested Loops)

Karar ifadelerinde (if) olduğu gibi döngü yapılarının da iç-içe oluşturulması mümkündür. Başka bir ifade ile bir döngü içerisinde başka döngü yapıları oluşturulabilir. Bu yapıyı daha iyi anlamak için aşağıdaki örneği inceleyiniz.

Örnek 7.9. Aşağıdaki ekran görüntüsünü verecek olan C++ programını iç-içe for (nested for) döngü yapısını kullanarak yazınız.



Ekran görüntüsü 3 satır, 3 sütun * (yıldız) karakterlerinden oluşturulmuştur. Bu ekran görüntüsünü elde edebilmek için iç-içe döngü yapısının kullanılması gerekmektedir.

ÖRNEK 7.9. C++ Kodlar

```
#include<iostream>
using namespace std;
int main () {
    for (int i=1;i<=3;i++) {
        for (int j=1;j<=3;j++){
            cout<<"*";
        }
        cout<<"\n";
    }
}
```

Örnek 7.9'dan görülebileceği gibi içteki döngü dıştaki döngünün etki alanı içerisinde. Program ilk çalıştırıldığında önce dıştaki döngü ve hemen ardından içteki döngü başlatılır. Bu aşamada i değişkeni 1 değerine sahip iken j değişkeni 1'den 3'e kadar değerler alır. J değişkenin her bir değerinde ekrana bir adet "*" karakteri basılır. Bu karakterler cout ifadesinde "/n" veya "endl" kullanılmadığı için yan yana yazdırılır. İçteki döngünün son değerinden sonra kontrol dıştaki döngüye geçer ve bu döngünün etki alanındaki cout<<"/n" ifadesi çalıştırılır ve ekranda bir alt satıra geçilir. Bu işlemin ardından i değişkeninin ikinci değeri (2) için döngü yeniden başlatılır. İçteki döngü yeniden başlatılırken j değişkeni yeniden 1 değerini alır ve bu işlem toplam 3 satır ve 3 sütun "*" karakterlerinin ekrana basılma işleminin sonuna kadar devam eder. Aşağıdaki tabloda bu durum ifade edilmiştir.

i=1	j=1	*
	j=2	**
	j=3	***
i=2	j=1	*** *
	j=2	*** **
	j=3	*** ***
i=3	j=1	*** *** *
	j=2	*** *** **
	j=3	*** *** ***

İç-içe döngü yapısında farklı döngüler birlikte kullanılabilir. Örneğin while döngüsü içerisinde for döngüsü, onun içerisinde do-while döngüsü gibi. Bu durum aşağıdaki örnek üzerinden incelenebilir.

Örnek 7.10. Aşağıdaki ekran görüntüsünü veren C++ kodlarını for ve while döngülerini iç-içe (nested) yapıda kullanarak kodlayınız.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
```

ÖRNEK 7.10. C++ Kodlar

```
#include<iostream>
using namespace std;
int main ( ) {
    for (int i=1;i<=10;i++)
    {
        int sayi=1;
        while(sayi<=i) {
            cout<<sayi<<" ";
            sayi+=1;
        }
        cout<<"\n";
    }
}
```

Bir programı yazmanın birden fazla yolu olabileceği unutulmamalıdır. Programlamada öğrenilen her yeni özellik ile program daha farklı şekillerde yazılabilir. Bu örnekte for döngüsü içerisinde (etki alanında) while döngüsü kullanılmıştır. Program sadece for döngüleri ya da while döngüleri ile oluşturulabilirdi. Örnekte, for döngüsü etki alanında sayı adında bir değişken kullanılarak başlangıç değeri 1 olarak atanmıştır (Bu değer for döngüsünün her yinelenmesinde yeniden 1 olacak şekilde düzenlenmiştir). İçteki döngü while yapısı ile oluşturulmuştur. Bu yapı ile sütun sayısı, dıştaki döngünün o anki değeri kullanılarak dinamik şekilde belirlenmiştir. Buradaki koşul (sayi<=i) ile sayının değeri i'den küçük olduğu sürece sayının alacağı değerler ekrana aralarında bir boşluk eklenerek yazdırılmıştır. Döngü denetimi yine sayı isimli değişken değeri ile sağlanmaktadır.

Çalışma Sorusu: Örnek 7.10'u aşağıdaki ekran görüntüsünü verecek şekilde düzenleyiniz.

```
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2
10 9 8 7 6 5 4 3
10 9 8 7 6 5 4
10 9 8 7 6 5
10 9 8 7 6
10 9 8 7
10 9 8
10 9
10
```


BÖLÜM ÖZETİ

Döngüler, programlama dillerinde yinelemeli işlemlerin gerçekleştirilmesinde temel bir role sahiptir. Programlarda tekrar eden işlemlerin etkin bir şekilde uygulanması gerektiğinde döngü yapıları kullanılmaktadır. Döngüler, programcılara daha esnek ve dinamik bir kod geliştirme imkânı sunarak, karmaşık algoritmaların daha sade bir biçimde ifade edilmesine olanak tanır.

Programlama süreçlerinde döngüler, sabit veya koşullu yapılar şeklinde tercih edilebilir. **Koşullu döngüler**, belirli bir koşul sağlandığı sürece çalışmaya devam eden yapılar olarak tanımlanır. Bu koşul, döngü oluşturulurken sabit bir değer olarak belirlenebileceği gibi, programın çalışma esnasında kullanıcı tarafından girilen dinamik bir değerle de şekillendirilebilir. **Sabit döngüler** ise döngünün yineleme sayısının önceden tanımlandığı yapılardır. Bu tür döngüler, belirli sayıda işlem gerçekleştirilmesi gerektiği durumlarda kullanılmaktadır.

Döngü yapıları, algoritmaların verimli bir şekilde uygulanması ve programların performansının artırılması açısından programlama dillerinin vazgeçilmez bir bileşenidir.

GÖZDEN GEÇİRELİM

1. Belirli bir koşul altında, bir döngüyü sonlandırmak için aşağıdaki komutlardan hangisi kullanılır?

- a) continue
- b) quit
- c) leave
- d) go
- e) break

2. Adım sayısı bilinen yinelemeli bir işlem için aşağıdaki döngülerden hangisi kullanılmalıdır?

- A) for
- b) while
- c) do-while
- d) while-while
- e) do-while-while

3. Aşağıdaki döngülerden hangisi döngünün devam koşulunu döngü sonunda kontrol eder?

- A) for
- b) while
- c) do-while
- d) for-while
- e) for-for

4. Program döngü içerisinde continue komutu ile karşılaşır aşağıdaki ifadelerden hangisi(leri) gerçekleşir.

I. Komuttan sonra gelen hiçbir ifade çalıştırılmaz döngü bir sonraki değer ile çalışmaya devam eder

II. Komuttan sonra gelen ifadeler bir defa çalıştırılır ve döngü sonlandırılır

III. Döngü hemen sonlandırılır

- a) Yalnız I
- b) Yalnız II
- c) Yalnız III
- d) II ve III
- e) Hiçbiri

5. Aşağıdaki program çalıştırıldığında aşağıdakilerden hangisi gerçekleşir?

```
#include<iostream>
using namespace std;
int main () {
    for(int i=1;i<10;i++){
        if(i==5 );
        break;
    }
    cout<<i;
```

- a) 1 2 3 4 5
- b) 1 2 3 4 5 6 7 8 9 10
- c) 5 10
- d) 5
- e) Program derleme zamanında i değişkeni döngü etki alanı dışında olduğu için hata verir.

6. Aşağıdaki program çalıştırıldığında ekranda gözükecek ifade aşağıdakilerden hangisidir?

```
#include<iostream>
using namespace std;
int main () {
    for(int i=1;i<=10;i++){
        if(i==5 || i==7)
            continue;
        cout<<i<<" ";
    }
}
```

- a) 1 2 3 4 5 6 7 8 9 10
b) 1 2 3 4 6 8 9 10
c) 1 2 3 4 5
d) 5 7
e) 5

7. Aşağıdaki program çalıştırıldığında ekranda gözükecek ifade aşağıdakilerden hangisidir?

```
#include<iostream>
using namespace std;
int main () {
    int a=0;
    int b;
    while(a<3){
        b=1;
        while(b<3){
            cout<<a<<" "<<b<<endl;
            b+=2;
        }
        a=3;
    }
}
```

- a) 0 1 3
b) 0 0 0
c) 0 2 3 3
d) 0 1
e) 0 1 1 2 2 2 3 3 3 3

8. Aşağıdaki program çalıştırıldığında ekranda gözükecek ifade aşağıdakilerden hangisidir?

```
#include<iostream>
using namespace std;
int main () {
    for(int i=5;i>3;i--){
        cout<<i<<" ";
    }
}
```

- a) 5 4 3 2 1
b) 1 2 3
c) 3
d) 5 4
e) 4 5

9. Aşağıdaki programda verilen while döngüsü kaç defa çalışır?

```
#include<iostream>
using namespace std;
int main () {
    int i=0;
    while(i){
        cout<<"Ankara\n";
        i++;
    }
}
```

- a) Sonuz döngü (infinite-loop)
b) Koşul sağlanmaz program döngü içine girmez
c) 1
d) 2
e) 3

10. Aşağıdaki ifadelerden hangisi bir döngü yapısı değildir?

- I. for
II. while
III. if
IV. do-while
a) Yalnız I
b) Yalnız II
c) Yalnız III
d) Yalnız IV
e) Hiçbiri

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

BÖLÜM 8

KORELASYON VE REGRESYON ANALİZLERİ

ANAHTAR KAVRAMLAR

Kullanıcı-Tanımlı Fonksiyon, Fonksiyon Aşırı Yükleme, Referanslar, Fonksiyon Şablonları, Özyinelemeli Fonksiyon, Rasgele Sayı Üretimi, Varsayılan Argüman



ÖĞRENME HEDEFLERİ

- 1 Kullanıcı tanımlı fonksiyon yazabilir
- 2 Matematik kütüphanesi fonksiyonlarını kullanabilir
- 3 Rasgele sayı üretebilir
- 4 Kapsam kurallarını uygulayabilir
- 5 Referans parametrelerini kullanabilir
- 6 Varsayılan argüman kullanabilir
- 7 Fonksiyon aşırı yükleme gerçekleştirebilir
- 8 Fonksiyon şablonları oluşturabilir
- 9 Özyinelemeli fonksiyon yazabilir ve kullanabilir

GİRİŞ

Uygulamaları hayata geçirmek için yazılan programlar büyüklük açısından fark-

lılıklar gösterir. Büyük programları oluşturmak için daha küçük kod parçaları yazmak ve bu parçaları birleştirerek çözüme ulaşmak gerekebilir. Bu bölümde program yazarken kullandığımız fonksiyonlar konusu anlatılacaktır. Öncelikli olarak C++'da program parçalarından bahsedilecek, daha sonra C++ standart kütüphanesinde yer alan matematik fonksiyonları gösterilecektir. Sonraki bölümlerde fonksiyon prototipleri ve nasıl kullanıldıkları örnekler ile anlatılacaktır. Rasgele sayı üretimi devam eden kısımlarda gösterilecek bir diğer konudur. Oyun programlarında veya simülasyon içeren programlarda şans faktörünü ifade etmek için rasgele sayı üretimi, bölümde yer alacak diğer bir alt başlık olacaktır. Kapsam kurallarında, farklı bloklarda tanımlanan değişkenlerin hangi öncelikler çerçevesinde erişilebileceği gösterilecektir. Global ve yerel değişkenlerle beraber statik değişkenler de bu konunun bir parçasıdır. Fonksiyonlar bölümünde değinilecek bir başka konu da fonksiyon aşırı yükleme olayıdır. Bu şekilde aynı isim ile birden fazla fonksiyon tanımlamasının ne şekilde yapılacağı açıklanacaktır. İlgili bir konuda fonksiyon şablonlarıdır. Tek bir tanım ile farklı veri türleri ile kullanılacak fonksiyonlar oluşturulacaktır. Bir problemi daha iyi bir şekilde çözüme kavuşturmak için kendi kendini çağıran fonksiyonlara ihtiyaç duyulabilir. Bu fonksiyonlara özyinelemeli fonksiyonlar ismi verilir ve bu konu bölümün son konusu olacaktır.



1. C++'DA PROGRAM PARÇALARI

Program yazmamızın temel amacı karşılaştığımız problemlere çözüm oluşturmaktır. Bazı problemleri çözmek için birden fazla alt probleme çözüm getirmek gerekebilir. Her bir alt problem için farklı bir kısmi kod oluşturmak mümkündür. Bazen bir alt problem için yazdığımız çözüm, programımızın farklı noktalarında birden fazla kez kullanılmak durumunda olabilir. Aynı kod parçasını programın farklı noktalarına tekrar tekrar yerleştirmek kod büyüklüğünü gereksiz yere artıracaktır. Aynı zamanda programımızda yer alan gereksiz kod miktarı hata yapma ihtimalini artırmakta, dayanıklı bir program oluşturmayı önlemektedir. Dolayısı ile bir alt probleme getirilen çözüm sadece bir kez yazılıp, programda ihtiyaç olan noktalarda kullanılabilir. Bu kullanım için yazılan kod parçasının çağırılması gerekir. Alt probleme getirilen çözüm bir fonksiyon olarak oluşturulur. Fonksiyonun sadece bir kez tanımlanması yeterlidir. Programın içerisinde defalarca çağırılarak kullanılabilir. Temel olarak fonksiyonlar bir programın gerçekleştirmesi gereken işlemin belirli parçalarını yapan bağımsız birimler olarak tanımlanırlar. C++ kütüphanelerinde tanımlanan kullanıma hazır fonksiyonlar mevcuttur. Ancak kullanıcılar isterlerse kendi fonksiyon tanımlamalarını gerçekleştirip programlarında kullanabilirler. Math Kütüphanesi Fonksiyonları C++ programlama dilinde yer alan matematik kütüphanesi bazı genel matematiksel hesaplamalar için kullanılan fonksiyonları içerir. Geliştiricilerin bu fonksiyonları kullanabilmeleri için `<cmath>` kütüphanesini programlarına eklemeleri gerekir. Örneğin, xy hesaplamasını gerçekleştirmek için `pow(xy)` fonksiyonu kullanılır. Eğer bu fonksiyonu `pow(2,3)` şeklinde çağırırsak sonuç 8 olarak döner. Aşağıda verilen Tablo 8.1'de matematik kütüphanesinde yer alan bazı önemli fonksiyonlar ve kullanımları verilmiştir.

2. MATH KÜTÜPHANESİ FONKSİYONLARI

C++ programlama dilinde yer alan matematik kütüphanesi bazı genel matematiksel hesaplamalar için kullanılan fonksiyonları içerir. Geliştiricilerin bu fonksiyonları kullanabilmeleri için `<cmath>` kütüphanesini programlarına eklemeleri gerekir. Örneğin, xy hesaplamasını gerçekleştirmek için

pow(xy)

fonksiyonu kullanılır. Eğer bu fonksiyonu `pow(2,3)` şeklinde çağırırsak sonuç 8 olarak döner. Aşağıda verilen Tablo 8.1'de matematik kütüphanesinde yer alan bazı önemli fonksiyonlar ve kullanımları verilmiştir.

Tablo 8.1 Matematik Kütüphanesi Fonksiyonları

Fonksiyon	Tanım	Örnek
<code>pow(x, y)</code>	x'in y. kuvvetini hesaplar (xy)	<code>pow(2, 5)</code> Sonuç:32
<code>sqrt(x)</code>	x'in karekökünü hesaplar	<code>sqrt(25.0)</code> Sonuç:5
<code>log(x)</code>	x'in doğal logaritması (e tabanında)	<code>log(15)</code> Sonuç: 2.70805
<code>log10(x)</code>	x'in logaritması (10 tabanında)	<code>log10(100)</code> Sonuç: 2
<code>fmod(x,y)</code>	x/y işleminin kalan değerini bulur	<code>fmod(2.48,1.2)</code> Sonuç: 0.08
<code>ceil(x)</code>	x'in değerini yukarıya yuvarlar	<code>ceil(2.7)</code> Sonuç: 3
<code>floor(x)</code>	x'in değerini aşağıya yuvarlar	<code>floor(2.7)</code> Sonuç: 2
<code>exp(x)</code>	üstel fonksiyon ex	<code>exp(4)</code> Sonuç: 54.5982
<code>fabs(x)</code>	x'in mutlak değerini hesaplar	<code>fabs(-2.45)</code> Sonuç: 2.45

3. FONKSİYON PROTOTİPLERİ

Program geliştiriciler kullanıcı tanımlı fonksiyonlar oluşturabilir ve bu fonksiyonu programlarında kullanabilirler. Aşağıdaki programda kullanıcı tanımlı maksimum fonksiyonu tanımlanmış ve kullanılmıştır. *maksimum* fonksiyonu 3 tam sayı parametre alır. Bu sayılardan en büyük değere sahip olanı bulunur ve çevrilir. Bir fonksiyonun prototipinde, döndürülen değerin veri türü, fonksiyonun ismi ve parantez içinde argümanların türleri ve isimleri sırasıyla yazılır. Eğer fonksiyon herhangi bir değer döndürmüyorsa döndürülen değerin türü void olarak yazılır. Bir fonksiyonun prototipi derleyiciye fonksiyonun ismini, dönüş türünü ve parametrelerin sıralı olarak türlerini söyler. Fonksiyonun prototipi aynı zamanda fonksiyonun tanımının ilk satırıyla birebir aynıdır. Tek fark prototipin noktalı virgül ile bitmesidir.

```
int maksimum(int sayi1, int sayi2, int sayi3);
```

Fonksiyonun tanımında, beklenen görevi tamamlamak için gerekli tüm işlemler gerçekleştirilir. Bu sayısal bir hesaplama olabileceği gibi sadece ekrana bilgi yazdıran bir görev de olabilir. Fonksiyonun tanımı, fonksiyon başlığından sonra kıvrımlı parantezler içerisine yerleştirilir.

```
//Program8.1.cpp
#include <iostream>
#include <iomanip>
using namespace std;
int maksimum(int sayi1, int sayi2, int sayi3);
int main()

{
    int i1, i2, i3;
    cout<<"3 tam sayi giriniz: ";
    cin>>i1>>i2>>i3;
    cout<<"Maksimum sayi: "<<maksimum(i1,i2,i3)<<endl;
    return 0;
}

int maksimum (int sayi1, int sayi2, int sayi3)
{
    int mSayi;
    if(sayi1>sayi2 && sayi1>sayi3)
        mSayi=sayi1;
    if(sayi2>sayi1 && sayi2>sayi3)
        mSayi=sayi2;
    if(sayi3>sayi1 && sayi3>sayi2)
        mSayi=sayi3;
    return mSayi;
}
```

Program8.1 Çıktısı:

3 tam sayi giriniz: 32 45 28

Maksimum sayi: 45

Program8.1.cpp'de *main* fonksiyonda 3 tam sayı değişkeni tanımlanmış ve kullanıcıdan bu değişkenler için değer girmesi istenmiştir. Daha sonra kullanıcının girdiği değerler argüman olarak kullanılmış ve *maksimum* fonksiyonu çağırılmıştır. Fonksiyonun çevirdiği değer *cout* ile ekrana yazdırılmıştır.

maksimum fonksiyonunun tanımında ise parametre olarak gelen 3 tam sayı içerisinde en büyük değer hesaplanmış ve *main* fonksiyona döndürülmüştür. En büyük değeri hesaplamak için farklı yöntemler kullanılabilir. Burada 3 farklı *if* ifadesi ile en büyük değer bulunmuştur. Her bir *if* ifadesi bir tam sayı değişkeni diğer 2 değişken ile karşılaştırmış ve söz konusu değişken diğer 2 değişkenden de büyükse en büyük değer olarak belirlenmiştir. Bu fonksiyon açısından tüm tam sayı değerlerin birbirinden farklı olduğunu kabul edebiliriz. Herhangi 2 değer birbirine eşit olmayacaktır şeklinde ön kabulümüz olabilir.



DİKKAT EDELİM

Bir fonksiyonun prototipinde, tanımlanmasında ve çağırılmasında kullanılan parametrelerin/argümanların sayısı, türleri ve sıraları birbirleri ile uyumlu olmalıdır.

Rasgele Sayı Üretimi

Bazı uygulamalar için yazılan programlarda rasgele sayılar üretmek gerekir. Simülasyon içeren veya bazı oyun programlarında şans faktörü uygulamanın bir parçasıdır. C++ programlama dilinde rasgele tamsayılar üretmek için *rand()* fonksiyonu kullanılır.

```
i=rand();
```

rand() fonksiyonu 0 ile *RAND_MAX* arasında bir tamsayı çevirir. *RAND_MAX* *<cstdlib>* kütüphanesi içerisinde tanımlanmıştır. Farklı programlar farklı rastgele sayı aralıklarına ihtiyaç duyarlar. Örneğin yazı-tura simülasyonu sadece 2 farklı rastgele sayıya ihtiyaç duyarken, zar atma simülasyonu 6 farklı değer içerisinde rasgele bir seçim gerektirir. Program8.2.cpp'de 15 defa zar atma işlemi gerçekleştirilmiş ve elde edilen değerler çıktı olarak verilmiştir. Program8.3.cpp'de ise 24000 kez zar atılmış ve bir zar yüzünün kaç kez tekrarladığı ekrana yazdırılmıştır.

```
//Program8.2.cpp
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
```

```
int main()
{
    int i;
    for(i=1; i<=15; i++)
    {
```

```

    cout<<setw(5)<<(1+rand()%6);
    if(i%3 == 0)
        cout<<endl;
}
return 0;
}

```

Program8.2.cpp Çıktısı

```

6 4 3
6 1 1
5 1 3
2 4 6
4 6 1

```

```

//Program8.3.cpp
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
int main()
{
    unsigned int frekans1=0;
    unsigned int frekans2=0;
    unsigned int frekans3=0;

```



ARAŞTIRALIM

rand fonksiyonunu kullanan bir programın her çalıştırıldığında farklı düzende rasgele sayı üretebilmesi için *srand* fonksiyonunu araştıralım.



GÜNLÜK HAYATLA İLİŞKİLENDİRELİM

Günlük hayatta karşılaştığımız hangi oyunların programlarını yazarken *rand* fonksiyonu gerekir?

4. KAPSAM KURALLARI

Bir fonksiyonda değişkenlere erişim kuralları, değişkenlerin tanımlandığı konuma göre belirlenir.

Değişkenler sadece tanımlandıkları fonksiyon veya blok içerisinde doğrudan kullanılabilirler. Bu değişkenlere yerel değişkenler denir. İlgili fonksiyon veya blok çalıştığında bu değişkenler için bellekte yer ayrılır. Fonksiyon veya blok sonlandığında ayrılan bu alan iptal olur ve bellekteki değişken silinir. Aynı fonksiyon veya aynı blok daha sonra tekrar çağrılrsa dahi değişkenler daha önceki değerlerine sahip olamazlar. main fonksiyonda veya diğer fonksiyonlarda tanımlanan tüm değişkenler yerel değişken olarak değerlendirilir.

Eğer tanımlanan bir değişkenin programda yer alan tüm fonksiyonlarda kullanılabilmesi amaçlanıyorsa bu değişkenler global değişken olarak tanımlanır. Tanımlanma işlemi fonksiyonların dışında gerçekleştirilir.

```
//Program8.4.cpp
#include <iostream>
using namespace std;
void local();
void staticLocal();
void global();
int i=1;
int main()
{
    cout<<"main fonksiyonda global degisken i: "<<i<<endl;
    int i=5;
    cout<<"main fonksiyonda yerel degisken i: "<<i<< endl;
    {
        int i=7;
        cout<<"main fonksiyonda ic blokda yerel degisken i: "<<i<<endl;
    }
    cout<<"main fonksiyonda yerel degisken i: "<<i<<endl;
    local();
    staticLocal();
    global();
    local();
    staticLocal();
    global();
    cout<<"main fonksiyonda yerel degisken i: "<<i<<endl;
    return 0;
}
void local()
{
    int i=10;
    cout<<"\nyerel degisken i "<<i<< " local fonksiyonu baslarken"<<endl;
    i++;
```

```

    cout<<"yerel degisken i "<<i<<" local fonksiyonu biterken"<<endl;
}
void staticLocal()
{
    static int i=100;
    cout<<"\nyerel statik degisken i "<<i<<" staticLocal fonksiyonu baslarken "<<endl;
    i++;
    cout<<"yerel statik degisken i "<<i<<" staticLocal fonksiyonu biterken"<<endl;
}
void global()
{
    cout<<"\nglobal degisken i "<<i<<" global fonksiyonu baslarken"<<endl;
    i *= 20;
    cout<<"global degisken i "<<i<<" global fonksiyonu biterken"<<endl;
}

```

Program8.4.cpp Çıktısı

main fonksiyonda global degisken i: 1
 main fonksiyonda yerel degisken i: 5
 main fonksiyonda ic blokda yerel degisken i: 7
 main fonksiyonda yerel degisken i: 5

yerel degisken i 10 local fonksiyonu baslarken
 yerel degisken i 11 local fonksiyonu biterken

yerel statik degisken i 100 staticLocal fonksiyonu baslarken
 yerel statik degisken i 101 staticLocal fonksiyonu biterken

global degisken i 1 global fonksiyonu baslarken
 global degisken i 20 global fonksiyonu biterken

yerel degisken i 10 local fonksiyonu baslarken
 yerel degisken i 11 local fonksiyonu biterken

yerel statik degisken i 101 staticLocal fonksiyonu baslarken
 yerel statik degisken i 102 staticLocal fonksiyonu biterken

global degisken i 20 global fonksiyonu baslarken
 global degisken i 400 global fonksiyonu biterken
 main fonksiyonda yerel degisken i: 5



DİKKAT EDELİM

Kapsam kurallarına dikkat edilmeden gerçekleştirilen değişken isimlendirmeleri, programın doğru olmayan sonuçlar üretmesine sebep olabilir.

5. REFERANSLAR VE REFERANS PARAMETRELERİ

Birçok programa dilinde parametreler fonksiyonlara iki farklı yöntem ile gönderilir. Birinci yöntem değer ile gönderim, ikinci metot ise referans ile gönderimdir. Bu bölümde ilk olarak gösterdiğimiz yaklaşım değer ile gönderimdir. Fonksiyona iletmek istenen parametrenin bellekte bir kopyası yapılır ve fonksiyon içerisinde bu kopya kullanılır. Dolayısı ile bu parametre için fonksiyonun içerisinde gerçekleştirilecek işlemlerden/değişikliklerden fonksiyonu çağıran diğer fonksiyondaki (genellikle main program/fonksiyon) orijinal değişken etkilenmez. İkinci yaklaşım olan referans ile gönderimde çağrılan fonksiyona gönderilen orijinal parametrenin bir kopyası oluşturulmaz. Fonksiyon doğrudan orijinal değişkene erişip, işlem yapma ve değiştirme hakkına sahip olur. Bu durumu gerçekleştirebilmek için referans parametreler kullanılır. Fonksiyon ve bu fonksiyonu çağıran fonksiyon ilgili parametreye doğrudan erişim hakkına sahiptir. Aslında bu değişken bellekte sadece bir defa yer almaktadır ve bu bellek alanı orijinal değişken ismi ile çağırılan fonksiyonda, referans parametre ile çağrılan fonksiyonda erişilebilmektedir.

C++ referans ile gönderim için referans parametrelerine sahiptir. Referans ile gönderim ile çağrılan fonksiyona çağırılan fonksiyonun verisine doğrudan erişim ve değiştirme hakkı verilir. Referans parametresi bir fonksiyon çağrıldığında karşılık gelen argümana bir takma ad olarak değerlendirilebilir. Bu durumu göstermek için, farklı bir ifade ile parametrenin referans ile gönderim metoduyla aktarıldığını ifade etmek için '&' sembolü kullanılır.

