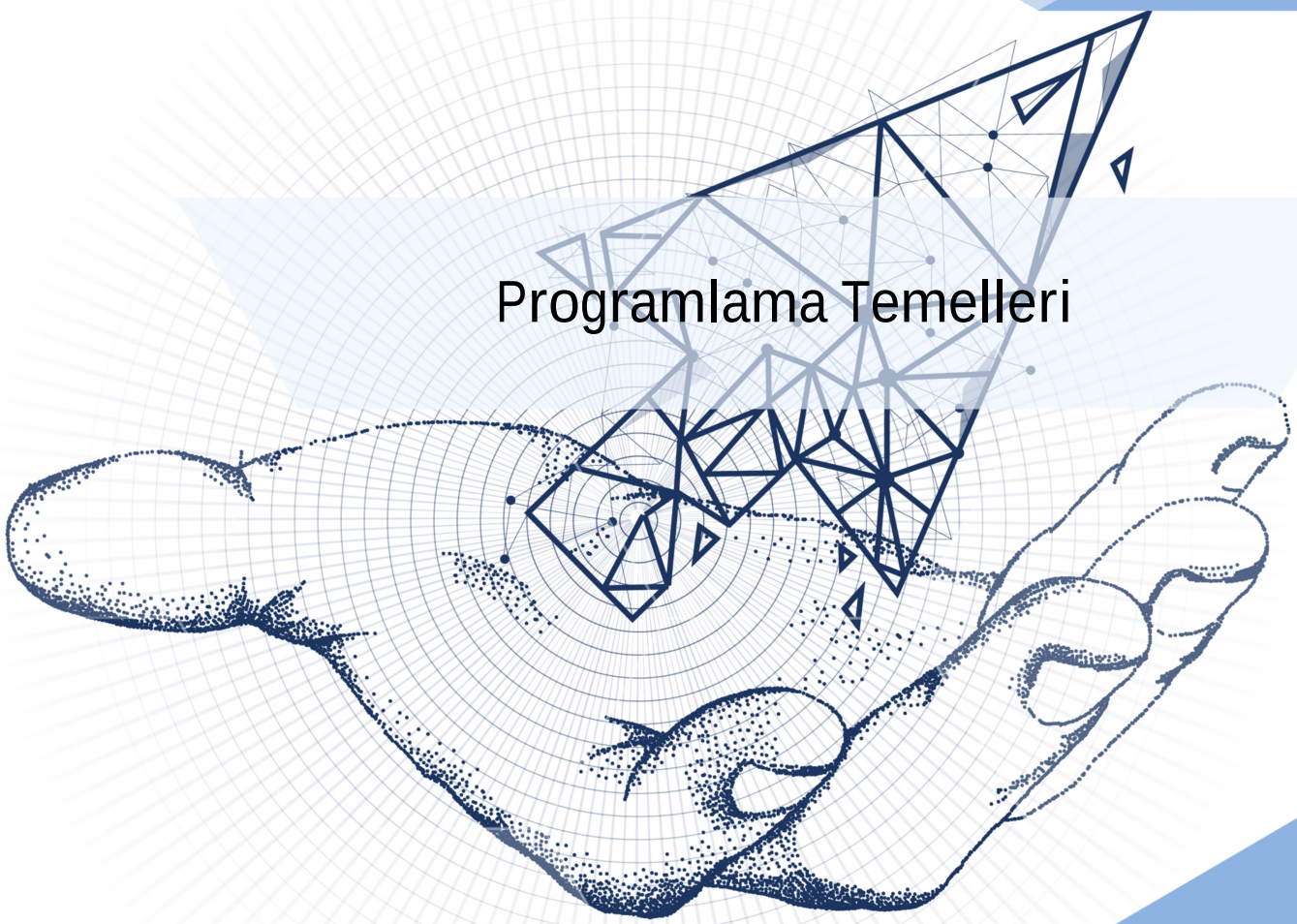
Atatürk Üniversitesi
Açıköğretim Fakültesi

Programlama Temelleri



ATATÜRK ÜNİVERSİTESİ
AÇIKÖĞRETİM FAKÜLTESİ

Programlama Temelleri

ISBN: 978-605-2278-77-2

ATATÜRK ÜNİVERSİTESİ AÇIKÖĞRETİM FAKÜLTESİ YAYINI

ERZURUM, 2019

İÇİNDEKİLER

Editör

Dr. Öğr. Üyesi DENİZ DAL

TEMEL KAVRAMLAR

İÇİNDEKİLER

- Bilgisayar Nedir?
- Donanım
- Sayı Sistemleri
- Yazılım
- Programlama Dilleri
- C++'ın Tarihçesi
- Bir C++ Programının Geliştirilmesi

HEDEFLER

- Bu üniteyi çalıştıktan sonra;
  - Bilgisayarın temel yapısını kavrayabilecek,
  - Donanım birimlerini özellikleriyle birlikte açıklayabilecek,
  - Bir sayıyı farklı bir sayı sistemine çevirebilecek,
  - C++ program geliştirme aşamalarını anlayabilecek,
  - Programlama dilleri seviyeleri arasındaki farklılıkları açıklayabileceksiniz.

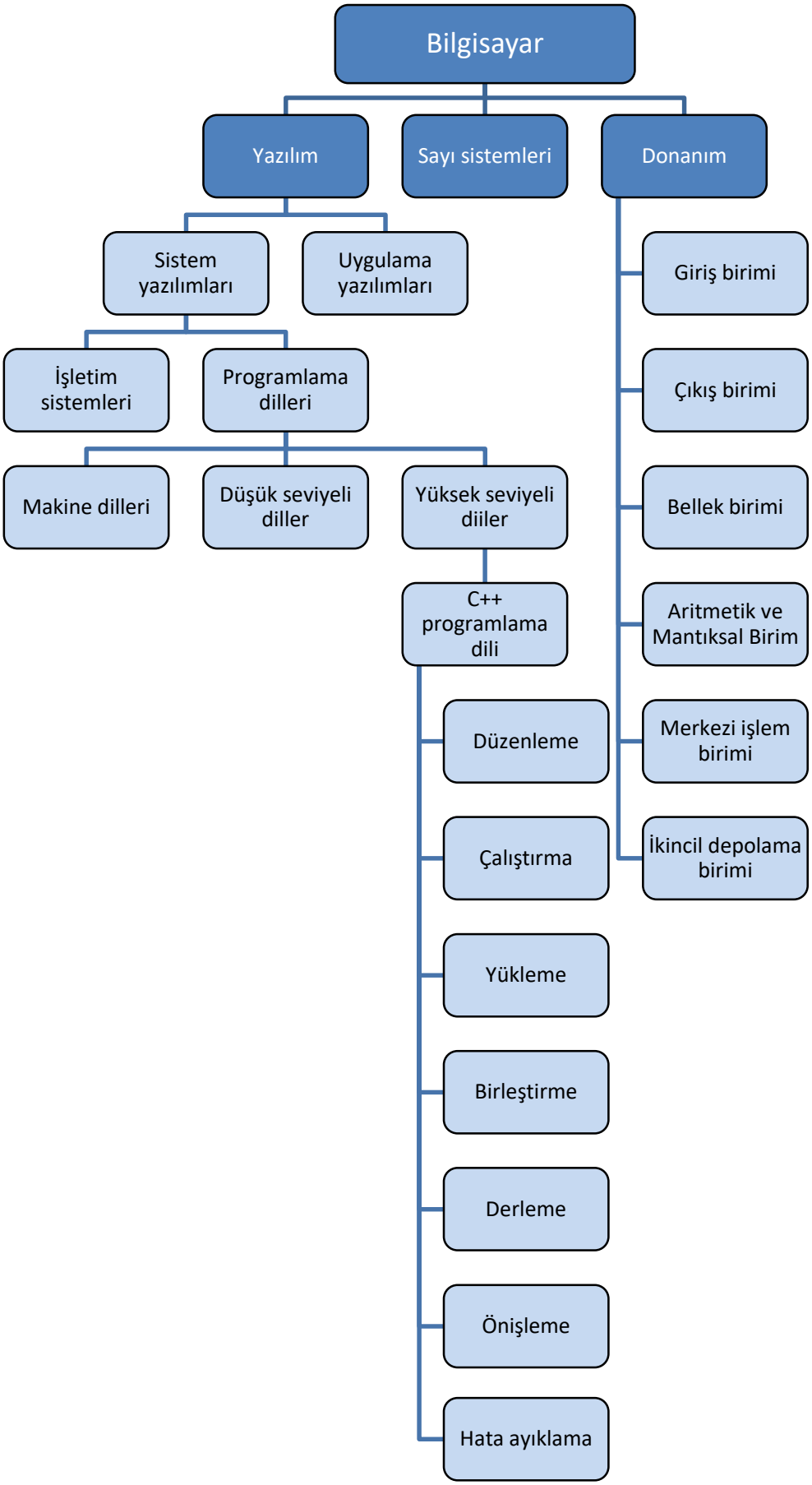
Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

Öğr. Gör. Salih DEMİR

ÜNİTE

1


Bilgisayar
Yazılım
Sayı sistemleri
Donanım
Sistem yazılımları
Uygulama yazılımları
İşletim sistemleri
Programlama dilleri
Makine dilleri
Düşük seviyeli diller
Yüksek seviyeli diiler
C++ programlama dili
Düzenleme
Çalıştırma
Yükleme
Birleştirme
Derleme
Önişleme
Hata ayıklama
Giriş birimi
Çıkış birimi
Bellek birimi
Aritmetik ve Mantıksal Birim
Merkezi işlem birimi
İkincil depolama birimi

GİRİŞ

Bilgi teknolojilerinin kullanımının yaygınlaşmasıyla birlikte üretilen veri miktarı da günden güne artmaktadır. Söz konusu verinin çok büyük olması birçok problemi de beraberinde getirmektedir. İlgili verilerin nerede saklanacağı, nasıl işleneceği ve sonuçlarının nasıl yorumlanacağı önemli problemlerdir. Ayrıca oluşan verilerden bilginin elde edilmesi ve saklanması kadar yorumlanması ve analiz edilmesi de oldukça önemlidir. Bilginin çok hızlı bir şekilde üretilmesi ve analiz edilmesi için bilgisayarların çok iyi programlanması gerekmektedir. İşletim sistemlerinin ve donanımların özelliklerine göre çeşitli programlama dilleri geliştirilmiştir. Üretilen veri türleri, bu veri üzerinde yapılacak işlemler ve bu verinin hangi elektronik cihaz üzerinde kullanılacağı, programcının programlama dili seçimi noktasında etkilidir.

Bilgisayar, *donanım* ve *yazılım* olmak üzere iki ana bileşenden oluşmaktadır ve yapısal olarak aldığı verileri işleyerek sonuçlar üreten ve saklayan bir cihaz şeklinde tanımlanmaktadır. Ayrıca makine dili adı verilen kodları kullanarak bu verileri almakta, işlemekte ve sonuçlar üretmektedir. Dolayısıyla verilerin, bilgisayarların anladıkları dille aktarılması gerekmektedir. Bahsi geçen nedenle de insanların bilgisayarları kullanabilmesi için işlemek istediği verileri makine diline çevirmesi gerekmektedir. Söz konusu çevirme işlemleri, bilgisayarların programlanması şeklinde gerçekleştirilmektedir. Bu çevirme işlemleri esnasında hata yapılmaması ve programların hafızada daha az yer tutması için çeşitli sayı sistemleri kullanılmaktadır.

Bu ünitede bilgisayar donanım birimlerinin özelliklerini, verilerin bilgisayara aktarılması için kullanılan sayı sistemlerini ve bu sayı sistemleri arasındaki dönüşümleri, yazılım türlerini, C++ programlama dilinin tarihsel gelişimini ve bir C++ programının geliştirim aşamalarını öğreneceksiniz.

BİLGİSAYAR NEDİR?

Bilgisayar; verileri belirli komutlar dizisine göre okuyup, bu verilerle aritmetiksel ve mantıksal işlemler yaparak onları kendi anlayabileceği bir dile çeviren ve sonuçları kullanıcıya sunan, ayrıca verileri belleğinde saklayabilen elektronik bir cihazdır [1].

Bilgisayarlar;

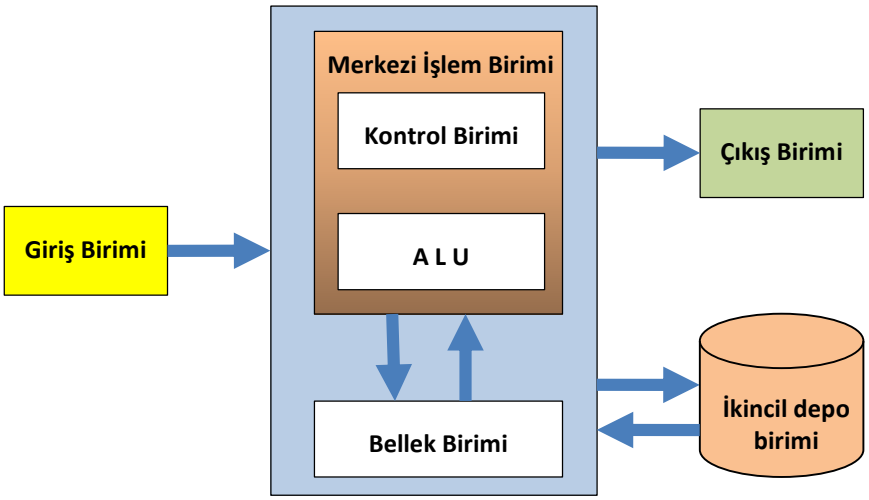
- Aldığı verileri kendisine verilen komutlara göre işleyerek bilgiyi oluştururlar.
- Aritmetiksel ve mantıksal işlemleri çok hızlı yapabilirler.
- Çok miktardaki veriyi çok hızlı işleyebilir ve üretilen bilgileri uzun süre saklayabilirler.

Bilgisayar donanım ve yazılım olmak üzere iki unsurdan meydana gelir.

Bilgisayar, donanım (hardware) ve yazılım (software) olmak üzere iki unsurdan meydana gelir. *Donanım*, bilgisayarı meydana getiren tüm fiziksel kısımlara (elektrik, elektronik ve mekanik birimler) verilen addır. *Yazılım* ise, bilgisayarların isteğe uygun işleri yapabilmeleri için onlara verilen tüm bilgiler ve komutlardır.

DONANIM

Günümüz bilgisayarının çalışma prensipleri, John von Neumann'ın 1945'de geliştirdiği mimariyi temel almaktadır. Bu mimaride basitçe bir işlemci, bir hafıza ve giriş/çıkış sistemleri yer almaktadır.



Şekil 1.1 John von Neumann mimarisi

Giriş birimi

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

Giriş birimi, bilgisayarda işlenecek verileri giriş aygıtlarından alarak merkezi işlem birimine aktarır. Bilgisayarlara bilgi çoğunlukla fare, klavye, dokunmatik ekran gibi cihazlardan aktarılmaktadır. Diğer veri girişi için kullanılan aygıtlar; mikrofon, kamera, tarayıcı, barkod okuyucu, ikincil bellekler (flaş bellekler, sabit diskler, DVD, Blue-ray gibi optik diskler) ve sensörlerdir.

Çıkış birimi

Çıkış birimi, bilgisayar tarafından işlenen verileri belirlenen çıkış aygıtına aktarır. İşlenen veriler ekranda görüntülenebilir, yazıcıdan kâğıda baskı yapılabilir, hoparlörden ses olarak çıkarılabilir, ağ üzerinden iletilebilir veya başka bir cihazı yönetmek için kontrol birimlerine aktarılabilir.

Bellek birimi

Bellek birimi, merkezi işlem biriminin işleyeceği verilerin tutulduğu hızlı erişim sağlayan geçici bir depolama birimidir. Çıkış birimine aktarılacak olan işlenmiş veriler de bu birimde tutulmaktadır.

Bilgisayarın elektrik enerjisi kesildiğinde belleğindeki veriler kaybolmaktadır. Bu yüzden bellek bilgisayarda kalıcı depolama alanı olarak kullanılamaz. Bellek birimi, RAM bellek veya birincil hafıza olarak da adlandırılır.

Aritmetik ve Mantıksal Birim (ALU-Arithmetic Logic Unit)

Merkezi işlem birimi (CPU-Central Processing Unit) içerisinde yer alan en önemli birimlerden birisidir. Toplama, çıkarma, çarpma, bölme gibi aritmetik işlemler ile AND-NAND, OR-NOR, XOR-XNOR gibi mantıksal ve eşitlik gibi karşılaştırma işlemleri bu birimde gerçekleştirilmektedir.

Merkezi işlem birimi (CPU)

Bu birim bilgisayar sistemindeki birimlerin çalışmalarını zamanlar ve kontrol eder. Giriş birimindeki verilerin bellek birimine yüklenmesi, yüklenen verilerin ALU biriminde nasıl işleneceği ve ilgili işlem sonuçlarının çıkış birimine nasıl ve ne zaman aktarılacağı gibi işlevleri üstlenir. Bir başka deyişle bilgisayardaki tüm birimlerin işleyişlerini organize etmekle sorumludur.

İkincil depolama birimi

Bu birim kalıcı ve yüksek kapasiteli bellek birimidir. Diğer birimlerde işlenen verilerin tekrar kullanılmak üzere saklanmasını sağlar. İkincil depolama birimindeki veriler kalıcıdır, başka bir deyişle bilgisayarın elektrik enerjisi kesildiğinde bu birim içerisindeki veriler kaybolmaz. İkincil depolama birimlerine veri yazmak ya da okumak bellek birimine göre daha yavaştır. DVD, Blue-ray gibi optik diskler, flaş bellek ve sabit disk ikincil depolama birimlerine örnek olarak verilebilir.

SAYI SİSTEMLERİ

Bilgisayarların çalışma prensiplerini anlamak, gerçekleştirilen işlemleri ve verilerin nasıl saklandığını kavramak adına sayı sistemlerini bilmemiz gerekir.

Mantık devreleri bilgi oluşturmak ve iletmek için 1'leri ve 0'ları kullanır. Bu iki değerli sayı sistemine, *ikilik* (binary) denir. İkilik sistem kullanmanın birçok avantajı vardır; ancak bize onluk sayı sistemini kullanarak sayma, ölçme ve işlem yapma öğretilmiştir. Onluk sayı sistemi, genellikle Arap rakamları olarak da adlandırılan 10 benzersiz sembol (0-9) içerir. Bu bölümde, farklı sayıda sembol içeren farklı sayı sistemlerini ve bunların birbirlerine nasıl çevrileceğini/dönüştürüleceğini öğreneceğiz.

Onaltılık ve sekizlik sayı sistemleri, daha az sayıda sembol kullanarak büyük ikilik değerler kümesini temsil etmek için kullanılır.

İlgili bölüm dört farklı sayı sistemini kapsamaktadır:

- Onluk (Decimal) sayı sistemi
- İkilik (Binary) sayı sistemi
- Sekizlik (Octal) sayı sistemi
- Onaltılık (Hexadecimal) sayı sistemi

Onluk sayı sistemi dünyayı nasıl yorumladığımızı ve ikilik sayı sistemi de bilgisayarların çalışma mantığını temsil ettikleri için kullanımlarını anlamak oldukça kolaydır. Onaltılık ve sekizlik sayı sistemleri ise daha az sayıda sembol kullanarak büyük ikilik değerler kümesini temsil etmek noktasında faydalıdır.

Sayıları oluşturan semboller bulundukları sıraya göre konumsal değer alırlar ve konum tabanlarına göre isimlendirilirler. Örneğin, onluk sayı sisteminde "6393" sayısında bulunan "3" rakamı iki ayrı yerde kullanılmıştır ve farklı değerliklere sahiptir. En sağdaki "3" rakamı "birler" basamağındadır ve sayısal değeri "3x1=3"'dür. Buna karşılık sağdan üçüncü sıradaki "3" rakamı "yüzler" basamağındadır ve sayısal değeri "3x100=300"'dür. Bu durum bütün sayı sistemlerinde aynıdır. Değişen sadece sayı ve o sayının konumuna göre aldığı değerdir.

Sayının sağ alt köşesine yazılan ifade, sayı değerinin hangi sistemle ifade edildiğini gösterir. Bir sayının sağ alt köşesinde bir tanımlama yoksa o sayının onluk sisteme göre yazıldığı kabul edilir. Sayıları oluşturan sembollerin basamak değerlikleri sağdan sola doğru yükselir. Yani en düşük değerli sembol en sağda, en yüksek değerli sembol ise en soldadır.

Bilgisayar sistemleri ikilik sayı sistemine göre çalışmaktadır.

Bir sayı sistemi, bir sayının farklı semboller kullanılarak nasıl temsil edilebileceğini tanımlar. Bir sayı farklı sistemlerde farklı şekilde temsil edilebilir. Örneğin; onluk sistemdeki $(52)_{10}$ sayısı, onaltılık sistemde $(34)_{16}$ ve sekizlik sistemde $(64)_8$ şeklinde gösterilir.

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 adlandırılır. Rakamlar taban değerinden küçük ve 0 dâhil tam sayı formundadır.

$$0 \leq a, b, c, d, e < T$$

$$(abcde)_T = a.T^4 + b.T^3 + c.T^2 + d.T + e.T^0$$

En büyük değerli basamak — En küçük değerli basamak

Onluk (Decimal) sayı sistemi

Decimal kelimesi Latince “decem” (on) kelimesinden türetilmiştir. Günlük yaşantımızda kullandığımız sayı sistemi onluk sayı sistemidir. Bu sistemde on tane sembol kullanılır. Bu semboller: {0,1,2,3,4,5,6,7,8,9}.

Örnek

- $275 = 2.10^2 + 7.10^1 + 5.10^0$
- $3847 = 3.10^3 + 8.10^2 + 4.10^1 + 7.10^0$

İkilik (Binary) sayı sistemi

Bit, bilgisayardaki en küçük bellek birimidir.

Binary kelimesi Latince “binarius” (iki) kelimesinden türetilmiştir. İkilik (Binary) sayı sistemi, sayısal elektronik sistemlerde yaygın olarak kullanılır. İki durum prensibi ile çalışır; açık veya kapalı. Bu durumlar, elektrik akımının ortamda var olup olmamasına karşılık gelir. Eğer elektrik akımı varsa, anlık durum 1 değerini alır, yoksa 0 değerini alır. İkilik sayı sisteminin sembolleri {0,1}’dir. Ayrıca bu sembollere ***Bi***nary Digi***t*** veya bu kelimelerden türetilmiş *Bit* denir. Bit bilgisayarda depolanabilen en küçük hafıza birimidir.