```
int &sayi;
```

Bu ifade; sayi değişkeni bir tam sayı değere referanstır şeklinde okunur. Fonksiyon çağrılırken orijinal değişken sadece ismi ile kullanılır. Çağrılan fonksiyonda sayi değişkeni referans olarak tanımlandığı için çağırılan fonksiyondaki orijinal değişkene doğrudan erişim sağlayabilir.

Program8.5.cpp değer ve referans ile gönderim yaklaşımları arasındaki farkı göstermek için yazılmıştır.

```
//Program8.5.cpp
#include <iostream>
using namespace std;
void fonksiyon1(int);
void fonksiyon2(int&);
int main()
{
    int n;
    cout<<"Bir tam sayi giriniz:\n";
    cin>>n;
```



```

fonksiyon1(n);
cout<<n<<endl;
fonksiyon2(n);
cout<<n<<endl;
return 0;
}
void fonksiyon1(int sayi)
{
    sayi = sayi + 5;
}
void fonksiyon2(int& sayi)
{
    sayi = sayi * 4;
}

```

Program8.5.cpp Çıktısı

Bir tam sayi giriniz:

10

10

40

**DİKKAT EDELİM**

Referans ile bir fonksiyona gönderilen bir parametrenin değerinde yapılan değişiklik, orijinal değışkide de gerçekleşecektir.

**ARAŞTIRALIM**

Referans ile gönderim yaklaşımında gerçekleşebilecek istenmeyen değışiklikleri önlemek amacıyla const kullanımını araştıralım.

6. VARSAYILAN ARGÜMANLAR

Bazı fonksiyonlar, aynı argüman için aynı değerle defalarca çağrılabilirler. Bu durumlarda varsayılan argüman belirlenebilir ve kullanılabilir. Varsayılan argüman fonksiyon çağrılırken tekrar yazılmak zorunda değildir. Ancak ilgili argüman için farklı bir değer kullanılmak istenirse, fonksiyon çağrılırken yeni değer parametre olarak kullanılabilir. Bu durumda varsayılan argüman değil, gönderilen parametre fonksiyonun içerisinde kullanılacaktır. Varsayılan argüman atama operatörü (0) ile varsayılan değerine sahip olur. Program8.6.cpp'de varsayılan argümanları içeren bir fonksiyonun kullanıldığı program yer almaktadır.

```
//Program8.6.cpp
#include <iostream>
using namespace std;
int alanDiktortgen(int a=1, int b=1)
{
    return a*b;
}

int main()
{
    cout<<"Varsayılan Argumanlar ile Diktortgen Alani:"<<alanDiktortgen()<<endl;
    cout<<"Kenarlardan bir tanesinin verildiği Diktortgen Alani:"<<alanDiktortgen(5)<<endl;
    cout<<"Her iki kenarin belirlendiği Diktortgen Alani:"<<alanDiktortgen(5,10)<<endl;
    return 0;
}
```

Program8.6.cpp Çıktısı:

```
Varsayılan Argumanlar ile Diktortgen Alani:1
Kenarlardan bir tanesinin verildiği Diktortgen Alani:5
Her iki kenarin belirlendiği Diktortgen Alani:50
```

7. TEKLİ KAPSAM ÇÖZÜMLEME OPERATÖRÜ

Bir kapsamda aynı isimde yerel bir değişken olduğu durumda, global değişkene erişmek amacıyla Tekli Kapsam Çözümleme Operatörü (::) kullanılır. Bu operatöre sadece yerel değişken ve global değişken aynı isimde ise ve global değişken kullanılmak isteniyorsa ihtiyaç vardır. Eğer global değişkenin ismi kapsamda yer alan yerel değişkenlerden farklı ise global değişken ilgili operatör olmadan doğrudan erişilebilir ve kullanılabilir. Tekli Kapsam Çözümleme Operatörü sadece global değişkene erişmek için kullanılır, bir dış kapsamdaki yerel değişkene erişmek için kullanılmaz. Program8.7.cpp isimli programda bu operatörün kullanımı gösterilmektedir.

```
//Program8.7.cpp
#include <iostream>
using namespace std;
int i=1;
int main()
{
    int i=5;
```

```

    cout<<"Yerel degisken: "<<i<<endl;
    cout<<"Global degisken: "<<::i<<endl;
return 0;
}

```

Program8.7.cpp Çıktısı

Yerel degisken: 5

Global degisken: 1

8. FONKSİYON AŞIRI YÜKLEME

Bazı benzer işlemler için, farklı fonksiyonlar tanımlamak gerekebilir. Bu durumlarda program geliştirici aynı fonksiyon ismini kullanmak isteyebilir. Farklı ancak benzer işlemler gerçekleştiren fonksiyonlar aynı isimde tanımlanabilirler. Bu tanımlamayı sağlayabilmek için fonksiyonların farklı imzalara sahip olması gerekir. Bu kavram fonksiyon aşırı yükleme olarak adlandırılır. Bir fonksiyonun imzası, fonksiyonun ismi ile beraber argümanların sayısı, türü ve sırası bilgilerini içerir. C++ derleyici uygun fonksiyonu, bu bilgileri kullanarak seçer. Matematik kütüphanesindeki fonksiyonlar için aşırı yükleme uygulanmıştır. Bu fonksiyonlar float, double ve long double türleri için tanımlanmışlardır.

Program8.8.cpp’de aşırı yükleme gerçekleştirilen topla fonksiyonları sırasıyla 2 tam sayı ve 2 kesirli sayıyı toplamak için kullanılmaktadır. main fonksiyonda ilk çağrılan topla fonksiyonu int argüman aldığı için, parametreleri int olan fonksiyon kullanılır. İkinci olarak çağrılan fonksiyondaki argümanlar double olduğu için, double parametrelere sahip olan fonksiyon tanımı kullanılır.

```

//Program8.8.cpp
#include <iostream>
using namespace std;
int topla(int sayi1, int sayi2)
{
    cout<<"İki tam sayinin toplamı: ";
    return sayi1 + sayi2;
}
double topla(double sayi1, double sayi2)
{
    cout<<"İki kesirli sayinin toplamı: ";
    return sayi1 + sayi2;
}
int main()
{
    int a=5, b=8;
    double c=7.5, d=9.2;
    cout<<topla(a,b)<<endl;
    cout<<topla(c,d);
    return 0;
}

```

Program8.8.cpp Çıktısı

İki tam sayinin toplamı: 13

İki kesirli sayinin toplamı: 16.7

9. FONKSİYON ŞABLONLARI

Aşırı yükleme fonksiyonlar farklı veri türleri üzerinde, farklı program mantığı kullanan benzer işlemler içerirler. Program mantığı ve işlemler farklı veri türleri için aynı ise aşırı yükleme işlemi yerine fonksiyon şablonları kullanılabilir. Aslında fonksiyon şablonları, fonksiyon aşırı yüklemenin daha kompakt bir şekilde yazılmasıdır. Bir başka ifade ile sadece bir fonksiyon tanımlayarak bir aşırı yükleme fonksiyon ailesini tanımlamış oluruz.

Aşağıda verilen kod, 3 parametreden en büyük değeri çeviren bir fonksiyon şablonu için yazılmıştır. Bütün fonksiyon şablonlarının tanımlanması template anahtar kelimesi ile gerçekleştirilir. Devamında üçgen parantezler içerisinde şablon parametre listesi sıralanır. Bu listede yer alan parametrelerin başında typename veya class anahtar kelimeleri yer alır. Burada yer alan tür parametreleri temel veri türleri veya kullanıcı tanımlı veri türleri için bir yer tutucu olarak değerlendirilebilirler. Genellikle büyük harf kullanılarak tanımlanırlar. Bizim örneğimizde T harfi kullanılmıştır. Tür parametreleri fonksiyonun parametrelerini belirtmede, fonksiyonun dönüş değerini ifade etmede ve fonksiyonun içerisinde yerel değişkenleri tanımlamada kullanılabilirler. Şablon fonksiyonu normal fonksiyonlar gibi tanımlanırlar, ancak tür parametrelerini gerçek veri türlerini temsil etmek için kullanırlar.

```
template <typename T>
T maksimum (T v1, T v2, T v3)
{
    T m = v1;
    if(v2>m)
        m= v2;
    if(v3>m)
        m = v3;
    return m;
}
```

Bu örnekte şablon sadece bir tür parametresi tanımlıyor. Parametre listesinde sadece farklı bir tür parametresi var. Derleyici programın kaynak kodunda maksimum fonksiyonunun çağrıldığını gördüğünde tür parametresini ilgili belirlenen tür ile değiştirir. Buradaki örnekte tüm tür parametreleri aynı olduğu için (T), tüm parametreler aynı tür olmalıdır. Program8.9.cpp’de yer alan main fonksiyon şablon fonksiyonunu 3 farklı veri türü ile çağırmaktadır (int, double ve char).

```
//Program8.9.cpp
#include <iostream>
using namespace std;
template <typename T>
T maksimum (T v1, T v2, T v3)
{
    T m = v1;
    if(v2>m)
        m= v2;
    if(v3>m)
        m = v3;
    return m;
}
```

```

}
int main()
{
    int i1, i2, i3;
    double d1, d2, d3;
    char c1, c2, c3;

    cout<<"3 tam sayi giriniz:\n";
    cin>>i1>>i2>>i3;

    cout<<"3 ondalikli sayi giriniz:\n";
    cin>>d1>>d2>>d3;

    cout<<"3 karakter giriniz:\n";
    cin>>c1>>c2>>c3;

    cout<<"Tam sayilarin maksimum olan degeri: "<<maksimum(i1,i2,i3)<<endl;
    cout<<"Kesirli sayilarin maksimum olan degeri: "<<maksimum(d1,d2,d3)<<endl;
    cout<<"Karakterlerin maksimum olan degeri: "<<maksimum(c1,c2,c3)<<endl;
    return 0;
}

```

Program8.9.cpp Çıktısı

```

3 tam sayi giriniz:
4 2 7
3 kesirli sayi giriniz:
2.3 5.4 1.2
3 karakter giriniz:
c a b
Tam sayilarin maksimum olan degeri: 7
Kesirli sayilarin maksimum olan degeri: 5.4
Karakterlerin maksimum olan degeri: c

```

9. ÖZYİNELEMELİ FONKSİYONLAR

Bazı problemlerde fonksiyonların kendilerini çağırımları daha kullanışlı olabilir. Özyinelemeli fonksiyonlar kendilerini ya doğrudan ya başka bir fonksiyon üzerinden çağırırlar. Sadece main fonksiyon/ program başka bir fonksiyon içerisinden çağrılmaz. Öncelikle özyineleme kavramı üzerinde tartışalım. Özyinelemeli problem çözme yaklaşımı bazı temel ortak öğelere sahiptir. Bir özyinelemeli fonksiyon çağrıldığında bu fonksiyon sadece temel durumu çözmeyi bilir. Eğer fonksiyon bu temel durumu çözmek için çağırılmışsa, cevabı çevirir. Eğer daha karmaşık bir durum için çağırılmışsa, özyinelemeli fonksiyon problemi iki parçaya ayırır. Birinci kısım fonksiyonun çözümünü bildiği kısım, ikinci kısım ise fonksiyonun problemin büyüklüğünü azaltıp kendisini çağırdığı parçadır. İkinci kısım aslında orijinal problemin aynısı veya çok benzeridir. Sadece bazı ölçütler açısından daha küçük bir

problemdir. Bu kısım için fonksiyonun tekrar çağırılması özyineleme adımı olarak değerlendirilir. Özyineleme adımı çalışırken, orijinal fonksiyon açıktır. Başka bir ifade ile çalışmasını tamamlamamıştır. Bu durumda dikkat edilecek bir nokta da özyineleme adımlarının sayısının birden fazla olabileceğidir. Dolayısı ile özyineleme fonksiyon çağırılmasında birçok açık, tamamlanmamış fonksiyon söz konusu olabilir. Bu özyineleme fonksiyon çağırılma sürecinde, orijinal problem giderek küçülür ve temel probleme ulaşır. Temel problemin çözümü de fonksiyon tarafından bilindiği için doğrudan çevrilir. Tabii ki bu çevrilme en son özyineleme fonksiyonunu çağırarak fonksiyon içine olur. Döndürülen bu değer çözümü bilinen birinci kısım ile birleştirilerek bir önceki fonksiyona döndürülür. Bir örnek olarak faktöriyel probleminin çözümünü tartışalım.

Bir pozitif tamsayının faktöriyeli aşağıdaki şekilde tanımlanır:

$$n! = n * (n-1) * (n-2) \dots 1$$

$$n! = n * (n-1)!$$

Burada görüldüğü gibi orijinal problemimizin büyüklüğü n değerinden $n-1$ e düşmüştür. Eğer $(n-1)!$ değerini bilirsek bu çözümü n ile çarparak $n!$ değerini hesaplayabiliriz. Sayısal bir örnek aşağıdaki gibi verilebilir.

$$5! = 5 * 4 * 3 * 2 * 1$$

$$5! = 5 * 4!$$

Program8.10.cpp özyinelemeli fonksiyon kullanarak, kullanıcıdan alınan bir değerın faktöriyelini hesaplar. Hesaplanan değer ekrana yazdırılmaktadır.

```
//Program8.10.cpp
#include <iostream>
#include <iomanip>
using namespace std;
unsigned long factorial(unsigned long);
int main()
{
    int n;
    cout<<"Bir tam sayi giriniz:\n";
    cin>>n;
    cout<<factorial(n)<<endl;
    return 0;
}
unsigned long factorial (unsigned long number)
{
    if(number==0 || number==1)
        return 1;
    else
        return number * factorial(number-1);
}
```

Program8.10.cpp Çıktısı

Bir tam sayi giriniz:

7

5040



DİKKAT EDELİM

Temel problemi doğru tanımlanmamış özyinelemeli fonksiyonlar sonsuz döngüye girip asla sonlanmayabilirler.

BÖLÜM ÖZETİ

Fonksiyonlar bir programda problemin çözümüne yönelik gerçekleştirilen işlemlerin yer aldığı kod parçalarıdır. Bazen bu çözümler program içerisinde birden fazla kez çağrılabilir. Küçük problemlere getirilen bu çözümler, fonksiyon olarak sadece bir kez kodlanırlar ve program içerisinde gerekli noktalarda defalarca çağrılabilirler. matematik kütüphanesi, bazı temel matematiksel işlemler için yazılmış kullanıma hazır fonksiyonları içerir. Örneğin; sqrt fonksiyonu fonksiyona gönderilen argümanın karekökünü hesaplar. C++ program geliştiricileri hazır kütüphane fonksiyonlarını kullanabilecekleri gibi, kendi fonksiyonlarını da tanımlayıp kullanabilirler. Eğer geliştiricinin tanımladığı fonksiyon main fonksiyondan sonra programa eklenirse, main'den önce derleyiciyi bilgilendirmek amacıyla fonksiyonun prototipinin verilmesi gerekir. Fonksiyonun prototipi, fonksiyonun ismi, döndürülen değerin türü, fonksiyonun aldığı argümanların türleri ve sırasından oluşur. Oyun veya simülasyon içeren uygulamalarda şans faktörünü programımıza eklemek gerekir. rand() fonksiyonu 0 ile RAND_MAX arasında bir sayı üretir. Mod operatörü '%' ve rand() kullanarak istenilen aralıkta sayı üretimi gerçekleştirilebilir. Örneğin; $1 + \text{rand}() \% 6$ işlemi 1 ile 6 arasında rasgele sayılar üretir. Bir fonksiyon veya blok içerisinde tanımlanan bir değişken, yerel değişken olarak kabul edilir ve sadece tanımlandığı alanda erişilip kullanılabilir. Fonksiyonların dışında tanımlanan global değişkenler, programda tüm fonksiyonlarda erişilip kullanılabilirler. Bir argüman fonksiyona referans (&) ile gönderilebilir. Bu durumda fonksiyon ilgili değişkenin bellekteki orijinal tanımlandığı alana erişebilir. Fonksiyonda değişken üzerinde gerçekleştirilen işlemler doğrudan orijinal değişken üzerinde gerçekleşir. Bu durum referans ile gönderim olarak adlandırılır. Ancak bir argüman referans ile gönderilmezse, fonksiyon içerisinde ilgili değişkenin bir kopyası oluşturulur ve bellekte farklı bir yere yerleştirilir. Bu değişken üzerinde yapılan değişiklikler orijinal değişkeni etkilemez ve durum değer ile gönderim olarak isimlendirilir. Bir fonksiyon tanımlanırken, bazı argümanlar için varsayılan değerler atanabilir. Fonksiyon çağrılırken ilgili argüman için bir değer gönderilmezse varsayılan değer kullanılır. Bir fonksiyon içerisinde, yerel değişkenler ile oluşabilecek bir karmaşıklık önlemek amacıyla, global değişkene ulaşmak için tekli kapsam çözümleme operatörü (::) kullanılır. Benzer işlemler gerçekleştiren ancak farklı imzalara sahip fonksiyonlar aynı isimde tanımlanabilirler. Derleyici hangi fonksiyonun çağrıldığına, gönderilen parametrelerin sayısına, türlerine ve sıralarına bakarak karar verir. Bu duruma fonksiyon aşırı yükleme adı verilir. Eğer bu fonksiyonlarda kullanılan program mantığı aynı ve fonksiyonlar sadece argümanların veri türleri açısından farklılık gösteriyorsa fonksiyon şablonları kullanılabilir. Bazı uygulamalarda problemin çözümü için özyinelemeli fonksiyonlar kullanılır. Özyinelemeli fonksiyonlar, fonksiyonların problemi daha küçük bir problem haline getirerek kendilerini çağırdıkları yapılardır. Problemin bir çözüme ulaşması için temel problemin doğru bir şekilde tanımlanması gerekir.

GÖZDEN GEÇİRELİM

1. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
int fonksiyon1(int b)
{
    b=b+5;
    return b+7;
}
int main()
{
    int a=5;
    a=fonksiyon1(a);
    cout<<a<<endl;
    return 0;
}
```

- a. 5
- b. 10
- c. 12
- d. 17
- e. 22

2. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
int fonksiyon1(int b)
{
    b=b+5;
    return b+7;
}
int main()
{
    int a=5;
    fonksiyon1(a);
    cout<<a<<endl;
    return 0;
}
```

- a. 5
- b. 10
- c. 12
- d. 17
- e. 22

3. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
int fonksiyon1(int &b)
{
    b=b+5;
    return b+7;
}
int main()
{
    int a=5;
    fonksiyon1(a);
    cout<<a<<endl;
    return 0;
}
```

- a. 5
- b. 10
- c. 12
- d. 17
- e. 22

4. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
void print(int a)
{
    if(a==0)
    {
        cout<<endl;
        return;
    }
    else
    {
        cout<<a%10<<" ";
        return print(a/10);
    }
}
int main()
{
    int i=4562;
    print(i);
    return 0;
}
```

- a. 2 6 5 4
- b. 4 5 6 2
- c. 5 6 4 2
- d. 2 4 5 6
- e. 6 5 4 2

5. Aşağıdakilerden hangisinde fonksiyon prototipi yanlış tanımlanmıştır?

- a. void yazdir();
- b. int sorular();
- c. void soru yazdir(int c);
- d. int yazdir(int a);
- e. void yazdir(int a, int b);

6. Aşağıdakilerden hangisi C++ programlama dilinde int hesaplama(int a, int b) prototipine sahip bir fonksiyonun aşırı yükleme eşlerinden birisi olabilir?

- a. int ortalamaHesaplama(int a, int b)
- b. double hesaplamal(double a, double b)
- c. int hesaplama(int a, int b)
- d. int hesaplamal(int a, int b)
- e. double hesaplama(double a, double b)

7. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
#include <cmath>
using namespace std;
```

```
int main()
```

```
{
```

```
    int a=25;
```

```
    cout<<pow(sqrt(a),3)<<endl;
```

```
    return 0;
```

```
}
```

- a. 5
- b. 25
- c. 125
- d. 250
- e. 625

8. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
```

```
int i=1;
```

```
int main()
```

```
{
```

```
    cout<<i<<" ";
```

```
    i=2;
```

```
    cout<<i<<" ";
```

```
    {
```

```
        int i=3;
```

```
        cout<<i<<" ";
```

```
    }
```

```
    cout<<i<<endl;
```

```
    return 0;
```

```
}
```

- a. 2 1 2 3
- b. 1 2 3 2
- c. 3 1 2 2
- d. 1 3 2 2
- e. 2 1 2 3

9. Aşağıdakilerden hangisinde C++ programlama dilinde double hesaplama(int a, int b) prototipine sahip bir fonksiyon doğru bir şekilde çağrılmamıştır?

- a. int d=heseplama(4,5);
- b. double w=hesaplama(3,2);
- c. cout<<hesaplama(5,8)<<endl;
- d. double k=hesaplama(1,2) + 4.0;
- e. cout<<5.7 + hesaplama(9,3)<<endl;

10. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
void fonksiyon1(int &a)
{
    a=a+10;
}
int fonksiyon2(int a)
{
    int b=a;
    fonksiyon1(b);
    return b;
}
int main()
{
    int i=5;
    cout<<fonksiyon2(i)<<endl;
    return 0;
}
```

a. 5
b. 10
c. 15
d. 20
e. 25

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

Kaynakça

- Deitel, P., Deitel, H. (2017). C++ How to Program. 10. Baskı. Pearson. Essex, İngiltere.
- Dikici, M. (2017). C++ Programlama Dili: Yapısal Programlama, Nesnelerle Programlama ve Jenerik Programlama. Seçkin Yayıncılık. İstanbul, Türkiye.
- Çobanoğlu, B. (2021). C/C++ Programlama Eğitim Kitabı. Kodlab. İstanbul, Türkiye.
- Demir, S., Eryiğit, R., Ar, Y., Tuğrul, B., Gök, O., Kul, S., . . . Yağanoğlu, M. (2019). Programlama Temelleri. Erzurum: Atatürk Üniversitesi Açıköğretim Fakültesi.
- Kaleli, C., Bilge, A., Ergin, S., & Şora Günel, E. (2016). Bilgisayar ve Programlamaya Giriş. Eskişehir: Anadolu Üniversitesi.

BÖLÜM 9

DİZİLER

ANAHTAR KAVRAMLAR

Tek Boyutlu Diziler, İndis, Karakter Dizileri, Sonlandırma Karakteri, Seriler, Arama, Sıralama, Çok Boyutlu Diziler, Matris

ÖĞRENME HEDEFLERİ

-  Dizileri nasıl tanımlayacağını ve dizilerin elemanlarına nasıl erişim sağlanabileceğini bilir
-  Tanımlı dizilerin elemanlarına atanmış değerlerin nasıl güncellenebileceğini bilir
-  Dizilerin fonksiyonlara nasıl aktarılabilceğini bilir
-  Tek boyutlu dizlerin yanı sıra çok boyutlu dizilerin de nasıl tanımlanacağını ve çok boyutlu dizilerin elemanlarına nasıl erişim sağlanabileceğini bilir
-  Dizileri kullanarak arama ve sıralama gibi belirli problemlerin çözümü için algoritmalar geliştirebilir
-  Diziler üzerinde matematiksel işlemler gerçekleştirilebilir

GİRİŞ

C++ programlama dilinde diziler, aynı veri türünden belirli sayıda elemanı bel-

lekte sıralı bir şekilde saklayan yapılardır. Aynı veri türünden çok sayıda elamanın işlendiği durum için diziler, diğer yapılara nazaran daha etkili ve pratik araçlar olarak öne çıkmaktadır. Dizilerde saklanacak elemanların sonlu sayıda ve aynı veri türünde olması gerekmektedir. Dizilerdeki elemanlara erişim ilgili elemanların bellekteki sırasını belirten indisler yoluyla yapılmaktadır. Bu bölümde C++ programlama dilinde tanımlanmış tek boyutlu ve iki boyutlu dizilerden bahsedilecektir.