İkilik-onluk çevrimi:

Her basamak 2’nin kuvveti ile çarpılır ve sonuçların toplamı onluk sayıyı verir.

Örnek

• $(101011)_2 = 1.2^5 + 0.2^4 + 1.2^3 + 0.2^2 + 1.2^1 + 1.2^0$
$= 1.32 + 0.16 + 1.8 + 0.4 + 1.2 + 1.1$
$= 32 + 0 + 8 + 0 + 2 + 1$
$= 43$

Örnek

• $(11100001)_2 = 1.2^7 + 1.2^6 + 1.2^5 + 0.2^4 + 0.2^3 + 0.2^2 + 0.2^1 + 1.2^0$
$= 1.128 + 1.64 + 1.32 + 0.16 + 0.8 + 0.4 + 0.2 + 1.1$
$= 128 + 64 + 32 + 0 + 0 + 0 + 0 + 1$
$= 225$

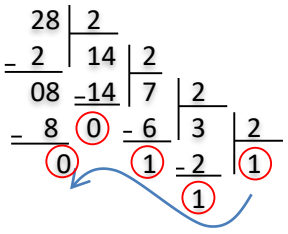
Onluk-ikilik çevrimi:

Onluk sayıları ikilik sayılara çevirirken bölme metodu kullanılır. Aşağıdaki örnekte görüleceği üzere sayı devamlı 2'ye bölünür ve kalan değerler tersten yazılarak ikilik sayı elde edilir.

Örnek

•$(28)_{10}$ sayısının ikilik sayı sistemindeki karşılığı kaçtır?

Çözüm:



$(28)_{10} = (11100)_2$

Sekizlik (Octal) sayı sistemi

Octal kelimesi Latince "octo" (sekiz) kelimesinden türetilmiştir. İkilik sayı sistemindeki sayıların daha kolay gösterilmesini sağlayan sayı sistemlerinden biri olarak kullanılır. $2^3 = 8$ olduğundan, iki tabanındaki üç bitlik bir sayı, bir dijit sekizlik sayı ile temsil edilebilir. Böylece veri girişi daha kolay olacak ve verilerin saklanması daha az yer kaplayacaktır. Semboller: {0,1,2,3,4,5,6,7}.

Sekizlik-onluk çevrimi:

Her basamak 8'in kuvveti ile çarpılır ve sonuçların toplamı onluk sayıyı verir.

Örnek

•$(765)_8$ sayısının onluk sayı sistemindeki karşılığı kaçtır?

Çözüm:

$7.8^2 + 6.8^1 + 5.8^0 =$

$7.64 + 6.8 + 5.1 =$

$448 + 48 + 5 = (501)_{10}$

Onluk-sekizlik çevrimi:

Onluk sayıları sekizlik sayı sistemine çevirirken bölme metodu kullanılır. Aşağıdaki örnekte görüleceği üzere sayı devamlı 8'e bölünür ve kalan değerler tersten yazılarak sekizlik sayı elde edilir.

Örnek

•$(225)_{10}$ sayısının sekizlik sayı sistemindeki karşılığı kaçtır?

Çözüm:

225 | 8
− 16 | 28 | 8
65 − 24 | 3
− 64 | 4
1

$(225)_{10} = (341)_8$

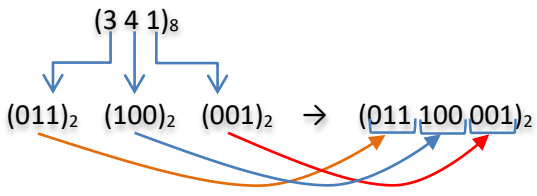
Sekizlik-ikilik çevrimi:

$2^3 = 8$ olduğundan üç bitlik ikilik bir sayı, bir dijit sekizlik sayı ile temsil edilebilir. Bu nedenle sekizlik sayılarda her bir dijitin ikilik sayı sistemindeki karşılıkları yan yana yazılarak çevrim gerçekleştirilir.

Örnek

•$(341)_8$ sayısının ikilik sayı sistemindeki karşılığı kaçtır?

Çözüm:



İkilik-sekizlik çevrimi:

İkilik sayı sekizlik sayı sistemine dönüştürülürken ikilik sistemdeki sayı sağdan üçer üçer gruplanarak sekizli sistemdeki karşılığı bulunur. Gruplama yapılırken eksik olan basamak yerinde 0 varmış gibi işlem yapılır.

Örnek

- $(11100)_2$ sayısının sekizlik sayı sistemindeki karşılığı kaçtır?

Çözüm:

$(011\ 100)_2$

$(3)_8$ $(4)_8$ → $(34)_8$

Onaltılık (Hexadecimal) sayı sistemi

Bilgisayar sistemleri ikilik sayıların girişini kolaylaştırmak ve hafızada daha az yer kaplaması için sekizlik veya onaltılık sayı sistemini kullanır.

Hexadecimal kelimesi Yunanca "hexa" (altı) ve Latince "decem" (on) kelimelerinden türetilmiştir. Bu sayı sisteminde 16 tane sembol bulunmaktadır. Bu semboller {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}'dir. 16'lık sayı sisteminde her bir harf bir rakamı temsil eder. Bunlar; A=10, B=11, C=12, D=13, E=14, F=15'tir.

0'lardan ve 1'lerden oluşan ikilik sistemdeki uzun sayıları hatırlamak veya klavyeden girmek oldukça zordur ve bu esnada hata yapma olasılığı yüksektir. Bilgisayar sistemleri, ikilik sayıların girişini kolaylaştırmak ve hafızada daha fazla yer harcamamak için onaltılık gösterimi kullanır. Örneğin, 24-bitlik 111100000000101011101001 RGB renk değerlerini ikilik sayı sistemi yerine F00AE9 gibi onaltılık sayı sisteminde kodlamak daha kolay olacaktır.

Onaltılık-onluk çevrimi:

Her basamak 16'nın kuvveti ile çarpılır ve sonuçların toplamı onluk sayıyı verir.

Çözüm:

Örnek

- $(9B4)_{16}$ sayısının onluk sayı sistemindeki karşılığı kaçtır?

$9.16^2 + 11.16^1 + 4.16^0 =$
$9.256 + 11.16 + 4.1 =$
$2304 + 176 + 4 = (2484)_{10}$

Onluk-onaltılık çevrimi:

Onluk sayıları onaltılık sayı sistemine çevirirken bölme metodu kullanılır. Aşağıdaki örnekte görüleceği üzere sayı devamlı 16'ya bölünür ve kalan değerler tersten yazılarak onaltılık sayı elde edilir.

Örnek

- $(2348)_{10}$ sayısının onaltılık sayı sistemindeki karşılığı kaçtır?

İkilik sayıyı onaltılık sayıya dönüştürmek için iki tabanlı sayı sağdan dörderli gruplara ayrılır. Eksik bitler varsa 0 kabul edilir.

Çözüm:

```
2348 |16
-16  |146 |16
 74  |144 |9
-64  |  2
 108
-96
 12  → C
```

$(2348)_{10} = (\ 92C\ )_{16}$

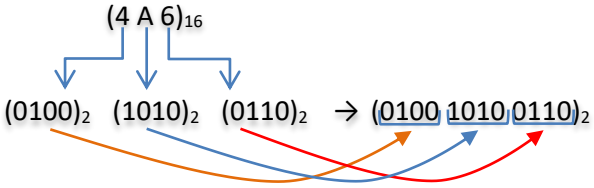
Onaltılık-ikilik çevrimi:

$2^4 = 16$ olduğundan 4 bitlik ikilik bir sayı, bir dijit onaltılık sayı ile temsil edilebilir. Bu nedenle onaltılık sayılarda her bir dijitin ikilik sayı sistemindeki karşılıkları yan yana yazılarak çevrim gerçekleştirilir.

Örnek

- $(4A6)_{16}$ sayısının ikilik sayı sistemindeki karşılığı kaçtır?

Çözüm:



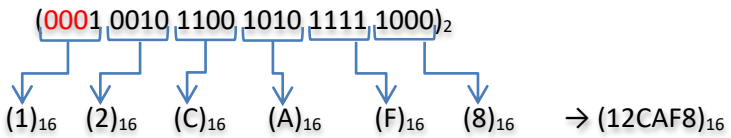
İkilik-onaltılık çevrimi:

İkilik sayı onaltılık sayı sistemine dönüştürülürken ikilik sistemdeki sayı sağdan dörder dörder gruplanarak onaltılık sistemdeki karşılığı bulunur. Gruplama yapılırken eksik olan basamak yerinde 0 varmış gibi işlem yapılır.

Örnek

- $(10010110010101111000)_2$ sayısının onaltılık sayı sistemindeki karşılığı kaçtır?

Çözüm:



YAZILIM

Program; belirli bir amaca yönelik olarak hazırlanmış, verilen komutlar doğrultusunda istenilen işlemleri belirli bir sırada yerine getiren komut dizisidir.

Genellikle yazılım ve program kavramları birbirlerinin yerine kullanılsa da temel olarak bir program belli işlevleri yerine getirmek üzere yazılmıştır ve komut satırlarından oluşmaktadır. Yazılım ise programlar bütününe verilen isimdir. Örnek olarak, bir veri tabanında ürün adlarını, fiyat ve adet bilgilerini kaydeden bir yazılıma sahip olduğumuzu varsayalım. Program ve veri tabanı yazılımın parçalarıdır; ancak veri tabanı program değildir. Veri tabanı programın daha kullanışlı olmasını sağlayan bir yardımcı araçtır.

Bilgisayar yazılımları sistem ve uygulama yazılımları olmak üzere iki ana gruba ayrılır.

Bilgisayar yazılımları, *sistem yazılımları* ve *uygulama* yazılımları olmak üzere temelde iki ana gruba ayrılır. *Sistem yazılımları,* bilgisayar sisteminin kaynaklarını yöneten ve uygulama programlarının işlevlerini yerine getirmesini sağlayan programlardır. İşletim sistemleri, ağ yazılımları, veri tabanı yönetim sistemleri ve programlama dilleri sistem yazılımlarıdır. *Uygulama yazılımları* ise son kullanıcının ihtiyaçlarını karşılamak için tasarlanmıştır. Kelime işlemci, hesap tablosu, fotoğraf düzenleme ve virüs programları gibi belirli bir alana ve işleme yönelik geliştirilmiş yazılımlardır.

PROGRAMLAMA DİLLERİ

Programlama dili, bilgisayara çeşitli işlemleri yaptırmak için uyulması zorunlu kurallar çerçevesinde komutların yazılmasını sağlayan her türlü karakter ve kurallar bütünüdür.

Günlük hayattaki bir sorunu bilgisayarla çözmek, rutin işlemleri kolaylaştırmak ve bilgisayarların isteğe uygun olarak özel bir takım işlemleri gerçekleştirmesi için programlanması gerekmektedir. Bir şirkette kullanılan stok uygulaması, sipariş takibi uygulaması, hastane otomasyonları ya da eğitim kurumlarının kullandığı öğrenci otomasyonları gibi programlamaya çok fazla örnek vermek mümkündür.

Bilgisayarların yapacağı görevlerin, bilgisayarın anladığı dil olan makine dili ile gerçekleştirilmesi gerekmektedir. İnsanların daha kolay yazıp okuyabileceği, ana diline yakın diller kullanılarak da bilgisayarlar programlanabilmektedir. Bu durumda bu tür bir programlama dili aracılığıyla programcının yazdığı komutlar bilgisayarın anlayabileceği makine diline dönüştürülmelidir.

Programlama dili, bilgisayara çeşitli işlemleri yaptırmak için uyulması zorunlu kurallar çerçevesinde komutların yazılmasını sağlayan her türlü karakter ve kurallar bütünüdür. Programlama dilleri, programcının bilgisayara hangi veri üzerinde nasıl işlem yapacağını, verinin nasıl depolanıp iletileceğini, hangi koşullarda hangi işlemlerin yapılacağını tam olarak anlatmasını sağlar.

Makine Dilleri, Düşük Seviyeli Diller ve Yüksek Seviyeli Diller

Bir programcı komutları yazmak için farklı programlama dilleri kullanabilir. Bazı diller bilgisayar tarafından doğrudan anlaşılırken bazıları çeviriciye ihtiyaç duymaktadır. Bir programlama dili konuştuğumuz doğal dile ne kadar yakın ise o kadar yüksek seviyeli dil, makine diline ne kadar yakın ise o kadar düşük seviyeli dil olarak sınıflandırılmaktadır.

Makine Dilleri

Makine dili sadece 1'lerden ve 0'lardan oluşan komut dizisidir.

Her bilgisayar sadece kendi mimarisi için tanımlanmış olan makine dilini anlayabilir. *Makine dili* sadece 1'lerden ve 0'lardan oluşan komut dizisidir. İnsanların komutları ve işlenecek verileri hata yapmadan ikilik sayı sisteminde girmeleri oldukça zordur. Tablo 1.1'de örnek bir makine dili kodu gösterilmektedir.

Tablo 1.1. Düşük seviyeli Assembly dilinde yazılmış kod ve makine dili karşılığı

Assembly dili	Makine dili
`SUB AX,BX`	`001010111000011`
`MOV CX,AX`	`100010111001000`
`MOV DX,0`	`10111010000000000000000`

Düşük Seviyeli Diller

Makine dillerinde program yazmak oldukça zordur ve uzun sürmektedir. Bu nedenle daha hızlı kod yazmak için her bir işlemi temsil etmek amacıyla İngilizce kısaltmalar kullanılmaya başlanmıştır. Bu kısaltmalar sembolik dillerin temelini oluşturmaktadır. Assembler adı verilen çevirici programlar, sembolik dilde yazılmış programları makine dillerine çevirmek için geliştirilmiştir. Tablo 1.1'de örnek bir Assembly kod ve makine dili karşılığı gösterilmektedir.

İnsanların konuşma diline yakın olan dillere "yüksek seviye dil", donanıma yakın dillere de "düşük seviye dil" denilmektedir.

Yüksek Seviyeli Diller

Program geliştirme sürecini hızlandırmak için yüksek seviyeli programlama dilleri geliştirilmiştir. C++, Java, C#, Visual Basic gibi dillerde yazılan komutlar İngilizceye ve hesaplama işlemleri de gerçek hayatta kullandığımız matematiksel ifadelere çok yakındır. Derleyici adı verilen programlar yazılan komutları makine diline çevirmektedirler.

Bir dilin seviyesi düştükçe zorluk seviyesi artmaktadır. Fakat donanıma daha yakın olması nedeniyle çalışma hızı da artmakta ve yüksek seviyedeki bir dile göre programı daha etkin kullanabilme açısından avantajları da olabilmektedir. Düşük seviyeli diller genelde sistem programlama, mikroişlemci programlama vb. alanlarda kullanılmaktadır. Zaman ve kolaylık açısından baktığımızda ise yüksek seviyeli diller göze çarpmaktadır. Tablo 1.2'de popüler bazı programlama dilleri ve bu diller hakkında kısa bilgiler yer almaktadır.

Bireysel Etkinlik

- Popüler yazılım projelerinin hangi programlama dili kullanılarak geliştirildiğini ve neden bu dilin tercih edildiğini araştırınız.

Tablo 1.2. Bazı popüler programlama dilleri [2]

Programlama Dili	Açıklama
Fortran	Fortran (FORmula TRANslator) 1954'te IBM tarafından üretilen IBM 704 için John Backus ve ekibi tarafından geliştirilmiştir. 1950'lerdeki bilgisayarlar için hız çok şey ifade ettiğinden yazması zor da olsa makine kodu popülerdir. İşte bu noktada Backus ve ekibi hem yüksek seviye programlama dilleri gibi kolay yazılabilen hem de makine kodunda yazılmış gibi hızlı çalışan bir programlama dili olarak Fortran'ı tanıtmışlardır. Fortran günümüzde hala birçok programcı tarafından kullanılmaktadır. Yeni sürümleri nesneye dayalı programlamayı desteklemektedir. Son sürümü 2008 yılında yayınlanmıştır.
COBOL	COBOL 1959 yılında Grace Hopper'ın geliştirdiği dil temel alınarak CODASYL tarafından tasarlanmıştır. Veri işleme için taşınabilir bir programlama dili oluşturmak amacıyla ABD Savunma Bakanlığı'nca geliştirilmiştir. 1968'de standartlaştırılmış ve dört kez revize edilmiştir. Nesneye yönelik programlama desteği sunmaktadır. Son sürümü 2014 yılında yayınlanmıştır.
Ada	Ada 1970'li yılların sonunda ABD Savunma Bakanlığı'nın desteğiyle Pascal temel alınarak geliştirilmiştir. Geliştirilen Pascal tabanlı bu dil, şair Lord Byron'un kızı Lady Ada Lovelace'in ismi ile adlandırılmıştır. Lady Ada dünyanın ilk bilgisayar programını 1800'lerde yazmıştır (Charles Babbage tarafından tasarlanan mekanik hesaplama makinesi Analitik Motor için yazmıştır.). Ada nesne yönelimli programlamayı desteklemektedir. Son sürümü 2016 yılında yayınlanmıştır.
Pascal	Pascal, Niklaus Wirth tarafından tasarlanan ve 1970 yılında yayınlanan bir yapısal programlama dilidir. Yapılandırılmış programlama, önceki tekniklerden daha açık olan ve kolayca test edilebilen ve değiştirilebilen programların yazılmasına olanak tanıyan bir programlama tekniğidir. Bu isim Fransız matematikçi, filozof ve fizikçi Blaise Pascal'ın onuruna verilmiştir. Nesne yönelimli programlama için tasarlanan ve Object Pascal olarak bilinen sürüm 1985'te geliştirilmiştir. Bu sürüm 1980'lerin sonlarına kadar Apple Computer ve Borland tarafından kullanılmıştır ve daha sonra Microsoft Windows platformunda Delphi haline gelmiştir.

Lady Ada Lovelace, dünyanın ilk bilgisayar programını 1800'lerde yazmıştır.

Basic	BASIC (Beginner's All-purpose Symbolic Instruction Code) (Yeni Başlayanlar için Çok Amaçlı Sembolik Talimat Kodu) tasarım amacı kullanım kolaylığı sağlamak olan genel amaçlı, yüksek seviyeli bir programlama dilidir. 1964'te John G. Kemeny, Thomas E. Kurtz ve Mary Kenneth Keller, ABD'nin New Hampshire kentindeki Dartmouth College'de orijinal BASIC dilini tasarlamıştır. Bu ekibin amacı, öğrencilerin fen ve matematik dışındaki alanlarda bilgisayarları kullanabilmelerini sağlamaktır.
C	C, ilk olarak 1969-1973 yılları arasında Bell Laboratuvarlarında Dennis Ritchie tarafından geliştirilmiştir ve UNIX işletim sisteminin geliştirilme dili olarak tanınmıştır. O zamandan beri mevcut bilgisayar mimarileri ve işletim sistemlerinin çoğunluğu için en yaygın kullanılan programlama dillerinden biri haline gelmiştir. C programlama dili, 1989'dan beri Amerikan Ulusal Standartlar Enstitüsü (ANSI) (ANSI C) ve daha sonra Uluslararası Standardizasyon Organizasyonu (ISO) tarafından standartlaştırılmıştır. Son sürümü 2011 yılında yayınlanan C11'dir.
Objective-C	Objective-C, temeli C programlama diline dayanan nesne yönelimli bir programlama dilidir. 1980'lerin başında geliştirilen Objective-C, önce NeXT ve ardından Apple tarafından satın alınmıştır. Bugün OS X işletim sistemleri ve iOS tabanlı cihazlar (iPod, iPhone ve iPad) için önemli bir programlama dili haline gelmiştir.
PHP	Başlangıçta 1994 yılında Rasmus Lerdorf tarafından oluşturulan PHP, daha sonra bir programcı komitesi tarafından geliştirilmiştir. PHP, web uygulamaları geliştirmek için tasarlanmıştır ve aynı zamanda genel amaçlı bir programlama dili olarak kullanılan açık kaynak kodlu nesne yönelimli bir "betik" dildir. Wikipedia ve Facebook gibi birçok web sitesi tarafından kullanılmaktadır. PHP platformdan bağımsız olarak herhangi bir web sunucuda çalıştırılabilmektedir.
Java	Sun Microsystems bünyesinde, 1991 yılında James Gosling önderliğindeki proje sonucunda C++ dili temel alınarak geliştirilen sınıf temelli, nesne yönelimli, genel amaçlı bir programlama dili olarak ortaya çıkmıştır. Java programlama dilinin temel amacı çok farklı bilgisayar sistemlerinde ve bilgisayar ile kontrol edilen cihazlarda çalışabilecek programlar yazabilmektir. Bu teknik "Bir kez yaz, her yerde çalıştır." olarak adlandırılmaktadır. Derlenmiş Java kodunun, Java'yı destekleyen tüm platformlarda yeniden derleme gerektirmeden çalışabileceği anlamına gelmektedir. Java uygulamaları genellikle bilgisayar mimarisine bakılmaksızın herhangi bir Java sanal makinesinde (JVM) çalışabilen bayt koduyla derlenir. Java, 2018 yılı itibariyle kullanımda olan en popüler programlama

Birçok programlama dili ve işletim sistemi C programlama dilinde yazılmıştır.

	dillerinden birisidir. Son Java 10 sürümü 2018 yılında yayınlanmıştır.
C#	C# programlama dili C++ ve Java dillerini temel alarak internet ve web'i bir bilgisayar uygulamasında birleştirmek için Microsoft tarafından geliştirilmiştir. Nesneye yönelik programlamayı desteklemektedir. Son 7.3 sürümü 2018 yılında yayınlanmıştır.
Perl	Perl (Practical Extraction and Report Language) web programlama için en çok kullanılan nesne yönelimli betik dillerinden birisidir. C, C++ dillerini temel alarak 1987 yılında Larry Wall tarafından geliştirilmiştir. Son 5.28 sürümü 2018 yılında yayınlanmıştır.
Phyton	Guido van Rossum tarafından geliştirilen ve ilk olarak 1991'de yayınlanan Python, özellikle anlamlı boşluklar kullanarak kod okunabilirliğini vurgulayan bir tasarım felsefesine sahiptir. Python, dinamik ve otomatik bellek yönetimine ve kapsamlı bir standart kütüphaneye sahiptir. Phyton bir sistem programlama dili olan Modula-3'ü örnek almıştır. Son 3.7 sürümü 2018 yılında yayınlanmıştır.