1. TEK BOYUTLU DİZİLER

Matematikteki vektör kavramının C++ programlama dilinde karşılığı olarak değerlendirilebilecek diziler aşağıdaki kod parçası üzerinden tanımlanmaktadır:

```
veri_turu diziAdi[boyut];
```

Burada veri_turu bellekte saklanacak verilerin türünü, boyut da verilerin sayısını belirtmektedir. Dizilerin bellekte kapladığı toplam alan dizinin boyutunun yanı sıra veri türü ile de doğrudan ilintilidir. Örneğin aynı boyuta sahip kesirli sayılar için oluşturulmuş bir dizi tam sayılar için oluşturulmuş bir diziye göre bellekte daha çok alan kaplayacaktır.

```
int dizi[4];
```

Örneğin yukarıdaki kod parçası ile 4 elemanlı bir tam sayı dizisi tanımlanmaktadır. Giriş kısmında da belirtildiği gibi tanımlanan dizinin elemanlarına, elemanların indislerini alt simge işleci [] içerisine yazarak erişim sağlanmaktadır. Her bir elemanın indisi dizideki sıralarının bir eksiği olarak belirlenmektedir. Örneğin yukarıda tanımlanan tam sayı dizisinde dizinin üçüncü elemanına erişim sağlamak için dizi[2] kullanılmalıdır. Yine aynı sayı dizisinin elemanları sırasıyla dizi[0], dizi[1], dizi[2] ve dizi[3] olacaktır.

Dizinin elemanlarına değer atama işlemi dizi tanımlandığında yapılabileceği gibi, dizi tanımlandıktan sonra ihtiyaca göre kullanıcıdan gelen girdiler veya program içerisinde uygulanan işlemler üzerinden de gerçekleştirilebilmektedir. C++ programlama dilinde bir dizinin elemanlarına tanımlama esnasında değer atama işlemi küme parantezi yoluyla yapılabilmektedir. Örneğin aşağıdaki kod parçasında 4 elemanlı bir tam sayı dizisi tanımlanmakta ve ilgili dizinin elemanlarına sırasıyla 1, 2, 3, 4 değerleri atanmaktadır.

```
int dizi[4] = {1, 2, 3, 4};
```

Küme parantezi yardımıyla atama işleminde dizinin boyutundan daha az sayıda değer girilmişse, program kalan elemanlara tam sayılarda tanımlanmış diziler için 0 değerini atayacaktır. Örneğin aşağıdaki kod bloğunda, program ilk tam sayı dizisinin ilk iki elemanına sırasıyla 1 ve 2 değerlerini atarken, son iki elemana 0 değerini atayacak; ikinci tam sayı dizisi için küme parantezi içerisine bir değer girilmediğinden, program ikinci sayı dizisinin bütün elemanlarına 0 değerini atayacaktır. Diğer bir ifadeyle ilk sayı dizisinin elemanlarının aldığı değerler sırasıyla dizi_1[0] = 1, dizi_1[1] = 2, dizi_1[2] = 0 ve dizi_1[3] = 0 iken, ikinci sayı dizisinin elemanlarının aldığı değerler sırasıyla dizi_2[0] = 0, dizi_2[1] = 0, dizi_2[2] = 0 ve dizi_2[3] = 0 tür.

```
int dizi_1[4] = {1, 2};
```

```
int dizi_2[4] = {};
```



DİKKAT EDELİM

Küme parantezi yardımıyla atama işleminde dizinin boyutundan daha az sayıda değer girilmişse, program kalan elemanlara tam sayılarda tanımlanmış diziler için 0 değerini atayacaktır.

Diğer yandan tanımlanan bir dizinin elemanlarına sonradan değer atamak ya da atanmış değerleri ihtiyaca göre değiştirmek aşağıdaki kod bloğunda gösterildiği gibi yine atama işleci yardımıyla yapılmaktadır. Kod bloğunda 4 elemanlı bir tam sayı dizisi tanımlanmış ve atama işleci yardımıyla dizinin ilk iki elemanına sırasıyla 1 ve 2 değerleri atanmıştır. Sonrasında 'dizi[1] = dizi[0] + dizi[1]' kod satırı ile dizinin ikinci elemanının değeri değiştirilmiştir.

```
int dizi[4];
dizi[0] = 1, dizi[1] = 2;
dizi[1] = dizi[0] + dizi[1];
```

Tanımlanmış dizilerin tüm elemanlarına değer atamak, atanmış değerleri değiştirmek ya da tüm elemanlara erişim sağlamak gibi işlemler for döngüsü yoluyla yapılabilir.

```
int dizi[4];
int indis;
for (indis = 0; indis < 4; indis++){
    dizi[indis] = indis*indis;
}
for (indis = 0; indis < 4; indis++){
    dizi[indis] = dizi[indis] + 1;
}
for (indis = 0; indis < 4; indis++){
    cout << "dizi[" << indis << "] = " << dizi[indis] << endl;
}
```

Yukarıdaki kod bloğunda 4 elemanlı bir sayı dizisi tanımlanmış ve ilk for döngüsünde 'dizi[indis] = indis*indis' kod satırı üzerinden dizinin her bir elemanına, ilgili elemanın indisinin karesi değer olarak atanmıştır. Yani ilk for döngüsü çalıştırdıktan sonra dizinin elemanlarının alacağı değerler sırasıyla dizi[0] = 0, dizi[1] = 1, dizi[2] = 4 ve dizi[3] = 9 olacaktır. Kod bloğundaki ikinci for döngüsünde ise 'dizi[indis] = dizi[indis] + 1' kod satırı üzerinden dizinin her bir elemanının değeri bir fazlası olarak yenilenmektedir. Kod bloğundaki son for döngüsünde ise dizinin her bir elemanının güncel değeri 'cout << "dizi[" << indis << "] = " << dizi[indis] << endl' kod satırı yoluyla ekrana yazdırılmaktadır. Dolayısıyla kod bloğunun ekran çıktısı Şekil 9.1'deki gibi olacaktır.

```
dizi[0] = 1
dizi[1] = 2
dizi[2] = 5
dizi[3] = 10
```

Şekil 9.1 Tam Sayı Dizisinin Elemanlarının Ekrana Yazdırılması

Hatırlanacağı üzere matematikteki seriler dizilerin belirli bir kesitindeki sayıların (ya da bütün sayıların) toplamı olarak ifade edilmektedir. Serilerin C++ programlama dilindeki karşılığı dizilere uygulanacak for döngüsü yardımıyla elde edilebilir. Örneğin aşağıdaki kod bloğunda 6 elemanlı bir tam sayı dizisi tanımlanmış ve bu dizinin elemanlarına ilk for döngüsündeki 'dizi[indis] = 10*indis + 5' kod

satırı üzerinden, başlangıç değeri 5 ve ortak farkı 10 olan aritmetik dizinin ilk 6 elemanı değer olarak atanmıştır. Dolayısıyla ilgili dizinin elemanları sırasıyla `dizi[0] = 5`, `dizi[1] = 15`, `dizi[2] = 25`, `dizi[3] = 35`, `dizi[4] = 45` ve `dizi[5] = 55` olacaktır. Kod bloğundaki ikinci for döngüsündeki `'toplam += dizi[indis]'` kod satırı yoluyla ilgili dizinin terimleri toplanacaktır. Kod bloğu çalıştırıldığında toplam değişkeninin alacağı değer 180 olacaktır. Kod bloğundaki son for döngüsünde ise `'ara_toplam += dizi[indis]'` kod satırı üzerinden dizinin ikinci elemanından beşinci elemanına kadar olan elemanların toplamı hesaplanacak ve bu toplam `ara_toplam` değişkeninde tutulacaktır. Kod bloğu çalıştırıldığında `ara_toplam` değişkeninin alacağı değer 120 olacaktır.

```
int dizi[6];
int indis;
int toplam = 0;
int ara_toplam = 0;
for (indis = 0; indis < 6; indis++){
    dizi[indis] = 10*indis + 5;
}
for (indis = 0; indis < 6; indis++){
    toplam += dizi[indis];
}
for (indis = 1; indis < 5; indis++){
    ara_toplam += dizi[indis];
}
```

2. KARAKTER DİZİLERİ

Yukarıda da belirtildiği gibi, karakter dizileri de tam sayı dizilerine benzer aşağıdaki kod satırı üzerinden tanımlanmaktadır.

```
char diziAdi[boyut];
```

Tanımlanmış karakter dizisinin elemanlarına değer atama işlemi, tam sayılarda olduğu küme parantezi yardımıyla yapılabileceği gibi atanacak değerlerin çift tırnak içinde tek bir kelime olarak yazılması ile de mümkündür. Yine tam sayılardakine benzer olarak, karakter dizilerinde ilk değer atama işleminde dizinin boyutundan daha az sayıda değer girilmişse, program kalan elemanlar için boş karakteri değer olarak atayacaktır.

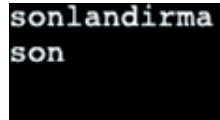
```
char dizi_1[4] = {'a', 'd', 'a'};
char dizi_2[4] = {};
char dizi_3[4] = "ad";
```

Örneğin yukarıdaki kod parçasında 4 uzunluklu 3 farklı karakter dizisi tanımlanmış ve küme parantezi yardımıyla ilk dizinin ilk üç elemanına sırasıyla 'a', 'd' ve 'a' değerleri atanmıştır. Dizinin son elemanı için bir değer belirtilmediğinden program son elemene boş karakter değerini atayacaktır. Dolayısıyla ilk dizinin elemanları sırasıyla `dizi_1[0] = 'a'`, `dizi_1[1] = 'd'`, `dizi_1[2] = 'a'` ve `dizi_1[3] = ' '` olacaktır. Küme parantezi içerisinde bir karakter girilmediğinden program ikinci dizinin bütün elemanlarına boş karakter değerini atayacaktır. Dolayısıyla ikinci dizinin elemanları sırasıyla `dizi_2[0]`

= ' ', dizi_2[1] = ' ', dizi_2[2] = ' ' ve dizi_2[3] = ' ' olacaktır. Üçüncü dizide ise değer atama işlemi çift tırnak içerisinde yazılan karakterler yoluyla yapılmaktadır. Çift tırnak içerisinde yazılan karakterlerin sayısı dizinin boyutundan az olduğu için program dizinin kalan elemanlarına boş karakter değerini atayacaktır. Çift tırnak değer atama yönteminde program, küme parantezinden farklı olarak dizinin değer girilmemiş kalan elemanlarından ilkine sonlandırma karakteri '\0' değerini atayacaktır. Sadece karakter dizileri için tanımlanmış sonlandırma karakteri '\0'; bellekte ayrılmış alanın nereye kadar kullanıldığını, bellekteki ilgili alanın tümünü kontrol etmeksizin tespit edebilmemize imkan sağlamaktadır. Son dizinin elemanları değerler atandıktan sonra sırasıyla dizi_3[0] = 'a', dizi_3[1] = 'd', dizi_3[2] = '\0' ve dizi_3[3] = ' ' olacaktır. Diğer yandan dizinin elemanlarını ekrana yazdırmak istediğimizde program sonlandırma karakterini pas geçecektir.

Sonlandırma karakteri, yukarıda belirtildiği gibi çift tırnak yöntemi üzerinden atanabileceği gibi, aşağı kod bloğunda olduğu gibi küme parantezi yöntemi üzerinden de değer olarak atanabilmektedir. Aşağıdaki kod bloğunda 15 uzunluklu bir karakter dizisi tanımlanmış ve dizinin elemanlarına küme parantezi yöntemi yardımıyla sırasıyla 's', 'o', 'n', 'l', 'a', 'n', 'd', 'i', 'r', 'm', 'a', ve '\0' değerleri atanmıştır. Sonrasında 'dizi[3] = '\0' ' kod satırı üzerinden dizinin dördüncü elemanı sonlandırma karakteri olarak yenilenmiştir. Kod parçası çalıştırıldığında alınacak ekran çıktısı Şekil 9.2'de verilmiştir. Şekil 9.2'den de anlaşılacağı gibi dizinin elemanları tekrar ekrana yazdırılmak istendiğinde program sadece '\0' sonlandırma karakterine kadar olan karakterleri ekrana yazdırmıştır. char dizi[15] = {'s', 'o', 'n', 'l', 'a', 'n', 'd', 'i', 'r', 'm', 'a', '\0' };

```
cout << dizi << endl;
dizi[3] = '\0';
cout << dizi << endl;
```



Şekil 9.2 Karakter Dizisinin Elemanlarının Ekrana Yazdırılması



DİKKAT EDELİM

Sadece karakter dizileri için tanımlanmış sonlandırma karakteri '\0'; bellekte ayrılmış alanın nereye kadar kullanıldığını, bellekteki ilgili alanın tümünü kontrol etmeksizin tespit edebilmemize imkan sağlamaktadır.

2.1. Dizilerde Arama

Aşağıdaki kod bloğu tanımlı bir dizinin kullanıcı tarafından girilen bir elemanı içerip içermediğini belirlemektedir. Kod bloğunda önce 15 uzunluklu bir tamsayı dizisi tanımlanmış ve bu dizinin elemanlarına sırasıyla 7, 12, 6, 21, 17, 5, 12, 32, 9, 2, 16, 31, 3, 18, 23 değerleri atanmıştır. Sonrasında kullanıcıdan pozitif bir tamsayı değeri girmesi istenmiş, while döngüsü ve indis tam sayı değişkeni yardımıyla dizinin girilen tam sayıyı içerip içermediği belirlenmiştir. Dikkat edilirse kod bloğunda dizinin boyutu

'(sizeof(diz)/sizeof(*diz))' formülü yardımıyla tanımlanmaktadır. Örneğin kullanıcı 17 değerini girerse; while döngüsünün beşinci adımında if kontrol yapısındaki 'diz[indis] ==x' koşulu doğru olacağından, program kontrol yapısının gövdesindeki kodu break anahtar kelimesine kadar çalıştıracak ve ekrana "Girilen deger dizinin 5. elemanidir" cümlesini yazdıracaktır. Diğer yandan kullanıcı 1 değerini girerse; herhangi bir indis değeri için döngü içerisindeki 'diz[indis] ==x' koşulu doğru olamayacak ve döngü sonunda indis tamsayı değişkeninin değeri 15 olacaktır. Dolayısıyla while döngüsü dışındaki if kontrol yapısının koşulu 'indis >= (sizeof(diz)/sizeof(*diz))' doğru olacak ve program kontrol yapısının gövdesindeki kodu çalıştırarak ekrana "Dizi, girilen tamsayı değerini icermemektedir" cümlesini yazdıracaktır.

```
int diz[15] = {7, 12, 6, 21, 17, 5, 12, 32, 9, 2, 16, 31, 3, 18, 23};
int indis = 0, x;
cout << "Pozitif bir tamsayı degeri giriniz : " << endl;
cin >> x;
int boyut = (sizeof(diz)/sizeof(*diz));
while (indis < boyut){
    if (diz[indis] == x){
        cout << "Girilen deger dizinin " << indis +1 << ". elemanidir" << endl;
        break;
    }
    indis++;
}
if(indis >= boyut)
    cout << "Dizi, girilen tamsayı degerini icermemektedir"<< endl;
```

2.2. Tek Boyutlu Dizilerin Fonksiyonlara Aktarılması

Tanımlı bir dizinin girilen bir elemanı içerip içermediğini belirleyen yukarıdaki program aşağıda gösterildiği gibi bir fonksiyon yardımıyla da tanımlanabilir. Aşağıdaki tanımlı verilen Arama fonksiyonu bir diziyi, dizinin boyutunu ve dizide aranacak x elemanını girdi olarak almakta; ve for döngüsü yardımıyla dizi x elemanını içeriyorsa ilgili elemanın dizideki sırasına, içermiyorsa 0 değerine dönmektedir.

```
int Arama(int dizi[], int boyut, int x) {
    for(int i = 0; i < boyut;i++){
        if(dizi[i] ==x)
            return (i+1);
    }
    return 0;
}
```

Aşağıda verilen kod bloğu ise yukarıda tanımlanan Arama fonksiyonu yardımıyla tanımlı bir dizinin girilen bir tamsayı değerini içerip içermediğini belirlemektedir. Örneğin kullanıcı 5 değerini girerse; Arama(diz, boyut, x) fonksiyonu verilen girdi değerleri için 6 değerine döneceğinden, 'Arama(diz, boyut, x) == 0' koşulu yanlış olacak ve program if..else kontrol yapısındaki alternatif kodu çalıştırarak

ekrana “Girilen deger dizinin 6. elemanidir” cümlesini yazdıracaktır. Diğer yandan kullanıcı 30 değerini girerse; dizinin herhangi bir elemanı 30 değerine eşit olmadığından Arama(diz, boyut, x) fonksiyonu verilen girdi değerleri için 0 değerine dönecek ve ‘Arama(diz, boyut, x) == 0’ koşulu doğru olacaktır. Dolayısıyla program if.. else kontrol yapısının gövdesindeki kodu çalıştırarak ekrana “Dizi girilen tam-sayı degerini icermemektedir” cümlesini yazdıracaktır.

```
int diz[15] = {7, 12, 6, 21, 17, 5, 12, 32, 9, 2, 16, 31,
3, 18, 23};
int x;
cout << “Pozitif bir tamsayı degeri giriniz : “ <<
endl;
cin >> x;
int boyut = (sizeof(diz)/sizeof(*diz));
if(Arama(diz, boyut, x) == 0)
    cout << “Dizi girilen tamsayı degerini icermemektedir”<< endl;
else
    cout << “Girilen deger dizinin “ << Arama(-diz, boyut, x) << “. elemanidir” << endl;
}
```

2.3. Dizilerde Sıralama

Burada tanımlı dizilerin elemanlarını küçükten büyüğe sıralamamıza imkan sağlayan sıralama algoritmalarına yer verilecektir. Algoritmalara geçmeden, önce diziler için sıralama problemini tanımlayalım: k dizinin boyutu olmak üzere kıyaslanabilir ve sıralanabilir elemanlardan oluşan tanımlı bir dizi[k] dizisi için sıralama problemi, dizinin elemanlarını $dizi[0] \leq dizi[1] \leq \dots \leq dizi[k-1]$ şartını sağlayacak şekilde yeniden sıralamak şeklinde tanımlanmaktadır. Dizilerin sıralı bir şekilde tutulmasının daha kullanışlı bir veri yapısı oluşturması, dizilerde sıralamanın arama algoritmaları gibi farklı algoritmaların alt yordamı olması ve sıralama problemine etkin çözüm geliştirmek için yapılan uğraşın algoritmik tasarım açısından zengin ve kullanışlı bir literatür oluşturuyor olması gibi sebeplerden ötürü sıralama problemi bilgisayar bilimlerindeki en temel problemlerden biri olagelmıştır.

Dizilerin elemanlarını küçükten büyüğe sıralamak mevzu olduğunda akla gelen en aşık ve kolay yaklaşım şüphesiz küçük elemanların bulunarak dizinin ilk sıralarına yerleştirilmesi ya da büyük elemanların bulunarak dizinin son sıralarına yerleştirilmesi olacaktır. Bu yaklaşımlardan ilki seçmeli sıralama (selection sort) ismiyle literatüre girmiş bir algoritmayken, ikincisi kabarcık sıralaması (bubble sort) ismiyle literatürde yer bulmuştur. Aşağıdaki kod bloğu küme parantezi yöntemi kullanılarak tanımlanmış 10 uzunluklu bir tam sayı dizisinin elemanlarını kabarcık sıralaması yöntemi kullanarak sıralamaktadır. Kod bloğundaki dıştaki for döngüsünün ilk adımında ($i = 0$ için) dizinin en büyük elemanı içteki for döngüsü yardımıyla dizinin son indisine taşınmaktadır. Dıştaki for döngüsünün ikinci adımında ise dizinin en büyük ikinci elemanı içteki for döngüsü yardımıyla dizinin sondan bir önceki indisine taşınmaktadır. Bu şekilde dizinin büyük elemanları dizinin sonlarına doğru taşınarak nihayetinde elemanları küçükten büyüğe sıralı bir dizi elde edilmektedir.

```
int diz[10] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
int boyut = (sizeof(diz)/sizeof(*diz));
```

```

int temp;
for(int i = 0; i < boyut - 1; i++){
    for(int j = 0; j < boyut - i; j++){
        if(diz[j] > diz[j+1]){
            temp = diz[j+1];
            diz[j+1] = diz[j];
            diz[j] = temp;
        }
    }
}

```



DİKKAT EDELİM

Yukarıda verilen tamsayı dizisini Seçmeli Sıralama Algoritması yardımıyla sıralayan bir program geliştirin.

3. İKİ BOYUTLU DİZİLER

Matematikteki matris kavramının C++ programlama dilindeki karşılığı olarak değerlendirilebilecek iki boyutlu diziler aşağıdaki kod satırı üzerinden tanımlanmaktadır.

```
veri_turu diziAdi[boyut_1][boyut_2];
```

Tek boyutlu dizilerde olduğu gibi veri_turu bellekte saklanacak verilerin türünü, boyut_1 tanımlanan dizinin satır sayısını ve boyut_2 ise dizinin sütun sayısını belirtmektedir. Kod satırı çalıştırıldığı anda program boyut_1*boyut_2*veri_turu boyutu kadarlık alanı dizi için ayırmaktadır. Örneğin Şekil 9.2'deki matris iki boyutlu bir diziyi göstermektedir. Şekilden de anlaşılacağı üzere iki boyutlu dizinin her elemanı, i ve j elemanı belirleyen indisler olmak üzere diz[i][j] şeklinde gösterilmektedir.

	Sütun 0	Sütun 1	Sütun 2
Satır 0	diz[0][0]	diz[0][1]	diz[0][2]
Satır 1	diz[1][0]	diz[1][1]	diz[1][2]
Satır 2	diz[2][0]	diz[2][1]	diz[2][2]

Şekil 9.3 İki Boyutlu Dizin Matris ile Gösterimi

İki boyutlu diziler bellekte tek boyutlu bir dizi olarak tutulmaktadır. Bu açıdan bakıldığında iki boyutlu diziler, her elemanı ayrı bir dizi olan tek boyutlu diziler olarak değerlendirilebilirler. Dolayısıyla iki boyutlu dizilerin satır ve sütun indisleri üzerinden yorumlanması bir kabule dayanmaktadır.

C++ programlama dilinde tanımlanan bir dizinin elemanlarına, elemanların matris içerisindeki konumu belirten satır ve sütun indislerinin sırasıyla alt simge işleçleri [] içerisine yazılmasıyla erişim sağlanmaktadır. Örneğin aşağıdaki kod satırı üzerinden iki satırlı ve iki sütunlu iki boyutlu bir tamsayı dizisi tanımlanmaktadır. Tanımlı bu dizinin ikinci satır birinci sütunundaki elemanına dizi[1][0] kodu yardımıyla erişim sağlanmaktadır. Dolayısıyla aynı sayı dizisinin birinci satırındaki elemanlar sırasıyla

dizi[0][0] ve dizi[0][1] olacakken, ikinci satırındaki elemanlar dizi[1][0] ve dizi[1][1] olacaktır.

```
int dizi[2][2];
```

Tek boyutlu dizilerde olduğu gibi iki boyutlu dizilerde de tanımlama esnasında dizinin elemanlarına değer atama işlemi küme parantezi yoluyla yapılabilmektedir. Küme parantezi ile değer atama işleminde atanacak değerler her satır için ayrı parantez kullanılarak tanımlanmaktadır. Örneğin aşağıdaki kod satırında üç satırlı ve üç sütunlu bir tamsayı dizisi tanımlanmış ve küme parantezi yardımıyla dizinin elemanlarına 1'den 9'a kadar tamsayı değerleri sırasıyla atanmıştır.

```
int dizi[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
```

İki boyutlu dizilerde küme parantezi yoluyla değer atama işlemi, aşağıda gösterildiği gibi her satır için ayrı bir parantez kullanılmaksızın da gerçekleştirilebilmektedir. Küme parantezi yardımıyla atama işleminde dizinin boyutundan daha az sayıda değer girilmişse, program kalan elemanlara tam sayılarda tanımlanmış diziler için 0 değerini atayacaktır. Dolayısıyla aşağıdaki kod satırı üzerinden tanımlanmış tamsayı dizisinin elemanları sırasıyla dizi[0][0] = 1, dizi[0][1] = 2, dizi[0][2] = 3, dizi[1][0] = 4, dizi[1][1] = 5, dizi[1][2] = 6, dizi[2][0] = 7, dizi[2][1] = 0 ve dizi[2][2] = 0 olacaktır.

```
int dizi[3][3] = {1, 2, 3, 4, 5, 6, 7};
```

Diğer yandan tanımlı iki boyutlu dizilerin elemanlarına iç içe döngüler yardımıyla erişim sağlanabilmektedir. Aşağıdaki kod bloğunda ise tanımlı bir dizinin elemanlarına iç içe iki for döngüsü yardımıyla nasıl değer atandığı gösterilmektedir. İç içe döngülerin ilkindeki 'diz[i][j] = i + j' kod satırı üzerinden dizinin elemanlarına satır ve sütun indislerinin toplamı değer olarak atanmaktadır. İç içe döngülerin ikincisindeki 'cout << "diz[" << i << "]"[" << j << "]" = " << diz[i][j] << " " ' kod satırı yardımıyla dizinin her bir elemanının güncel değeri ekrana yazdırılmaktadır. Dolayısıyla kod bloğu çalıştırıldığında oluşacak ekran görüntüsü Şekil 9.4'deki gibi olacaktır.

```
int diz[3][3];
for (int i = 0 ; i < 3; i++){
    for (int j = 0; j < 3; j++){
        diz[i][j] = i + j;
    }
}
for (int i = 0 ; i < 3; i++){
    for (int j = 0; j < 3; j++){
        cout << "diz[" << i << "]"[" << j << "]" = " << diz[i][j] << " ";
    }
}
cout << "\n" << endl;
```

```
diz[0][0]=0 diz[0][1]=1 diz[0][2]=2
diz[1][0]=1 diz[1][1]=2 diz[1][2]=3
diz[2][0]=2 diz[2][1]=3 diz[2][2]=4
```

Şekil 9.4 İki Boyutlu Tamsayı Dizisinin Elemanlarının Ekrana Yazdırılması.