C++'IN TARİHÇESİ

C++ (Si pılas pılas diye okunur.) programlama dili, Bjarne Stroustrup'un doktora tezine çalıştığı 1979'a kadar uzanan bir tarihe sahiptir. Stroustrup, Simula adı verilen bir dil üzerinde çalışıyordu ve isminden de anlaşılacağı üzere, bu dil simülasyonlar için kullanılmaktaydı. Stroustrup'un çalıştığı sürüm olan Simula 67 dili, nesne yönelimli programlamayı destekleyen ilk dil olarak kabul edilmektedir. Stroustrup, bu özelliğin yazılım geliştirme için çok yararlı olduğunu ortaya koydu; ancak Simula dili pratik kullanım için çok yavaştı. Kısa bir süre sonra, "C ile Sınıflar" üzerine çalışmaya başladı. Amacı, C dilinin hızından ödün vermeden ona nesne yönelimli bir özellik eklemekti.

Onun geliştirdiği hali, C dilinin tüm özelliklerine ek olarak sınıflar ve güçlü tip denetimleri içeriyordu. Sınıfların olduğu derleyiciye sahip ilk C, **CPre** denen C derleyicisinden türetilen **Cfront** olarak adlandırıldı. Cfront'un, çoğunlukla Sınıflar ile yazılmasından kaynaklı, kendisini derleyen bir derleyici haline gelmesi dikkat çekici bir noktaydı. Cfront, daha sonra 1993'te, yeni özelliklerin bütünleştirmesi zorlaştıktan sonra terk edildi. Bununla birlikte Cfront, gelecekteki derleyicilerin ve Unix işletim sistemindeki uygulamaların üzerinde büyük bir etki yarattı.

1983'te, dilin adı C++ olarak değiştirildi. C dilinde ++ operatörü, Stroustrup'un dili nasıl değerlendirdiğine dair bir fikir vermektedir ve bir değişkenin değerini arttırmak için kullanılan yeni bir operatördür. Bu zamana kadar başka birçok yeni özellik de eklenmiştir. Bunların en önemlileri sanal fonksiyonlar, fonksiyonların aşırı yüklenmesi, yani aynı yaşam alanında aynı isme sahip birden fazla fonksiyonun kullanılabilmesi, & sembollü referanslar, sabitler için kullanılan **const** anahtar kelimesi ve iki eğik çizgi ile açıklama satırının eklenmesi şeklindedir (Bu özellik, BCPL'den kazandırılmıştır.).

1985'te, “Stroustrup'un C++ Programlama Dili” adlı referansları yayınlandı. Aynı yıl, C++ ticari bir ürün olarak tanıtıldı. Bu dil resmi olarak henüz standartlaştırılmamıştı; ancak kitabı çok önemli bir referans haline geldi. Dil, 1989'da korumalı ve statik üyelerin yanı sıra çeşitli sınıflardan miras alma özellikleri ile güncellendi.

1990 yılında, “Açıklamalı C++ Referans Kılavuzu” yayımlandı ve aynı yıl, Borland'ın Turbo C++ derleyicisi ticari bir ürün olarak piyasaya sürüldü. Turbo C++, C++'ın gelişimi üzerinde önemli bir etkisi olan çok sayıda ek kütüphane ile çıkmıştır. Turbo C++'ın son kararlı sürümünün 2006 yılında yayınlanmış olmasına rağmen, derleyici halen yaygın olarak kullanılmaktadır.

1998 yılında, C++ standartlar komitesi, C++ 98 olarak bilinen ve C++ ISO/IEC 14882:1998 adı verilen ilk uluslararası standardı yayınlamıştır. Açıklamalı C++ Referans Kılavuzunun da standardın geliştirilmesinde büyük bir etkisi olduğu bilinmektedir. Ayrıca kavramsal gelişimini 1979'da başlatan Standart Şablon Kütüphanesi de dâhil edilmiştir. 2003 yılında komite, 1998 standartları ile bildirilen çok sayıda soruna yanıt vermiş ve buna göre revize etmiştir. Değiştirilen dil C++ 03 olarak adlandırılmıştır.

2017 yılında C++ programlama dilinin son sürümü olan C++ 17 yayınlanmıştır.

2005 yılında, C++ standartlar komitesi, en son C++ standardına eklemeyi planladıkları çeşitli özellikleri detaylandıran bir teknik rapor (TR1 olarak adlandırıldı) yayınlamıştır. Yeni standart, ilk on yılın sonuna kadar piyasaya sürüleceği tahmin edilen C++ 0x olarak adlandırılmıştır. 2011'in ortalarında ise hem dilin yapısı hem de standartlarında birçok eklemeler içeren yeni C++ standardı (C++ 11) tamamlanmıştır. 2014 yılında, C++ 14 (C++ 1y olarak da bilinir.), çoğunlukla hata düzeltmeleri ve küçük iyileştirmeler içeren C++ 11'e küçük bir uzantı olarak piyasaya sürülmüştür. C++ 14'ten sonra, C++ 1z olarak da bilinen büyük bir düzeltme C++ 17 2017'de yayınlanmıştır [3].

BİR C++ PROGRAMININ GELİŞTİRİLMESİ

Bir C++ programı genellikle *düzenleme*, *önişleme*, *derleme*, *birleştirme*, *yükleme*, *çalıştırma* ve *hata ayıklama* olmak üzere yedi aşamadan geçmektedir. Aşağıda temel bir C++ programının geliştirme aşamalarında neler gerçekleştiği anlatılmaktadır.

Bir C++ kod dosyası genellikle “.cpp” uzantılıdır.

- **DÜZENLEME (EDITING)**

Programcı bir düzenleyici program (editör) yardımı ile C++ programını (kaynak kod) yazar, gerekli düzenlemeleri yapar ve diske dosya olarak kaydeder. Bir C++ kaynak kod dosyası genellikle *“.cpp”* uzantılıdır. Düzenleyici program olarak herhangi bir metin düzenleme programı kullanılabilir. Örneğin, Linux sistemlerinde “vi” editörü, Windows işletim sistemlerinde “Not defteri” programı bu amaçla kullanılabilir.

Programcılara kaynak kodlarını yazmada kolaylık sağlamak amacıyla çeşitli yazılım geliştirme araçları geliştirilmiştir. Bu araçlarda yer alan hata ayıklayıcıları sayesinde programcı yazım hatalarını ve mantıksal hataları rahatlıkla tespit edebilir. Günümüzde en çok tercih edilen yazılım geliştirme araçlarına örnek

olarak Microsoft Visual Studio, Dev C++, NetBeans, Eclipse, Apple's Xcode verilebilir.

- **ÖNİŞLEME (PREPROCESSING)**

Önişleme, kaynak kodun derlenmesi aşamasına geçilmeden önce kaynak kodla ilgili yapılması gereken düzenlemelerin önişleme komutları kullanılarak gerçekleştirilmesidir. Örneğin, programcının kodunun daha anlaşılır olması için yazmış olduğu açıklamalarının derlenecek koddan çıkartılmasıdır. Ya da program içinde "#define pi 3.14;" söz dizimi gibi sabit ifadelerin tanımlandığı yerler dikkate alınarak kaynak kod içerisindeki "pi" görülen tüm yerlere "3.14" ün yazılmasıdır.

- **DERLEME (COMPILING)**

C++ ön işlemcisi tarafından oluşturulan genişletilmiş kaynak kod, önce derleyici tarafından platformdaki işlemciye uygun makine diline ve daha sonra da makine kodu işletim sistemine uygun amaç koda dönüştürülür.

Bir C++ programı düzenleme, önişleme, derleme, birleştirme, yükleme, çalıştırma ve hata ayıklama aşamalarından geçmektedir.

- **BİRLEŞTİRME (LINKING)**

Bir C++ programı tipik olarak başka yerlerde tanımlı fonksiyonlara (standart kütüphanedeki fonksiyonlar gibi) referanslar vermektedir. Derleyicinin oluşturduğu "*amaç kod*" da olmayan fonksiyonların yeri sonradan tamamlanmak üzere boş bırakılır. Birleştirici, amaç kod ile fonksiyonların ait oldukları kütüphaneleri birleştirir ve çalıştırılabilecek program dosyasını oluşturur.

- **YÜKLEME (LOADING)**

Bir programın çalışabilmesi için bilgisayarın birincil belleğine yüklenmesi gerekir. İşletim sisteminin yükleyici isimli bileşeni, çalışmaya hazır program dosyasını diskten okuyarak bilgisayarın birincil belleğine yükler. Yükleyici ayrıca programın çalışması için gerekli kütüphane dosyalarını da birincil belleğe aktarır.

- **ÇALIŞTIRMA (EXECUTING)**

Bu aşamada bilgisayar, merkezî işlem biriminin kontrolünde bellekteki programın komutlarını birer birer çalıştırır.

- **HATA AYIKLAMA (DEBUGGING)**

Genellikle programlar ilk deneme esnasında çalışmazlar. Yukarıda bahsedilen 6 aşamanın herhangi birinden kaynaklanan bir sorunun olup olmadığı test edilir. Çalıştırılmak istenilen programda hata varsa editör penceresinde hatanın hangi satır ya da satırlardan kaynaklandığı programcıya bildirilecektir. Sebep araştırılarak başa dönülür ve her işlem yeniden tekrarlanır.

Genellikle üç tip hata ile karşılaşılır:

- *Yazım hataları*: C++ yazma kuralları ihlal edildiğinde oluşan hatalardır. Bir parantezi eksik ya da fazla yazmak, noktalı virgül ";" işaretini unutmak, C++ diline özgü ifadelerin yanlış yazılması çok sık karşılaşılan yazım hatalarındandır. Bu tip hataları tespit etmek kolaydır. Derleyici bu hataları

Program geliştirmede üç tip hata ile karşılaşılır: Yazım hataları, çalışma zamanı hataları ve mantıksal hatalar.

algılayarak kullanıcıya hatanın yerini bildirir. Bu nedenle bu tip hatalar derleme zamanı hataları olarak da bilinir.

- *Çalışma zamanı hataları*: Genellikle kullanıcı kaynaklı hatalardır. Örneğin, kullanıcı sayısal değer girilmesi gereken bir değişkene metin girdiğinde hata oluşacaktır. Çok sık karşılaşılan bir hata da matematiksel işlemlerin yürütüldüğü yerde gerçekleşebilecek sıfıra bölüm hatasıdır. Programcının geliştirme aşamasında bu tür istisnai durumları dikkate alarak kodlamasını yapması, çalışma zamanı hataları ile karşılaşmasının önüne geçerek programın kırılmasını engelleyecektir.
- *Mantıksal hatalar*: Bir programın yürütülmesi sırasında, belirli girdi değerleri verildiğinde istenilen çıktının elde edilemediği durumlarda karşılaşılan hatalardır. Beklenen çıkışın üretilmediği; ancak programın çalışmasında bir problemin olmadığı bu tip hatalara, mantıksal hata denir. Bunun sebebi hatalı yazılan, örneğin gerçekleştirilmesi istenilen işlemde + yerine – operatörünün yazıldığı matematiksel formüller olabilmektedir. Komutların yürütme sırası adım adım takip edilerek programın istenilen sonucu neden almadığı tespit edilebilir.

Özet

- Bilgisayar, donanım ve yazılım olmak üzere iki bileşenden meydana gelir. Donanım, bilgisayarı oluşturan tüm fiziksel kısımlara (elektrik, elektronik ve mekanik birimler) verilen addır. Yazılım ise bilgisayarların isteğe uygun işleri yapabilmeleri için onlara iletilen tüm bilgiler ve komutlardır. Yazılımı sistem ve uygulama yazılımları olarak iki gruba ayırabiliriz.
- Günümüz bilgisayarlarının çalışma prensipleri, John von Neumann'ın 1945'de geliştirdiği mimariyi temel almaktadır. Bu mimaride basitçe bir işlemci, bir hafıza ve giriş/çıkış sistemleri yer almaktadır. Bir bilgisayar giriş, çıkış, bellek, aritmetik ve mantıksal, merkezi işlem ve ikincil depolama birimlerinden meydana gelmektedir.
- Giriş birimleri bilgisayara veri aktarmak için kullanılan aygıtlardır. Çıkış birimleri ise bilgisayarda işlenen verinin sonucunun gösterildiği aygıtlardır. Bellek işlenecek ve işlenmiş verilerin geçici olarak saklandığı bir hafıza birimidir. Aritmetik ve mantıksal birim işlemci içerisinde yer alan veriler üzerinde artitmetik ve mantıksal işlemler yapan birimdir. Merkezi işlem birimi bilgisayardaki tüm birimleri kontrol eder ve giriş biriminden alınan verilerin ve işlenen verilerin belleğe aktarılmasını sağlar. İkincil depolama birimleri kalıcı ve yüksek kapasiteli bellek birimleridir.
- Bilgisayar sistemleri veri oluşturmak ve iletmek için 1'leri ve 0'ları kullanır. Bu iki değerli sayı sistemine, ikilik (binary) denir. 1 ve 0 değerleri Binary Digit olarak ifade edilir. Bu ifadeden türetilmiş bit kelimesi en küçük bellek birimi olarak adlandırılır. Bilgisayarda verilerin aktarılmasında hata yapmamak, veri girişini hızlandırmak ve daha az dijit ile daha çok değer gösterebilmek için sekizlik ve onaltılık sayı sistemleri kullanılmaktadır. $2^3=8$ olduğundan üç dijitlik/bitlik bir ikilik sayı bir dijitlik sekizlik sayı ile gösterilebilmektedir. Aynı şekilde $2^4=16$ olduğundan 4 bitlik bir ikilik sayı bir dijitlik onaltılık sayı ile gösterilebilir. Onaltılık sayı sisteminde 10-15 arasındaki değerleri tek sembolle ifade edebilmek için sayılar yerine harfler kullanılmaktadır. Bu harfler A=10, B=11, C=12, D=13, E=14 ve F=15 şeklindedir.
- Sayı sistemleri arasında çevrim işlemi oldukça basittir. İkilik sayılar sekizlik sayıya çevrilirken ikilik sayı sağdan üçerli gruplara ayrılır ve eksik kalan bitler 0 kabul edilir. Her bir grubun sayısal değerleri aynı sırada birleştirilerek sekizlik karşılık elde edilir. Sekizlikten ikilik sayı sistemine çevrilirken ise her bir sekizlik dijitin ikilik sistemdeki karşılıkları aynı sırada yan yana yazılarak ikilik karşılık elde edilir. İkilik sayıdan onaltılık sayıya çevirme işleminde ikilik sayı sağdan dörderli gruplara ayrılır ve eksik kalan bitler 0 kabul edilir. Her bir grubun sayısal değerleri aynı sırada birleştirilerek onaltılık karşılık elde edilir. Onaltılıktan ikilik sayı sistemine çevrilirken ise her bir onaltılık dijitin ikilik sistemdeki karşılıkları aynı sırada yan yana yazılarak ikilik karşılık elde edilir.
- İkilik sayı sistemi kullanarak bilgisayarları programlamak oldukça zordur. Bu nedenle çeşitli programlama dilleri geliştirilmiştir. Programlama dilleri aracılığıyla çeşitli semboller, ifadeler kullanılarak oluşturulan komutlar grubuyla bilgisayara yapılması istenen işlemler ve veriler aktarılır. İnsanların konuşma diline yakın olan dillere "yüksek seviye dil", donanıma yakın dillere de "düşük seviye dil" denilmektedir. Programlama dillerini makine dilleri, düşük seviyeli diller ve yüksek seviyeli diller olarak üç gruba ayırabiliriz. Derleyiciler yüksek seviyeli programlama dilleri ile yazılmış programları bilgisayarların anladığı makine diline dönüştüren sistem yazılımlarıdır.

Özet (devamı)

- 1979 yılında Bjarne Stroustrup tarafından Bell laboratuvarlarında C programlama diline eklemeler yapılarak yeni bir dil geliştirilmiştir. Bu yeni dile "Sınıflarla C" adı verilmiş ve daha sonra 1983 yılında dilin adı C++ olarak değiştirilmiştir ve ilk kullanım kılavuzu yayınlanmıştır. 1989 yılında resmi olarak 2.0 versiyonu yayınlanmıştır. 1998 yılında C++ standart komitesi ilk uluslararası standart ISO C++ 98'i yayınlamıştır. 2003 yılında, C++ 98'in hataları düzeltilen dil C++ 03 olarak adlandırılmıştır. Daha sonra dile yeni düzeltmeler ve eklemeler yapılarak 2011 yılında C++ 11, 2014 yılında C++ 14 ve 2017 yılında da C++ 17 sürümleri yayınlanmıştır.
- C++ dilinde program geliştirme düzenleme, önişleme, derleme, birleştirme, yükleme, çalıştırma, hata ayıklama aşamalarından oluşmaktadır. Düzenleme, kaynak kodların bir editör aracılığıyla oluşturulduğu aşamadır. Önişleme, kaynak kod içerisindeki açıklamaların kaldırılması, sabit değerlerin kaynak kod içerisinde yerlerine değerlerinin yazılması gibi işlemlerin yapıldığı aşamadır. Derleme aşamasında, önişleme ile oluşturulan kaynak kod uygulamanın çalışacağı işletim sistemine göre makine diline çevrilerek amaç kod üretilir. Birleştirme aşamasında, kullanılan kütüphane fonksiyonları amaç kod ile birleştirilir. Yükleme aşamasında üretilen program bilgisayarın belleğine yüklenir. Çalıştırma aşamasında merkezi işlem birimi bellekteki programı adım adım işletir. Hata ayıklama aşaması yazılan programda hata olması durumunda işletilecek adımdır. Yazım hatası, çalıştırma zamanı hatası ve mantık hatası olmak üzere üç tip hata vardır.

DEĞERLENDİRME SORULARI

1. Bir bilgisayarın fiziksel kısmına ne ad verilmektedir?
   a) Ana kart
   b) Kasa
   c) Donanım
   d) Ekran
   e) İşletim sistemi

2. Belirli bir amaca yönelik olarak hazırlanmış ve istenilen işlemleri belirli bir sırada yerine getiren komut dizilerinin genel adı aşağıdakilerden hangisidir?
   a) İşlemci
   b) Kayıt defteri
   c) Metin editörü
   d) Program
   e) Bilgisayar

3. Giriş birimi aşağıdakilerden hangisi olamaz?
   a) Dokunmatik ekran
   b) Web kamerası
   c) Mikrofon
   d) Barkod okuyucu
   e) Hoparlör

4. Bir bilgisayarın birimlerini kontrol eden ve verileri işleyerek sonuçları belleğe kaydeden birim aşağıdakilerden hangisidir?
   a) RAM
   b) Sabit disk
   c) Flaş bellek
   d) İşlemci
   e) Optik disk

5. $(465)_8$ sayısının ikilik sistemdeki karşılığı aşağıdakilerden hangisidir?
   a) $(100101100)_2$
   b) $(100110101)_2$
   c) $(111011101)_2$
   d) $(101110100)_2$
   e) $(100101110)_2$

6. İnsanların konuşma diline yakın programlama dillerine verilen genel ad aşağıdakilerden hangisidir?
   a) Yüksek seviyeli diller
   b) Makine dilleri
   c) Düşük seviyeli diller
   d) Yapay zekâ dilleri
   e) Assembly dilleri

7. C++ programının çalışması sırasında sıfıra bölüm hatası meydana gelmesi hangi tür hatadır?
   a) Mantık hatası
   b) Yazım hatası
   c) Çalışma zamanı hatası
   d) Derleme hatası
   e) İşlem hatası

8. C++ program geliştirme aşamasında birleştiricinin görevi aşağıdakilerden hangisidir?
   a) Kaynak kodu amaç koda dönüştürmek
   b) Kaynak kodu makine diline dönüştürmek
   c) Kaynak kodla amaç kodu birleştirmek
   d) Amaç kodu çalıştırmak üzere diskteki dosya ile birleştirmek
   e) Amaç kod ile fonksiyonların ait oldukları kütüphaneleri birleştirmek

9. $(111110000011010)_2$ sayısının onaltılık sistemdeki karşılığı aşağıdakilerden hangisidir?
   a) FC1B
   b) 7C1A
   c) F832
   d) 7B1F
   e) F68C

10. C++ kaynak kodunun makine diline dönüştürülmesi hangi geliştirme aşamasında olmaktadır?
    a) Derleme
    b) Birleştirme
    c) Yükleme
    d) Çalıştırma
    e) Önişleme

Cevap Anahtarı

1.c, 2.d, 3.e, 4.d, 5.b, 6.a, 7.c, 8.e, 9.b, 10.a

YARARLANILAN KAYNAKLAR

[1] Seferoğlu, S. S. (2006). *Öğretim teknolojileri ve materyal tasarımı*. Pegem A Yayıncılık.

[2] Deitel, P. J., & Deitel, H. M. (2016). *C++ ile Programlama*. (C. Öz, Çev.) Palme Yayınevi.

[3] Albatross. 2006. *History of C++.* Erişim tarihi:18.07.2018, http://www.cplusplus.com/info/history/

ALGORİTMALAR VE AKIŞ DİYAGRAMLARI

İÇİNDEKİLER

- Algoritma
- Sözde Kodlar
- Akış Diyagramları
- Uygulamalar

HEDEFLER

- Bu üniteyi çalıştıktan sonra;
  - Algoritma kavramını açıklayabilecek,
  - Bir problemin çözümüne yönelik algoritma tasarlayabilecek,
  - Tasarlanmış algoritmaların sözde kodunu yazabilecek,
  - Algoritmaları akış diyagramları ile görselleştirebilecek,
  - Algoritma testi yapabileceksiniz.

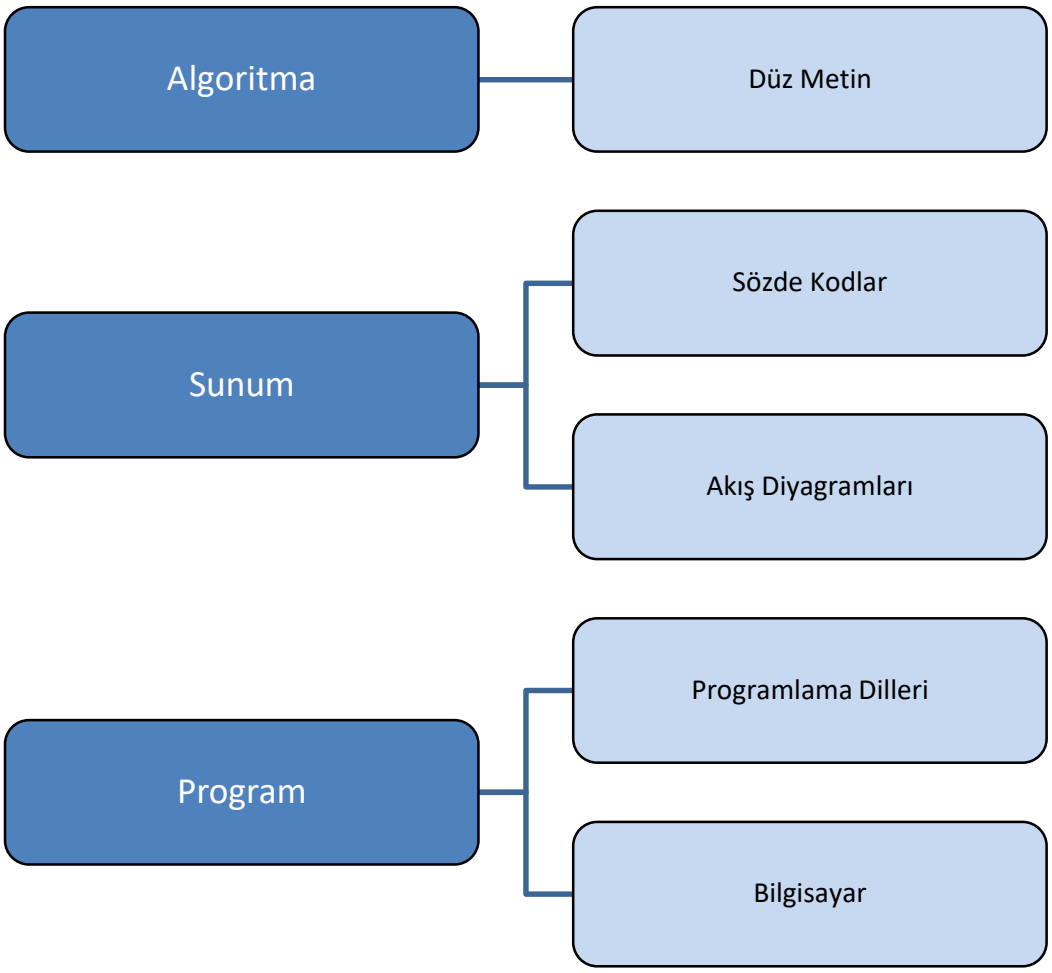
Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

Doç. Dr. Recep ERYİĞİT

ÜNİTE

2


Algoritma
Düz Metin
Sunum
Sözde Kodlar
Akış Diyagramları
Program
Programlama Dilleri
Bilgisayar

GİRİŞ

Programlama ya da diğer bir deyişle yazılım geliştirme, bilgisayar donanımına nasıl davranacağını anlatan veya bilgisayara yön veren komutlar dizisi oluşturmaktır. Program geliştirme programlama dilleri kullanılarak yapılmaktadır. Geliştirmede kullanılan programlama dili C++ veya Java gibi yüksek seviyeli bir dil olabileceği gibi bazı durumlarda makine dili de olabilir. Yazılan kod kaynak kod olarak da adlandırılır. Çoğu programlama dilinde kaynak kodlar derleyici tarafından bilgisayarda çalışır hale getirilir. Derleyiciler; yazılan programları okuyup içerisinde yazımsal hatalar olup olmadığını bulan, bulduğu hataları kullanıcıya göstererek programın düzeltilmesine yardım eden, hata yoksa programı çalıştırıp sonucunu gösteren program geliştirme platformlarıdır.

Programlama dilleri bazı yönleriyle doğal dillere benzeseler de aktarım güçleri doğal dillere oranla çok sınırlıdır. İnsanlar arası iletişimde dinleyen taraf duyduğu sözleri kendi bilgileri ile yorumlayarak anlamlandırır. Programları konuşmacı, bilgisayarı da dinleyici olarak kabul edersek, bilgisayarlar programların söylediklerini (henüz !) yorumla(ya)mazlar. Bu nedenle programlar, yapmak istedikleri işin bütün detaylarını programlama dilinin içerdiği yapıları kullanarak bilgisayara aktarmalıdır. Toplama işlemini bilen birisine 3 ile 2'nin toplamının kaç olduğu sorulduğunda, işlemin yapılması için başka bir bilgi vermeye gerek yoktur. Aynı iş program aracılığı ile yapılmak istendiğinde toplama işleminin derleyicinin hangi operatörü veya fonksiyonu ile yapılması gerektiği kodda belirtilmek zorundadır.

Algoritma, bir sorunun çözüm adımlarının sıralı halidir.

Programlama (Yaygın şekliyle kodlama olarak da adlandırılır.), sorunun analizi, çözüm algoritmasının tasarlanması ve seçilen bir programlama dilinde algoritmanın gerçekleşmesini içerir.

Algoritma bir problemin çözümünde izlenecek sıralı kurallar listesidir. Günlük hayatta yaptığımız işler dâhil bütün iş süreçleri belli kurallar çerçevesinde gerçekleşir. Bir işe ait algoritmanın tasarlanması dendiğinde işin alt adımlara bölünmesi ve adımlar arasındaki geçiş kurallarının belirlenmesini ifade ediyoruz. Örneğin diş fırçalamayı; fırçaya diş macunu sürülmesi, fırçanın 30 kez dişler üzerinde aşağı yukarı hareket ettirilmesi ve ağzın su ile çalkalanması adımlarından oluşturabiliriz. Algoritmada iş adımlarının uygulanma sırasının belirlenmesi en az iş adımlarının belirlenmesi kadar önemlidir.

Algoritmaların bilgisayar programı olarak yazılması farklı programlama biçimlerini destekleyen farklı diller ile olabilir. Farklı programlama biçiminden kasıt, dilin programcıya sunmuş olduğu yapılardır. Örneğin; C dilinin en büyük programlama birimi her biri bir algoritmaya karşılık gelen fonksiyonlar iken, C++ dilinde fonksiyon ve değişkenlerin bir arada tutulduğu sınıflar programcının daha büyük programlar oluşturabilmesine olanak sağlamaktadır. Bir problemin çözümünde kullanılacak dilin seçimi, işe uygunluk, programda kullanılacak üçüncü parti paketleri destekleme ve bireysel tercih gibi hususlara göre belirlenir.

Algoritmaların ifade edilmesi sözel olabileceği gibi sözde kod ve akış diyagramları ile de olabilmektedir. Bu bölümde algoritma, sözde kod ve akış diyagramları konuları anlatılacaktır.

ALGORİTMA

Sıradan bir günde yataktan kalkmak, el yüz yıkamak, diş fırçalamak gibi binlerce farklı sorunu hiç düşünmeden çözeriz. Bu tür sorunların çözümü çok az entelektüel çaba gerektirir ve göreceli olarak önemsizdir. Bununla birlikte daha fazla önem taşıyan bir sorunun çözümü genellikle problemi anlaşılabilir bir şekilde ifade etmeyi ve çözümü başkalarına iletmeyi içerir. Bilgisayarların problemin çözüm aracı olması durumunda, çözüm adımlarını açıkça belirten işlevlerin bilgisayara aktarılması gerekir. Bu türden bir problem çözümü bilgisayar bilimleri için önemli olan algoritmaların incelenmesini gerektirir.

Algoritma tasarlanırken çözüm yolunun birden fazla olabileceği düşünülmelidir.

Algoritma, bir problemin çözümü için net olarak tanımlanmış iş adımları listesidir. Algoritmanın her adımı programlama dilinde yazıldığında bilgisayar tarafından yürütülebilen açık bir talimat olmalıdır. Adımların sırası önemlidir; çünkü çoğu programlama dilinde işlemler programda bulundukları sıraya uygun olarak çalıştırılır. Çözülecek problem, iki sayının toplanması gibi basit veya sıkıştırılmış bir video dosyasının oynatılması gibi karmaşık da olsa çözüm adımlarının belirlenmesi için öncelikle sorun alanının analiz edilerek çözüme ait algoritmanın tasarlanması gerekir. Algoritma tasarımında,

- Nihai hedef nedir?
- Hedefe giden yolda hangi işlemlerin yapılması gerekir?

sorularının cevapları aranır.

Sorunu Anlamak

Algoritma tasarlanmadan önce sorunun bütün yönleri ile anlaşılması, sorunu gerçekten anlamak için ise soruna ait aşağıdaki soruların cevaplanması gerekir.

- Problemin girdileri neler?
- Çözüm çıktıları ne olacak?
- Talimatlar hangi sırayla gerçekleştirilecek?
- Problemde hangi kararların alınması gerekiyor?
- Çözümde tekrarlayan kısımlar var mı?

Yukarıda listelenen soruların cevaplanması sorunun anlaşılması ve çözümün ortaya konulması açısından önemlidir.

Bireysel Etkinlik

- Çay demlemeyi bilmeyen bir kişinin anlayacağı şekilde bir çay demleme algoritması oluşturun.

Algoritma ile Program Arasındaki İlişki

Algoritmalar bilgisayar programlarında fonksiyonlara karşılık gelir.

Algoritmaların bilgisayar programlarındaki karşılıkları genellikle fonksiyonlar olarak karşımıza çıkmaktadır. Fonksiyonlar daha büyük programlar içerisinde alt iş adımları olarak değerlendirilebilecek programlama birimleridir. Fonksiyonlar aynı işlevin tekrar tekrar yazılmasının engellenmesi için oluşturulabilecekleri gibi önceden geliştirilmiş olan fonksiyon kütüphaneleri de programcılar tarafından sıklıkla kullanılabilirler. Örneğin, bir görüntü işleme programında her biri farklı görüntü dosyası formatlarını işlemek için tasarlanmış bir algoritmaya karşılık gelen fonksiyon kütüphanesi geliştiricilerce kullanılabilir. Aynı şekilde liste veri yapısında tutulan verilerin sıralanması, yeni veri eklenmesi veya listeden veri silinmesi farklı algoritmalara karşılık gelmektedir. Liste, vektör ve harita veri yapıları C++ dilinde standart şablon yapıları olarak bilinir ve Ünite 12'de bu veri yapıları ile ilgili algoritmaların fonksiyon hallerinin nasıl kullanılacağı anlatılmıştır.

Hemen her işin birden fazla gerçekleştirilme yöntemi olup, yöntemlerin her biri farklı bir algoritmaya karşılık gelmektedir. Bilgisayar programları algoritmaların bir programlama dilinin kurallarına uygun olarak bilgisayara aktarılmış halleri olduğuna göre bir işin bilgisayarda gerçekleştirilme yolunun da birden fazla olabileceğini söyleyebiliriz. Bu nedenle, programcılar mümkün olan en verimli algoritmayı tasarlamaya çalışırlar. Elbette tüm algoritmalar en mükemmel şekilde tasarlanamaz. Bu nedenle, programcılar mevcut algoritmaları geliştirir ve gelecekteki program güncellemelerine dâhil ederler.

Çözüm:

Örnek

- Sıralı olmayan bir sayı listesindeki en büyük sayıyı bulacak algoritmayı geliştiriniz.

Sayılar gözümüzle kontrol edebileceğimiz kadar az (10-20) olsaydı bu problemin çözümünde bilgisayara ihtiyaç olmayabilirdi. Sayıların olduğu listeye bakıldığında karşılaştırma yaparak listedeki en büyük sayı belirlenebilirdi. Listede bir milyon sayı olduğunu varsayarsak aralarındaki en büyük değere sahip olan sayının belirlenmesi için kesinlikle bir algoritmaya ve onu bilgisayarda gerçekleyecek programa ihtiyaç duyulacağı açıktır.

Algoritmanın girdisi sayılar listesi olup, listeyi L olarak adlandıralım. Listedeki ilk sayı L1 olsun. Algoritmanın çıktısı listedeki en büyük değere sahip olan sayıdır. Algoritma çıktısını En Büyük olarak adlandıralım. Algoritma girdisini ve çıktısını belirlediğimize göre artık algoritmanın çalışma adımlarını tespit etmemiz gerekir.

Sayı listesindeki en büyük sayının bulunması algoritmasının adımları aşağıda verilmiştir.

Adım 1: En Büyük = L1

Birinci adımda listedeki L1'i En Büyük olarak atayalım.

Adım 2: Listedeki elemanları sırayla En Büyük'le karşılaştır.

Bu, bütün listeyi ardışık olarak tarayacağımız anlamına gelir.

Adım 3: Eğer bir eleman En Büyük'ten büyükse:

En Büyük'ten büyük sayı bulunursa 4. adıma geç. En Büyük'ten küçükse bir sonraki elemana geç.

Adım 4: En Büyük = eleman

Bulunan yeni büyük sayının değerini En Büyük'e ata. Yeni En Büyük değer ile listede sayı kalmayıncaya kadar ikinci adıma dön.

Algoritma tasarlarken iş adımlarının sırası önemlidir.

Step 5: En Büyük belirlenir.

Listenin bütün elemanları En Büyük'le karşılaştırılınca En Büyük değer bulunmuş olur.

Algoritmanın anlaşılır şekilde bir dizi mantıksal adım olarak tanımlandığına dikkat ediniz. Bir bilgisayarın bu talimatları anlayıp kullanabilmesi için bu adımların bir programlama dili ile yazılması ve bilgisayarda çalıştırabilmesi için derleyici ile derlenerek makine diline çevrilmesi gerekir.

SÖZDE KOD

Sözde kod, doğal dil ile programlama dillerinin bileşkesi kullanılarak algoritmaların sunulma tekniğidir. Algoritmanın, programcı ve diğer ilgili kişilere anlatılmasını sağlamaya yönelik bir çalışmadır. Programcının gerçek kodlama söz dizimini takip etmeden algoritmayı test etmesine izin verir. Aslında bir bilgisayarda çalışmamasına rağmen bir programlama diline kolayca aktarılabilir. Sözde kod için çok katı kurallar listesi olmasa da sözde kodlamada yaygın olarak kullanılan deyimler gruplanarak aşağıda verilmiştir.

Sözde kod yazılırken algoritmanın ileride bilgisayara aktarılacağı programlama dilinin anahtar kelimeleri kullanılır. Bundan dolayı özellikle döngü yapıları ve alt işlemleri ifade eden sözde kod deyimleri farklı çalışmalarda değişiklikler gösterir.

Aritmetik işlemlerde: +, -, *, / ,%, <,>,>=,<=, =,==, !=

Girdi: READ, OBTAIN, GET

Çıktı: PRINT, DISPLAY, SHOW

Hesaplama: COMPUTE, CALCULATE, DETERMINE

Başlatma: SET, INIT

Seçim: IF - THEN - ELSE - ENDIF

Döngü: WHILE - ENDWHILE, DO - WHILE, REPEAT - UNTIL

Aritmetik işlemlerde, matematikteki dört işlem (+, -,*, /), bölümden kalan (mod %) ve karşılaştırma operatörleri kullanılabilir.

Sözde kod yazarken program yazdığınız dilin anahtar kelimelerini kullanın.

Örnek

- Bir öğrencinin notu 50'ye eşitse veya büyükse bilgisayar ekranında "Geçti", 50'den küçükse "Kaldı" yazacak algoritmanın sözde kodunu yazınız.

Çözüm:

```
IF ogrencininNotu >= 50 THEN
  PRINT “Geçti”
ELSE
  PRINT “Kaldı”
ENDIF
```

Bu örnekte seçim ifadesi olan IF – THEN – ELSE – ENDIF yapısı kullanılmıştır. Bu yapıda IF kelimesinden sonra cevabı yalnızca doğru veya yanlış olacak bir karşılaştırma ifadesi vardır. Karşılaştırma sonucu doğru ise THEN – ELSE arasındaki işlemler, yanlış ise ELSE – ENDIF arasındaki işlemler yapılır.

IF yapısı karşılaştırma sonucuna bağlı olarak hangi işin yapılacağına karar verir. Karşılaştırma aynı türdeki iki büyüklüğün eşitliğinin veya eşitsizliğinin kontrol edilmesidir. Karşılaştırma yaparken matematikteki karşılaştırma işaretleri kullanılabilir. IF'in genel yapısı,

```
IF Karşılaştırma THEN
  Karşılaştırma sonucu “doğru” olduğunda yapılacak işlemler
ELSE
  Karşılaştırma sonucu “yanlış” olduğunda yapılacak işlemler
ENDIF
```

Örnek

- 10 öğrencinin notlarını okuyup ortalamasını hesaplayacak algoritmanın sözde kodunu yazınız.

Çözüm:

```
SET toplam = 0
SET sayac = 1
WHILE sayac <= 10
    INPUT  ogrencininNotu
```

Cevabını bildiğiniz soruların algoritmalarını tasarlayın.

COMPUTE toplam=toplam + ogrencininNotu

COMPUTE sayac = sayac + 1

ENDWHILE

SET ortalama = toplam / 10

PRINT ortalama

Bu örnekte "10 öğrencinin notunun girilmesi ve girilen notların toplanması" işlemi bir döngü ifadesi ile gerçekleştirilmiştir. Bir döngü ifadesi olan WHILE – ENDWHILE genel yapısı aşağıda verilmiştir.

Karşılaştırma ifadesinin ilk değeri

WHILE Karşılaştırma

Karşılaştırma sonucu "doğru" olduğu sürece yapılacak işlemler

Karşılaştırma ifadesinin değerinin değiştirilmesi

ENDWHILE

Bireysel Etkinlik

- 1'den 100'e kadar olan tam sayıların toplamını verecek algoritmayı tasarlayarak sözde kodunu yazınız.

Bireysel Etkinlik

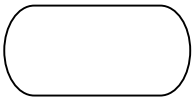
- Öğrencilerin sınavdan almış oldukları not 50 - 75 arasında ise "İyi", 75 - 100 arasında ise "Çok iyi" ve 50'nin altındaysa "Daha Fazla Çalışmalı" ifadelerini ekrana yazacak algoritmanın sözde kodunu oluşturun.

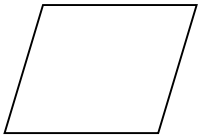
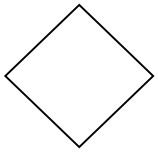
Akış diyagramlarında; Başla, Bitir, Girdi, Çıktı, Karar ve İşlem sembolleri kullanılır.

AKIŞ DİYAGRAMLARI

Bir problemin çözümüne yönelik geliştirilen algoritmanın görsel sembollerle ifade edilmiş şekline akış diyagramı denir. Akış diyagramları program yazarken sürecin izlenmesi ve başkalarına algoritmaları açıklarken yardımcı olur. Akış diyagramları sözde kodların görselleştirilmiş halleri olarak düşünülebilir.

Tablo 2.1 Akış diyagramı oluşturulurken yaygın olarak kullanılan semboller

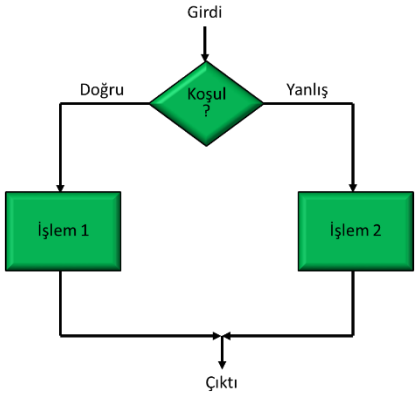
Sembol	Açıklaması
	Akış yönü: Diyagramda şekiller arasındaki akışı gösterirler. Ok yönü akış yönünü ifade eder.
	**Başla/Bitir:** Programın başladığı ve bittiği konumu gösterir. Her programda başla ve dur diyagramları mutlaka olmalıdır.

	**Girdi/Çıktı:** Çevre birimleri ile yapılan bilgi alışverişini simgeler. Yapılacak işlem şekil içerisine yazılır.
	**İşlem/Atama:** Değişkenlere değer atamaları ve matematiksel veya dizgisel işlemlerin yapıldığı aşamalarda kullanılır. İşlemler bu şekil içerisine kısa olarak yazılır.
	**Karşılaştırma ve karar:** Karşılaştırma işlemi ve sonuçta varılan karar durumuna göre akış yönünü belirleyen işlemlerde kullanılır. Kıyaslama ifadesi şekil içine yazılır, karar E (evet) veya H (hayır) simgesi ile belirtilen bir uçtan çıkan akış ile başka bir diyagrama gider.
	**Alt süreç:** Bir işin tamamlanması için alt süreçler ve uygulamalar varsa bu süreçleri simgeler. Sürecin kendisi değil ancak tanımı şekil içine yazılır.
	**Belge:** Basılı belge veya okunabilir ekran çıktısının ismi şekil içerisine yazılır. Çoklu belge çıktısı için birden fazla şekil kullanılır.

Algoritma tasarımında işlem adımları kadar işlem sırası da önemli olup, algoritmaların şekilsel gösterimi olan akış diyagramlarında ok ile gösterilen akış yönüne dikkat edilmelidir. Akış yönüne göre kavşakların olduğu yerler karar verme noktalarıdır. Algoritmalarda sıklıkla karşılaşılan karar yapılarının sözde kod ve akış diyagramlarının anlaşılması algoritma tasarımı açısından çok önemlidir.

Karşılaştırma sonucunun doğru olduğu durumda işlem 1, yanlış olduğu durumda işlem 2'yi yapacak algoritmanın sözde kodu ve akış diyagramı aşağıda verilmiştir.

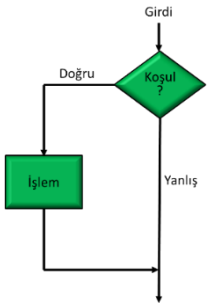
```
SET Girdi
IF Koşul THEN
  COMPUTE işlem 1
ELSE
  COMPUTE işlem 2
ENDIF
```



Akış diyagramlarında ilerleme yönü ok ile gösterilir.