4. İKİ BOYUTLU DİZİLERİN ÇARPIMI

Hatırlanacağı gibi verilen iki matrise çarpma işleminin uygulanabilmesi için, ilk matrisin sütun sayısı ile ikinci matrisin satır sayısının birbirine eşit olması gerekmektedir. Bu kısıt sağlandığında verilen m satır ve n sütun içeren A matrisi ile n satır ve k sütun içeren B matrisinin çarpımı aşağıda gösterildiği şekilde olmaktadır:

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ a_{i1} & a_{i2} & \dots & a_{in} \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{1j} & b_{1k} \\ \vdots & \vdots & \vdots \\ b_{n1} & b_{nj} & b_{nk} \end{pmatrix} = \begin{pmatrix} c_{11} & \dots & c_{1k} \\ \vdots & \ddots & \vdots \\ c_{m1} & \dots & c_{mk} \end{pmatrix}$$

$$c_{ij} = a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + \dots + a_{in} \cdot b_{nj}$$

Şekil 9.5 Matrislerin Çarpımı,

Şekil 9.5'ten de anlaşılacağı gibi m satır ve k sütunlu C sonuç matrisinin her bir cij elemanı A matrisinden i. satır ve B matrisinden j. sütunun elemanları yukarıda gösterildiği gibi bir işleme tabi tutularak bulunmaktadır. Aşağıdaki kod bloğu iç içe üç for döngüsü yardımıyla verilen iki matrisin çarpımını hesaplamaktadır. Kod bloğunda önce 2 satır ve 3 sütunlu iki boyutlu bir tamsayı dizisi AMatrisi tanımlanmış ve bu matrisin satırlarına sırasıyla {1, 2, 3} ve {4, 5, 6} değerleri atanmıştır. Yine benzer şekilde 3 satır ve 2 sütunlu iki boyutlu bir tamsayı dizisi BMatrisi tanımlanmış ve bu matrisin satırlarına sırasıyla {1, 2,}, {3, 4} ve {5, 6} değerleri atanmıştır. Son olarak çarpım matrisini saklayacak 2 satır ve 2 sütunlu iki boyutlu bir tamsayı dizisi CMatrisi tanımlanmış ve bu matrisin elemanlarına 0 değeri atanmıştır. Kod bloğundaki dıştaki iki for döngüsü sırasıyla sonuç matrisinin satır ve sütun indislerini tararken, içteki for döngüsü yukarıda verilen

'cij = ai1.b1j + ai2.b2j + ... + ain.bnj'

formülünü uygulayarak sonuç matrisinin elemanlarının alacağı değerleri hesaplamaktadır.

```
int AMatrisi[2][3] = {{1,2,3},{4,5,6}};
int BMatrisi[3][2] = {{1,2},{3,4},{5,6}};
int CMatrisi[2][2] = { };
for (int i = 0; i < 2; i++){
    for (int j = 0; j < 2; j++){
        for (int n = 0; n < 3; n++){
            CMatrisi[i][j] += AMatrisi[i][n] * BMatrisi[n][j];
        }
    }
}
for (int i = 0 ; i < 2 ; i++){
    for (int j = 0; j < 2; j++){
        cout << "CMatrisi[" << i << "][" << j << "]"<< CMatrisi[i][j] << " ";
    }
    cout<<" "<<endl;
}
```

Yukarıdaki kod bloğunda iç içe döngülerin ikincisi ise 'cout << "CMatrisi[" << i << "]" << j << "]" << "]= "<< CMatrisi[i][j] << " , ' kod satırı üzerinden sonuç matrisinin elemanlarını ekrana yazdırmaktadır. Kod bloğu çalıştırıldığında oluşacak ekran görüntüsü Şekil 9.6'daki gibi olacaktır.

```
CMatrisi[0][0]=22 CMatrisi[0][1]=28
CMatrisi[1][0]=49 CMatrisi[1][1]=64
```

Şekil 9.6 İki Boyutlu Dizilerin Çarpımı.

5. İKİ BOYUTLU DİZİLERİN FONKSİYONLARA AKTARILMASI

Burada fonksiyonlar yardımıyla satır ve sütun sayıları eşit kare bir matrisin transpozuna eşit olup olmadığı, diğer bir ifadeyle matrisin köşegenine göre simetrik olup olmadığını belirleyen bir program incelenecektir. Aşağıda tanımı verilen Simetrik fonksiyonu satır ve sütun sayısı eşit iki boyutlu bir diziyi ve dizinin satır sayısını girdi olarak almakta; ve iç içe iki for döngüsü yardımıyla bütün satır ve sütun indisleri için 'dizi[i][j] = dizi[j][i]' eşitliğinin sağlanıp sağlanmadığını kontrol ederek verilen iki boyutlu dizinin simetrik olup olmadığına karar vermektedir. Fonksiyon dizi simetrikse 1 değerine, simetrik değilse 0 değerine dönmektedir. Tek boyutlu dizilerden farklı olarak, iki boyutlu diziler için tanımlanan fonksiyonlarda ikinci boyut mutlaka belirtilmelidir. Dikkat edilirse; aşağıdaki fonksiyon tanımında dizinin ikinci boyutu 4 olarak tanımlanmıştır.

```
int Simetrik(int dizi[][4], int boyut) {
    for (int i = 0; i < boyut; i++){
        for (int j = 0; j < boyut; j++){
            if(dizi[i][j] != dizi[j][i]){
                return 0;
            }
        }
    }
    return 1;
}
```



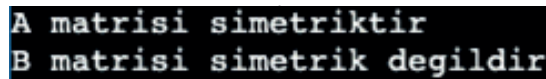
DİKKAT EDELİM

Çok boyutlu diziler üzerinde fonksiyon tanımlanırken, ilk boyut dışındaki diğer boyutlar mutlaka belirtilmelidir.

Aşağıda verilen kod bloğu ise yukarıda tanımlanan Simetrik fonksiyonu yardımıyla iki boyutlu tanımlı A ve B tamsayı dizilerinin simetrik olup olmadığını belirlemektedir. Örneğin tanımlı dizilerden ilki simetrik olduğundan 'Simetrik(A, 4) == 1' koşulu doğru olacak ve program ilk if...else kontrol yapısının gövdesindeki kodu çalıştırarak ekrana "A matrisi simetriktir" cümlesini yazdıracaktır. Diğer yandan ikinci dizi için 'A[0][1] == A[1][0]' ifadesi doğru olmadığından 'Simetrik(B, 4) == 1' koşulu doğru olmayacak ve program ikinci if...else kontrol yapısının alternatif gövdesindeki kodu çalıştırarak ekrana

“B matrisi simetrik degildir” ifadesini yazdıracaktır. Dolayısıyla aşağıdaki kod bloğu çalıştırıldığında ekran çıktısı Şekil 9.7’deki gibi olacaktır.

```
int A[4][4] = {{1,2,1,2},
               {2,1,2,1},
               {1,2,1,2},
               {2,1,2,1}};
int B[4][4] = {{1,2,1,2},
               {3,1,2,1},
               {1,2,1,2},
               {2,1,2,1}};
if(Simetrik(A, 4) == 1){
    cout << “A matrisi simetriktir” << endl;
}
else {
    cout << “A matrisi simetrik degildir” << endl;
}
if(Simetrik(B, 4) == 1){
    cout << “B matrisi simetriktir” << endl;
}
else {
    cout << “B matrisi simetrik degildir” << endl;
}
```



Şekil 9.7 Matrislerin Simetrik Olup Olmadığını Belirleyen Kodun Ekran Çıktısı.

6. ÇOK BOYUTLU DİZİLER

C++ programlama dili ikiden fazla boyuta sahip diziler de tanımlanabilmesi imkan vermektedir. İki boyutlu dizilere benzer şekilde çok boyutlu diziler de aşağıda gösterildiği gibi dizinin farklı boyutlarıdaki eleman sayıları alt simge işleci içerisinde belirtilerek tanımlanmaktadır.

```
veri_turu diziAdi[boyut_1][boyut_2]...[boyut_n];
```

Örneğin aşağıdaki kod bloğunda, birinci boyutunda 3, ikinci boyutunda 2 ve üçüncü boyutunda 3 eleman barındıran üç boyutlu bir tamsayı dizisi tanımlanmış; ve bu dizinin elemanlarına i, j ve k sırasıyla birinci, ikinci ve üçüncü boyuttaki indisler olmak üzere ‘i + j * k’ formülü kullanılarak iç içe üç for döngüsü yardımıyla tamsayı değerleri atanmıştır. Dolayısıyla kod bloğu çalıştırıldığında ekran çıktısı Şekil 9.8’deki gibi olacaktır.

```
int dizi[3][2][3] = { };
for (int i = 0; i < 3; i++){
    for (int j = 0; j < 2; j++){
        for (int k = 0; k < 3; k++){
            dizi[i][j][k] = i + j * k;
        }
    }
}
```

```

    cout << "dizi[" << i << "]" << j << "]" << k << "]" << "=" << dizi[i][j][k] << " ";
}
cout << "\n" << endl;

```

```

dizi[0][0][0]=0 dizi[0][0][1]=0 dizi[0][0][2]=0
dizi[0][1][0]=0 dizi[0][1][1]=1 dizi[0][1][2]=2
dizi[1][0][0]=1 dizi[1][0][1]=1 dizi[1][0][2]=1
dizi[1][1][0]=1 dizi[1][1][1]=2 dizi[1][1][2]=3
dizi[2][0][0]=2 dizi[2][0][1]=2 dizi[2][0][2]=2
dizi[2][1][0]=2 dizi[2][1][1]=3 dizi[2][1][2]=4

```

Şekil 9.8 Üç Boyutlu Tamsayı Dizisinin Elemanlarının Ekrana Yazdırılması.

BÖLÜM ÖZETİ

Bu bölümde aynı veri türünden belirli sayıda elemanı bellekte sıralı bir şekilde saklamaya imkân veren dizilerden bahsedilmiştir. Dizilerde saklanacak elemanların sonlu sayıda ve aynı veri türünde olması gerekmektedir. Dizilerdeki elemanlara erişim ilgili elemanların bellekteki sırasını belirten indisler yoluyla yapılmaktadır. Dizinin elemanlarına değer atama işlemi dizi tanımlandığında küme parantezi yardımıyla yapılabileceği gibi, dizi tanımlandıktan sonra ihtiyaca göre kullanıcıdan gelen girdiler veya program içerisinde uygulanan işlemler üzerinden atama işlemiyle de gerçekleştirilebilmektedir. Diğer yandan tanımlanmış karakter dizisinin elemanlarına değer atama işlemi, tam sayılarda olduğu küme parantezi yardımıyla yapılabileceği gibi atanacak değerlerin çift tırnak içinde tek bir kelime olarak yazılması ile de mümkündür. Bunların yanı sıra dizi kavramını pekiştirmek ve dizilerin farklı problemlerin çözümünde nasıl kullanılabildiğini göstermek amacıyla dizilerde arama ve sıralama algoritmaları da incelenmiştir. Dahası dizilerin fonksiyonlara aktarılmasının nasıl yapıldığı örnekler üzerinden anlatılmıştır. Bölümde ayrıca matematikteki matris kavramının C++ programlama dilindeki karşılığı olarak değerlendirilebilecek iki boyutlu dizilerden de bahsedilmiştir. Ayrıca iki boyutlu dizi kavramını pekiştirmek adına, iki boyutlu diziler kullanılarak verilen iki matrisin çarpımını hesaplayan bir uygulamaya da yer verilmiştir.

GÖZDEN GEÇİRELİM

1. Aşağıdakilerden hangisinde dizi doğru şekilde tanımlanmıştır?

- a) `int dizi[5] = {5, 3, 1, 10, 8, 7};`
- b) `double dizi[5] = {3.2, 0.5, c};`
- c) `int dizi = {5, 2, 1, 3, 4};`
- d) `char dizi[5];`
- e) `char dizi[5] = {a, d, a, n, a};`

2. Aşağıdakilerden hangisinde dizi yanlış tanımlanmıştır?

- a) `int dizi[5] = {10, 6, 5, 7};`
- b) `bool dizi[5];`
- c) `char dizi[5] = 'c';`
- d) `char dizi[5] = "c";`
- e) `float dizi[5];`

3. Aşağıdakilerden hangisinde iki boyutlu tamsayı dizisi doğru şekilde tanımlanmıştır?

- a) `int dizi[2][4] = {{4, 4, 4, 4}, 3, 3, 3, 3};`
- b) `int dizi[2];`
- c) `int dizi = {{4, 4, 4, 4}, {3, 3, 3, 3}};`
- d) `int dizi[2][4] = {{4, 4, 4, 4, 3, 3, 3, 3}};`
- e) `int dizi[2][4] = {4, 4, 4, 4, 3, 3, 3, 3};`

4. Aşağıdakilerden hangisinde iki boyutlu dizi yanlış tanımlanmıştır?

- a) `int dizi[2][4] = {{2, 2, 2, 2}, {4, 4, 4, 4}};`
- b) `bool dizi[2][4];`
- c) `char dizi[2][4] = {"ada", "na"};`
- d) `char dizi[2][2] = "adana";`
- e) `int dizi[2][2] = {1, 3, 5};`

5. `int dizi[6] = {1, 2};`
`for(int i = 2; i < 6; i++){`
`dizi[i] = 2*dizi[i - 1] - dizi[i - 2];`
`}`

Yukarıdaki kod bloğu çalıştırıldıktan sonra

dizinin son elemanının değeri aşağıdakilerden hangisi olacaktır?

- a) 3
- b) 4
- c) 5
- d) 6
- e) 7

6. `int dizi[5] = {1, 2};`
`for(int i = 2; i < 5; i++){`
`if(i%2 == 1)`
`dizi[i] = 2*dizi[i - 1] - dizi[i - 2];`
`else`
`dizi[i] = 3*dizi[i - 2] - dizi[i - 1];`
`}`

Yukarıdaki kod bloğu çalıştırıldıktan sonra dizinin son elemanının değeri aşağıdakilerden hangisi olacaktır?

- a) 0
- b) 1
- c) 2
- d) 3
- e) 4

7. `char dizi[8] = "adana";`
`dizi[1] = 't';`
`dizi[3] = '\0';`
`cout<<dizi<<endl;`

Yukarıdaki kod bloğu çalıştırıldığında ekran çıktısı aşağıdakilerden hangisi olacaktır?

- a) adana
- b) ada\0
- c) ada
- d) ata\0
- e) ata

8. `int diz[12] = {25, 13, 10, 21, 8, 7, 17, 1, 4, 5, 19, 18};`
`int indis = 0, x = 5;`
`int boyut = (sizeof(diz)/sizeof(*diz));`
`while (indis < boyut){`
`if (diz[indis] == x){`
`cout << "Aranan tam sayi degeri`
`dizinin "<< indis + 1 << ". elemanidir"`
`<< endl;`
`break;`
`}`
`indis++;`
`}`
`if(indis >= boyut)`
`cout << "Dizi aranan tam sayi degerini`
`icermemektedir"`
`<< endl;`

Yukarıdaki kod bloğu çalıştırıldığında ekrana yazdırılacak olan ifade aşağıdakilerden hangisi olacaktır?

- a) Aranan tam sayi degeri dizinin 9. elemanıdır
- b) Aranan tam sayi degeri dizinin 10. elemanıdır
- c) Aranan tam sayi degeri dizinin 11. elemanıdır
- d) Aranan tam sayi degeri dizinin 12. elemanıdır
- e) Dizi aranan tam sayi degerini icermemektedir

9. `int diz[12] = {25, 13, 10, 21, 8, 7, 17, 1, 4, 5, 19, 18};`
`int temp;`
`for(int i = 0; i < (sizeof(diz)/sizeof(*diz))-1; i++){`
`for(int j = 0; j < (sizeof(diz)/sizeof(*diz))-1; j++){`
`if(diz[j] < diz[j+1]){`
`temp = diz[j];`
`diz[j] = diz[j+1];`
`diz[j+1] = temp;`
`}`
`}`
`}`
`for(int i = 0; i < (sizeof(diz)/sizeof(*diz)); i++){`
`cout << diz[i] << " ";`
`}`

Yukarıdaki kod bloğu çalıştırıldığında ekrana yazdırılacak olan ifade aşağıdakilerden hangisi olacaktır?

- a) 25 13 10 21 8 7 17 1 4 5 19 18
- b) 1 4 5 7 8 10 13 17 18 19 21 25
- c) 25
- d) 25 21 19 18 17 13 10 8 7 5 4 1
- e) 1

10. `int diz[4][4];`
`for (int i = 0 ; i < 4; i++){`
`for (int j = 0; j < 4; j++){`
`diz[i][j] = i * j;`
`}`
`}`

Yukarı kod bloğu çalıştırıldıktan sonra dizinin 3. satır ve 4. sütunundaki elemanın alacağı değer aşağıdakilerden hangisi olacaktır?

- a) 1
- b) 2
- c) 6
- d) 12
- e) 16

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

BÖLÜM 10

İŞARETÇİLER

ANAHTAR KAVRAMLAR

İşaretçiler, Adres İşleci, Referanstan Ayırma İşlemi, İşaretçi İfadesi, İşaretçi Aritmetiği



ÖĞRENME HEDEFLERİ

-  1 İşaretçi tanımlamayı ve kullanmayı bilir.
-  2 İşaretçi tanımlamayı ve kullanmayı bilir.
-  3 Fonksiyonlara argümanları işaretçi kullanarak referans ile gönderebilir.
-  4 İşaretçiler ile beraber const kullanmayı gerçekleştirebilir.
-  5 İşaretçi ifadeleri ve aritmetiği konularında bilgi sahibi olur.
-  6 Diziler ve işaretçiler arasındaki yakın ilişkiyi anlar ve kullanır.

GİRİŞ

Bu bölümde C++ programlama dilinde yer alan işaretçiler anlatılacaktır.

İşaretçiler, diğer tüm değişkenler gibi belirli bir bellek alanı kullanırlar. Diğer değişkenlerden farklı olarak gerçek veri değeri yerine adres değeri tutarlar. Bu adres bir veri tipinde tanımlanmış bir değişkenin bellekte yer aldığı alanın adresidir. Bu nedenle işaretçiler de uygun tiplerde tanımlanırlar. Tam sayı olarak tanımlanmış bir işaretçi, tam sayı olarak tanımlanmış bir değişkenin bellekteki adresini gösterir. Farklı veri tipleri bellekte farklı miktarda yer kullandıkları için işaretçinin hangi tipte tanımlandığı önemlidir. İşaretçiler kullanılarak parametreler fonksiyonlara referans ile gönderilebilirler. Bu durumu gerçekleştirmek için fonksiyon çağırma işleminde adres işleci kullanılarak (&) bir değişkenin adresi alınır ve fonksiyona gönderilir. Fonksiyonun parametre listesinde ise gönderilen bu adres değerinin atanacağı bir işaretçi tanımlanır. İşaretçilerin tanımlanmasında const nitelleyicisi kullanılarak işaretçinin veya gösterdiği değerin sabit olarak tanımlanması sağlanabilir. İşaretçiler aritmetik ifadelerde, atama ifadelerinde ve karşılaştırma işlemlerinde işlenen olarak yer alabilirler. Artırma ve azaltma işlemleri işaretçiler üzerinde kullanılabilir. Dizinin ilk elemanının adresi bir işaretçi değişkenine atanabilir. Bu şekilde dizinin ismi ve adresin atandığı işaretçi kullanılarak iki farklı gösterim yöntemiyle dizinin farklı elemanlarına ulaşmak mümkün olabilir. Bu yöntemler işaretçi sapma ve işaretçi alt simge gösterimleridir.



1. İŞARETÇİ DEĞİŞKENİ TANIMLAMA VE İLK DEĞER ATAMA

İşaretçiler C++ programlama dilinde adres değerine sahip değişkenler olarak tanımlanırlar. Bir işaretçi değişkeninin aldığı değer bellekte yer alan herhangi bir baytlık alanın adres değeridir. Bir işaretçi C++ da yer alan diğer değişkenler gibi tanımlanır ancak tanımlamada '*' işleci kullanılır. İşaretçiler herhangi bir veri tipinde tanımlanabilirler. int olarak tanımlanan bir işaretçi tam sayı bir değişkenin yer aldığı bellek alanının ilk baytının adresini gösterir. Daha önceki bölümlerde ifade edildiği gibi farklı tipteki değişkenler bellekte farklı miktarlarda alan kullanırlar. Örneğin, char 1 bayt, int 4 bayt (veya 8 bayt) ve double 8 baytlık alanlar kullanır. Bir işaretçi değişkeni (bu örnekte ptr) aşağıdaki şekilde tanımlanabilir:

```
int a=5;
int* ptr;
ptr=&a;
```

ptr bir tam sayı değişkenin adresini tutmak amacıyla işaretçi değişkeni olarak tanımlanmıştır. Bu örnekte a tam sayı değişkeninin adresi (&) işleci kullanılarak elde edilmiş ve ptr işaretçi değişkenine atanmıştır. Bu noktadan itibaren ilgili tam sayı değişkenine erişmek için değişken ismi a kullanılabileceği gibi, aynı değişkenin bellek adresini tutan ptr işaretçi değişkeni de kullanılabilecektir. Yalnız değişken ismi diğer bölümlerde de anlatıldığı üzere doğrudan kullanılırken, ilgili değişkenin değerine ulaşmak için işaretçi değişkeni referanstan ayırma işleci (*) ile birlikte kullanılacaktır.

```
//Program10.1.cpp
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    int a=5;
    int* ptr;
    ptr=&a;
    cout<<"Dogrudan degisken ismi (a) kullanimi:"<<a<<endl;
    cout<<"Dolayli olarak isaretci (*ptr) kullanimi:"<<*ptr<<endl;
    return 0;
}
```

Program10.1 Çıktısı

Dogrudan degisken ismi (a) kullanimi:5

*Dolayli olarak isaretci (*ptr) kullanimi:5*

Program10.1.cpp'de tanımlanan a tam sayı değişkeninin değeri ekrana önce değişken ismi kullanılarak sonraki satırda ise işaretçi kullanılarak yazdırılmıştır. İşaretçi kullanarak a değişkenine erişebilmek amacıyla öncelikle bu tam sayı değişkenin adresinin, işaretçi değişkene atanması gerekir. Bu atama işlemi main fonksiyonda 3. satırda gerçekleştirilmiştir.

**DİKKAT EDELİM**

Bir işaretçi değişkeni kullanabilmek için, bu işaretçi değişkene herhangi bir adres atamasının gerçekleştirilmiş olması gerekir.

Program10.2.cpp’de ptr işaretçi değişkeninin değeri ekrana yazdırılmıştır. Programda a tam sayı değişkeninin adresini tutan işaretçi ile a değişkeninin adresinin aynı olması beklenmektedir. Doğrudan a değişkeninin adresi (&) işleci ile elde edilebilir.

```
//Program10.2.cpp
#include <iostream>
using namespace std;
int main()
{
    int a=5;
    int* ptr;
    ptr=&a;
    cout<<"a degiskeni bellek adresi:"<<&a<<endl;
    cout<<"ptr degiskeni degeri:"<<ptr<<endl;
    return 0;
}
```

Program10.2 Çıktısı

a degiskeni bellek adresi:0xffffcbf4
ptr degiskeni degeri:0xffffcbf4

**DİKKAT EDELİM**

İşaretçi değişkenleri, program farklı ortamlarda veya farklı zamanlarda çalıştığında farklı değerler alabilir. Programın çalıştığı ortama göre programda kullanılan değişkenler farklı bellek alanlarını kullanabilirler.

Adres işleci (&) ve referanstan ayırma işleci (*) birbirlerinin tersi işleçlerdir.

```
int a=5;
int* ptr;
ptr=&a;
```

Yukarıdaki kod parçası verilmiş olsun. *(&ptr) ile &(*ptr) ifadeleri aynı değeri üretir. Bu değer a değişkeninin adresidir. Program10.3.cpp bu durumu gösteren bir program içermektedir.

```
//Program10.3.cpp
#include <iostream>
using namespace std;
int main()
{
    int a=5;
    int* ptr;
    ptr=&a;
    cout<<"*ptr: "<<*ptr<<endl;
    cout<<"&ptr: "<<&ptr<<endl;
    return 0;
}
```

Program10.3.cpp Çıktısı

```
*ptr: 0xffffcbfc
&ptr: 0xffffcbfc
```

2. İŞARETÇİ KULLANARAK REFERANS İLE FONKSİYON ÇAĞIRMA

Bölüm 8’de fonksiyonların iki farklı şekilde çağrılabilirliği anlatılmıştı. Bu yaklaşımlardan birisi parametreleri referans ile gönderme işlemiydi. Bu şekilde fonksiyon içerisinde, doğrudan çağırılan fonksiyondaki orijinal değişkene erişim söz konusu olabiliyordu. Ayrıca boyutu büyük nesneler, tekrar oluşturulmaya gerek olmadan fonksiyonlara aktarılabilirdi. Bu yaklaşımı gerçekleştirmek için referans parametreleri kullanılmıştı. Bu kısımda işaretçileri kullanarak referans ile fonksiyon çağırma işleminin nasıl gerçekleştirileceği anlatılacaktır.

```
//Program10.4.cpp
#include <iostream>
using namespace std;
void toplaIsaretcisi(int *a, int *b, int *t)
{
    *t=*a+*b;
}
int main()
{
    int i=5, j=15;
    int toplam;
    toplaIsaretcisi(&i, &j, &toplam);
    cout<<"İki Sayının Toplami:"<<toplam<<endl;
    return 0;
}
```

Program10.4.cpp Çıktısı

```
İki Sayının Toplami:20
```

Program10.4.cpp’de main fonksiyonda 2 tam sayı değişken (i ve j) tanımlanmış ve bu değişkenlere değer atanmıştır. Bu değişkenlerin toplamı için farklı bir tam sayı değişken (toplam) oluşturulmuştur

toplaIsaretci fonksiyonuna tanımlanan 3 tam sayı değişkenin adresleri gönderilmiştir. Fonksiyonun başlığında da görüleceği gibi parametre listesinde 3 işaretçi tanımlanmıştır. Gönderilen adresler sırası ile bu listede yer alan işaretçilere atanmıştır. Fonksiyonun içinde yer alan matematiksel ifadede referanstan ayırma işleci (*) ile ilgili adreslerin gösterdiği bellek alanındaki değerlere ulaşılmış ve toplama işlemi gerçekleştirilmiştir. Dolayısı ile fonksiyonun herhangi bir değer çevirmesine gerek yoktur ve bu nedenle dönüş tipi void olarak tanımlanmıştır.