Karşılaştırma sonucunun doğru olduğu durumda işlem yapılan sözde kod ve akış diyagramı aşağıda verilmiştir.

```
SET Girdi
IF Koşul THEN
  COMPUTE işlem
ENDIF
```



Örnek

- Genel ortalaması 60'a eşit veya daha yüksek olan öğrenciler bir dersten 50 ile 60 arasında not alırlarsa kalırlar. Eğer 60'a eşit veya daha yüksek not alırlarsa genel ortalamaya bakılmaksızın dersten geçerler.
- Yukarıda verilen ifadeye ait algoritmayı sözde kod ve akış diyagramı ile tasarlayınız.

Çözüm:

Sorudaki koşulları tabloya aktarırsak, ogNot >= 60 ve ogNot <50 durumlarında ortalamanın kontrol edilmediğini rahatlıkla görürüz.

ogNot >= 60		Geçti
ogNot < 50		Kaldı
50 <= ogNot < 60	ortalama >= 60	Geçti
	ortalama < 60	Kaldı

Algoritmalar, akış diyagramları veya sözde kodlar üzerinden farklı girdi değerleri için test edilmelidir.

```
SET ortalama
SET ogNot
IF ogNot  >=  60 THEN
  PRINT “Geçti”
ELSE
  IF ogNot >=50 THEN
    IF ortalama >= 60 THEN
      PRINT “Öğrencinin notu 50 ile 60 arasında, ortalaması 60’dan büyük, Geçti ”
    ELSE
      PRINT “Öğrencinin notu 50 ile 60 arasında, ortalaması 60’dan küçük, Kaldı ”
    ENDIF
  ELSE
    PRINT “Öğrencinin notu 50’den küçük, Kaldı”
  ENDIF
ENDIF
```

Algoritmalar, akış diyagramları veya sözde kodlar üzerinden farklı girdi değerleri için test edilmelidir.

Algoritma Testi

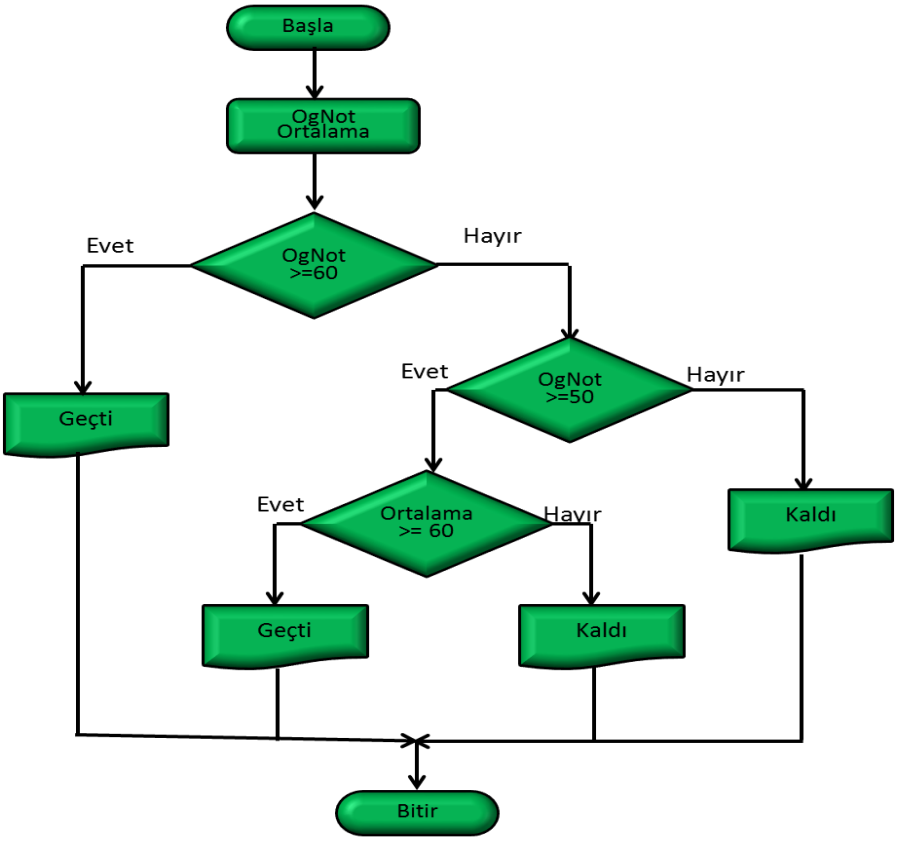
Akış diyagramı üzerinden algoritma testi,
ogNot = 65, ortalama=45 durumu;

ogNot >= 60 → Evet → Geçti

ogNot = 45, ortalama=90 durumu;
ogNot >= 60 → Hayır → ogNot >=50 →Hayır →Kaldı

ogNot = 55, ortalama=60 durumu;
ogNot >= 60 → Hayır → ogNot >=50 →Evet →ortalama >= 60 →Evet →Geçti

ogNot = 55, ortalama=55 durumu;
ogNot >= 60 → Hayır → ogNot >=50 →Evet →ortalama >= 60 →Hayır→Geçti



Algoritmayı tasarlamadan program yazmaya çalışmayın.

Çözüm:

Sorudaki koşulları tabloya aktarırsak,

Örnek

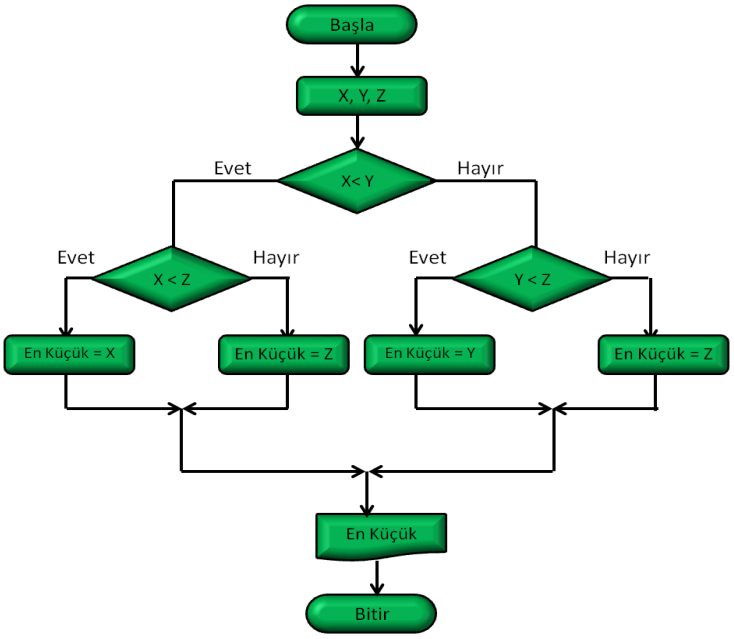
•X, Y ve Z sembolleri ile verilen 3 sayı arasındaki en küçük sayıyı bulacak algoritmaya ait sözde kodu yazınız ve akış diyagramını çiziniz.

X < Y	X < Z	En küçük X
		En küçük Z
	Y < Z	En küçük Y
		En küçük Z

İlişki tablosu verilen algoritmanın sözde kodu;

```
IF X < Y THEN
    IF X < Z THEN
        En Küçük  = X
    ELSE
       En Küçük  = Z
    ENDIF
ELSE
    IF Y < Z THEN
        En Küçük  = Y
    ELSE
        En Küçük  = Z
   ENDIF
ENDIF

PRINT En Küçük
```

X, Y, Z sayılarının en küçüğünü bulan algoritmanın akış diyagramı aşağıda verilmiştir.

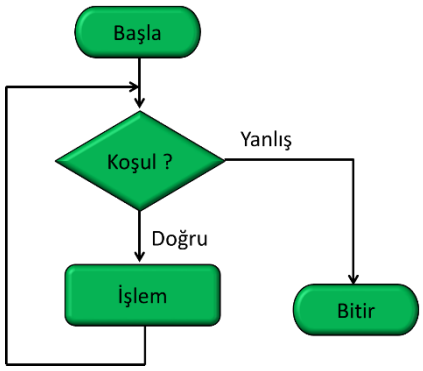


Sözde kodların bir programlama diline aktarılması çok kolaydır.

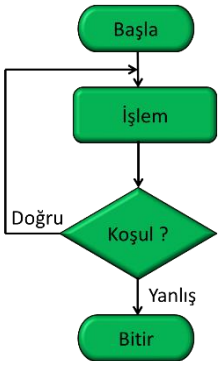
Akış diyagramları, algoritmaların görsel sunumudur.

Akış diyagramlarında diğer bir karışık yapı ise döngülerdir. Sırasıyla WHILE – ENDWHILE ve DO – WHILE döngülerinin sözde kodları ile birlikte akış diyagramları verilmiştir.

```
WHILE Koşul
  COMPUTE İşlem (Koşul doğru)
ENDWHILE
```




```
DO
  COMPUTE İşlem (Koşul doğru)
WHILE Koşul
```



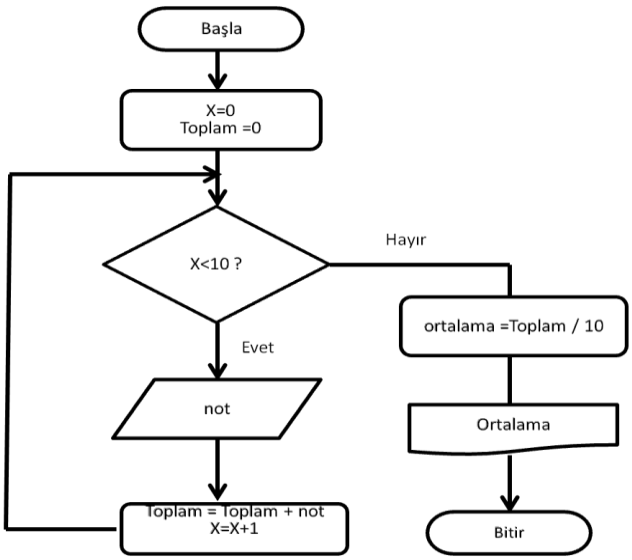
DO – WHILE yapısının koşulu işlem bir kez yapıldıktan sonra kontrol etmesi nedeniyle, işlem en az bir kez yapılacaktır.

Örnek

- 10 öğrencinin notunu okuyup ortalamasını hesaplayacak bir sistemin algoritmasının akış diyagramını çiziniz.

Akış diyagramlarını karar noktalarında farklı yönlere gidecek şekilde test ediniz.

Çözüm:



Örnek

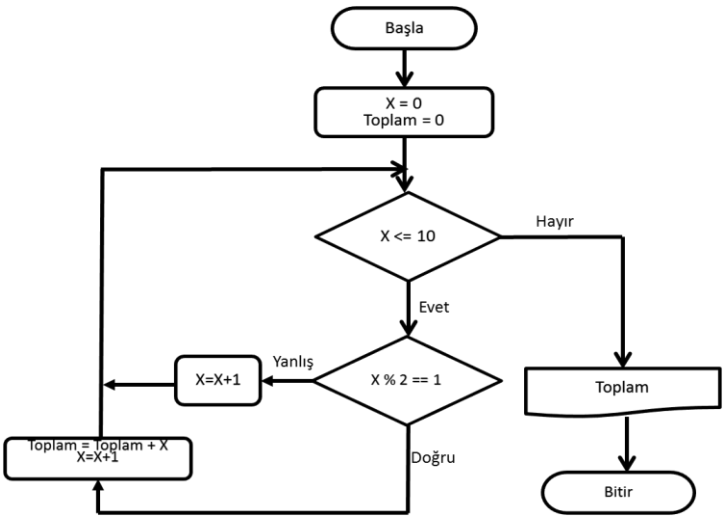
•0 ile 10 arasındaki tek sayıların toplamını verecek algoritmayı tasarlayarak sözde kodu yazınız ve akış diyagramını çiziniz.

Çözüm:

0 ile 10 arasındaki tek sayılar 1, 3, 5, 7 ve 9'dur. Tek sayıların ortak özellikleri 2'ye bölümlerinden kalanın 1 olmasıdır. İki sayının bölümünden kalan işlem mod işlemi olarak bilinir. C++ dilinde mod operatörü % işaretidir. İki büyüklük arasındaki eşitlik ise == işareti ile test edilir.

Algoritma girdisi 0 ile 10 arasındaki tam sayı dizisidir. Çıktının dizi elemanlarından tek olanlarının toplamı olmasını istiyoruz. Başlangıçta çıktımızı dizinin ilk elemanı olan 0 alalım. Sonra sıra ile dizi elemanlarının tekliğini kontrol edip tek olanları var olan Toplama ekleyelim (Toplam = Toplam + tek eleman).

```
SET X =0, SET Toplam = 0
WHILE X <= 10
    IF X % 2 == 1 THEN
       COMPUTE Toplam = Toplam + X
        X = X + 1
    ELSE
        X = X + 1
    ENDIF
ENDWHILE
PRINT Toplam
```



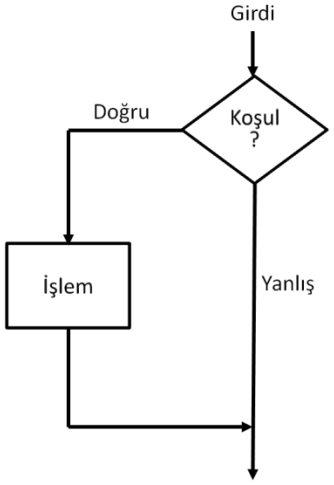
Bireysel Etkinlik

• 10 ile 100 arasındaki tam sayılardan 3'e tam olarak bölünenlerin toplamını bulacak algoritmanın akış diyagramını çiziniz.

Özet

- Algoritma bir problemi çözmek için sınırlı sayıda ve net tanımlanmış adımdan oluşan talimatlar listesidir. Algoritmanın her adımı bir programlama dilinde yazıldığında bilgisayar tarafından yürütülebilen açık bir talimat olmalıdır. Algoritma tasarımında;
- Büyük resim - Nihai hedef nedir?
- Alt aşamalar - Hedefe giden yolda hangi işlemlerin yapılması gerekir? sorularının cevapları aranır.
  Algoritması tasarlanacak sorunu anlamak için cevaplanması gereken temel sorular:
  - Problemin girdileri neler?
  - Problemin çözüm çıktıları ne olacak?
  - Talimatların hangi sırayla gerçekleştirilmesi gerekiyor?
  - Problemde hangi kararların alınması gerekiyor?
  - Çözümde tekrarlayan alanlar var mı?
- Algoritmaların bilgisayar programlarındaki karşılıkları genellikle fonksiyonlar olarak karşımıza çıkmaktadır. Fonksiyonlar daha büyük programlar içerisinde alt iş adımları olarak değerlendirilebilecek programlama birimleridir.
- Algoritmaların başkalarına aktarılması sözde kod ve akış diyagramları ile olmaktadır.
- Sözde kod, doğal dil ile programlama dillerinin bileşkesi kullanılarak algoritmaların sunulma tekniğidir. Algoritmanın geliştirici ve diğer ilgili kişilere anlatılmasını sağlamaya yönelik bir çalışmadır. Programcının gerçek kodlama söz dizimini takip etmeden algoritmayı test etmesine izin verir. Aslında bir bilgisayarda çalışmamasına rağmen bir programlama diline kolayca aktarılabilir. Sözde kodlamada kullanılan anahtar kelimelere örnek olarak SET, INPUT, IF-THEN-ELSE ve WHILE-ENDWHILE verilebilir.
- Bir problemin çözümüne yönelik geliştirilen algoritmanın görsel sembollerle ifade edilmiş şekline akış diyagramı denir. Akış diyagramları, program yazarken sürecin izlenmesine ve başkalarına algoritmayı ve programı açıklarken yardımcı olur. Akış diyagramları, sözde kodların görselleştirilmiş halleri olarak düşünülebilir. Akış diyagramlarında ok işareti akış yönünü, dikdörtgen şekli hesaplamayı, baklava şekli ise karar vermeyi ifade etmektedir.

DEĞERLENDİRME SORULARI

1. 

Yukarıdaki C++ programı çalıştırıldığında aşağıdaki ekran çıktılarından hangisi oluşur?

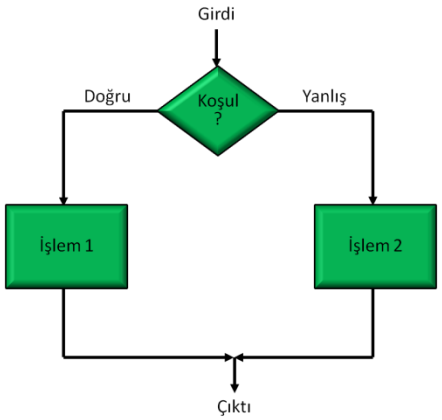
a) IF Koşul THEN İşlem ENDIF
b) IF Koşul ELSE İşlem ENDIF
c) WHILE Koşul ELSE İşlem ENDWHILE
d) WHILE Koşul THEN İşlem ENDWHILE
e) DO İşlem WHILE Koşul

2. Algoritma için aşağıdakilerden hangisi söylenemez?
   a) Algoritma iş adımlarının sıralı listesidir.
   b) Algoritmalar sözde kodlarla ifade edilebilirler.
   c) Algoritmalar akış diyagramları ile gösterilebilirler.
   d) Algoritmalar bilgisayar programlarında fonksiyonlarla ifade edilebilirler.
   e) Algoritmada iş adımlarının sırası önemsizdir.

3. Bir akış diyagramında aşağıdakilerden hangisi mutlaka olmalıdır?
   a) İşlem
   b) Karşılaştırma
   c) Döngü
   d) Girdi
   e) Başla

4. Akış diyagramlarında girdi ve çıktılar aşağıdakilerden hangisi ile gösterilebilir?
   a) Kare
   b) Baklava
   c) Paralel kenar
   d) Daire
   e) Elips

5.



Yukarıda verilen şeklin sözde kod karşılığı aşağıdakilerden hangisidir?

a) IF Koşul THEN İşlem 1 ELSE İşlem 2 ENDIF
b) IF Koşul ELSE İşlem ENDIF
c) WHILE Koşul ELSE İşlem ENDWHILE
d) WHILE Koşul THEN İşlem ENDWHILE
e) DO İşlem WHILE Koşul

6.
```
READ yaş
IF ? THEN
        DISPLAY “Genç”
ENDIF
```

Yukarıda verilen sözde koddaki ? yerine aşağıdakilerden hangisi yazılmalıdır?

a) yas>=13 ve yas <=19
b) yas < 19
c) yas > 19
d) yas<13 ve yas > 19
e) yas <13 veya yas > 19

7. SET x=1, y=10
   DO
   COMPUTE x = x + 2
   COMPUTE y = y – 1
   PRINT X, Y
   WHILE x < y

   Yukarıdaki sözde kodunun çıktısı aşağıdakilerden hangisidir?

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

8. SET toplam = 0
   SET sayac = 0
   WHILE sayac < 4
   toplam = toplam + sayac*2
   sayac = sayac+1
   ENDWHILE
   DISPLAY toplam

   Yukarıda verilen sözde kodun çıktısı aşağıdakilerden hangisidir?

   a) 0
   b) 4
   c) 6
   d) 8
   e) 12

9. Akış diyagramlarında baklava şekli neyi temsil eder?
   a) Başla
   b) Bitir
   c) Karar
   d) Girdi
   e) Hesaplama

10. Algoritmaların grafiksel gösterimi aşağıdakilerden hangisi ile yapılabilir?
    a) Sözde kod
    b) İlişki grafikleri
    c) Akış diyagramları
    d) Çizgi diyagramları
    e) Algoritma

Cevap Anahtarı

1.a, 2.e, 3.e, 4.c, 5.a, 6.a, 7.a, 8.e, 9.c

YARARLANILAN KAYNAKLAR

[1] Cevizci, A. (2002). Etiğe Giriş, (1. Baskı), İstanbul: Paradigma Yayıncılık.

(2]Cevizci, A. (2014). Etik, Ahlak Felsefesi, (1. Baskı), İstanbul: Say Yayıncılık.

[3] Maclntyre, A. (2001). Ethik'in Kısa Tarihi, (1.Baskı),çev. Ahmet Cevizci, İstanbul: Paradigma Yayınları.

[4]Hazlitt, H. (2006). Ahlakın Temelleri, (1. Baskı) çev. Mehmet Aydın, Recep Tapramaz, Ankara: Liberte Yayınları.

[5]Cevizci, A. (2009). Felsefe Tarihi, (1. Baskı), İstanbul: Say Yayıncılık.

[6] Gökberk, M. (2012). Felsefe Tarihi, (24. Baskı), Ankara: Remzi Kitabevi.

İLK C++ PROGRAMI (MERHABA DÜNYA)

Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

Dr. Yılmaz AR

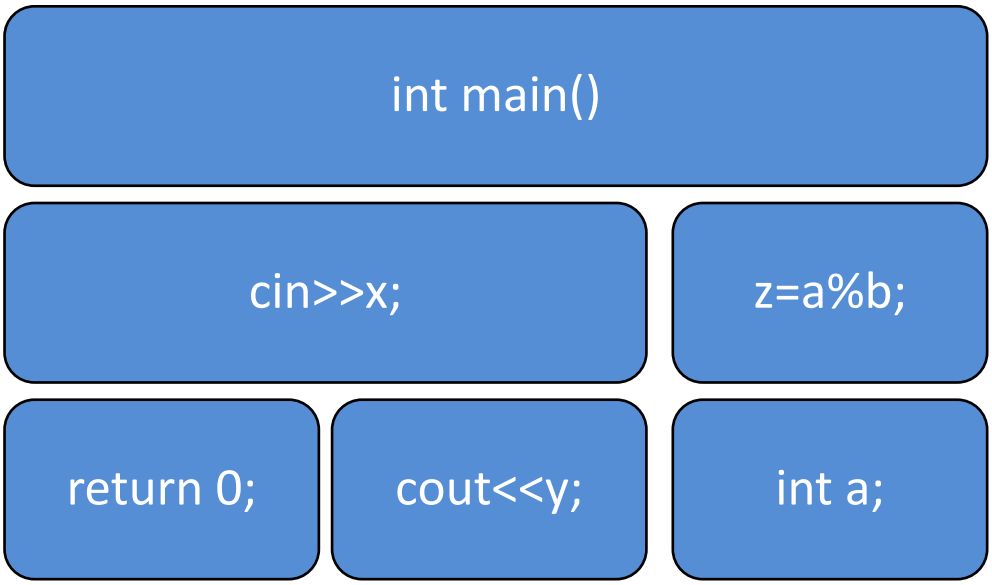
İÇİNDEKİLER

- İlk C++ Programı
  - Kaçış Karakterleri
- İki Tam Sayıyı Toplayan C++ Programı
  - Bellek Kavramları
- Aritmetik İşlemler

HEDEFLER

- Bu üniteyi çalıştıktan sonra;
  - Ekrana karakter dizisi basan C++ programları yazabilecek,
  - Ekrana yazdırılacak karakter dizisindeki kaçış karakterlerini kullanabilecek,
  - Kullanıcıdan aldığı veriyi ilgili değişkenlere aktarabilen C++ programları oluşturabilecek,
  - Aritmetik işlemler yapabilen C++ programları yazabileceksiniz.

ÜNİTE
3


int main()
cin>>x;
z=a%b;
return 0;
cout<<y;
int a;

GİRİŞ

Bu ünite kapsamında C++ ile programlama için geçerli bazı temel kavramlar basit programlar yardımıyla açıklanacaktır. İlk örnek program ekrana *Merhaba Dunya* cümlesini yazdıran programdır. Bu örnek kullanılarak bir C++ programının temel özellikleri anlatılacaktır. Ekrana sadece bir cümlelik bir mesaj yazdıran bu program kullanıcıdan herhangi bir girdi istememekte, ayrıca matematiksel bir hesaplama da içermemektedir. Bu ilk örnek bu tür farklı işlemlere sahip olmamasına rağmen C++ programlarının temel özelliklerini göstermesi açısından önemlidir. Bu program kullanılarak bir C++ programında yer alması zorunlu *main* fonksiyonu anlatılacaktır. Ayrıca basit bir programda karşımıza çıkabilecek, ekrana bilgi yazdırmak için kullanılan akış yerleştirme operatörü, açıklama satırları ve ön işlemciye yönelik bilgilendirme komutları gösterilecektir. Her C++ ifadesinin noktalı virgül ile bitmesi gibi bazı söz dizimi kurallarına da bu örnek kapsamında yer verilecektir. Ayrıca bu kitapta yer alan programlarda kullanılan deve ve paskal notasyonları anlatılacaktır. İlk örnekten sonra kaçış karakterleri gösterilecektir. İkinci örnek program klavyeden girilen iki tam sayının toplamını bulan ve ekrana yazdıran bir C++ programıdır. Bu program ile aynı zamanda değişken tanımlama ve kullanıcıdan değer alma gibi ilk programda olmayan özelliklere de değinilecektir. Bu programda aynı zamanda matematiksel bir işlem de yer almaktadır. Bu işlem de detaylandırılacak ve ilgili her bir programın çalıştırılması sonrasında elde edilen çıktılara yer verilecektir.

Birçok programlama dilinde ilk örnek olarak ekrana Merhaba Dunya yazdırılır.

Örnek programlardan sonra C++ ile programlamada basit bellek kullanımı açıklanacaktır. Tanımlanan değişkenler için bellekte yer ayrılması konusu işlenecektir. Sonrasında basit aritmetik operatörler tanıtılacaktır. Toplama, çıkarma, çarpma, bölme ve modül operatörlerinin C++ programlarında kullanımları bir tablo üzerinde gösterilecektir. Aynı deyim içerisinde yer alan farklı operatörlerin çalışma sırasını belirleyen kurallar üzerinde (işlem önceliği) durulacaktır. Parantez kullanılarak istenilen işlemlerin öncelikli olarak çalışması gösterilecektir. Daha sonra kullanıcıdan bir öğrencinin final notunu, ara sınav notunu ve ödev notunu alan ve dönem sonu ortalama notunu hesaplayan bir C++ programına yer verilecektir. Bu program aritmetik işlemlerin bir C++ programında kullanımına bir örnek olacaktır. Ayrıca kullanıcıdan bir dikdörtgenin kısa ve uzun kenarını alan, sonrasında bu dikdörtgenin alanını-çevresini hesaplayan ve sonucu ekrana yazdıran bir C++ programı da bu ünite kapsamında kendine yer bulacaktır.

main() fonksiyonu her C++ programında olması zorunlu tek fonksiyondur.

Bu bölümde yer alan programlar yapısal programlama yaklaşımının en basit yapı taşı olan ardışık çalışma şeklinde oluşturulacaktır. Programları oluşturan ifadeler yazıldıkları sıra ile çalıştırılırlar. Karar verme ve aynı ifadeyi tekrar tekrar çalıştıran döngü yaklaşımları ilerleyen bölümlerde gösterilecektir. Bu bölümdeki programlar C++ dilinin temel öğelerini göstermek amacıyla yazıldıkları için, mümkün olduğunca basit olacaklardır. C++ dilinin sahip olduğu operatörler bu bölümde verilenler ile sınırlı değildir. İlerleyen bölümlerde sık kullanılan diğer C++ operatörlerinin tamamına değinilecektir.

İLK C++ PROGRAMI

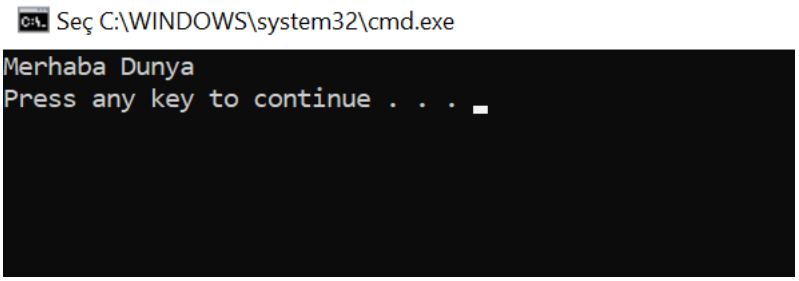
Aşağıda verilen program ekrana *Merhaba Dünya* yazdıran bir C++ programıdır. Şekil 3.1'de ise bu program çalıştırıldığında elde edilen çıktı ekranı verilmektedir.

Örnek

•Ekrana "Merhaba Dunya" yazdıran bir C++ programı yazınız.

Çözüm:

```
// Ilk C++ Programi
// IlkProgram.cpp
#include <iostream>
using namespace std;

int main()
{
	cout<<"Merhaba Dunya\n";

	return 0;
}
```



Şekil 3.1. İlk C++ Programının Ekran Çıktısı

Bu kitaptaki tüm programlarda deve ve paskal notasyonu kullanılmıştır.

Bu ünitede ve kitabın tamamında yer alan tüm programlarda deve ve paskal notasyonu kullanılmıştır (Deve notasyonundan değişken isimlendirmelerinde; paskal notasyonundan ise sınıf, dosya ve fonksiyon isimlendirmelerinde faydalanılmıştır.). Deve notasyonunda eğer değişken ismi bir kelimeden fazla kelime içeriyorsa ilk kelimenin tüm harfleri küçük harf, takip eden diğer kelimelerin ilk harfleri büyük diğer harfleri küçük harf olacaktır. Örneğin, bir öğrencinin numarası için belirlenen değişken *ogrenciNumarasi* olabilir. *toplamSatis* değişkeni de başka bir programda aylık gerçekleşen toplam satış miktarını gösterebilir. Paskal notasyonunda ise dosya, fonksiyon ve sınıf isimlerini oluşturan tüm kelimelerin ilk harfleri büyük diğer harfleri küçük harf olarak

kullanılır. *IlkProgram.cpp* bu bölümde yazılan C++ programının ismi olarak kullanılabilir. *OrtalamaNotHesapla* ismi ise not ortalamaları hesaplayan bir programda bu işlemi gerçekleştiren fonksiyonun ismi olarak tasarlanabilir.

// Ilk C++ Programi satırı açıklama satırıdır. İki adet '/' karakteri ile başlayan satırlar açıklama satırı olup, derleyici tarafından derleme sürecinde dikkate alınmazlar (Açıklama satırlarının derleme öncesinde gerçekleştirilen önişleme adımı esnasında devre dışı bırakıldıklarından 1. Ünite'de bahsedildiğini hatırlayınız.). Başka bir deyişle programın çalışmasına herhangi bir etkide bulunmazlar. Sadece programın kullanıcılarını bilgilendirmek amaçlı yazılan kısımlardır. Açıklama satırları ile programların okunabilirliği arttırılır. Kullanıcılar programı daha kolay bir şekilde okuyup anlayabilirler. Örnekte verildiği gibi bir programda sadece programı tanımlayan bir açıklama satırı olabileceği gibi programın farklı kısımlarını tanımlayan birden fazla açıklama satırı da olabilir. Ayrıca '*/**' ile başlayan ve '**/*' ile biten çok satırlı açıklama alanları da vardır. Eğer yazacağınız açıklama bir satırdan daha uzun ise bu şekilde çok satırlı açıklama alanları kullanılabilir. Örneğin, ilk C++ programında yer alan ilk açıklama satırı şu şekilde değiştirilebilir:

/ Bu örnekte verilen C++ programı bir programcinin bu dilde yazabilecegi ilk programlardan birisidir.*/*

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

#include <iostream> satırı önişlemciye bir bilgilendirme mesajıdır. # karakteri ile başlayan satırlar C++ programı derlenmeden önce ön işlemci tarafından çalıştırılırlar. Bu satırların sonuna C++ deyimlerinin sonunda muhakkak kullanılan *';'* karakteri eklenmez. Bu örnekte verilen ve ön işlemci tarafından içeriğinin eklenmesi istenen *iostream* başlık dosyası C++ programlarında girdi/çıktı, başka bir deyişle okuma/yazma işlemleri için gereklidir. Okuma işlemi varsayılan olarak klavye, yazma işlemi de varsayılan olarak ekran üzerinden gerçekleştirilir. Daha sonraki satırda yazılan *using namespace std;* satırı programda kullanılan *cout* gibi fonksiyonların önüne herhangi bir şey yazılmazsa ilgili fonksiyonun *std* isim uzayında olduğunu belirtir.

int main() satırı her C++ programının bir parçasıdır. *main* anahtar kelimesinden sonra yazılan () karakterleri *main*'in bir fonksiyon olduğunu gösterir. C++ programları bir veya birden fazla fonksiyon içerir ve bu fonksiyonlardan birisi *main* olmak zorundadır. C++ programları çalışmaya *main* fonksiyonunun ilk satırı ile başlar. Programda verilen ilk fonksiyon *main* olmayabilir fakat bu çalışma sırasını değiştirmez. Programın çalışması *main* fonksiyonunun ilk satırından başlar ve herhangi bir fonksiyon çağrıldığında program çalışma sırası ilgili fonksiyona geçer. *main* anahtar kelimesinin sol tarafında yer alan *int* anahtar kelimesi *main* fonksiyonunun dönüş tipini gösterir. *int* tam sayıları gösteren veri tipidir. *main* fonksiyonu başarılı bir şekilde çalışmasını tamamladığında geriye bir tam sayı çevirecektir. Bir sonraki satırda yer alan sol ayraç *{* her fonksiyonda gereklidir ve

Bir fonksiyonda dönüş veri tipi belirlenmişse, fonksiyon içerisinde bu tipte bir sonuç çevirmek zorunludur.

fonksiyon gövdesinin başladığını gösterir. Karşılık gelen sağ ayraç } ise fonksiyon gövdesinin bittiğini gösterir. Her iki ayraç da her fonksiyonda mutlaka kullanılmak zorundadır.

cout<<"Merhaba Dunya\n"; satırı çift tırnak işaretleri arasında yer alan karakter dizisini (string) ekrana yazdırmak için kullanılır. Tüm satır çalıştırılabilir ifade olarak isimlendirilir ve noktalı virgül karakteriyle sonlanmak zorundadır. << operatörü akış yerleştirme operatörü olarak isimlendirilir ve operatörün sağ tarafında kalan değerler çıktı akışına eklenir. *cout* nesnesi ise << operatörünün sol işlenen değeridir ve varsayılan olarak ekrana bağlıdır. *\n* bir sonraki alt bölümde anlatılacak kaçış dizilerinden birisidir ve göstergeyi (imleci) bir sonraki satırın başlangıç noktasına getirir. Dolayısıyla *cout<<"Merhaba\nDunya\n";* deyimi

Merhaba
Dunya
şeklinde bir çıktı üretecektir.

return 0; ifadesi programın başarılı bir şekilde sonlandığını gösterir. *main* fonksiyonunun dönüş tipi tam sayı olarak belirlendiği için *return* anahtar kelimesinden sonra 0 değeri dönüş değeri olarak verilmiştir. 0 yerine farklı bir tam sayı değeri de kullanılabilir. *main* fonksiyonunda geriye döndürülen değer programın başarılı bir şekilde tamamlandığını gösterir ve başka bir amaç için kullanılmaz. Fakat diğer fonksiyonlarda döndürülen değerler *main* programda fonksiyonun çağrıldığı noktaya yönlendirilirler ve bir amaç için kullanılabilirler.

Kaçış Karakterleri

Bir karakter dizisinde '\' veya '%' karakterlerini tek başına kullanırsanız söz dizimi hatasıyla karşılaşırsınız.

Ekrana yazdırmak istediğimiz karakter dizisi içerisinde yer alan \ kaçış karakteridir ve kendisinden sonra gelen karakterle beraber kaçış dizisini oluşturur. *\n* kaçış dizisi göstergeyi bir sonraki satırın başına getirir. *\t* kaçış dizisi göstergeyi bir tab boyutu sonrasına götürür. *\a* sistem çanını çalar. \\ ise ekrana \ karakterini yazdırmak istediğimizde kullanılır. Doğrudan \ karakterini ekrana yazdırmamız mümkün değildir. Derleyici \ karakterini gördüğünde sonraki karakterin özel karakterlerden birisi olmasını bekler (n, t, a vb.). Eğer \ karakteri tek başına görülürse derleyici söz dizimi hatası üretir. Aynı şekilde " karakteri de C++ akış yerleştirme işleminde özel bir karakterdir ve bu karakteri ekrana yazdırmak için \" kaçış dizisi kullanılır.

İKİ TAM SAYIYI TOPLAYAN C++ PROGRAMI

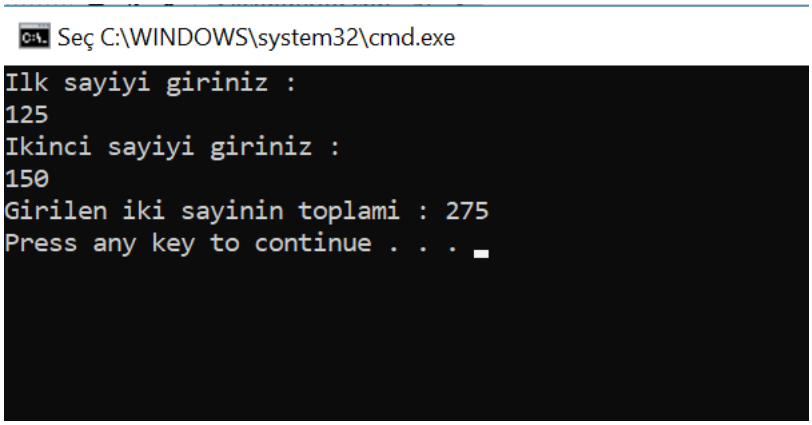
Aşağıda verilen C++ programı kullanıcıdan iki tam sayı alan, bu sayıları toplayan ve sonucu ekrana yazdıran bir programdır. Şekil 3.2.'de bu programın ekran çıktısı gösterilmektedir.

Örnek

- Kullanıcıdan iki tam sayı alan, bu sayıları toplayan ve sonucu ekrana yazdıran bir C++ programı yazınız.

Çözüm:

```
// Iki Tam Sayiyi Toplayan C++ Programi
// IkiSayiyiTopla.cpp

#include <iostream>
using namespace std;

int main()
{
	int sayi1;
	int sayi2;
	int toplam;

	cout<<"Ilk sayiyi giriniz:\n";
	cin>>sayi1;
	cout<<"Ikinci sayiyi giriniz:\n";
	cin>>sayi2;

	toplam = sayi1 + sayi2;

	cout<<"Girilen iki sayinin toplami: "<<toplam<<endl;
	return 0;
}
```



Şekil 3.2. İki Tam Sayıyı Toplayan C++ Programının Ekran Çıktısı

Bir değişkenin ismi, veri tipi ve değeri vardır. Veri tipi ile doğrudan ilgili boyut bilgisi de diğer bir özellik olarak düşünülebilir.

int sayi1;, *int sayi2;*, ve *int toplam;* deyimleri üç tam sayı değişken tanımlamak amacıyla kullanılmıştır. Değişken isimleri harf, rakam ve alt çizgi karakterleri kullanılarak belirlenir. *int* anahtar kelimesi kullanılarak *sayi1*, *sayi2* ve *toplam* değişkenleri tanımlanmıştır. Değişkenler bir program içerisinde herhangi bir yerde tanımlanabilirler. Bu noktada tek kısıt değişkenlerin program içerisinde kullanılmadan önce muhakkak tanımlanmış olmalarıdır. Değişken tanımlarken anlamlı isimler seçilmesi programın anlaşılabilmesi için önemlidir. Değişkenlerin isim, veri tipi, boyut ve değer olmak üzere 4 özelliği vardır. Yukarıdaki programda değişkenler sadece tanımlanmış, bu değişkenlere herhangi bir başlangıç değeri

atanmamıştır. Programın farklı satırlarında tanımlanmış 3 değişken *int sayi1, sayi2, toplam;* şeklinde tek bir ifade ile de tanımlanabilir (Değişkenlerin aynı ifadede (satırda) tanımlanabilmeleri için aynı tipte olmaları gerekir.). Değişken isimleri harfe duyarlıdır. Örneğin, *Sayi1* ve *sayi1* değişkenleri farklı değişkenlerdir ve bellekte farklı lokasyonları gösterirler. Değişkenlere tanımlandıklarında = operatörü ile ilk değer atanabilir. *int a=25;* deyimi ile a değişkeni tam sayı olarak tanımlanmış ve 25 değeri a değişkenine ilk değer olarak atanmıştır. Burada tek bir satırda gerçekleştirilen işlem *int a;* ve *a=25;* olmak üzere 2 farklı deyim ile de yazılabilir. Bu programda sadece tam sayı değişkenlere ihtiyaç duyulmuştur ve veri tipi olarak int kullanılmıştır. C++ programlama dili farklı ilkel veri tiplerine de sahiptir. Şekil 3.3.'te ilkel veri tipleri, bu veri tiplerinin bellekte ne kadarlık bir alan kapladıkları ve alabilecekleri değer aralıkları gösterilmiştir.

Tablo 3.1. C++ İlkel Veri Tipleri [2]

Veri Tipi	Boyut (Bayt)	Değer Aralığı
bool	1	true, false
char	1	-127 ile 127
signed char	1	-127 ile 127
unsigned char	1	0 ile 255
short	2	-32768 ile 32767
unsigned short	2	0 ile 65535
int	4	-2147483648 ile 2147483647
unsigned int	4	0 ile 4294967295
float	4	+/- 3.4e +/- 38 (~7 digits)
double	8	+/- 1.7e +/- 308 (~15 digits)

>> operatörü akış çıkarma operatörü olarak isimlendirilir ve operatörün sağ tarafında kalan değerler girdi akışına eklenir. *cin* nesnesi ise >> operatörünün sol işlenen değeridir ve varsayılan olarak klavyeye bağlıdır. *cin>>sayi1;* deyimi ile klavyeden girilen tam sayı *sayi1* değişkenine atanır. *sayi1* değişkeninin daha önceden sahip olduğu değer (eğer varsa) kaybolur. Her bir değişken ayrı *cin* deyimi kullanılarak okunabilir. Bunun yanında sadece bir *cin* deyimi kullanılarak birden fazla değişken için klavyeden değer okuma işlemi de gerçekleştirilebilir. Örneğin, *cin>>sayi1>>sayi2;* deyimi ile sayi1 ve sayi2 değişkenlerine klavyeden değer girilebilir. Genel olarak *cin* ve *cout* deyimleri yardımıyla kullanıcılar ve bilgisayar sistemi arasında etkileşim gerçekleştirilir.

Atama operatörü kullanılarak sağ işlenenin değeri sol işlenene aktarılabilir. Bu atama sonrası sağ işlenenin değeri değişmez.

toplam = sayi1 + sayi2; deyimi sayi1 ve sayi2 değerlerini toplama ve elde edilen sonucu toplam değişkenine atama işlemini gerçekleştirir. = operatörü ikili bir operatördür ve sağ işleneni sol işlenene atama işlemi için kullanılır. Bu esnada sağ işlenende bir değişiklik olmazken, sol işlenen eski değerini kaybeder. = operatöründe sol işlenen mutlaka bir değişken olmalıdır; sabit değerler = operatörünün sol tarafında yer alamazlar. Dolayısıyla *10 = b;* deyimi doğru yazılmış bir C++ ifadesi değildir.

cout<<"Girilen iki sayinin toplami: "<<toplam<<endl; deyimi önce *"Girilen iki sayinin toplami: "* karakter dizisini ekrana yazdırır. Sonra *toplam* değişkeninin değeri ekranda görüntülenir. *endl* satır sonu anlamındadır ve göstergeyi yeni satırın başına getirir (cout ile ekrana veri yazdırma işleminin soldan sağa doğru ilerlediğine dikkat ediniz.). Hesaplamalar çıktı ifadelerinin içinde de gerçekleştirilebilir. *cout<<"Girilen iki sayinin toplami: "<<sayi1 + sayi2<<endl;* ifadesi kullanılarak da istenen sonuç elde edilebilirdi. Bu şekilde bir ifade kullanılmış olsaydı *toplam* değişkeninin tanımlanmasına gerek kalmazdı.