3. İŞARETÇİLERDE CONST KULLANIMI

Eğer bir fonksiyon tanımında bir parametrenin değeri değişmiyorsa veya değişmemesi gerekiyorsa bu parametre const olarak tanımlanabilir. const niteleyicisinin fonksiyonların parametre listesinde yer alan işaretçiler ile 4 farklı kullanımı vardır. Ayrıca bu farklı durumlar bir fonksiyon içindeki işaretçi tanımlamalarında da geçerlidir.

- Sabit olmayan işaretçi ile sabit olmayan veri (const niteleyicisi kullanılmıyor.)
- Sabit olmayan işaretçi ile sabit veri
- Sabit işaretçi ile sabit olmayan veri
- Sabit işaretçi ile sabit veri

Sabit olmayan işaretçi, sabit olmayan veriyi gösterdiğinde const niteleyicisi kullanılmaz (örneğin, `int *ptr;`). Bu durum tanımlanabilecek en üst seviye erişim hakkıdır. Adresten ayırma işleci ve işaretçi ile gösterdiğimiz veriyi değiştirebiliriz. Aynı zamanda işaretçimizin bir başka bellek alanının göstermesini de sağlayabiliriz. Bu bölümde şu ana kadar kullanılan işaretçiler bu kısımda değerlendirilebilir.

```
int x=10, y =20;
```

```
int * ptr=&x;
```

Yukarıdaki tanımlamaların gerçekleştiğini varsayalım. Bu noktadan sonra programımızda, `*ptr=30;` ifadesi ile x tam sayı değişkeninin değerini 30 olarak değiştirebiliriz. Ayrıca `ptr=&y;` ifadesi ile ptr işaretçisinin ayrı bir değişkeni göstermesini de sağlayabiliriz. Bu haklara sahip olmamızın nedeni const niteleyicisinin kullanılmamış olmasıdır.

Sabit olmayan işaretçi ile sabit veriyi gösterdiğimizde const niteleyicisi veri tipinden önce yazılır (örneğin, `const int *ptr;`). Bu ifade “ptr işaretçisi sabit bir tam sayı değerini” gösterir şeklinde okunabilir. Adresten ayırma işleci ve işaretçi ile gösterdiğimiz veriyi değiştirebilmemiz söz konusu değildir. Ancak işaretçimizin bir başka bellek alanını göstermesini sağlayabiliriz.

```
int x=10, y =20;
```

```
const int * ptr=&x;
```

Yukarıdaki tanımlamaların gerçekleştiğini varsayalım. Bu noktadan sonra programımızda, `*ptr=30;` ifadesi ile x tam sayı değişkeninin değerini 30 olarak değiştirmeye çalıştığımızda derleyici hata üretir. Ancak `ptr=&y;` ifadesi ile ptr işaretçisinin ayrı bir değişkeni göstermesini sağlayabiliriz.

Sabit işaretçi ile sabit olmayan veriyi gösterdiğimizde const niteleyicisi işaretçiden önce yazılır (örneğin, `int * const ptr;`). Bu ifade “ptr sabit işaretçisi bir tam sayı değerini” gösterir şeklinde okunabilir. Adresten ayırma işleci ve işaretçi ile gösterdiğimiz veriyi değiştirebiliriz. Ancak işaretçimizin bir başka bellek alanını (bir başka değişken) göstermesini sağlamak mümkün olmaz.


```
int x=10, y =20;
int * const ptr=&x;
```

Yukarıdaki tanımlamaların gerçekleştiğini varsayalım. Bu noktadan sonra programımızda, *ptr=30; ifadesi ile x tam sayı değişkeninin değerini 30 olarak değiştirebiliriz. Ancak ptr=&y; ifadesi ile ptr işaretçisinin ayrı bir değişkeni göstermesini istediğimizde derleyici hata üretir. ptr const olarak tanımlanmıştır.

Sabit işaretçi ile sabit veriyi gösterdiğimizde const niteleyicisi işaretçiden önce ve veri tipinden önce olmak üzere 2 kez yazılır (örneğin, const int * const ptr;). Bu ifade “ptr sabit işaretçisi sabit bir tam sayı değerini” gösterir şeklinde okunabilir. Adresten ayırma işleci ve işaretçi ile gösterdiğimiz veriyi değiştirmemiz mümkün değildir. Aynı zamanda işaretçimizin bir başka bellek alanını (bir başka değişken) göstermesini de sağlayamayız.

```
int x=10, y =20;
const int * const ptr=&x;
```

Yukarıdaki tanımlamaların gerçekleştiğini varsayalım. Bu noktadan sonra programımızda, *ptr=30; ifadesi ile x tam sayı değişkeninin değerini 30 olarak değiştirmek istediğimizde derleyici hata üretir. Aynı zamanda ptr=&y; ifadesi ile ptr işaretçisinin ayrı bir değişkeni göstermesini istediğimizde derleyici hata üretir.

4. İŞARETÇİ İFADELERİ VE İŞARETÇİ ARİTMETİĞİ

İşaretçiler aritmetik ifadelerde, atama ifadelerinde ve karşılaştırma işleçlerinde işlenen olarak kullanılabilirler. Aşağıda verilen az sayıda aritmetik işlem işaretçiler üzerinde kullanılmak üzere tanımlanmışlardır.

- Artırma işleci (++)
- Azaltma işleci (--)
- Bir tam sayı, işaretçi değişkene eklenebilir (+ veya +=)
- Bir tam sayı, işaretçi değişkenden çıkartılabilir (- veya -=)
- Bir işaretçi diğer bir işaretçiden (aynı tipte tanımlanmaları şartı ile) çıkartılabilir (ancak bu çıkartma işlemi, işaretçiler bir dizinin elemanlarını gösterdiği takdirde anlamlı olabilir).



DİKKAT EDELİM

Günümüzde bilgisayar sistemleri 4 veya 8 baytlık tam sayılar kullanırlar. Bu nedenle işaretçi aritmetik işlemlerinin sonucu kullanılan sisteme göre değişebilir.

Bir işaretçiye bir dizi içinde herhangi bir dizi elemanının adresi atanmış olsun. Bu durumda eğer işaretçinin değerini 1 artırırsak işaretçi dizideki bir sonraki elemanı gösterir. Burada önemli olan durum işaretçinin gerçek değeri olan adres değerine 1 eklenmemiş, işaretçinin tipine göre işaretçinin değeri bir sonraki dizi elemanını gösterecek şekilde artırılmıştır. Eğer dizi double tipinde tanımlanmış ise ve double tipindeki işaretçi bu dizideki herhangi bir elemanını gösterecek şekilde tanımlanmışsa bu işaretçi değişkenini 1 artırmak adres değerini 8 bayt artırmak demektir. Bu durumun nedeni her bir

double değişkenin bellekte 8 baytlık alan kullanmasıdır.

```
//Program10.5.cpp
#include <iostream>
using namespace std;
int main()
{
    int a[10]={7,3,-8,12,5,-1,0,16,10,8};
    int *ptr;
    ptr=&a[2];
    cout<<*ptr<<endl;
    ptr++;
    cout<<*ptr<<endl;
    ptr=ptr+3;
    cout<<*ptr<<endl;
    ptr--;
    cout<<*ptr<<endl;
    return 0;
}
```

Program10.5.cpp Çıktısı

```
-8
12
0
-1
```

Program10.5.cpp’de 10 elemanlı bir tam sayı dizi tanımlanmış ve dizinin elemanlarına başlangıç değerleri atanmıştır. Daha sonra bir tam sayı işaretçi tanımlanmış ve dizinin 3. elemanının adresi bu işaretçiye atanmıştır. Programın devam eden kısmında bazı aritmetik işlemler ilgili işaretçi üzerinde uygulanmıştır. Her işlemten sonra işaretçinin gösterdiği tam sayı değeri ekrana yazdırılmıştır.

```
//Program10.6.cpp
#include <iostream>
using namespace std;
int main()
{
    double a[10]={7.0,3.2,-8.1,12.5,5.7,-1.3,0.0,16.2,10.8,8.5};
    double *ptr1;
    double *ptr2;
    ptr1=&a[4];
    ptr2=&a[1];
    cout<<ptr1-ptr2<<endl;
    if(ptr1>ptr2)
        cout <<"İlk isaretcici ikinci isaretciden daha büyüktür."<<endl;
    return 0;
}
```

Program10.6.cpp Çıktısı

```
3
```

İlk isaretci ikinci isaretciden daha büyüktür.

Program10.6.cpp'deki programda bir double dizi ve 2 ayrı double işaretçi değişken tanımlanmıştır. 1. double işaretçi 4. dizi elemanını, 2. double işaretçi ise 1. dizi elemanını gösterecek şekilde başlangıç değerleri almıştır. ptr2 değişkeni ptr1 değişkeninden çıkartıldığında 3 değerini elde ederiz. ptr1 değişkeni 5. elemanı, ptr2 değişkeni 2. elemanı gösterdiği için arada 3 elemanlık bir fark vardır. Dolayısı ile çıkarma işlemi bayt cinsinden bellek adresleri üzerinden gerçekleşmemiştir. Ayrıca aynı programda 2 işaretçi karşılaştırılmış ve daha yüksek indise sahip dizi elemanını gösteren işaretçinin büyük olduğu görülmüştür. Bayt adresi cinsinden bellekte yüksek indise sahip elemanın daha yüksek bir değere sahip olması beklenen bir durumdur.

5. İŞARETÇİLER VE DİZİLER ARASINDAKİ İLİŞKİ

İşaretçiler dizi alt simge işleci [] ile gerçekleştirilen tüm işlemlerde kullanılabilirler. Aşağıdaki şekilde bir tam sayı dizi ve bir tam sayı işaretçisi tanımlandığını düşünelim.

```
int a[5];
```

```
int *ptr;
```

a dizi ismi burada sabit bir işaretçi olarak değerlendirilebilir. Aynı zamanda dizinin ilk elemanının adresini tutar. Bir dizi tanımlandıktan sonra a'nın değeri (dizinin ilk elemanının adresi) program sonlanıncaya kadar değişmez. Ancak a değeri ile beraber işaretçi sapma ve işaretçi alt simge gösterimleri ile dizinin her bir elemanına erişmek mümkündür. ptr bir işaretçi değişkenidir ve const olarak tanımlanmadığı sürece program süresince farklı bellek alanlarında yer alan değişkenleri göstermek için kullanılabilir.

ptr=a; veya ptr=&a[0]; ifadeleri ile dizinin ilk elemanının adresi ptr işaretçi değişkenine atanır.

a dizisinin 3. elemanı dizi ismi ve alt simge işleci ile a[2] şeklinde erişilebilir. İşaretçi kullanarak ise *(ptr + 2) şeklinde işaretçi ifadesi ile aynı değere ulaşmak mümkündür. Bu gösterim işaretçi sapma gösterimi olarak isimlendirilir. Bu örnekte olduğu gibi &a[2] değerine (ptr +2) ifadesi ile erişilebilir. Aynı zamanda dizinin ismi bir işaretçi gibi kullanılarak *(a+2) şeklinde işaretçi sapma gösterimi ile de dizinin 3. elemanına ulaşmak mümkündür. Bunun yanında alt simge gösterimi de işaretçi olarak tanımlanan değişkenler ile birlikte kullanılabilir. Bu örnek için ptr[2] ifadesi ile aynı dizi elemanına erişilebilir. Dolayısı ile hem dizi ismi hem işaretçi ile işaretçi sapma ve işaretçi alt simge gösterimleri kullanılabilir. Bu da aşağıdaki 4 farklı durumu oluşturur:

- Dizi alt simge gösterimi
- İşaretçi sapma gösterimi (dizi ismi işaretçi)
- İşaretçi alt simge gösterimi
- İşaretçi sapma gösterimi (işaretçi değişkeni ile)

Program10.7.cpp'de yer alan programda bu 4 farklı durum gösterilmiştir. Farklı gösterimler ile dizinin 4. elemanına erişim sağlanmış ve elemanın değeri ekrana yazdırılmıştır.

```
//Program10.7.cpp
```

```
#include <iostream>
```

```
using namespace std;
```

```

int main()
{
    int a[5]={5, 10, 15, 20, 25};
    int *ptr;
    ptr=&a[0];
    cout<<"Dizi alt simge gosterimi: "<<a[3]<<endl;
    cout<<"Isaretci sapma gosterimi (dizi ismi isaretci): "<<*(a+3)<<endl;
    cout<<"Isaretci alt simge gosterimi: "<<ptr[3]<<endl;
    cout<<"Isaretci sapma gosterimi (isaretci degiskeni ile): "<<*(ptr+3)<<endl;
    return 0;
}

```

Program10.7.cpp Çıktısı

Dizi alt simge gosterimi: 20

Isaretci sapma gosterimi (dizi ismi isaretci): 20

Isaretci alt simge gosterimi: 20

Isaretci sapma gosterimi (isaretci degiskeni ile): 20



DİKKAT EDELİM

*(dizilsmi + i) ifadesinde dizilsmi değerini değiştirmiyoruz. Sadece bu değeri kullanarak yeni bir adres değeri oluşturuyoruz. Dizi isimleri sabittir ve program süresince değiştirilemezler.

BÖLÜM ÖZETİ

Bu bölümde C++ programlama dilinde kullanılan işaretçiler anlatılmıştır. İşaretçi herhangi bir veri tipinde tanımlanabilen ve adres tutan değişkenlerdir. İşaretçinin değeri, bellekte bir adrestir. İşaretçi tanımlandıktan sonra hangi veri tipinden tanımlandıysa o veri tipinden bir değişkenin adresini değer olarak alır. Bir başka ifade ile tanımlandığı veri tipi, sahip olduğu adres değerinin gösterdiği bellek alanında hangi veri tipinde değer olacağını gösterir. İşaretçiler kullanılarak değişkenler fonksiyonlara referans ile gönderilebilir. Fonksiyonlarda yer alan parametre listesinde, işaretçi tanımlamasında const kullanarak işaretçinin veya gösterdiği değerin değiştirilmesini engellemek mümkündür. Bazı aritmetik işlemler işaretçiler üzerinde kullanılabilir. Bu işlemler arasında; 1 artırma (++), 1 azaltma (--) ve atama işlemleri sayılabilir. Ayrıca aynı dizinin elemanlarını gösteren işaretçiler arasında çıkarma işlemi gerçekleştirilebilir. Diziler ve işaretçiler arasındaki ilişki de bu bölümde anlatılan bir diğer konudur. İşaretçi sapma ve işaretçi alt simge gösterimleri kullanarak hem dizi ismi hem de işaretçi değişkeni ismi ile ilgili dizinin elemanlarına erişmek mümkündür.

GÖZDEN GEÇİRELİM

1. `int a=10;`
`int *aptr=&a;`

Yukarıdaki kod parçası dikkate alındığında aşağıdakilerden hangisi bir adres değeri göstermez?

- a. a
- b. aptr
- c. &a
- d. &(*aptr)
- e. *(&aptr)

2. `int a=10;`
`int *aptr=&a;`
`int *bptr;`
`bptr=aptr;`

Yukarıdaki kod parçası dikkate alındığında aşağıdakilerden hangisi a değişkeninin değerini ekrana yazdırır?

- a. `cout<<&a<<endl;`
- b. `cout<<*aptr<<endl;`
- c. `cout<<*bptr<<endl;`
- d. `cout<<&aptr<<endl;`
- e. `cout<<aptr<<endl;`

3. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include<iostream>
using namespace std;
int main()
{
    int a=7;
    int b=5;
    int *ptr=&a;
    ptr=&b;
    cout<<*ptr+a+2*b<<endl;
    return 0;
}
```

- a. 12
- b. 19
- c. 20
- d. 22
- e. 26

4. `int x[7]={-2, 4, 11, 7, 6, -5, 8};`
`int *ptr=&x[2];`

Aşağıdaki ifadelerden hangisi x[6] değerini ekrana yazdırmak için kullanılamaz?

- a. `ptr+=4;`
`cout<<*ptr;`
- b. `cout<<*(x+6);`
- c. `cout<<ptr[4];`
- d. `cout<<x[6];`
- e. `x+=6;`
`cout<<*x;`

5. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
void fonksiyon(int *b)
{
    int i;
    for(i=0; i<=4; i++)
        b[i]=b[i]+5;
}
int main()
{
    int a[5]={4,8,3,5,6};
    fonksiyon(a);
    fonksiyon(a);
    cout<<a[4]<<endl;
    return 0;
}
```

- a. 11
- b. 16
- c. 17
- d. 20
- e. 21

6. Aşağıdakilerden hangisinde C++ programlama dilinde `int hesap(int *ptr)` prototipine sahip bir fonksiyon doğru bir şekilde çağrılmamıştır?

```
int a[5]={10,20,40,60,10};
```

```
int i=5;
```

```
int *j=&a[3];
```

a. `hesap(a[3])`

b. `hesap(a)`

c. `hesap(j)`

d. `hesap(&a[3])`

e. `hesap(&i)`

7. `int a=5;`
`int b=10;`
`int * const ptr=&a;`
Aşağıda verilen ifadelerden hangisi hata üretir?

a. `ptr=&b;`

b. `a=a+1;`

c. `b=b*2;`

d. `*ptr=*ptr+5;`

e. `b=a+b;`

8. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include<iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int a[5]={5, 10, 15, 11, 8};
```

```
    int *ptr;
```

```
    ptr=&a[1];
```

```
    cout<<*(&ptr)<<endl;
```

```
    return 0;
```

```
}
```

a. 5

b. 9

c. 10

d. 15

e. 16

9. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int a[5]={18, 20, 15, 14, 16};
```

```
    int *ptr1=&a[4];
```

```
    int *ptr2=&a[2];
```

```
    cout<<ptr1-ptr2<<endl;
```

```
    return 0;
```

```
}
```

a. 1

b. 2

c. 3

d. 4

e. 5

10. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
```

```
using namespace std;
```

```
void fonksiyon(int *b)
```

```
{
```

```
    *b=*b+15;
```

```
}
```

```
int main()
```

```
{
```

```
    int a=10;
```

```
    int b=20;
```

```
    fonksiyon(&a);
```

```
    cout<<a+b<<endl;
```

```
    return 0;
```

```
}
```

a. 10

b. 15

c. 20

d. 25

e. 45

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

Kaynakça

- Deitel, P., Deitel, H. (2017). C++ How to Program. 10. Baskı. Pearson. Essex, İngiltere.
- Dikici, M. (2017). C++ Programlama Dili: Yapısal Programlama, Nesnelerle Programlama ve Jenerik Programlama. Seçkin Yayıncılık. İstanbul, Türkiye.
- Çobanoğlu, B. (2021). C/C++ Programlama Eğitim Kitabı. Kodlab. İstanbul, Türkiye.
- Demir, S., Eryiğit, R., Ar, Y., Tuğrul, B., Gök, O., Kul, S., . . . Yağanoğlu, M. (2019). Programlama Temelleri. Erzurum: Atatürk Üniversitesi Açıköğretim Fakültesi.
- Kaleli, C., Bilge, A., Ergin, S., & Şora Günel, E. (2016). Bilgisayar ve Programlamaya Giriş. Eskişehir: Anadolu Üniversitesi.

BÖLÜM 11

SOYUT VERİ TİPLERİ

ANAHTAR KAVRAMLAR

Enum (Numalandırma), Fonksiyon İşaretçileri, Struct (Yapı)

ÖĞRENME HEDEFLERİ

-  1 Enum (numalandırma) tanımlamayı ve kullanmayı bilir.
-  2 Fonksiyon işaretçilerini kullanarak fonksiyonlara veri tipi oluşturur.
-  3 Oluşturulmuş veri tipini başka bir fonksiyonda parametre olarak kullanır.
-  4 Struct (yapı) tanımlamayı ve kullanmayı bilir.
-  5 Struct içerisinde fonksiyon oluşturur.
-  6 Struct değişkenini fonksiyonda parametre olarak kullanır.

GİRİŞ

Bu bölümde C++ programlama dilinde soyut veri tanımlamada kullanılan,

enum (numaralandırma),

Fonksiyon işaretçileri,

struct (yapı)

kavramları anlatılmıştır. Numaralandırma aynı küme-
deki elamanlara sabit tamsayı değerler atanması şeklin-
de yapılmaktadır. Küme elemalarının isimleri doğrudan
kendilerine atanan tamsayı değerlerine karşılık olarak
kullanılmaktadır. Fonksiyon işaretçileri ile veri türü ve
parametre sayısı aynı olan fonksiyonları değer olarak
alan fonksiyon veri tipi tanımlanır. Yapılar C++'da sınıf-
lara benzer özelliklere sahip olup, bir kavrama ait değiş-
kenler ve fonksiyonlarla oluşturulabilir.



1. SOYUT VERİ TİPLERİ

Soyut veri tipleri geliştirici tarafından çözülmeye çalışılan probleme özgü oluşturulan veri tipleridir. Problem çözümünün ilk aşamasındaki kavramsallaştırmanın programlama dillerindeki karşılıkları olan soyut veri tipleri, genel olarak kütüphane dosyalarının oluşturulması aşamasında geliştirilirler. Programlama dillerinin aktarım gücünün ölçütlerinden biri olan soyut veri tanımlama C++ dilinde,

- enum (numaralandırma),
- fonksiyon işaretçileri,
- struct (yapı),
- class (sınıf)

yapıları ile geliştirilmektedir.

Bu bölümde enum, fonksiyon işaretçileri ve struct ile veri tipleri açıklanacaktır. Nesne yönelimli dillerin temel programlama yapıları olan sınıflar bir sonraki bölümün konusudur.

2. ENUM (NUMARALANDIRMA)

C++ programlamada, enum (numaralandırma) tamsayı sabitlerini temsil eden adlandırılmış değerlerden oluşan bir veri türüdür. Değişkenlerin alabileceği farklı değerlerin sınırlı sayıda olması halinde programın okunabilirliğini artırmak için tercih edilir. Değişkenlere tek tek sayısal değerler vermek bazı durumlarda hataya sebep olabilmektedir. Bu durumu ortadan kaldırmak için enum kullanılabilir.

Numaralandırmanın C++'daki sözdizimi;

```
enum enum_adı {sabit1, sabit2, ...};
```

enum_adı'nın elemanları sabit1, sabit2, ... durumlarını içeren küme ismini ifade etmektedir. Örneğin günleri tutacak bir enum yapısında veri tipi (enum) ismi Gunler olacaktır.

Örnek:

```
#include <iostream>
using namespace std;
enum Gunler {Pazartesi, Sali, Carsamba, Persembe, Cuma, Cumartesi, Pazar};
int main () {
    cout<<"Pazartesi:"<<Pazartesi<<endl;
    cout<<"Sali:"<<Sali<<endl;
    cout<<"Carsamba:"<<Carsamba<<endl;
    cout<<"Persembe:"<<Persembe<<endl;
    cout<<"Cuma:"<<Cuma<<endl;
    cout<<"Cumartesi:"<<Cumartesi<<endl;
    cout<<"Pazar:"<<Pazar<<endl;
    return 0;
}
```

Çıktı:

Pazartesi:0

Sali:1

```
Carsamba:2
Persembe:3
Cuma:4
Cumartesi:5
Pazar:6
```

Program çıktısı enum içerisinde verilen isimlerin (const int Pazartesi = 0, Salı =1, ...) gibi değerleri enum'daki bildirim sırasına göre aldıklarını göstermektedir. Bu anlamda enum içerisindeki değerler Gunler kümesini oluşturan tam sayı sabitlerdir.

Gunler içerisindeki isimlerden herhangi birine değer atanırsa bir sonraki ismin değeri, atanan değerden bir büyük sayı olacaktır. Örneğin,

```
enum Gun {Pazartesi, Sali=3, Carsamba, Persembe, Cuma, Cumartesi, Pazar};
```

enum yapısı verilmiş olması durumunda Pazartesi değeri değişmezken Sali= -3, Carsamba=-2 Persembe =-1 şeklinde devam edecektir.

Çıktı:

```
Pazartesi:0
Sali:-3
Carsamba:-2
Persembe:-1
Cuma:0
Cumartesi:1
Pazar:2
```

Enum içerisindeki isimlerin program tarafından sabit tamsayı değerlerine karşılık geldiğini gördüğümüze göre, bu değerlerden oluşturulan Günler veri tipi nasıl kullanılacak?

C++'da bir değişken bildirimi;

Veri_tipi değişken_ismi;

Değer ataması;

değişken_ismi = veri kümesi içerisindeki değerlerden biri;

şeklinde olmaktadır. Bu hatırlatmaya dayanarak,

```
Gun gun; // değişken bildirimi
gun = Persembe;
```

veya

```
Gun gun = Persembe;
```

şeklinde yapılabilir.

Programlamada enum'ların yaygın kullanıldığı durumlardan biri küme elemanlarının temsil ettiği değişkenlerde yapılması gereken işlerin farklılık göstermesidir. Renk kümesinin elemanlarının değerlerine göre farklı kod gruplarının seçimi aşağıda verilen örnek kod üzerinde gerçekleştirilmiştir.

Örnek:

```
#include <iostream>
using namespace std;
enum Renk{beyaz=3, siyah=5, turuncu};
int main() {
    Renk r = turuncu;
    switch(r)
    {
        case turuncu: cout << "Turuncu\n"; break;
        case beyaz: cout << "Beyaz\n"; break;
        case siyah : cout << "Siyah\n"; break;
        default: cout << "Renk yok\n"; break;
    }
    return 0;
}
```

Çıktı:

Turuncu

**DİKKAT EDELİM**

Yapı (struct) ile oluşturulan veri tipindeki değişkene değer ataması Veri tipi değişken = {değer1, değer2 ,...} biçiminde yapılabilir.

3. YAPI İÇERİSİNDE FONKSİYON KULLANMA

C++ programlama dilinde yapı içerisinde fonksiyon geliştirilerek tanımlanan kavrama ait yapılabilecek işler tek bir programlama biriminde tamamlanabilir. Yukarıdaki Dikdörtgen örneğini fonksiyon içeren yapıya dönüştürürsek;

Örnek:

```
#include<iostream>
using namespace std;
struct Dikdortgen{
    int yukseklik;
    int genislik;
    void alan(){
        int al = yukseklik * genislik;
        cout<<" Alan :"<<al<<endl;
    }
    void cevre(){
        int cev = 2* (yukseklik + genislik);
        cout<<" Cevre :"<<cev<<endl;
    }
}
```

```

    }
};
int main(){
    Dikdortgen d = {3,4}; //değişken bildirim ve değer atama
    d.cevre(); // fonksiyon çağırma
    d.alan();
    return 0;
}

```

Çıktı:

Cevre :14

Alan :12



DİKKAT EDELİM

Yapı (struct) içerisinde fonksiyon kullanarak veri tipindeki değişkenin hesaplanmak istenilen değerleri yapıyla bütünleştirilebilir.

4. YAPI ÜRETİCİ FONKSİYONU

Yapı içerisinde yapı ismiyle aynı isimli kod grubuna üretici fonksiyon denir. Üretici fonksiyonların geri dönüş değeri yoktur. Yapı değişkeni oluşturulurken ilk değerlerin atamasının yapılmasında kullanılır.