Bellek Kavramları

Değişken isimleri olarak kullanılan *sayi1*, *sayi2* ve *toplam* bilgisayar belleğinde belirli lokasyonları gösterir. Her bir değişkenin ismi, veri tipi, boyutu ve değeri vardır. *cin>>sayi1;* deyimi çalıştığında kullanıcı tarafından klavyeden girilen değer, tam sayı olarak tanımlanan *sayi1* değişkeninin derleyici tarafından belirlenen bellek alanına yazılır. Bu alanda daha önce herhangi bir değer varsa bu değer kaybolur. Bir başka ifade ile yeni değer eski değerin üzerine yazılır. Aynı şekilde *cin>>sayi2;* deyimi ile kullanıcının girdiği değer *sayi2* değişkeninin bellekteki alanına yazılır. *toplam = sayi1 + sayi2;* deyimi ise önce *sayi1* ve *sayi2* değerlerini toplar ve sonucu atama operatörü ile *toplam* değişkeninin bellekteki alanına aktarır. Bu esnada *toplam* değişkeninin önceki değeri kaybolur.

ARİTMETİK İŞLEMLER

Aritmetik operatörlerin öncelik sırasından emin değilseniz parantez kullanarak istediğiniz işlemlere öncelik sağlayabilirsiniz.

Birçok program istenen çözümü gerçekleştirmek için aritmetik işlemlerden faydalanır. C++ programlama dilinde kullanılan aritmetik operatörler Tablo 3.2.'de verilmiştir.

Tablo 3.2.'de verilen tüm operatörler ikili operatörlerdir. Örneğin, *c-d* şeklindeki C++ ifadesinde çıkarma operatörü ikilidir; c ve d olarak iki işleneni vardır. Tam sayı bölme işlemi (pay ve paydanın tam sayı olması) tam sayı bir sonuç üretir. Örneğin, 9 / 4 işleminin sonucu 2'dir. Modül operatörü birinci işlenenin ikinciye bölünmesinden kalan değeri verir. Örneğin, 19 % 4 işleminin sonucu 3'tür. Modül operatörü bir tam sayının başka bir tam sayının katı olup olmadığını bulmak için kullanılabilir. Bu operatörün sık kullanım alanlarından bir başkası ise bir tam sayının tek veya çift olduğuna karar vermektir.

Tablo 3.2. C++'ın Aritmetik Operatörleri [1]

Aritmetik İşlem	C++ Operatörü	Cebirsel İfade	C++ İfadesi
Toplama	+	a + b	a + b
Çıkarma	-	c - d	c - d
Çarpma	*	ef	e * f
Bölme	/	g / h	g / h
Modül	%	i mod j	i % j

Cebirsel ifadelerdeki parantezler C++ ifadelerinde de kullanılır. Aynen cebirsel ifadelerde olduğu gibi parantezin diğer tüm operatörlere önceliği vardır. Parantez operatöründen sonra öncelik sırası çarpma, bölme ve modül

operatörlerindedir. Bu üç operatör aynı seviyede önceliğe sahiptir. Eğer bir ifadede söz konusu üç operatör birden fazla kez kullanılmışsa, operatörlerin çalışma sırası soldan sağa şeklindedir. Öncelik sırası daha sonra toplama ve çıkarma operatörlerinindir. Aynı şekilde bir ifadede birden fazla toplama ve çıkarma operatörü kullanılıyorsa işlem sırası soldan sağa doğrudur.

Birden fazla değişkene bir cin deyimi kullanılarak değer okunabilir.

Aşağıdaki örnekte bir öğrencinin ödev notu, ara sınav notu ve final notu kullanıcıdan alınmakta ve örnekte verilen oranlar kullanılarak dönem sonu notu ağırlıklı ortalama olarak hesaplanmaktadır. Bu örnekte her bir değer ayrı *cin* deyimi kullanılarak okunmuştur. Bunun yanında tek bir *cin* kullanılarak 3 değer arka arkaya okunabilir. Bu durum için *cin>>odevNotu>>arasinavNotu>>finalNotu;* yazmak yeterlidir.

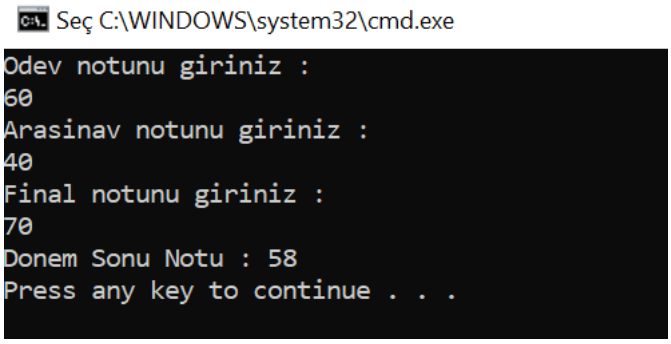
Örnek

- Bir dersin dönem sonu notu hesaplanırken aşağıdaki oranlar kullanılmaktadır:
- Final: %50
- Ara Sınav: %35
- Ödev: %15
- Kullanıcıdan final, ara sınav ve ödev notunu alan ve dönem sonu notunu hesaplayıp ekrana yazdıran bir C++ programı yazınız.

Çözüm:

```
// Donem Sonu Notunu Hesaplayan C++ Programi
// DonemSonuNotu.cpp
#include <iostream>
using namespace std;
int main()
{
        int odevNotu,
            araSinavNotu,
            finalNotu,
            donemSonuNotu;
        cout << "Odev notunu giriniz: \n";
        cin >> odevNotu;
        cout << "Ara sinav notunu giriniz: \n";
        cin >> araSinavNotu;
        cout << "Final notunu giriniz: \n";
        cin >> finalNotu;
        donemSonuNotu=0.15*odevNotu+0.35*araSinavNotu+0.50*finalNotu;
        cout << "Donem Sonu Notu: " << donemSonuNotu << endl;
        return 0;
}
```

Aynı veri tipindeki değişkenler bir satırda tanımlanabileceği gibi, ayrı satırlarda da tanımlanabilir.



Şekil 3.3. Dönem Sonu Notunu Hesaplayan C++ Programının Ekran Çıktısı

Örnek

- y = 4 * 4 * 4 - 2 * 5 * 5 % 3 + 4 ; deyiminde yer alan işlemler C++ dilinde hangi sırayla gerçekleştirilir?

Çözüm:

1. Adım: y = 16 * 4 - 2 * 5 * 5 % 3 + 4;
2. Adım: y = 64 - 2 * 5 * 5 % 3 + 4;
3. Adım: y = 64 - 10 * 5 % 3 + 4;
4. Adım: y = 64 - 50 % 3 + 4;
5. Adım: y = 64 - 2 + 4;
6. Adım: y = 62 + 4;
7. Adım: y = 66;

Birden fazla atama operatörü (=) aynı ifade içinde kullanıldığında bu operatörler için işlem sırası sağdan sola olarak gerçekleşir.

Yukarıda verilen çözüm adımları incelendiğinde * ve % operatörlerinin – ve + operatörlerinden öncelikli olarak çalıştığı görülmektedir. Aynı öncelik sırasına sahip operatörlerin öncelik sırası soldan sağadır. Yukarıdaki tabloda verilen operatörler için aynı seviye operatörlerin öncelik sırası soldan sağadır; ama bu durum tüm operatörler için aynı değildir. Örneğin *x = y = z;* aritmetik işleminde iki adet = operatörü kullanılmıştır. Fakat bu operatörlerin işlem sırası soldan sağa değil, sağdan sola şeklindedir. Öncelikle *z* değişkeninin değeri *y* değişkenine, daha sonra *y* değişkeninin bu yeni değeri *x* değişkenine atanmaktadır. Bu durumda verilen bu ifade çalıştırıldığında tüm değişkenlerin değeri *z* değişkeninin değeri olacaktır.

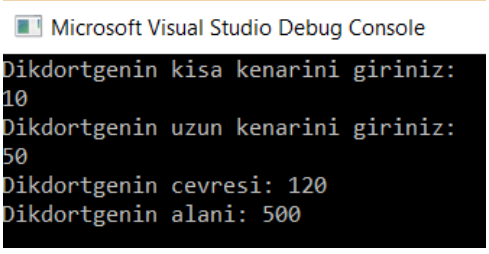
Aşağıdaki örnek C++ programı bir dikdörtgenin kısa kenarını ve uzun kenarını kullanıcıdan alıp, dikdörtgenin çevresini ve alanını hesaplayarak ekrana yazdırmaktadır.

Örnek

•Kullanıcıdan bir dikdörtgenin kısa ve uzun kenarlarını alan, dikdörtgenin çevresini ve alanını hesaplayarak ekrana yazdıran bir C++ programı yazınız.

Çözüm:

```
// Bir Dikdortgenin Cevresini ve Alanını Hesaplayan C++ Programi
// DikdortgenAlanCevre.cpp

#include <iostream>
using namespace std;

int main()
{
	int kisaKenar;
	int uzunKenar;
	int cevre;
	int alan;

	cout << "Dikdortgenin kisa kenarini giriniz: \n";
	cin >> kisaKenar;
	cout << "Dikdortgenin uzun kenarini giriniz: \n";
	cin >> uzunKenar;

	cevre = 2*kisaKenar + 2*uzunKenar;
	alan = kisaKenar * uzunKenar;

	cout << "Dikdortgenin cevresi: " << cevre << endl;
	cout << "Dikdortgenin alani: " << alan << endl;

	return 0;
}
```



Şekil 3.4. Bir Dikdörtgenin Çevresini ve Alanını Hesaplayan C++ Programının Ekran Çıktısı

Bu örnekte de daha önce bu bölümde yer alan C++ programlarında olduğu gibi ardışık çalışma gerçekleştirilmiştir. Programda yer alan deyimler ardışık bir şekilde işletilmiştir. İlerleyen bölümlerde farklı deyimler arasında seçim yapmamızı

sağlayan karar ifadeleri ve aynı deyimleri tekrarlayan döngü ifadeleri kullanılacak ve bu şekilde ardışık çalışma dışında gerçekleşen programlara da yer verilecektir.

Yazılan C++ programlarının doğruluğu yanında okunabilirliliği de önemlidir.

Bireysel Etkinlik

- Ekrana "C++'a Hosgeldiniz!" yazdıran bir C++ programı yazınız.
- Ekrana çarpım tablosunu yazdıran bir C++ programı yazınız.
- Kullanıcıdan iki tam sayı alan, bu sayıların farkını, çarpımını, ikinci sayının birinci sayıya bölümünden elde edilen sonucu ve kalanı hesaplayan bir C++ programı yazınız.
- Kullanıcıdan bir öğrencinin 4 doğru cevabın 1 yanlış cevabı götürdüğü sınavda yaptığı doğru ve yanlış sayılarını alarak öğrencinin netini hesaplayan bir C++ programı yazınız.

Özet

- Bu bölümde C++ programlamanın temel öğeleri anlatılmıştır. Bu ünitede yazılan programlar Microsoft Visual Studio 2017 Community sürümünde çalıştırılmış ve ekran çıktısı eklenmiştir. İlk program sayesinde bir karakter dizisinin ekrana ne şekilde yazdırılacağı anlatılmıştır. Dizilerin içine kaçış karakterleri eklenmesi gösterilmiş ve farklı kaçış karakterleri sıralanmıştır. Değişken, dosya, sınıf ve fonksiyon isimlerinde kullanılan deve ve paskal notasyonları tanıtılmıştır.
- Ekrana "Merhaba Dunya" yazdıran C++ programı bu ünitede çalışılmış ilk örnektir ve C++ dilinde yazılan programların temel bileşenleri bu program kullanılarak anlatılmıştır. // Ilk C++ Programi satırı bir açıklama satırıdır ve derleyici tarafından derleme sürecinde dikkate alınmaz. Başka bir deyişle programın çalışmasına herhangi bir etkide bulunmaz. Sadece programın kullanıcılarını bilgilendirmek amaçlı yazılan kısımdır.
- #include <iostream> satırı önişlemciye bir bilgilendirme mesajıdır. # karakteri ile başlayan satırlar C++ programı derlenmeden önce ön işlemci tarafından çalıştırılırlar. Bu örnekte verilen ve önişlemci tarafından içeriğinin eklenmesi istenen iostream başlık dosyası C++ programlarında okuma/yazma işlemleri için gereklidir.
- int main() satırı her C++ programının bir parçasıdır. main anahtar kelimesinden sonra yazılan () karakterleri main'in bir fonksiyon olduğunu gösterir. C++ programları bir veya birden fazla fonksiyon içerir ve bu fonksiyonlardan birisi main olmak zorundadır. C++ programları çalışmaya main fonksiyonunun ilk satırından başlar. main anahtar kelimesinin sol tarafında yer alan int anahtar kelimesi main fonksiyonunun dönüş tipini gösterir. int tam sayıları gösteren veri tipidir. main fonksiyonu başarılı bir şekilde çalışmasını tamamladığında bir tam sayıyı geriye çevirecektir. Bir sonraki satırda yer alan sol ayraç { her fonksiyonda gereklidir ve ilgili fonksiyon gövdesinin başladığını gösterir. Karşılık gelen sağ ayraç } ise fonksiyon gövdesinin bittiğini gösterir.
- cout<<"Merhaba Dunya\n"; satırı çift tırnak işaretleri arasında yer alan karakter dizisini ekrana yazdırmak için kullanılır. << operatörü akış yerleştirme operatörü olarak isimlendirilir ve operatörün sağ tarafında kalan değerler çıktı akışına eklenir. cout nesnesi ise << operatörünün sol işlenen değeridir ve varsayılan olarak ekrana bağlıdır. return 0; ifadesi programın başarılı bir şekilde sonlandığını gösterir. main fonksiyonunun dönüş tipi tam sayı olarak belirlendiği için return anahtar kelimesinden sonra 0 değeri dönüş değeri olarak kullanılmıştır.
- İkinci örnek programda kullanıcıdan klavye yardımıyla değer okunması için gerekli komutlar anlatılmıştır. Ayrıca toplama ve atama işlemleri de bu program içerisinde gösterilen yeni özelliklerdir. Bellek kullanımı, üç tam sayı tanımlama işleminin de olduğu bu örnek program üzerinde gösterilmiştir. Programda sadece tam sayı değişkenler kullanılmış, diğer ilkel C++ değişken tipleri bir tablo üzerinde gösterilmiştir. Bu tabloda her bir veri tipinin bellekte kapladığı alan ve alabileceği minimum ve maksimum değerler yer almaktadır. Diğer basit aritmetik işlemler ve işlemlerdeki öncelik sıraları detaylı olarak ifade edilmiştir. Aritmetik işlemlere örnek olması amacıyla final, ara sınav ve ödev notlarını kullanarak dönem sonu notunu ağırlıklı ortalama ile hesaplayan bir C++ programı yazılmıştır.

DEĞERLENDİRME SORULARI

1. Geçerli bir kaçış dizisi aşağıdakilerden hangisi olamaz?
   a) \a
   b) \%
   c) \n
   d) *
   e) \\

2. unsigned char veri tipinin alabileceği minimum ve maksimum değerler aşağıdakilerden hangisinde doğru verilmiştir?
   a) -127 ve 127
   b) -127 ve 0
   c) -127 ve 255
   d) 0 ve 255
   e) 0 ve 127

3. Bir C++ programında m = n + 10; deyiminde verilen işlem gerçekleştiğinde m ve n değişkenlerinin yeni değerleri için aşağıdakilerden hangisi söylenebilir?
   a) m değişkeninin yeni değeri n değerinin 10 fazlasıdır. n değişkeninin değeri değişmez.
   b) m değişkeninin yeni değeri n değerinin 10 fazlasıdır. n değişkeninin yeni değeri kendi değerinin 10 fazlasıdır.
   c) m değişkeninin değeri değişmez. n değişkeninin yeni değeri kendi değerinin 10 fazlasıdır.
   d) m ve n değişkenlerinin değeri değişmez.
   e) m değişkeninin yeni değeri kendi değerinin 10 fazlasıdır. n değişkeninin değeri değişmez.

4. C++ deyimlerinden hangisinde söz dizimi hatası yapılmıştır?
   a) cout<<"C++\n dersine hosgeldiniz\n";
   b) cout<<"Ders notunun %%20 si uygulamalardan olusacak\n";
   c) cout<<"Haftalik olarak odevler verilecek\n";
   d) cout<<"Basarilar dilerim\n";
   e) cout<<"Final sinavinda 10 soru olacak\n;

5. cout<<"Bugun hava\ncok guzel.\n"; deyimi çalıştırıldığında ekrana aşağıdakilerden hangisini yazdırır?
   a) Bugun hava cok guzel.
   b) Bugun
      hava
      cok
      guzel.
   c) Bugun
      hava
      cok guzel.
   d) Bugun hava
      cok guzel.
   e) Bugun hava cok
      guzel.

6. C++ deyimlerinden hangisinde söz dizimi hatasına neden olan bir durum söz konusu değildir?
   a) cout>>a;
   b) cin<<b+5;
   c) Char a='d';
   d) 4=b+c;
   e) m=5;

7. C++ deyimlerinden hangisi söz dizimi hatasına neden olur?
   a) Return 0;
   b) i = j - 2;
   c) int c, d;
   d) cin>>c;
   e) cout<<"*\n";

8. Bir C++ programında verilen y=42%5+6/2*5-3; ifadesinde y değişkeninin alacağı değer aşağıdakilerden hangisidir?
   a) 14
   b) 16
   c) 18
   d) 22
   e) 26

9. a ÷ b cebirsel ifadesinin C++ ifadesi olarak yazılmış hali aşağıdakilerden hangisidir?
   a) a / b
   b) a ÷ b
   c) a * b
   d) a|b
   e) a & b

10. Operatörlerden hangisinin önceliği diğerlerinden düşüktür?
   a) Bölme
   b) Modül
   c) Toplama
   d) Parantez
   e) Çarpma

Cevap Anahtarı

1.d, 2.d, 3.a, 4.e, 5.d, 6.e, 7.a, 8.a, 9.a, 10.c

YARARLANILAN KAYNAKLAR

[1] Deitel H. M., Deitel P. J. (1998). *C++ How to Program*. Second Edition.

[2] C++ Language Reference. 24 Temmuz 2018 tarihinde https://msdn.microsoft.com/en-us/library/3bstk3k5.aspx adresinden erişildi.

BİR C++ PROGRAMININ YAZILMASI, DERLENMESİ VE ÇALIŞTIRILMASI

İÇİNDEKİLER

- Bütünleşik Geliştirme Ortamı
  - Microsoft Visual Studio Community 2017 Programının Kurulması
  - Bütünleşik Geliştirme Ortamının Tanıtılması
- Proje Oluşturma
- Derleme ve Çalıştırma
- Hata Düzeltme

HEDEFLER

- Bu üniteyi çalıştıktan sonra;
  - Microsoft Visual Studio Community 2017 programı üzerinde bir C++ projesi oluşturabilecek,
  - Projenize kaynak dosya ekleyerek program yazabilecek,
  - Yazdığınız programı derleyip çalıştırabilecek,
  - Programınızdaki hataları tespit edip düzeltebileceksiniz.

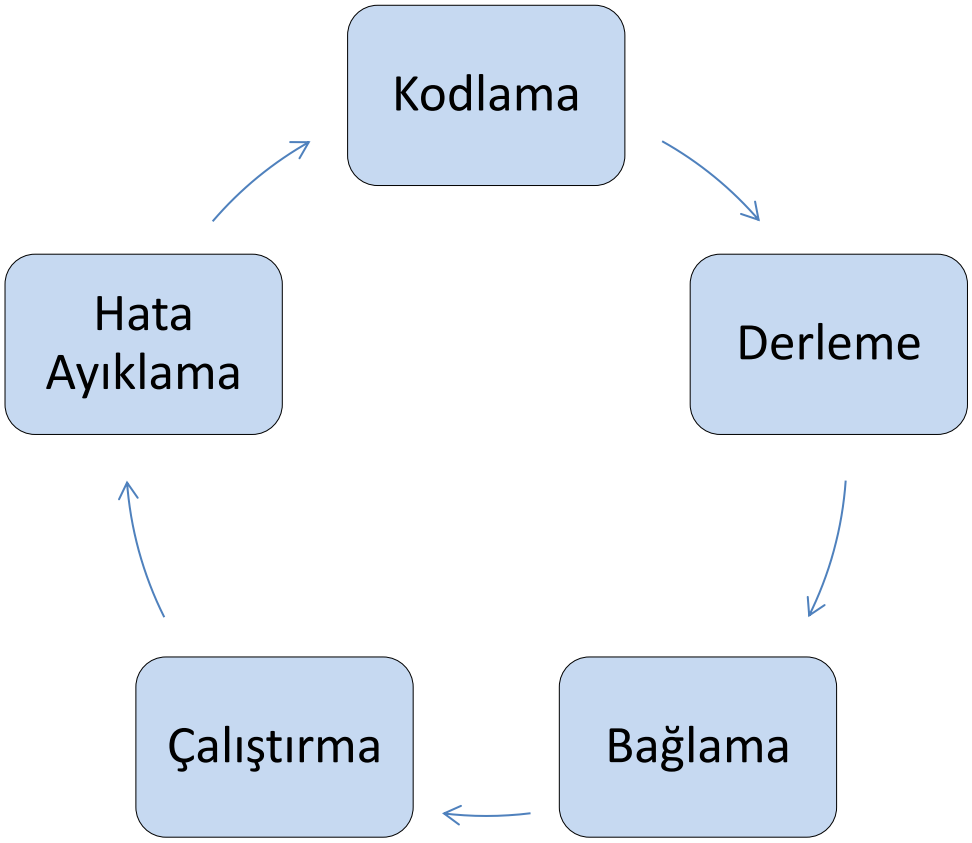
Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

Dr. Yılmaz AR

ÜNİTE

4


Kodlama
Derleme
Bağlama
Çalıştırma
Hata Ayıklama

GİRİŞ

Bütünleşik Geliştirme Ortamları (BGO), programlama dilleri ile uygulama geliştirmeyi kolaylaştıran yazılımlardır. Bir program yazmak için gerekli tüm araçlar BGO'lar bünyesinde tek bir çalışma ortamında bir araya getirilmiştir. BGO'lar gelişmiş kullanıcı arayüzlerine sahiptirler ve uygulamaların daha kolay ve hızlı geliştirilmelerine imkân sağlarlar. Önceleri kullanılan komut tabanlı yazılımların aksine menü tabanlı ortamlardır. Sadece bir programlama dilini destekleyen geliştirme ortamları olduğu gibi birden fazla programlama diline ev sahipliği yapan BGO'lar da vardır. Bu ünitede ve kitabın tamamında bir C++ programı oluşturmak için Microsoft Visual Studio 2017 (Community Sürümü) isimli BGO kullanılmıştır. Bu ortam ücretsiz olarak indirilebilir, kurulabilir ve kullanılabilir. Bu bölümde söz konusu geliştirme ortamının ne şekilde indirileceği ve kurulacağı da anlatılmıştır. Ayrıca Visual Studio 2017'nin en fazla kullanılan özellikleri de detaylı olarak gösterilmiştir. Bu kitapta geliştirilen tüm programlar C++ konsol uygulaması olarak tasarlanmıştır. Konsol uygulamalarının çıktıları komut penceresi üzerinden takip edilebilir. Visual Studio 2017 ortamı kullanılarak bir C++ konsol uygulaması geliştirmek amacıyla takip edilmesi gereken tüm aşamalar detaylı olarak anlatılmıştır. Bir projenin oluşturulması, bu proje içinde bir C++ programının yazılması ve daha sonra bu programın çalıştırılması, basit bir uygulamanın temel adımlarıdır. Ayrıca program yazımı esnasında karşılaşılabilecek bir söz dizimi hatası sonrasında yapılabilecekler iki örnek kullanılarak detaylandırılmıştır. Bu tür hatalar programın derlenmesi aşamasında ortaya çıkar ve kullanıcı arayüz ekranı üzerinden kullanıcıya bildirilir.