```

struct Dikdortgen{
    int yukseklik;
    int genislik;
    // üretici fonksiyon tanımlama
    Dikdortgen (int y, int g){
        yukseklik = y;
        genislik = g;
    }
};

```

Üretici fonksiyona sahip bir yapı tipindeki değişkene değer atama;

Dikdortgen d = Dikdortgen(4, 5);

Dikdortgen d1 = {4, 5};

veya

Dikdortgen d2;

d2.yukseklik = 4;

d2.genislik = 5;

şeklinde yapılabilir.

**DİKKAT EDELİM**

Yapıya ait üretici fonksiyon yapının ismiyle aynı isimdeki özel bir fonksiyon olup geri dönüşü yoktur.

```
struct Uretici{  
    int numara;  
    string ad;  
    Uretici(int x, string a){  
        numara =x;  
        ad = a;}  
}  
};
```

BÖLÜM ÖZETİ

Bu bölümde, geliştiricilerin programlarında yeni veri tipleri tanımlamak amacıyla kullandıkları **enum**, **fonksiyon işaretçileri** ve **struct kavramları** ele alınmıştır.

Enum (enumeration), bir küme içerisindeki değişkenlerin sabit tamsayılarla ifade edilmesini sağlayan bir veri tipi tanımlama mekanizmasıdır. Bu yapı, değişkenlerin daha anlamlı ve okunabilir bir şekilde temsil edilmesine olanak tanır.

Fonksiyon işaretçileri, aynı dönüş tipine ve parametre yapısına sahip fonksiyonları temsil etmek için kullanılan veri tipleridir. Bu yapılar, fonksiyonların birer veri nesnesi gibi işlenmesini sağlar ve esnek programlama imkanları sunar.

Yapılar (struct), C++ dilindeki en karmaşık veri yapılarından biridir ve sınıflarla (class) birlikte gelişmiş veri modellemelerine olanak tanır. Bu bölümde, bir yapı (struct) tanımlama, bu yapıdan türetilen değişkenlerin bildirilmesi ve değer atanması üzerinde durulmuştur. Ayrıca, yapılarda fonksiyon tanımlama, üretici fonksiyon (constructor) oluşturma ve kullanma süreçleri detaylı bir şekilde açıklanmıştır.

Bu kavramlar, C++ programlamasında özelleştirilmiş veri tipleri oluşturmak ve bu tipler üzerinde etkin işlemler gerçekleştirmek için önemli bir temel oluşturmaktadır.

GÖZDEN GEÇİRELİM

1.

```
#include <iostream>
using namespace std;
enum Renk{sari, mavi=5, kirmizi, yesil, siyah};
int main() {
    Renk r = siyah;
    cout<<"Renk : "<< r <<endl;
    return 0;
}
```

Yukarıda verilen kodun çıktısı nedir?

- a. 8
- b. 0
- c. 3
- d. 4
- e. 5

```
#include <iostream>
using namespace std;
enum Gunler {pazar, pazartesi, sali, carsamba, persembe, cuma, cumartesi};
int main () {
    Gunler g = sali;
    switch(g){
        case pazartesi:{
            cout<<"Pazartesi ";
            break;
        }
        case sali:{
            cout<<"Sali";
        }
        case carsamba:{
            cout<<"Carsamba ";
        }
        case persembe:{
            cout<<"Persembe ";
        }
        case cuma: {
            cout<<"Cuma ";
            break;
        }
    }
    return 0;
}
```

2.

```
#include <iostream>
using namespace std;
enum Renk{sari=-1.2, mavi, kirmizi, yesil, siyah};
int main() {
    Renk r = kirmizi;
    cout<<"Renk : "<<r <<endl;
    return 0;
}
```

Yukarıda verilen kodun çıktısı nedir?

- a. Program derleme hatası verir.
- b. -2
- c. -1
- d. 0
- e. 1

3. Yukarıda verilen programın çıktısı nedir?

- a. Sali Carsamba Persembe Cuma
- b. persembe
- c. sali
- d. carsamba
- e. Cuma

```
float cikar(float x, float y){
    return x - y;
}
float carp(float x, float y){
    return x * y;
}
```

4. Fonksiyonlarını değer olarak alacak işaretcı fonksiyonu aşağıdakilerden hangisinde doğru tanımlanmıştır?

- typedef float (*Islem)(float x ,float y);
- typedef float Islem(float x ,float y);
- typedef float (*Islem)(int x ,int y);
- typedef (*Islem)(float x ,float y);
- typedef vod (*Islem)(float x ,float y);

5. void hesapla(float x, float y, Islem op,string islem){
cout<<islem<<" "<<op(x,y)<<endl;
}

Fonksiyonunun çağrılması aşağıdakilerden hangisinde doğru yapılmıştır.

- Islem op = topla; hesapla(1.0, 2.0, op, "topla");
- Islem op = topla; cout<<hesapla(1.0, 2.0, op, "topla");
- op Islem = topla; hesapla(1.0, 2.0, op, "topla");
- op topla = Islem; hesapla(1.0, 2.0, op, "topla");
- Islem topla = op; hesapla(1.0, 2.0, op, "topla");

6. Kişilerin ad, soyad, yas bilgilerini tutacak yapı aşağıdakilerden hangisinde doğru tanımlanmıştır.

- struct Kisi{
string ad;
string soyad;
int yas;
};
- Kisi struct {
string ad;
string soyad;
int yas;
};
- Kisi {
string ad;
string soyad;
int yas;
} struct;
- Kisi{
int ad;
int soyad;
string yas;
};
- struct Kisi(
string ad;
string soyad;
int yas;
);

7. struct Daire{
float r;
};

Yukarıda verilen Daire koduna aşağıdakilerden hangisi eklenirse, Daire d = Daire(1.5); ifadesi çalışır.

- Daire (float x) {r = x;}
- void Daire(float x){r =x;}
- void Daire(float x){x =r;}
- Daire (float x) {x = r;}
- Yeni kod eklemek gerekmez.

8. İki boyutlu düzlemde bir nokta x ve y eksen değerleri ile verilmektedir. Bir noktanın değerlerinin girileceği veri tipi aşağıdakilerden hangisinde tanımlanmıştır.

- a. struct Nokta{
float x;
float y;
};
- b. enum Nokta{
float x;
float y;
};
- c. Nokta(float x, float y);
- d. Nokta enum {
float x;
float y;
};
- e. enum Nokta{
float x;
float y;
};

10.

```
#include <iostream>
using namespace std;
int main() {
    student{
        int num;
        string name;
    };
    student stu;
    stu.num = 123;
    stu.name = "Ali";
    cout << stu.num << endl;
    cout << stu.name << endl;
    return 0;
}
```

Yukarıda verilen kodun çıktısı nedir?

- a. Derleme hatası verir.
- b. 123 Ali
- c. 123
- d. Ali
- e. Ali 123

9. C++ dilinde yapı (struct) aşağıdakilerden hangisi ile sonlandırılır?

- a. ;
- b. :
- c. }
- d. .
- e. :

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

Kaynakça

- Deitel, P., Deitel, H. (2017). C++ How to Program. 10. Baskı. Pearson. Essex, İngiltere.
- Dikici, M. (2017). C++ Programlama Dili: Yapısal Programlama, Nesnelerle Programlama ve Jenerik Programlama. Seçkin Yayıncılık. İstanbul, Türkiye.
- Çobanoğlu, B. (2021). C/C++ Programlama Eğitim Kitabı. Kodlab. İstanbul, Türkiye.
- Demir, S., Eryiğit, R., Ar, Y., Tuğrul, B., Gök, O., Kul, S., . . . Yağanoğlu, M. (2019). Programlama Temelleri. Atatürk Üniversitesi Açıköğretim Fakültesi.
- Kaleli, C., Bilge, A., Ergin, S., & Şora Günel, E. (2016). Bilgisayar ve Programlamaya Giriş. Eskişehir: Anadolu Üniversitesi.

BÖLÜM 12

NESNEYE YÖNELİK PROGRAMLAMA

ANAHTAR KAVRAMLAR

Sınıf, Üye Fonksiyonlar, Veri Üyeleri, Yapıcı Fonksiyon, Yıkıcı Fonksiyon, İşleç Aşırı Yükleme, Bileşim, Katılım, Çok Biçimlilik

ÖĞRENME HEDEFLERİ

- 1 Üye fonksiyonları ve veri üyeleri ile sınıf tanımlı yapılabilir.
- 2 Sınıf kullanarak kullanıcı tanımlı veri tipi oluşturulabilir.
- 3 Oluşturulan yeni veri tipinden nesneler tanımlanabilir.
- 4 İşleç aşırı yükleme gerçekleştirilerek, işleçleri sınıflardan tanımlanan nesneler üzerinde kullanılabilir.
- 5 Bileşim işlemi ile bir sınıfta, başka bir sınıfın nesnesini veri üyesi olarak kullanılabilir.
- 6 Kalıtım kullanarak temel sınıflardan türetilmiş sınıflar oluşturulabilir.
- 7 Çok biçimlilik kullanarak türetilmiş sınıf nesnelerine temel sınıf işaretçileri veya referansları ile ulaşılabilir.

GİRİŞ

Bu bölümde önce sınıf tanımlamasının nasıl gerçekleştirileceği anlatılacaktır.

Sınıf kullanarak kullanıcı tanımlı yeni veri tipleri oluşturulabilir. Bu şekilde C++ programlama dilinde var olan yerleşik veri tiplerine yeni veri tipleri eklenebilir ve dilin kapasitesi genişletilebilir. Sınıf tanımlamalarında gerekli üye fonksiyonlar ve veri üyeleri ve bu üyelere erişim seviyeleri detaylı olarak tartışılacaktır. Yapıcı ve yıkıcı fonksiyonlar ve kullanım amaçları sınıflar açısından önemlidir. Nesneler tanımlandığında otomatik olarak yapıcı fonksiyonlar, yok edildiğinde otomatik olarak yıkıcı fonksiyonlar çağrılır. En son oluşturulan nesne, ilgili kapsam alanı tamamlandığında ilk yok edilecek nesne olacaktır. İşleç aşırı yükleme gerçekleştirilerek yerleşik veri tiplerinde kullanılan işleçler, sınıflardan tanımlanabilecek nesneler üzerinde kullanılabilir. Bu bölümde toplama ve ekrana yazdırma işleçlerinin aşırı yükleme ile Rasyonel sınıfına ait nesneler üzerinde kullanılması örnek olarak gösterilecektir. Bir sınıfın nesnesinin bir başka sınıf içerisinde veri üyesi olarak tanımlanması bileşim olarak isimlendirilir. Bir Tarih sınıfı nesnesinin Personel sınıfında veri üyesi olarak tanımlanması bu bölümdeki örneklerden birisidir. Bazı sınıflar var olan tanımlanmış sınıflar kullanılarak tanımlanabilir. Var olan sınıf temel sınıf olarak tanımlanır ve üyeleri yeni tanımlanan türemiş sınıfın de üyeleri olur. Türemiş sınıfın kendine ait üye fonksiyonları ve veri üyeleri olabilir. Tüm kalıtım hiyerarşisindeki sınıf nesnelerine temel sınıf işaretçileri veya referansları kullanarak erişmek mümkündür. Bu şekilde tüm nesneler farklı türemiş sınıflara da ait olsalar tek bir listede veya dizide tutulabilir. Sanal fonksiyonlar yardımı ile dinamik olarak doğru nesnenin ilgili fonksiyonu çağrılarak tek bir liste üzerinden tüm sınıfların nesnelerine erişmek mümkün olmaktadır. Bu yaklaşım çok biçimlilik olarak tanımlanır.



1. SINIFLAR VE NESNELER

C++ programlama dilinde geliştiricilerin kullanabilmesi için tanımlanmış yerleşik veri tipleri vardır. Bu veri tipleri int, float, double ve char başta olmak üzere daha önceki bölümlerde anlatılmıştır. Bu veri tipleri sayesinde tam sayı, kesirli sayı ve karakter gibi nesneleri tanımlamak ve programlarda uygun durumlarda kullanmak mümkündür. Bu veri türleri üzerinde gerçekleştirilebilecek işlemler de tanımlıdır. Örnek olarak; tam sayılar üzerinde, toplama, çıkartma, çarpma, bölme vb. işlemler tanımlıdır. Bu işlemler çoğunlukla ikili işlemlerdir ve iki adet işlenen gereklidir. Ayrıca tekli işlemler de tam sayılar üzerinde kullanılabilir. Örneğin (++) ve (--) işlemleri sadece bir tam sayı işlenen üzerinde çalışırlar.

Bir C++ programında kullanabileceğimiz yerleşik veri tipleri tüm ihtiyaçları karşılama konusunda bazı durumlarda yetersiz kalabilmektedir. Birçok problemin çözümünde tam sayı, kesirli sayı, karakter vb. veri tipleri ile oluşturabileceğimiz nesnelerin dışında farklı nesnelere gereksinim söz konusudur. Karmaşık sayılar ve rasyonel sayılar gibi matematiksel nesneleri tanımlamak için yeni veri tipleri kullanabilme yeteneği programlama diline, problemlerin çözümünde büyük kolaylıklar getirecektir. Ayrıca yeni tanımlanabilecek nesneler sadece matematiksel sayılar ile sınırlı olmayacaktır. Günlük hayatta karşımıza çıkabilecek her türlü nesne yeni bir veri tipi olarak tanımlanabilme imkanına sahip olabilecektir. Örneğin, bir öğrenci nesnesi, bir çalışan nesnesi, koordinat sisteminde bir nokta nesnesi vb. nesneler rahatlıkla tanımlanabilecek ve kullanılacaklardır. Bu tür durumlardan dolayı yerleşik veri tiplerine ek olarak kullanıcı tanımlı yeni veri tipleri söz konusu olmuştur. Kullanıcı tanımlı yeni veri tipleri C++ programlama dilinde sınıf yapıları kullanılarak gerçekleştirilir. Bu veri tipleri için, ilgili nesneyi tanımlayan veri üyeleri ve bu nesne üzerinde gerçekleştirilebilecek işlemleri gösteren üye fonksiyonlar sınıf yapısı içinde tanımlanır.

Program12.1.cpp rasyonel sayılar için yazılmış bir sınıf tanımı içermektedir.

```
//Program12.1.cpp
#include <iostream>
using namespace std;
class Rasyonel
{
public:
    Rasyonel(){
        pay=1;
        payda=2;
    }
    void setRasyonel(int a, int b)
    {
        pay=a;
        payda=b;
    }
    void yazdirRasyonel()
    {
        cout<<pay<<"/"<<payda<<endl;
    }
}
```

```

private:
    int pay;
    int payda;
};
int main()
{
    Rasyonel r1;
    r1.yazdirRasyonel();
    r1.setRasyonel(3,5);
    r1.yazdirRasyonel();
    return 0;
}
Program12.1.cpp Çıktısı
1/2
3/5

```

Burada verilen Rasyonel sınıfını kullanarak bazı temel kavramları açıklayalım. Bir sınıf tanımlamak için class anahtar kelimesi kullanılır. class kelimesinden sonra sınıfın adı yazılır. Daha sonra kıvrımlı parantezler içerisinde sınıf tanımı gerçekleştirilir. public ve private olmak üzere temel iki farklı erişim seviyesi mevcuttur. Üye fonksiyonlar genellikle public olarak tanımlanırlar. public tanımlanan bir fonksiyon veya değişken sınıf tanımının dışında da ulaşılabilir. private erişim seviyesinde yer alan üyeler ise sadece sınıfın kapsam alanında erişilebilirler. Varsayılan erişim seviyesi private olarak belirlenmiştir. Dolayısıyla herhangi bir erişim seviyesi belirtilmemişse ilgili üyeler private olur.

Sınıfın ismi ile aynı isme sahip özel bir üye fonksiyon vardır. Bu fonksiyon yapıcı fonksiyon (constructor) olarak isimlendirilir. Her hangi bir sınıf için bir değişken tanımlandığında yapıcı fonksiyon çağrılır. Program12.1.cpp'de main fonksiyonda r1 isminde bir Rasyonel nesnesi oluşturulur. Bu nesne oluşturma işlemi Rasyonel sınıfının yapıcı fonksiyonunu çağırır. Yapıcı fonksiyonun içinde pay ve payda veri üyelerinin değerleri sırasıyla 1 ve 2 olarak atanmıştır. Burada da görüldüğü gibi private olarak tanımlanan üyeler sınıf fonksiyonları içinde doğrudan kullanılabilirler. setRasyonel üye fonksiyonu iki tam sayı parametre alır. Parametreler fonksiyon tanımı içerisinde veri üyelerine atanır. Fonksiyon her hangi bir değer çevirmedeği için dönüş tipi void olarak belirlenmiştir.

yazdirRasyonel fonksiyonu parametre almaz. Fonksiyonu çağırın nesnenin veri üyeleri ekrana rasyonel sayı formatında yazdırılır. Bu fonksiyon da her hangi bir değer çevirmez ve bu nedenle dönüş tipi void olarak yazılmıştır.

Bir üye fonksiyonu çağırmak için ilgili sınıf nesnesi '.' işleci ve fonksiyon ismi ile beraber kullanılır.

```
r1.yazdirRasyonel();
```

ifadesinde yazdirRasyonel fonksiyonu r1 nesnesi ile çağrılır. r1 nesnesi ilk oluşturulduğunda yapıcı fonksiyonlar çağrılmış; pay ve payda değerleri 1 ve 2 olarak atanmıştı. Bir sınıftan oluşturulan tüm nesnelerin kendilerine ait veri üyeleri vardır. Dolayısı ile bu örnekte de r1 nesnesinin kendine ait pay ve payda değerleri bulunmaktadır. Bir nesnenin bellekte kapladığı alan bir anlamda sahip olduğu veri üyelerinin boyutu kadardır. yazdirRasyonel fonksiyonu çalıştığında Rasyonel sınıfının r1 nesnesine ait pay ve payda değerleri ekrana yazdırılır.

```
r1.setRasyonel(3,5);
```

ifadesi ile setRasyonel fonksiyonu r1 nesnesi ile çağırılır. Yapıcı fonksiyon ile başlangıç değeri 1 ve 2 olarak atanan r1 nesnesinin pay ve payda veri üyelerine parametre olarak gönderilen 3 ve 5 değerleri atanır. yazdirRasyonel fonksiyonu tekrar çağrıldığı zaman ekrana r1 nesnesinin veri üyelerinin yeni değerleri yazdırılır.

r1 nesnesine ait veri üyeleri private erişim seviyesi ile tanımlandığından doğrudan main fonksiyon içinde ulaşılamazlar. Ancak üye fonksiyonlar public olarak tanımlandığından main fonksiyon içinde doğrudan erişilebilirler. private olarak tanımlanan erişime sahip üyeler sınıf kapsama alanında, örneğin public tanımlanan üye fonksiyonlar içinde doğrudan erişilebilirler.



DİKKAT EDELİM

Yapıcı fonksiyonlar bir nesne oluşturulduğunda otomatik olarak çağırılırlar. Bu fonksiyonlar sınıfın ismi ile aynı isme sahiptirler.

```
//Program12.2.cpp
#include <iostream>
using namespace std;
class Rasyonel
{
public:
    Rasyonel(){
        pay=1;
        payda=2;
    }
    Rasyonel(int i, int j){
        pay=i;
        payda=j;
    }
    void setRasyonel(int a, int b){
        pay=a;
        payda=b;
    }
    void setPay(int a){
        pay=a;
    }
    void setPayda(int a){
        payda=a;
    }
}
```

```

        int getPay(){
            return pay;
        }
        int getPayda(){
            return payda;
        }
        void yazdirRasyonel()
        {
            cout<<pay<<"/"<<payda<<endl;
        }
private:
        int pay;
        int payda;
};

int main()
{
    Rasyonel r1;
    Rasyonel r2(4,7);
    r1.yazdirRasyonel();
    r2.yazdirRasyonel();
    r1.setPay(5);
    r1.setPayda(9);
    r1.yazdirRasyonel();
    r1=r2;
    r1.yazdirRasyonel();
    cout<<r1.getPay()<<"/"<<r1.getPayda()<<endl;
    return 0;
}

```

Program12.2.cpp Çıktısı

```

1/2
4/7
5/9
4/7
4/7

```

Program12.2.cpp’de sınıf tanımı içerisinde bireysel olarak üye verilere değer atayan ve değerleri çeviren üye fonksiyonlar tanımlanmıştır. Bu fonksiyonlar set ve get fonksiyonları olarak isimlendirilirler. private olarak tanımlanan üye verileri doğrudan sadece sınıf kapsama alanında erişilebilirler. Bu nedenle sınıf kapsama alanı dışında örneğin main fonksiyonda bu verilere erişebilmek için bu verileri bireysel olarak döndüren fonksiyonlara ihtiyaç vardır. Örnek programımızda pay ve payda üye verilerini çeviren getPay ve getPayda fonksiyonları verilmiştir. Aynı zamanda bu verilere bireysel olarak değer atayan setPay ve setPayda üye fonksiyonlarının kullanımı da gösterilmiştir.

```

//Program12.3.cpp
#include <iostream>
using namespace std;
class Rasyonel
{
public:
    Rasyonel(){
        pay=1;
        payda=2;
        cout<<"Yapici (constructor) fonksiyon cagrildi."<<" Pay:"<<pay<<"
        Payda:"<<payda<<endl;
    }
    Rasyonel(int i, int j){
        ppay=i;
        ppayda=j;
        pcout<<"Yapici (constructor) fonksiyon cagrildi."<<"
        Pay:"<<pay<<" Payda:"<<payda<<endl;
    }
    ~Rasyonel()
    {
        cout<<"Yikici (destructor) fonksiyon cagrildi."<<" Pay:"<<pay<<"
        Payda:"<<payda<<endl;
    }
    void yazdirRasyonel()
    {
        cout<<pay<<"/"<<payda<<endl;
    }
private:
    int pay;
    int payda;
};

int main()
{
    Rasyonel r1;
    Rasyonel r2(4,7);
    Rasyonel *r1Ptr, *r2Ptr;
    r1Ptr=&r1;
    r2Ptr=&r2;
    r1Ptr->yazdirRasyonel();
    r2Ptr->yazdirRasyonel();
    return 0;
}

```

Program12.3.cpp Çıktısı

Yapici (constructor) fonksiyon cagrildi. Pay:1 Payda:2

Yapici (constructor) fonksiyon cagrildi. Pay:4 Payda:7

1/2

4/7

Yikici (destructor) fonksiyon cagrildi. Pay:4 Payda:7

Yikici (destructor) fonksiyon cagrildi. Pay:1 Payda:2

Program12.3.cpp'de yıkıcı fonksiyon (destructor) kullanımı gösterilmiştir. Bir nesne tanımlandığı kapsama alanı tamamlanınca bellekten silinir. Yıkıcı fonksiyonlar tam bu noktada otomatik olarak çağrılırlar. Yapıcı fonksiyonlar gibi özel fonksiyonlardır ve sınıfın ismi ile aynı isimde olmak zorundadırlar. İsim olarak yapıcı fonksiyondan farkları '~' karakteri ile başlamalarıdır. Nesne ile ilgili gerçekleştirilmesi gereken bellek yönetimi işlemlerinin tamamlanmasını sağlayan ifadeler bu fonksiyonlar içinde yer alır. Örneğin bir üye verisi dinamik bellek alanı kullanıyorsa bu alanın geriye verilme işlemi bu fonksiyonlar içine yazılabilir.

Yapıcı fonksiyonlar bir nesne yaratıldığında, yıkıcı fonksiyonlar ise bir nesne silindiğinde çağrılırlar. Program.12.3.cpp'de yapıcı ve yıkıcı fonksiyonlar içinde ekrana, ilgili fonksiyonların çağrıldığına yönelik ifadeler yazdırılmıştır. Bu şekilde yapıcı ve yıkıcı fonksiyonların çağırılma sıraları takip edilebilir. Bu örnekte de görüldüğü gibi bir kapsam alanında en son oluşturulan nesne ilk yok edilen nesne olmaktadır. Önce r1 nesnesi daha sonra r2 nesnesi oluşturulmuş ancak önce r2 nesnesi daha sonra r1 nesnesi silinmiştir.

Ayrıca Program12.3.cpp'de bir sınıf tipinden tanımlanan işaretçilerin kullanımı gösterilmiştir. Diğer yerleşik veri tiplerinde olduğu gibi kullanıcı tanımlı veri tipleri için de işaretçiler tanımlanabilir. Bu örnekte r1Ptr ve r2Ptr işaretçilerine sırasıyla r1 ve r2 nesnelerinin adresleri atanmıştır. Bir işaretçi kullanarak üye fonksiyonu çağırmak için '->' işleci kullanılır. Bu örnekte yazdirRasyonel fonksiyonu r1Ptr ve r2Ptr işaretçileri ile çağırılmıştır.

```
//Program12.4.cpp
#include <iostream>
using namespace std;
class Rasyonel
{
    public:
        Rasyonel(){
            pay=1;
            payda=2;
        }
        Rasyonel(int i, int j){
            pay=i;
            payda=j;
        }
        void setRasyonel(int a, int b){
```



```

        pay=a;
        payda=b;
    }
    Rasyonel toplaRasyonel(Rasyonel r)
    {
        Rasyonel temp;
        temp.pay=payda*r.pay+r.payda*pay;
        temp.payda=payda*r.payda;
        return temp;
    }
    void yazdirRasyonel()
    {
        cout<<pay<<"/"<<payda<<endl;
    }
private:
    int pay;
    int payda;
};
int main()
{
    Rasyonel r1;
    Rasyonel r2(4,7);
    Rasyonel r3;
    r1.yazdirRasyonel();
    r2.yazdirRasyonel();
    r3=r1.toplaRasyonel(r2);
    r3.yazdirRasyonel();
    return 0;
}

```

Program12.4.cpp Çıktısı

```

1/2
4/7
15/14

```

Program12.4.cpp’de toplaRasyonel fonksiyonunun kullanımı gösterilmiştir. Bu fonksiyon ile iki rasyonel sayının toplanması gerçekleştirilmiştir. toplaRasyonel fonksiyonu bir sınıf nesnesinin fonksiyonlarda dönüş tipi ve parametre olarak kullanılabilmesini göstermesi açısından iyi bir örnektir. toplaRasyonel fonksiyonunun dönüş tipi Rasyonel sınıfının nesnesi olarak belirlenmiştir. Aynı zamanda fonksiyon bir Rasyonel sınıfı nesnesini parametre olarak almaktadır.

```
r3=r1.toplaRasyonel(r2);
```

ifadesinde toplaRasyonel fonksiyonu r1 nesnesi tarafından çağırılmıştır. Dolayısı ile bu fonksiyonun içinde doğrudan ulaşılan pay ve payda veri üyeleri r1 nesnesinin üyeleridir. toplaRasyonel fonksiyonu-

nun aldığı parametre r2 nesnesi (parametre listesinde r olarak listelenmiş, değer ile çağırma yaklaşımı kullanılmıştır), r1 nesnesi ile toplanmak istenen rasyonel sayıdır. Bu nesnenin pay ve paydasına fonksiyon içinde erişmek için r.pay ve r.payda ifadeleri kullanmak gerekir. Fonksiyonun içinde temp isminde bir Rasyonel nesnesi oluşturulmuş ve bu nesnenin pay ve paydası iki nesnenin toplanması ile bulunmuştur. Elde edilen nesne toplaRasyonel fonksiyonunun sonucu olarak çevrilmiştir. main fonksiyonda çevrilen bu değer r3 nesnesine atanmıştır ve yazdirRasyonel fonksiyonu ile ekrana yazdırılmıştır.



ARAŞTIRALIM

Sınıf üye fonksiyonlarının sınıf tanımlaması dışında geliştirilmesi konusunu araştırınız.

2. İŞLEÇLERİN AŞIRI YÜKLENMESİ

Program14.4.cpp’de toplaRasyonel fonksiyonu kullanılarak iki rasyonel sayı toplanmıştır. Ancak iki tam sayıyı toplamak için ‘+’ işleci kullanılabilirken, iki rasyonel sayıyı toplayabilmek için sayılardan biri ile bir üye fonksiyon çağırarak gerekmektedir. Yerleşik veri tipleri ile kullanılabilen işleçleri kullanıcı tanımlı veri tipleri ile kullanabilmek için işleçlerin aşırı yüklenmesi yaklaşımı geliştirilmiştir. Temel amaç; $r3=r1+r2$; ifadesi ile iki rasyonel sayıyı toplamaktır.

Bir tam sayıyı ekrana yazdırmak için ‘<<’ işleci kullanılmaktadır. Ancak bir rasyonel sayıyı ekrana yazdırmak için Program14.4.cpp’de yazdirRasyonel fonksiyonu kullanılmıştır. Yerleşik veri tiplerini yazdırmak için kullandığımız ‘<<’ işlecini kullanıcı tanımlı tipler için de kullanılabilmesi beklenmektedir. ‘+’ işlecinde olduğu gibi ‘<<’ işleci de, işleçlerin aşırı yüklemesi yaklaşımı ile kullanıcı tanımlı tipler için de tanımlanabilir. Buradaki temel amaç yazdirRasyonel fonksiyonunu çağırarak yerine `cout<<r;i-fadesi` ile bir rasyonel sayıyı ekrana yazdırmaktır.

İşleç aşırı yükleme için tanımlanacak fonksiyonun ismi operator anahtar kelimesi ve işleç karakterinden oluşur. Program12.5.cpp’de `operator<<` ve `operator+` fonksiyonları işleç aşırı yükleme için tanımlanmışlardır. `operator+` fonksiyonunun tanımı toplaRasyonel fonksiyonunun tanımına benzerdir. Bu fonksiyon ‘+’ işlecinin sol tarafındaki rasyonel sayı tarafından çağırılır. İşlecın sağ tarafındaki rasyonel sayı ise `operator+` fonksiyonuna parametre olarak gönderilir.

`operator<<` fonksiyonu friend olarak tanımlanmış bir işleç aşırı yükleme fonksiyonudur. friend fonksiyonlar bir sınıfın üye fonksiyonu olmamakla beraber, sınıfın public üyelerinin yanında private üyelerine de erişme yetkisine sahip fonksiyonlardır. `cout<<r` ifadesinde ‘<<’ işlecın sol tarafında bir ostream sınıf nesnesi olduğu için bu fonksiyon doğrudan üye fonksiyon olarak tanımlanamaz. İşleç aşırı yükleme için tanımlanan fonksiyonlar işlecın sol tarafındaki nesne ile çağırıldığı için, `cout<<r` ifadesinin üye fonksiyon olması mümkün değildir. Bu fonksiyon iki parametre alır. Birinci parametre işlecın sol tarafındaki ostream nesnesi, ikinci parametre işlecın sağ tarafındaki rasyonel sayı nesnesidir. Aynı zaman da bu fonksiyonun doğrudan sınıfın private üyelerine de erişmesi gerekir. Bu nedenle ilgili işleç aşırı yükleme fonksiyonu friend olarak tanımlanmıştır. Bu fonksiyonun dönüş tipi ostream nesnesidir. Bu şekilde birden fazla nesne aynı `cout` ifadesi içinde yer alabilir. Örneğin `cout<<r1<<" "<<r2<<endl;` ifadesi ile birden fazla nesne ekrana yazdırılabilir.

```

//Program12.5.cpp
#include <iostream>
using namespace std;
class Rasyonel
{
    friend ostream& operator<<(ostream& o, Rasyonel r);
public:
    Rasyonel(){
        pay=1;
        payda=2;
    }
    Rasyonel(int i, int j){
        pay=i;
        payda=j;
    }
    void setRasyonel(int a, int b){
        pay=a;
        payda=b;
    }
    Rasyonel operator+(Rasyonel r)
    {
        Rasyonel temp;

        temp.pay=payda*r.pay+r.payda*pay;
        temp.payda=payda*r.payda;

        return temp;
    }
private:
    int pay;
    int payda;
};

ostream& operator<<(ostream& o, Rasyonel r)
{
    o<<r.pay<<"/"<<r.payda;

    return o;
}

int main()
{
    Rasyonel r1;
    Rasyonel r2(4,7);

```

```

Rasyonel r3;
r3=r1+r2;
cout<<r1<<endl;
cout<<r2<<endl;
cout<<r3<<endl;
return 0;
}
Program12.5.cpp Çıktısı
1/2
4/7
15/14

```



ARAŞTIRALIM

Üye fonksiyon veya friend fonksiyon kullanmadan işleç aşırı yükleme gerçekleştirme konusunu araştıralım.

3. BİLEŞİM (COMPOSITION)

Bileşim (composition) bir sınıfın, kullanıcı tanımlı bir başka sınıfın nesnesini veri üyesi olarak içermesi durumudur. Program12.6.cpp’da iki farklı sınıf tanımlanmıştır. Tarih sınıfı gun, ay ve yıl veri üyelerine sahip, yapıcı ve yazdırTarih üye fonksiyonları olan bir sınıftır. Personel sınıfı ad, soyad ve doğum-Tarihi veri üyelerine sahip bir sınıf olarak tanımlanmıştır. ad ve soyad üyeleri string sınıfını tipinden tanımlanmışken, dogumTarihi veri üyesi daha önce oluşturulmuş Tarih sınıfının bir nesnesidir. Bu nesneye başlangıç değeri atamak için üye değer atama listesi kullanılır. Bu yapıcı fonksiyonun başlığından hemen sonra ‘:’ karakteri ile başlayan listedir. Üye değer atama listesi aynı zamanda yerleşik veri tiplerinden oluşan veri üyelerine de değer atamak için kullanılabilir. Veri üyesi ismi ve daha sonra parantez içinde ilgili üyenin değeri yazılarak bu işlem gerçekleştirilir.

```

//Program12.6.cpp
#include <iostream>
#include <string>
using namespace std;

class Tarih
{
public:
    Tarih(int a, int b, int c)
    {
        gun=a;
        ay=b;
        yıl=c;
    }
    void yazdirTarih()
    {
        cout<<gun<<"-"<<ay<<"-"<<yil;
    }
private:
    int gun;

```

```

        int ay;
        int yıl;
    };
    class Personel
    {
    public:
        Personel(string personelAdi, string personelSoyadi, int g, int a, int y):dogumTarihi(g,a,y)
        {
            ad=personelAdi;
            soyad=personelSoyadi;
        }

        void yazdirPersonel()
        {
            cout<<ad<<" "<<soyad<<endl;
            cout<<"Dogum Tarihi: ";
            dogumTarihi.yazdirTarih();
        }
    private:
        string ad;
        string soyad;
        Tarih dogumTarihi;
    };

    int main()
    {
        Personel al("Ahmet","Kartal",10,2,2000);
        al.yazdirPersonel();
        cout<<endl;
        return 0;
    }

```

Program12.6.cpp Çıktısı

Ahmet Kartal

Dogum Tarihi: 10-2-2000

4. KALITIM (INHERITANCE)

Kalıtım kullanarak var olan sınıflardan yeni sınıflar oluşturulabilir. Kalıtım kullanmanın amacı geliştirilmiş olan kodların yeniden kullanılabilmesidir. Var olan sınıfa temel sınıf, temel sınıftan miras yolu ile oluşturulmuş yeni sınıfa türetilmiş sınıf denir. Temel sınıfın tüm üye fonksiyonları ve veri üyeleri aynı zamanda türetilmiş sınıfın da üye fonksiyonu ve veri üyeleri olarak kabul edilir. Türetilmiş sınıfta, temel sınıfın bazı üye fonksiyonları yeniden yazılabilir. Temel sınıfta yer almayan yeni veri üyeleri veya yeni üye fonksiyonları türetilmiş sınıfa eklenebilir. public, private ve protected olmak üzere üç farklı kalıtım yöntemi vardır. private ve protected kalıtım çok kullanılan yöntemler değildir, bu ne-

denle bu bölümde temel olarak public kalıtım anlatılacaktır. Bu kalıtım şeklinde türetilmiş sınıfın nesneleri aynı zamanda temel sınıfın nesneleri olarak değerlendirilir. Ancak temel sınıfın nesneleri aynı zamanda türetilmiş sınıfın nesneleri olarak düşünülemez. Örneğin Araç sınıfından Otomobil sınıfını kalıtım ile oluşturduğumuzu düşünelim. Tüm otomobiller araçtır ancak tüm araçlar otomobil değildir. Bu kalıtım türünde temel sınıfın private olmayan üyelerine türetilmiş sınıfta doğrudan ulaşılabilir. Ancak private üyelere, private olmayan üye fonksiyonları kullanarak ulaşılabilir. Bu durum için ayrı bir çözüm de geliştirilmiştir. Türetilmiş sınıflarda temel sınıfın veri üyelerine ulaşabilmek ve bu üyelerin aynı zamanda sınıfların kapsama alanları dışında ulaşılabilmesini engellemek amacıyla protected erişim seviyesi geliştirilmiştir. protected seviye public ve private seviyeler arasında erişim düzeyi sağlar. Sınıfların dışında protected erişim seviyesindeki üyelere ulaşmak mümkün olmazken türetilmiş sınıflarda temel sınıfın protected verisine ulaşılabilir. Program12.7.cpp’de Personel temel sınıfından AkademikPersonel ve IdariPersonel sınıfları public kalıtım ile oluşturulmuştur. Bu işlem için türetilmiş sınıf tanımlanırken sınıfın isminden sonra ‘:’ karakteri ve public anahtar kelimesi ile temel sınıfın ismi yazılır. AkademikPersonel türetilmiş sınıfı Personel temel sınıfının ad, soyad ve maas veri üyelerine sahiptir. Ek olarak dersSaati veri üyesi AkademikPersonel sınıfında tanımlanmıştır. AkademikPersonel sınıfında hesaplaToplamMaas temel sınıf fonksiyonu yeniden yazılmış toplam maaş hesabına dersSaati veri üyesi kullanılarak ek ders ücreti ilave edilmiştir. yazdirPersonel fonksiyonunda herhangi bir değişiklik yapılmadığı için AkademikPersonel tipinde tanımlanmış nesneler, temel sınıf versiyonunu doğrudan kullanabilirler. IdariPersonel türetilmiş sınıfında ise temel sınıf üye verilerine ek olarak görev veri üyesi tanımlanmıştır. IdariPersonel sınıfında yazdirPersonel üye fonksiyonu tekrar yazılmış ancak hesaplaToplamMaas fonksiyonu üzerinde bir değişiklik yapılmamıştır. Türetilmiş AkademikPersonel ve IdariPersonel sınıflarında temel sınıfın veri üyelerine doğrudan erişebilmek amacıyla, bu üyelerin erişim düzeyi protected olarak belirlenmiştir. Bu şekilde bu veriler türetilmiş sınıflarda erişilebilirken, sınıfların kapsama alanları dışında erişilemezler.

```
//Program12.7.cpp
#include <iostream>
#include <string>
using namespace std;
class Personel
{
    public:
        Personel(string personelAdi, string personelSoyadi, double m)
        {
            ad=personelAdi;
            soyad=personelSoyadi;
            maas=m;
        }
        void yazdirPersonel()
        {
            cout<<ad<<" "<<soyad<<endl;
        }
        double hesaplaToplamMaas()
```

```

        {
            return maas;
        }
protected:
    string ad;
    string soyad;
    double maas;
};
class AkademikPersonel: public Personel
{
    public:
        AkademikPersonel(string pAdi, string pSoyadi, double m, int dSaati):Personel(pAdi,p-
        Soyadi,m)
        {
            dersSaati=dSaati;
        }
        double hesaplaToplamMaas()
        {
            return maas + dersSaati*40;
        }
private:
    int dersSaati;
};
class IdariPersonel: public Personel
{
    public:
        IdariPersonel(string pAdi, string pSoyadi, double m, string g):Personel(pAdi,pSoyadi,m)
        {
            gorev=g;
        }
        void yazdirPersonel()
        {
            cout<<ad<<" "<<soyad<<" "<<gorev<<endl;
        }
private:
    string gorev;
};
int main()
{
    AkademikPersonel al("Ahmet","Kartal",10000,75);
    IdariPersonel il("Ali","Kanarya",8000,"Ogrenci Isleri");

```



```

al.yazdirPersonel();
cout<<"Toplam Maas:"<<al.hesaplaToplamMaas()<<endl;

il.yazdirPersonel();
cout<<"Toplam Maas:"<<il.hesaplaToplamMaas()<<endl;

return 0;
}

```

Program12.7.cpp Çıktısı

Ahmet Kartal

Toplam Maas:13000

Ali Kanarya Ogrenci Isleri

Toplam Maas:8000



DİKKAT EDELİM

Türetilmiş sınıflarda temel sınıfın private üyelerine doğrudan erişim söz konusu değildir.

5. ÇOK BİÇİMLİLİK (POLYMORPHİSM)

Kalıtım hiyerarşisinde yer olan tüm sınıfların nesnelere temel sınıf nesnesi gibi davranarak çok biçimlilik oluşturulur. Hiyerarşide yer alan her bir nesne kendi sınıfının görevini gerçekleştirir. Örneğin, Animal temel sınıfını düşünelim. Bu temel sınıftan türetilmiş her bir sınıfın move fonksiyonu vardır. Hiyerarşide yer alan farklı Animal nesneleri bir Animal temel sınıfı işaretçi listesi kullanılarak erişilebilir. Her bir türetilmiş sınıf nesnesinin kendine özgü move fonksiyonunu kullanması beklenmektedir. Ancak normal şartlarda bir temel sınıf işaretçisi veya referansı ile türetilmiş sınıf nesnesini gösterdiğimizde, herhangi bir üye fonksiyon çağrıldığında temel sınıf fonksiyonu kullanılır. Ancak burada temel amacımız her bir türetilmiş sınıf nesnesini aynı listede tutarak (işaretçi veya referans kullanarak), nesnelerin kendilerine özgü fonksiyonu kullanabilmelerini sağlamaktır. Bu durum için sanal (virtual) fonksiyonlar geliştirilmiştir. Temel sınıf işaretçisi veya referansı kullanarak sanal fonksiyonlar üzerinden türetilmiş sınıf fonksiyonlarına ulaşarak çok biçimlilik gerçekleştirilir. Program çalıştığında, dinamik olarak nesnenin doğru fonksiyonuna erişim sağlanacaktır. Temel sınıf işaretçisi (veya referansı) ile türetilmiş sınıfın nesnesini gösterdiğimizde, bu işaretçi ile sanal fonksiyon çağrıldığında temel sınıfın değil türetilmiş sınıfın fonksiyonu kullanılır.

Temel sınıftaki sanal fonksiyonlardan herhangi biri saf sanal (pure virtual) olarak tanımlanmışsa bu sınıfa soyut temel sınıf denir. Bir fonksiyonun saf sanal olarak tanımlanması, fonksiyon başlığına atama işleci ile 0 atanması sayesinde gerçekleştirilir. Soyut temel sınıflardan herhangi bir nesne tanımlanması söz konusu değildir. Bu sınıflar diğer sınıfların türetilmesi için bir temel olarak kullanılırlar. Program12.8. cpp'de yer alan Sekil sınıfına ait hesaplaAlan fonksiyonu saf sanal olarak tanımlanmış ve bu şekilde Sekil sınıfı soyut temel sınıf olmuştur. Bu programda Daire ve Dikdortgen sınıfları Sekil sınıfından public kalıtım ile türetilmiştir. Sekil sınıfında saf sanal olarak belirlenmiş hesaplaAlan fonk-

siyonu türetilmiş sınıflarda tanımlanmıştır. Daire sınıfını yaricap, Dikdortgen sınıfının kısaKenar ve uzunKenar isminde veri üyeleri vardır. Her iki sınıfta da veri üyelerine değer ataması, üye değer atama listesi ile gerçekleştirilmiştir. main fonksiyonda Daire sınıfından bir nesne, Dikdortgen sınıfından dört nesne oluşturulmuştur. Bu nesnelerin adresleri temel sınıf işaretçisi dizisine atanmıştır. Dolayısı ile farklı türetilmiş sınıf nesneleri aynı liste içinde tutulmuştur. Aynı liste üzerinde bir for döngüsü ile temel sınıf işaretçileri kullanarak hesaplaAlan fonksiyonu çağırılmıştır. Bu fonksiyon sanal olarak tanımlandığından, dinamik olarak doğru sınıfın hesaplaAlan fonksiyonu çağırılmış ve kullanılmıştır.

```
//Program12.8.cpp
#include <iostream>
#include <string>
using namespace std;

class Sekil
{
public:
    virtual double hesaplaAlan() =0;
};
class Daire: public Sekil
{
public:
    Daire(double r):yaricap(r){}
    double hesaplaAlan(){return 3.14*yaricap*yaricap;}
private:
    double yaricap;
};
class Dikdortgen: public Sekil
{
public:
    Dikdortgen(double a=1.0, double
b=2.0):kisaKenar(a),uzunKenar(b){}
    double hesaplaAlan(){return kısa-
Kenar*uzunKenar;}
private:
    double kısaKenar;
    double uzunKenar;
};
int main()
{
    Daire c1(20);
    Dikdortgen r1(2,7);
    Dikdortgen r2, r3(4);
    Dikdortgen r4(r1);
```

```

        Sekil *arr[]={&c1,&r1,&r2,&r3,&r4};
        for(int i=0; i<5; i++)
            cout<<arr[i]->hesaplaAlan()<<endl;
        return 0;
    }

```

Program12.8.cpp Çıktısı

1256

14

2

8

14



DİKKAT EDELİM

Soyut temel sınıftan bir nesne tanımlamak mümkün değildir. Eğer soyut sınıftan nesne tanımlanırsa, derleyici hatası oluşur.

BÖLÜM ÖZETİ

Sınıf kullanarak kullanıcı tanımlı yeni veri tipleri oluşturulabilir. Bir sınıfı oluşturmak için üye fonksiyonlar ve veri üyeleri tanımlanır. Yapıcı ve yıkıcı fonksiyonlar sınıf ismi kullanılarak tanımlanan özel fonksiyonlardır. Yapıcı fonksiyon bir sınıf nesnesi oluşturulduğunda, otomatik olarak çağrılır. Bu fonksiyon içinde veri üyelerine ilk değerleri atanabilir. Bir nesnenin tanımlandığı alan çalışmasını tamamladığında nesne silinir ve otomatik olarak yıkıcı fonksiyon çağrılır. Bir sınıf içerisinde kullanılacak public, protected ve private olmak üzere üç farklı erişim seviyesi vardır. private üyeler sadece sınıfın kapsam alanında erişilip kullanılabilirler. public üyeler sınıfın kapsam alanının yanında main fonksiyon ve diğer fonksiyonlar içerisinde ulaşılabilirler. protected olarak tanımlanmış üyeler kalıtım hiyerarşisinde türetilmiş sınıflar içerisinde erişilip kullanılabilirler. Bir sınıfın nesnesinin bir başka sınıf içerisinde veri üyesi olarak kullanılması bileşim olarak isimlendirilir. İşleç aşırı yükleme ile yerleşik veri tipleri üzerinde kullanılan işleçlerin kullanıcı tanımlı veri tipleri üzerinde kullanılabilmesi sağlanabilir. Temel sınıflardan yeni türetilmiş sınıflar oluşturulmasına kalıtım denir. Türetilmiş sınıf temel sınıfın üye fonksiyonlarına ve veri üyelerine sahip olur. Gerekğinde ek veri üyeleri tanımlayabilir veya var olan temel sınıf fonksiyonlarında değişiklik gerçekleştirebilir. Temel sınıf işaretçileri veya referansları ile türetilmiş sınıf nesnelerini göstererek ve sanal fonksiyonlar üzerinden türetilmiş sınıf fonksiyonlarına dinamik olarak erişerek çok biçimlilik gerçekleştirilir.

GÖZDEN GEÇİRELİM

1. Aşağıdakilerden hangisi Rasyonel sınıfından r1 isimli nesneyi doğru olarak tanımlar?

- a. class Rasyonel r1;
- b. Rasyonel r1;
- c. public Rasyonel r1;
- d. R r1;
- e. Rasyonel class r1;

2.

```
class Rasyonel {
    public:
        void yazdirRasyonel();
};
```

Rasyonel sınıfındaki yazdirRasyonel() fonksiyonunun sınıf dışında geliştirilmesi hangi şıkta doğru verilmiştir?

- a. void Rasyonel->yazdirRasyonel(){};
- b. void Rasyonel.yazdirRasyonel(){};
- c. Rasyonel::yazdirRasyonel(){};
- d. void yazdirRasyonel(){};
- e. void Rasyonel::yazdirRasyonel(){};

3. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
class Dikdortgen
{
    public:
        int x;
        int y;
};
int main()
{
    D ikdortgen d1;
    D ikdortgen d2;
    d1.x=5;
    d1.y=10;
    d2.x=1;
    d2.y=2;
    d2=d1;
    d1.x=3;
    d1.y=4;
    cout <<d1.x<<" "<<d2.y<<endl;
    return 0;
}
```

- a. 3 10
- b. 5 10
- c. 1 4
- d. 1 2
- e. 3 4

4. Aşağıdakilerden hangisinde Personel sınıfı doğru olarak tanımlanmıştır?

- a. class Personel{string ad;};
- b. public class Personel{string ad;};
- c. private class Personel{string ad;};
- d. protected class Personel{string ad;};
- e. class Personel(string ad){ };

5. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
class Test {
    public:
        int x;
        Test(){x=10;}
};
int main()
{
    Test t1,t2,t3;
    t1.x=7;t3.x=8;t1.x=5;
    Test dizi[] = {t1,t2,t3};
    cout<<dizi[1].x;
    return 0;
}
```

- 5
- 7
- 8
- 10
- Derleme hatası üretir.

- 6.

```
int main(){
    Personel p("Ali","Ahmet");
}
```

Yukarıda verilen kodun çalışabilmesi için gerekli yapıcı fonksiyon aşağıdakilerden hangisidir?

- Personel (string a, string s){
ad = a;
soyad = s;}
- constructor Personel (string a, string s){
ad = a;
soyad = s;}
- Personel.Personel (string a, string s){
ad = a;
soyad = s;}
- Personel (string a, string s) public {
ad = a;
soyad = s;}
- public Personel (string a, string s){
ad = a;
soyad = s;}

7. ArastirmaGorevlisi sınıfını Personel ve Ogrenci sınıflarından miras alarak oluşturacak C++ kodu aşağıdakilerden hangisidir?

- class ArastirmaGorevlisi ->public Personel, public Ogrenci{ };
- class Personel, class Ogrenci : class ArastirmaGorevlisi { };
- class Personel, class Ogrenci: class ArastirmaGorevlisi (){ };
- class ArastirmaGorevlisi: public Personel, public Ogrenci{ };
- class ArastirmaGorevlisi: : public Personel, public Ogrenci(){ };

8. Aşağıda verilen ifadelerden hangisi ++ işlecini aşırı yüklemek için kullanılan fonksiyonun ismidir?

- operator++
- ++
- ++operator
- overload++
- ++overload

9. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
class Dikdortgen{
public:
    int x;
    int y;
    virtual void alan()
{cout<<x*y<<endl;}
};
class Kare:public Dikdortgen{
public:
    void alan(){cout<<x*x<<endl;}
};
int main()
{
    Kare k;
    Dikdortgen *d=&k;
    d->x =2; d->y =3;
    d->alan();
    return 0;
}
```

- Derleyici hata üretir
- 5
- 6
- 9
- 4

10. Aşağıdakilerden hangisi hesapla fonksiyonunu saf sanal (pure virtual) olarak tanımlar?

- pure virtual void hesapla();
- virtual void hesapla()=0;
- void virtual hesapla();
- void hesapla()=0;
- virtual void hesapla();

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

Kaynakça

- Deitel, P., Deitel, H. (2017). C++ How to Program. 10. Baskı. Pearson. Essex, İngiltere.
- Dikici, M. (2017). C++ Programlama Dili: Yapısal Programlama, Nesnelerle Programlama ve Jenerik Programlama. Seçkin Yayıncılık. İstanbul, Türkiye.
- Çobanoğlu, B. (2021). C/C++ Programlama Eğitim Kitabı. Kodlab. İstanbul, Türkiye.
- Demir, S., Eryiğit, R., Ar, Y., Tuğrul, B., Gök, O., Kul, S., . . . Yağanoğlu, M. (2019). Programlama Temelleri. Erzurum: Atatürk Üniversitesi Açıköğretim Fakültesi.
- Kaleli, C., Bilge, A., Ergin, S., & Şora Günel, E. (2016). Bilgisayar ve Programlamaya Giriş. Eskişehir: Anadolu Üniversitesi.

BÖLÜM 13

STRİNG SINIFI VE KARAKTER DİZİSİ
İŞLEMLERİ

ANAHTAR KAVRAMLAR

String Kütüphanesi, String Kapasite
Metotları, String Erişim Metotları, String
Düzenleyici Metotları

ÖĞRENME HEDEFLERİ

- 1 Bir string değişkeni tanımlayabilir.
- 2 Bir string değişkenine değer ataması yapabilir.
- 3 String sınıfına ait kapasite metotlarını bilir.
- 4 String sınıfına ait erişim metotlarını bilir.
- 5 String sınıfına ait düzenleyici metotlarını bilir.
- 6 Struct değişkenini fonksiyonda parametre olarak kullanır.

GİRİŞ

String veri türü “Merhaba” gibi iki veya daha fazla karakterden oluşan metin

içeriğine sahip nesneleri saklamak için kullanılır. C++ programlama dilinde tanımlı diğer temel veri türleri gibi kullanmadan önce tanımlanmalı ve değer ataması yapılmalıdır. String veri türüne ait bir değişken tanımlayabilmek ve metotlarına erişebilmek için `<string>` kütüphanesinin programa eklenmesi gerekmektedir. String değişkeninin değeri çift tırnak içine alınması gerekmektedir.

Metin içerikli değişkenler aynı zamanda karakter dizisi (C-tarzı string) olarak da oluşturulabilir. Ancak zamanla bu kullanımdan kaynaklanan bellek yönetimi sorunları nedeniyle daha güvenilir ve verimli string sınıfının kullanılması tavsiye edilmektedir.

String kütüphanesinde tanımlı çok çeşitli metotlar vardır. Bu metotlar string nesneleri üzerinde işlemler yapabilmeyi sağlar. Bu metotlar genel olarak üç ana başlık altında değerlendirilebilir. Kapasite metotları string nesnesinin uzunluğu, hafızada ne kadar alan kullandığı gibi bilgilere ulaşmayı sağlar. Erişim metotları string değişkenin belirli bir karakterine veya bir grup karakterine erişme amacıyla tanımlanmıştır. Düzenleyici metotları, iki farklı string nesnesini birleştirmeye veya üzerinde ekleme veya çıkarma gibi karakter işlemleri gerçekleştirmeye izin verir. Bu üç gruba girmeyen daha genel kullanıma sahip metotlar diğer metotlar başlığı altında incelenmiştir.

Metotların kullanımı aldıkları parametre sayısı ve veri türüne göre farklılıklar göstermektedir. Metotların kullanımı en genel ve sıklıkla kullanılan şekliyle anlatılmıştır. Diğer kullanım yöntemlerine bölüm sonunda verilen kaynaklardan ulaşmak mümkündür.



1. STRING DEĞİŞKENİ TANIMLAMA VE DEĞER ATAMA

Bir string nesnesi oluşturmak için kullanılabilecek farklı tür ve sayıda girdiye sahip birkaç yapıcı yöntemi vardır. En genel haliyle, bir string nesnesi üç farklı yöntem ile oluşturulabilir. İlk yöntem string sınıfının tek parametre alan yapıcı metodu kullanılarak gerçekleştirilir.

```
string str("Merhaba");
```

İkinci yöntem benzer şekilde yine string sınıfının iki parametre alan yapıcı metodu kullanarak gerçekleştirilir. Aşağıda verilen örnek kod bloğu üç adet 'w' karakterinden oluşan "www" değerine sahip string nesnesini oluşturur.

```
string str(3, 'w');
```

Üçüncü yöntem atama işleci kullanılarak gerçekleştirilir.

```
string str="Merhaba";
```

Yukarıda anlatılan string nesnesi oluşturma ve değer atama yöntemleri ve diğer yöntemler aşağıda verilen örnek kod bloğunda gösterilmiştir.

Örnek:

```
string str1 ("Merhaba Ankara");
string str2 (str1);
string str3 (str1, 8, 6);
string str4 ("Merhaba Ankara", 8);
string str5 (3, 'w');
string str6 (3, 119); //'w' karakterinin ASCII karşılığı olan değeri 119
string str7 (str1.begin(), str1.begin()+7);
cout<<"str1= "<<str1<<endl;
cout<<"str2= "<<str2<<endl;
cout<<"str3= "<<str3<<endl;
cout<<"str4= "<<str4<<endl;
cout<<"str5= "<<str5<<endl;
cout<<"str6= "<<str6<<endl;
cout<<"str7= "<<str7<<endl;
```

Çıktı:

```
str1= Merhaba Ankara
str2= Merhaba Ankara
str3= Ankara
str4= Merhaba
str5= www
str6= www
str7= Merhaba
```



DİKKAT EDELİM

Bir string değişkeninin değeri "..." (çift tırnak) içerisine alınmalıdır.



ARAŞTIRALIM

String kütüphanesinde tanımlı metotlara ve farklı kullanım yöntemlerine <https://www.cplusplus.com/reference/string/string/> adresinden ulaşılabilir.

2. KAPASİTE METOTLARI

Üzerinde çalışılan string nesnesinin uzunluğu, hafızada ne kadar alana gereksinim duyduğu gibi bilgilere ulaşmak için kapasite metotları altında size, length, max_size, ve capacity metotları tanımlanmıştır. Ayrıca string nesnesinin boş olup olmadığını öğrenmek için empty metodu ve gerektiğinde değişkenin sakladığı bütün karakterlerini silmek için tanımlanmış clear metotları mevcuttur.

size ve length Metotları

Hem size hem de length metotları bir string değişkenindeki boşluklar ve noktalama işaretleri dahil olmak üzere toplam karakter sayısını döndürür.

Örnek:

```
string str = "Bu metinde 33 karakter mevcuttur. ";
cout<<str.length()<<endl;
cout<<str.size()<<endl;
```

Çıktı:

33

33

max_size Metodu

Bir string değişkenin saklayabileceği en büyük uzunluk değerini döndürür. Bu sistem tarafından sağlanabilecek uzunluğa karşılık gelmektedir. Bu değer aşılmaması durumunda length_error hatası alınır. Metot çalıştırılan bilgisayarın özelliklerine ve kullanılan işletim sistemine bağlı olarak farklı değer üretebilmektedir.

Örnek:

```
string str = "Merhaba Ankara!";
cout<<str.size()<<endl;
cout<<str.max_size()<<endl;
```

Çıktı:

15

4611686018427387903

capacity Metodu

Capacity metodu string değişkeni için ne kadarlık bir alanın hafızada ayrıldığı bilgisini döndürür. Dönen değer string uzunluğu ile aynı olması gerekmemektedir. Değişkene sonradan eklenebilecek karakter işlemlerini optimize etmek için değişken uzunluğuna eşit veya daha uzun olabilir.

Örnek:

```
string str = "Merhaba Ankara!";
cout<<str.capacity()<<endl;
cout<<str.length()<<endl;
str += "!";
cout<<str.capacity()<<endl;
cout<<str.length()<<endl;
```

Çıktı:

15

15

30

16

Yukarıdaki örnekte oluşturulan string nesnesinin hem capacity hem de uzunluğu 15 iken değişkene bir karakter eklenmesi durumunda capacity 30'a uzunluğu ise 16'a çıkmıştır.

empty Metodu

Bir string değişkenin içeriğinin boş olup olmadığını kontrol eder. Eğer çağrıldığı değişkenin uzunluğu sıfır ise 1 değeri döndürür değilse 0 değeri döndürür.

Örnek:

```
string str1 = "";
string str2 = "Merhaba";
if(str1.empty())
cout<<"Boş string";
if(str2.empty())
cout<<"Dolu string";
```

Çıktı:

Boş string

clear Metodu

Bir string değişkenin içeriğini boşaltır. Böylece uzunluğu sıfır olan bir string halini alır.

Örnek:

```
string str = "Merhaba";
str.clear();
if(str.empty())
cout<<"Boş string";
```

Çıktı:

Boş string

Erişim Metotları

Bazı durumlarda string değişkenini oluşturan karakterlere erişmek ve üzerinde birtakım değişiklikler yapmak gerekmektedir. String nesnesinin son karakterine erişmek için back metodu tanımlanmıştır. Benzer şekilde ilk karakterine front metodu ile erişilir. Herhangi bir karaktere erişmek için at ve [indeks] yapıları tercih edilebilir.

back Metodu

Bir string değişkeninin en son karakterine erişmek için kullanılır.

Örnek:

```
string str = "0123456789";
cout<<str.back()<<endl;
```

Çıktı:

9

front Metodu

Bir string değişkeninin ilk karakterine erişmek için kullanılır.

Örnek:

```
string str = "0123456789";
cout<<str.front()<<endl;
```

Çıktı:

0

at Metodu

Köşeli parantez veya “at” metodu kullanarak iki farklı yöntem ile bir string değişkenin bütün karakterlerine erişilebilir. Değişkenin ilk karakterinin indeks değeri sıfırdır. String değişkenin sınır değerleri dışındaki konumlara erişmemeye dikkat edilmesi gereklidir.

Örnek:

```
string str = "0123456789";
cout<<str.at(0)<<endl;
cout<<str.at(5)<<endl;
cout<<str.at(9)<<endl;
```

Çıktı:

0

5

9

Benzer sonucu aşağıda verilen kod bloğu da gerçekleştirmektedir.

Örnek:

```
string str = "0123456789";
cout<<str[0]<<endl;
cout<<str[5]<<endl;
cout<<str[9]<<endl;
```

Çıktı:

0

5

9

**DİKKAT EDELİM**

Bir string değişkeninin ilk karakterinin indeksi sıfırdır. En son karakterine “size()-1” değeri ile erişilebilir.

3. DÜZENLEYİCİ METOTLAR

Oluşturulan string nesnesine bazı durumlarda eklemeler ve çıkarmalar yapmak mümkündür. Hem append metodu hem de aşırı yüklenmiş olan “+” işleci kullanarak var olan string değişkenine ekleme yapmak mümkündür. Eğer string nesnesinin sonuna tek bir karakter eklenmek isteniyorsa push_back metodu kullanılabilir. En sondan tek bir karakter silme işlemini ise pop_back metodu gerçekleştirir. İki farklı string değişkeninin değerleri değiştirilmek isteniyorsa swap metodu kullanılmalıdır.

append Metodu

Bir string nesnesinin sonuna ilave karakterler ekleyerek string nesnesinin genişletilmesini sağlar. Aynı sonuç ‘+’ işleci kullanılarak da elde edilebilir.

Örnek:

```
string str1 = "Merhaba ";
string str2 = "Ankara";
string str3 = "!";
str1.append(str2);
```



```
string str4 = str1 + str3;
cout<<str1<<endl;
cout<<str4<<endl;
```

Çıktı:

Merhaba Ankara

Merhaba Ankara!

push_back Metodu

Bir string nesnesinin sonuna ilave bir karakter ekler. Böylece string nesnenin uzunluğu bir artar.

Örnek:

```
string str1 = "Merhaba Ankara";
cout<<str1<<" "<<str1.length()<<" karakterden oluşmaktadır."<<endl;
str1.push_back('!');
cout<<str1<<" "<<str1.length()<<" karakterden oluşmaktadır."<<endl;
```

Çıktı:

Merhaba Ankara 14 karakterden oluşmaktadır.

Merhaba Ankara! 15 karakterden oluşmaktadır.

pop_back Metodu

Bir string nesnesinin sonundan bir karakter siler. Böylece string nesnenin uzunluğu bir azalır. pop_back metodu parametre almaz.

Örnek:

```
string str1 = "Merhaba Ankara!";
cout<<str1<<" "<<str1.length()<<" karakterden oluşmaktadır."<<endl;
str1.pop_back();
cout<<str1<<" "<<str1.length()<<" karakterden oluşmaktadır."<<endl;
```

Çıktı:

Merhaba Ankara! 15 karakterden oluşmaktadır.

Merhaba Ankara 14 karakterden oluşmaktadır.

**DİKKAT EDELİM**

pop_back gibi bazı metotları kullanabilmek için string değişkenine değer ataması yapılması gerekmektedir. Eğer string değişkeni boş ise hata mesajı ile karşılaşılır.

insert ve erase Metotları

Bir string nesnesinin istenilen yerine ilave karakterlerin yerleştirilmesi insert metodu ile gerçekleştirilir. Eğer string nesnesinden bazı karakterlerin çıkarılması isteniyorsa erase metodu tercih edilir.

Örnek:

```
string str1 = "Merhaba Ankara";
cout<<str1<<endl;
str1.erase(8,6);
```



```
cout<<str1<<endl;
str1.insert(8, "Antalya");
cout<<str1<<endl;
```

Çıktı:

```
Merhaba Ankara
Merhaba
Merhaba Antalya
replace Metodu
```

Bir string nesnesinin istenilen kısmın çıkarılmasını ve yerine yeni karakterlerin eklenmesini sağlar.

Örnek:

```
string str1 = "Merhaba Ankara";
cout<<str1<<endl;
str1.replace(8, 6, "Mersin");
cout<<str1<<endl;
```

Çıktı:

```
Merhaba Ankara
Merhaba Mersin
swap Metodu
```

İki farklı string nesnesinin birbirleriyle değerlerinin değiştirilmesini sağlar.

Örnek:

```
string str1 = "Ankara";
string str2 = "Antalya";
cout<<"str1= "<<str1<<endl;
cout<<"str2= "<<str2<<endl;
str1.swap(str2);
cout<<"swap metodundan sonra"<<endl;
cout<<"str1= "<<str1<<endl;
cout<<"str2= "<<str2<<endl;
```

Çıktı:

```
str1= Ankara
str2= Antalya
swap metodundan sonra
str1= Antalya
str2= Ankara
Diğer String Metotları
```

Bu bölümde yukarıda anlatılan ana gruplara girmeyen diğer string metotlarına yer verilmiştir. Hem find hem de rfind metotları bir string değişkeni içerisinde başka bir string nesnesinin olup olmadığını arar. Find metodu arama işlemine string değişkenin başından başlarken rfind metodu sondan başlar. Tek bir karakter bir string nesnesi içerisinde aranacak ise find_first_of veya find_last_of metotları kullanılmalıdır. Bazı durumlarda bir string nesnesinin bir bölümü elde edilmek istenir. Bazen daha büyük bir string nesnesini bölerek daha küçük alt nesneler oluşturulması gerekebilir. Bu gibi durumlarda substr metodu tercih edilir. String nesnelerin alfabetik olarak sıralanması için karşılaştırılması en sık

ihtiyaç duyulan yöntemlerden biridir. Compare metodu iki string değişkenini karşılaştırarak üç farklı sonuç değeri döndürür. Eğer dönen değer sıfır ise bu değişkenlerin aynı karakterlerden oluştuğu anlaşılır.

find Metodu

Bir string değişkeni içerisinde parametre olarak gönderilen karakter dizisi var ise bulunduğu ilk konum indeksini döndürür. Bulunmaması durumunda string değişkenin uzunluğundan daha büyük bir değer döndürür. Parametre olarak ya bir karakter ya da başka bir string nesnesi alabilir.

Örnek:

```
string str1 = "Merhaba Ankara!";
string str2 = "kara";
cout<<str1.find(str2)<<endl;
cout<<str1.find("Anka")<<endl;
cout<<str1.find("Kara")<<endl;
```

Çıktı:

10

8

18446744073709551615

rfind Metodu

Bir string değişkeni içerisinde parametre olarak gönderilen karakter dizisi var ise bulunduğu en son konum indeksini döndürür. Bulunmaması durumunda string değişkenin uzunluğundan daha büyük bir değer döndürür.

Örnek:

```
string str1 = "Merhaba Ankara Ankara!";
string str2 = "kara";
cout<<str1.rfind(str2)<<endl;
cout<<str1.rfind("Anka")<<endl;
cout<<str1.rfind("Kara")<<endl;
```

Çıktı:

17

15

18446744073709551615

find_first_of Metodu

Bir string değişkeni içerisinde parametre olarak aldığı karakterin bulunduğu ilk indeks değerini döndürür.

Örnek:

```
string str1 = "Merhaba Ankara";
cout<<str1.find_first_of('a')<<endl;
```

Çıktı:

4

find_last_of Metodu

Bir string değişkeni içerisinde parametre olarak aldığı karakterin bulunduğu en son indeks değerini döndürür.

Örnek:

```
string str1 = "Merhaba Ankara";
cout<<str1.find_last_of('a')<<endl;
```

Çıktı:

13

substr Metodu

substr metodu çağrıldığı string değişkeninden parametre olarak aldığı başlangıç ve uzunluk değerini kullanarak bir alt parçasını oluşturur. Metodun aldığı ilk parametre başlangıç indeksini ve ikinci parametre uzunluğu ifade eder. Eğer uzunluk parametresi verilmez ise başlangıç indeksinden değişkenin sonuna kadar olan parça alınır.

Örnek:

```
string str1 = "Merhaba Ankara";
string str2 = str1.substr(8, 6);
string str3 = str1.substr(8);
cout<<str2;
cout<<str3;
```

Çıktı:

Ankara

Ankara

**DİKKAT EDELİM**

Başlangıç parametresi için negatif bir değer göndermek çalışma zamanı hatasına sebep olacaktır. Eğer başlangıç parametresi için string nesnesinin uzunluğundan daha büyük bir değer girilmesi durumunda metod boş string döndürecektir.

compare Metodu

String nesnesi ile parametre değerini karşılaştırır. Eğer her ikisi aynı değere sahip ise sıfır, nesne parametre değerinden sözlükte daha önce ise negatif sonra ise pozitif değer döndürür. Burada dikkat edilmesi gereken husus, karşılaştırma işlemi ASCII tablosundaki karakterlerin diziliş sırasına göre yapılmaktadır. 'A' karakterinin değeri 65 iken 'a' karakterinin değeri 97'dir. Bu durumda "Armut" kelimesi "armut" kelimesinden önce gelmektedir. Benzer şekilde "Elma" kelimesi "armut" kelimesinden öncedir. Karşılaştırma işlemi '<', '<=', '>', ve '>=' işleçleri kullanılarak da gerçekleştirilebilir.

Örnek:

```
string str1 = "abc";
string str2 = "abc";
string str3 = "bcd";
cout<<str1.compare(str2)<<endl;
cout<<str1.compare(str3)<<endl;
cout<<str1.compare("aaa")<<endl;
```

Çıktı:

0

-65793

258

**DİKKAT EDELİM**

Eğer karşılaştırılan iki string değişkeni birbirinden farklıysa, programın her çalıştırılmasında farklı negatif ve pozitif değerler görmek mümkündür.

BÖLÜM ÖZETİ

Bu bölümde, C++ programlama dilinin standart string kütüphanesi detaylı bir şekilde ele alınmıştır. **String** veri türü, temel veri türlerinden biri olmayıp, metin tabanlı verileri saklamak ve işlemek amacıyla tanımlanmıştır. C++ dilinde string oluşturmanın iki temel yöntemi bulunmaktadır.

Birinci yöntem, **karakter dizileri** kullanılarak oluşturulan ve “C-tarzı string” olarak adlandırılan yapılardır. Bu tür string yapısı, önceki bölümlerde incelenmiştir. İkinci yöntem ise, C++ standart kütüphanesinde tanımlı **string sınıfı** kullanılarak string oluşturulmasıdır. Bu yaklaşım, daha güvenilir ve verimli bellek yönetimi sağladığından, modern C++ uygulamalarında yaygın olarak tercih edilmektedir.

Bölümde öncelikle bir string nesnesinin nasıl tanımlanacağı ve bu nesnelere değer atanmasının farklı yolları ele alınmıştır. String kütüphanesindeki metotlar, işlevselliklerine göre üç ana kategori altında incelenmiştir: **kapasite metotları**, **erişim metotları** ve **düzenleyici metotlar**. Bu kategorilere girmeyen işlevler ise “diğer metotlar” başlığı altında açıklanmıştır.

String nesnelerinin tanımlanması ve farklı değer atama yöntemlerinin yanı sıra, bir string nesnesinin uzunluk gibi özelliklerine erişim sağlama yöntemleri ele alınmıştır. Ardından, string nesnelerinin nasıl birleştirileceği, karşılaştırılacağı ve değerlerinin nasıl değiştirileceği üzerinde durulmuştur. Son olarak, bir string içerisinde belirli bir örüntünün nasıl aranacağına dair yöntemler detaylandırılmıştır.

Bu bölüm, C++ string sınıfının sunduğu özellikleri ve bu özelliklerin nasıl kullanılacağını anlamak için kapsamlı bir kaynak niteliğindedir.

GÖZDEN GEÇİRELİM

1. Verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
string txt = "Merhaba Ankara!";
```

```
cout << txt.size();
```

- a. 12
- b. 13
- c. 14
- d. 15
- e. 16

4. Verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
string str ("Merhaba Ankara!");
```

```
str.pop_back();
```

```
cout << str;
```

- a. Merhaba
- b. Merhaba Ankara
- c. Ankara
- d. Ankara!
- e. !

2. Verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
string txt = "Merhaba Ankara!";
```

```
txt.erase(7, 7);
```

```
cout << txt;
```

- a. Merhaba
- b. Merhaba!
- c. Ankara
- d. Ankara!
- e.

5. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
string str ("Merhaba Ankara!");
```

```
str.replace(10, 5, "talya");
```

```
cout << str;
```

- a. Merhaba Antalya
- b. Merhaba Antalya!
- c. Merhaba talya!
- d. Merhaba Ankaratalya!
- e. Merhaba Ankaratalya

3. Verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
string str1 ="3";
```

```
string str2="3";
```

```
string str3 = str1 + str2;
```

```
cout << str3;
```

- a. 3
- b. 6
- c. 33
- d. "3""3"
- e. "33"

6. Verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
string str1 ("Merhaba Ankara!");
```

```
string str2 ("Antalya");
```

```
str1.swap(str2);
```

```
cout << str1;
```

- a. Merhaba Antalya!
- b. Merhaba Ankara!
- c. Merhaba Ankara Antalya!
- d. Antalya
- e. Antalya!

7. Verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
string txt = "Merhaba Ankara!";
```

```
cout << txt.length();
```

- a. 2
- b. 3
- c. 14
- d. 15
- e. 16

9. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
string txt = "Merhaba Ankara!";
```

```
cout << txt.find_last_of('a');
```

- a. 10
- b. 11
- c. 12
- d. 13
- e. 14

8. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
string txt = "Merhaba Ankara!";
```

```
cout << txt.find_first_of('a');
```

- a. 3
- b. 4
- c. 5
- d. 6
- e. 7

10. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
string str1 ("Merhaba Ankara!");
```

```
string str2 = str1.substr(8,7);
```

```
cout << str2;
```

- a. Merhaba
- b. Ankara
- c. Merhaba!
- d. kara!
- e. Ankara!

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

Kaynakça

- Deitel, P., Deitel, H. (2017). C++ How to Program. 10. Baskı. Pearson. Essex, İngiltere.
- Dikici, M. (2017). C++ Programlama Dili: Yapısal Programlama, Nesnelerle Programlama ve Jenerik Programlama. Seçkin Yayıncılık. İstanbul, Türkiye.
- Çobanoğlu, B. (2021). C/C++ Programlama Eğitim Kitabı. Kodlab. İstanbul, Türkiye.
- Demir, S., Eryiğit, R., Ar, Y., Tuğrul, B., Gök, O., Kul, S., . . . Yağanoğlu, M. (2019). Programlama Temelleri. Erzurum: Atatürk Üniversitesi Açıköğretim Fakültesi.
- Kaleli, C., Bilge, A., Ergin, S., & Şora Günel, E. (2016). Bilgisayar ve Programlamaya Giriş. Eskişehir: Anadolu Üniversitesi.