Bütünleşik geliştirme ortamları, herhangi bir programlama dilinde bir uygulama geliştirmek ve bu uygulamayı çalıştırmak için gerekli ortamı sağlarlar.

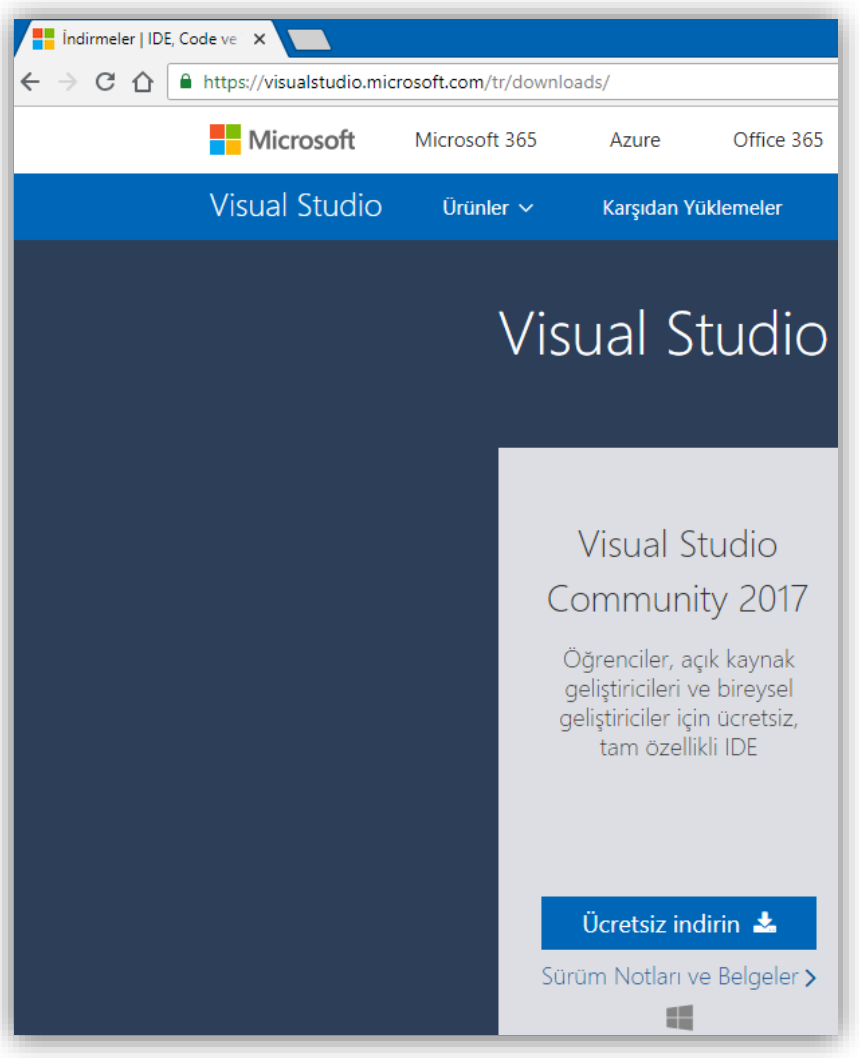
BÜTÜNLEŞİK GELİŞTİRME ORTAMI

Bir bütünleşik geliştirme ortamı, program yazmak ve test etmek amacıyla geliştiriciler tarafından kullanılan bir yazılım geliştirme aracıdır. Bu ortamlar yardımıyla yazılacak programlar daha hızlı ve verimli bir şekilde düzenlenebilir. BGO'lar için belirlenmiş standartlar yoktur. Her BGO'nun kendine ait güçlü yönleri veya zayıf noktaları olabilir. Bunun yanında genel olarak tüm BGO'lar kullanımı kolay bir arayüz sağlar, geliştirme adımlarını otomatikleştirir ve geliştiricilerin programları tek bir ekrandan çalıştırıp hata ayıklamasına imkân tanır. Bir açıdan ilkel metin düzenleyicilerin geliştirilmiş hali olarak düşünülebilirler. Kod yazma konusunda getirdikleri birçok yenilik vardır. Yazılacak kod kelimesini ilk bir kaç harften tahmin etme, bazı durumlarda açılan bir listeden seçim yapabilme gibi kodlama işlemini kolaylaştıran özellikler bu yeniliklerden sadece birkaç tanesidir. Ayrı bir metin düzenleyici kullanılan durumlarda derleme sırasında ortaya çıkabilecek hataları takip etmek zor olacaktır. Bunun yanında bütünleşik geliştirme ortamlarına entegre edilmiş metin düzenleyicilerde bu tür hatalar daha kolay takip edilebilir. Öncelikli olarak bir programlama dilinde bir program yazıp çalıştırmak için metin düzenleyici, derleyici ve hata ayıklayıcı kullanılır. Çalıştırılabilir bir yazılım oluşturabilmek için öncelikle kaynak dosyalar oluşturmanız, kaynak kodlardan makine kodu (nesne dosyaları) oluşturmak için derleme yapmanız, nesne dosyalarını birbirleriyle ve herhangi bir kütüphane veya diğer kaynaklarla

ilişkilendirmeniz gerekir. Tüm bu işlevler bütünleşik geliştirme ortamlarında yer alır. Grafiksel bir kullanıcı arayüzü üzerinden söz konusu işlemler gerçekleştirilir. Bu ortamların fonksiyonları sadece bir dilde yazılan programı derleyip çalıştırmak ile sınırlı değildir. Bütünleşik geliştirme ortamları kullanılarak çok farklı işlemler gerçekleştirilebilir. Veritabanı yönetim sistemlerinden mobil uygulama geliştirmeye, ağ sayfaları geliştirmeden ofis uygulamalarına kadar farklı işlemler bütünleşik geliştirme ortamlarında kendilerine yer bulabilirler.

Microsoft Visual Studio Community 2017 programı https://visualstudio.microsoft.com/tr/downloads/ adresinden indirilebilir. Bu adreste Şekil 4.1'de görüldüğü gibi en solda yer alan *Visual Studio Community 2017* kısmında *Ücretsiz İndirin* butonuna tıklanarak öncelikle yükleyici programı indirilebilir. Daha sonra yükleyici program çalıştırılarak geliştirme ortamının bilgisayara kurulması tamamlanabilir.

Bu kitapta yer alan tüm programlar Microsoft Visual Studio 2017 Community sürümünde hazırlanmıştır ve çalıştırılmıştır.

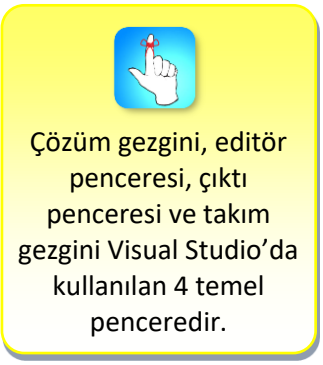


Şekil 4.1. Microsoft Visual Studio Community 2017 Programının İndirilebileceği Ağ Sayfası

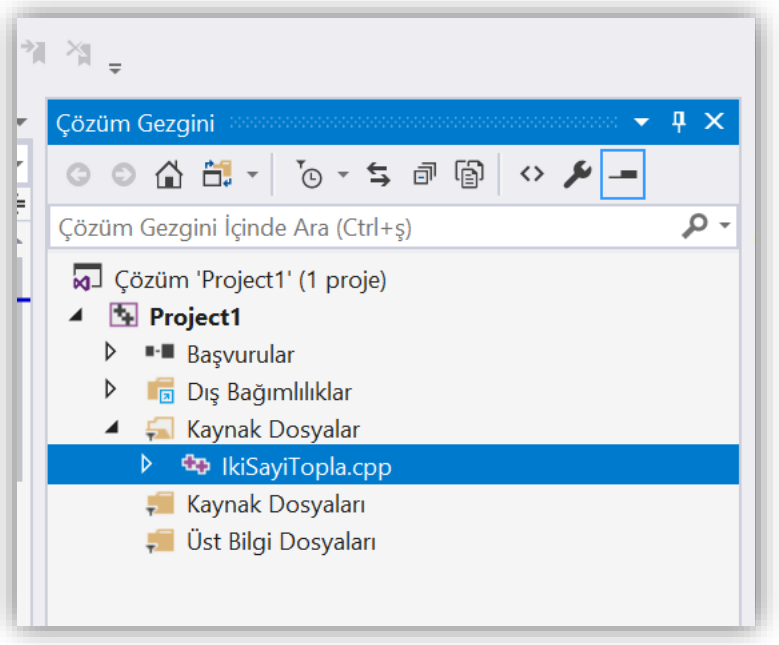
Bütünleşik Geliştirme Ortamının Tanıtılması

Bu kitapta yer alan tüm programlar ve kod parçaları Microsoft Visual Studio Community 2017 bütünleşik geliştirme ortamında oluşturulmuştur ve çalıştırılmıştır. Bu nedenle bu ünitede Visual Studio programlama ortamı tanıtılacaktır. Bu ortamda temel olarak program geliştirme, derleme, hata ayıklama ve test etme görevleri gerçekleştirilir. Bu görevlerin yanında geliştirilen uygulamaların dağıtılması, buluta bırakılması, veri tabanı uygulamaları ve uygulamaların gerçek zamanlı paylaşma ortamlarında işbirliği ile düzenlenmesi gibi görevler de gerçekleştirilmektedir. Bu ortamda sekiz adet programlama dilinde uygulama geliştirilebilmektedir. Bu diller; C#, Visual Basic, C++, F#, JavaScript, TypeScript, Python ve R olarak sıralanabilir.

Visual Studio Community 2017 isimli BGO'da temel olarak kullanılan pencereler; *çözüm gezgini, editör penceresi, çıktı penceresi* ve *takım gezginidir*. Şekil 4.2'de verilen çözüm gezgini kod dosyalarını bulmayı, görüntülemeyi ve yönetmeyi sağlar. Aynı zamanda kod dosyalarını çözümler veya projeler içinde gruplayarak düzenlemeye yardımcı olur.



Çözüm gezgini, editör penceresi, çıktı penceresi ve takım gezgini Visual Studio'da kullanılan 4 temel penceredir.



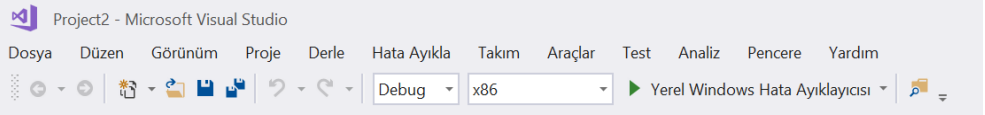
Şekil 4.2. Çözüm Gezgini

Editör penceresi, kodu gösterir ve kaynak kodunu düzenlemeye ve bir kullanıcı arayüzü tasarlamaya olanak tanır. Çıktı penceresi, Visual Studio'nun hata ayıklama ve hata iletileri, derleyici uyarıları gibi bildirimlerini gönderdiği yerdir. Her mesaj kaynağının kendi sekmesi vardır. Takım gezgini iş öğelerini takip edip, diğerleriyle sürüm kontrol teknolojilerini kullanarak kod paylaşmayı sağlar.

Visual Studio 2017 ortamında kod geliştirmeyi kolaylaştıran bazı özellikler vardır. *IntelliSense* kod düzenleyicide kod ile ilgili bilgi verir ve bazı durumlarda küçük kod parçalarını kullanıcı için tamamlar. *IntelliSense* özelliğinin getirdiği en önemli fayda anahtar kelime, fonksiyon ismi ve değişken ismi gibi bazı isimleri

tamamlamasıdır. *IntelliSense* ayrıca bir kod içerisinde yanlış yazılan kelimelerin altını dalgalı çizerek bir hatanın var olduğu konusunda kullanıcıyı uyarır.

Bütünleşik geliştirme ortamında yer alan menüler Şekil 4.3.'de gösterilmektedir.



Şekil 4.3. Visual Studio 2017'de Menüler

Dosya menüsünde yeni bir proje oluşturma, son açılan projelere ulaşma, proje kaydetme, yazdırma ve Visual Studio ortamını kapatma gibi seçenekler bulunur. Kopyalama, kesme, yapıştırma, silme, gerçekleştirilen işlemi geri alma veya ileri alma, bulma ve değiştirme işlemlerinin yapıldığı menü *Düzen* menüsüdür. *Görünüm* menüsünde Visual Studio ortamında yer alan alanların görüntülenme ayarları gerçekleştirilir. İstenilen alan gösterilir, istenilmeyen alanlar görüntülenmez. *Proje* menüsü oluşturulan bir projeye yeni öğe ekle, öğe çıkar gibi işlemlerin gerçekleştirildiği menüdür. *Derle* menüsünde projenin derlenmesi veya yapılandırılılması ile ilgili işlemler gerçekleştirilir. Projenin veya programların hatalarının ayıklanması ve çalıştırılması işlemleri *Hata Ayıkla* menüsünde yer alır. *Takım* menüsü kullanılarak birçok yazılımcının ortak bir sunucu üzerindeki projede çalışması için gerekli işlemler gerçekleştirilebilir. Visual Studio ortamının kullanıcı ihtiyaçları doğrultusunda kişiselleştirilmesi işlemleri *Araçlar* menüsünden gerçekleştirilir. Projelerin test işlemleri için kullanılan menü *Test* menüsüdür. Geliştirilen yazılımın performans ve analizleri için *Analiz* menüsü kullanılır. *Pencere* menüsü ile Visual Studio ortamındaki pencerelerin saklanması, gizlenmesi ve gösterilmesi işlemleri yapılır. *Yardım* menüsü ile Visual Studio hakkında yardım bilgilerine ulaşılır.

Hata Ayıkla menüsü kullanılarak yazılan program çalıştırılabilir.

Bireysel Etkinlik

- Araçlar menüsünde yer alan Seçenekler... sayesinde ortamın yazı tipi ve boyutu değiştirilebilir. Bu işlemin nasıl gerçekleştirilebileceğini araştırınız.

PROJE OLUŞTURMA

Kaynak dosyalar ilgili programa ait kod parçalarını içerir. Her kaynak dosya bünyesinde kullanılan programlama diline özgü ifadeler barındırır. Genellikle kaynak dosya adlarının içerdikleri kodu belirten uzantıları vardır. Bir C++ kaynak dosyasının uzantısı *.cpp*'dir. Derleyiciler, kaynak dosyaları hedef ortam için uygun makine seviyesi koduna çevirir. Bağlayıcılar, bir program için gereken tüm nesne dosyalarını alır ve bunları birbirine bağlar, bellek atar ve değişkenlere kayıt yapar, verileri ayarlar. Ayrıca işletim sistemi görevlerini ve programın ihtiyaç duyduğu

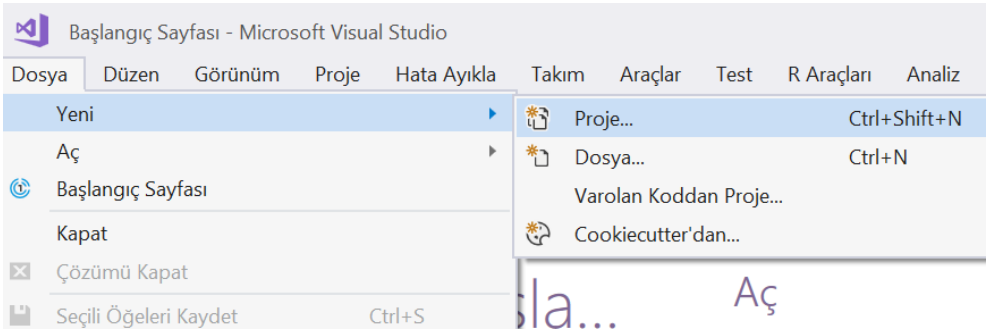
diğer dosyaları desteklemek için kitaplık dosyalarına da bağlanırlar. Bağlayıcılar çalıştırılabilir dosyaları üretirler. Tüm bu işlemler bu ve bir sonraki bölümde örnek bir program üzerinden gösterilmektedir. Örnek program kullanıcıdan iki tam sayı alacak ve bu sayıların toplamını bularak ekrana yazdıracaktır. Bu programı yazmak ve gerekli işlemleri gerçekleştirerek çalıştırmak için Visual Studio 2017 Community sürümünde takip edilecek işlemler sırasıyla gösterilmektedir. Örnek program geliştirilirken derleyici, bağlayıcı ve çalıştırma işlemleri kaynak dosya yazıldıktan sonra tek bir adımda uygulanacaktır.

Bir C++ uygulaması geliştirmek için ilk olarak Visual Studio 2017 programı Şekil 4.4.'te görüldüğü gibi *Başlat* menüsü aracılığıyla çalıştırılır.

Şekil 4.4. Visual Studio 2017 Programının Başlatılması

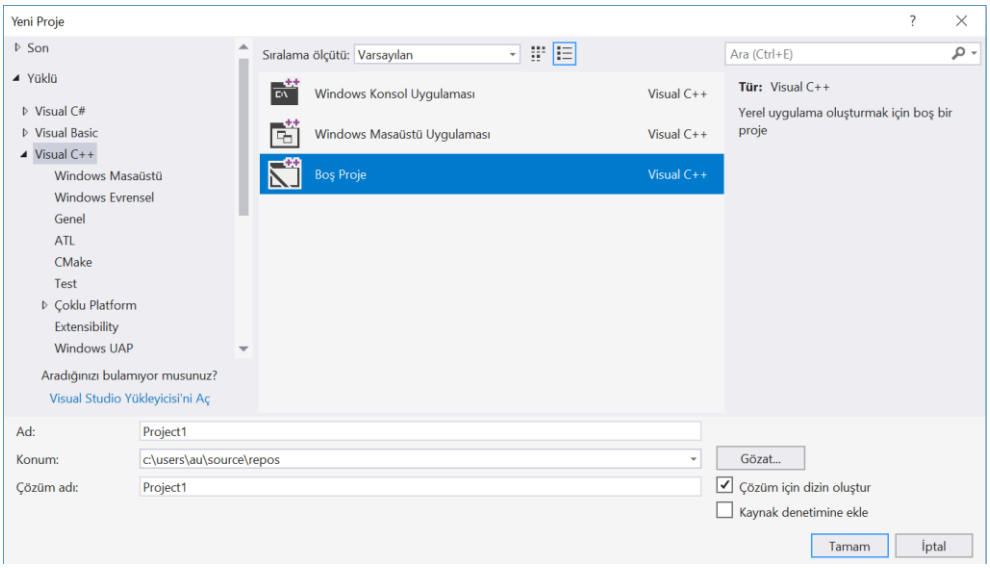
CTRL + SHIFT + N kısa yol tuş birleşimi ile yeni bir proje oluşturulabilir.

Daha sonra açılan Visual Studio programında Şekil 4.5'te yer aldığı gibi *Dosya* menüsünden önce *Yeni* daha sonra *Proje* seçenekleri seçilir.



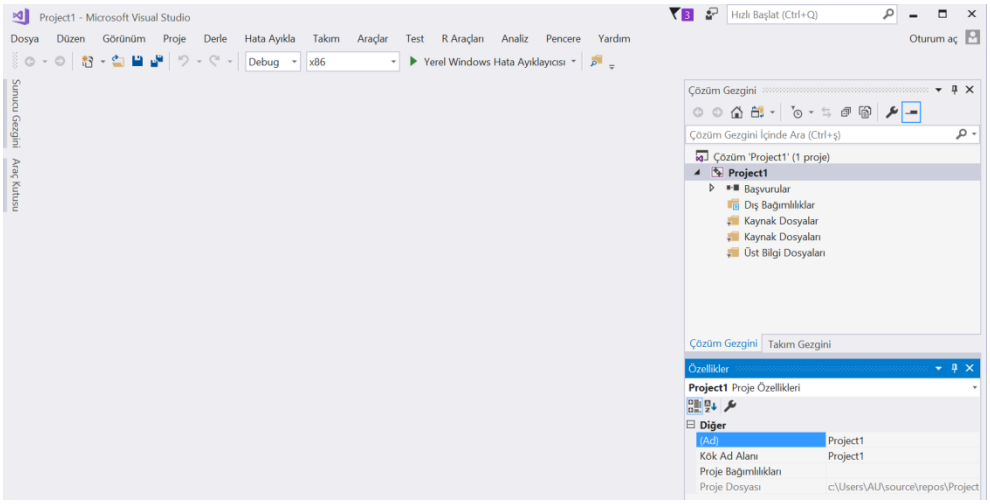
Şekil 4.5. Dosya Menüsünden Yeni ve Proje Seçeneklerinin Seçilmesi

Şekil 4.6.'da karşımıza gelen *Yeni Proje* ekranından *Visual C++* sekmesi seçilir. Şekilde de gösterildiği gibi *Boş Proje* seçeneği tıklanır. Şeklin alt kısmında *Ad* ve *Konum* alanları kullanılarak projenin ismi ve bilgisayar üzerinde oluşturulacağı konum belirlenir. Sonra *Tamam* butonu yardımıyla gerçekleştirilen seçimler onaylanır.



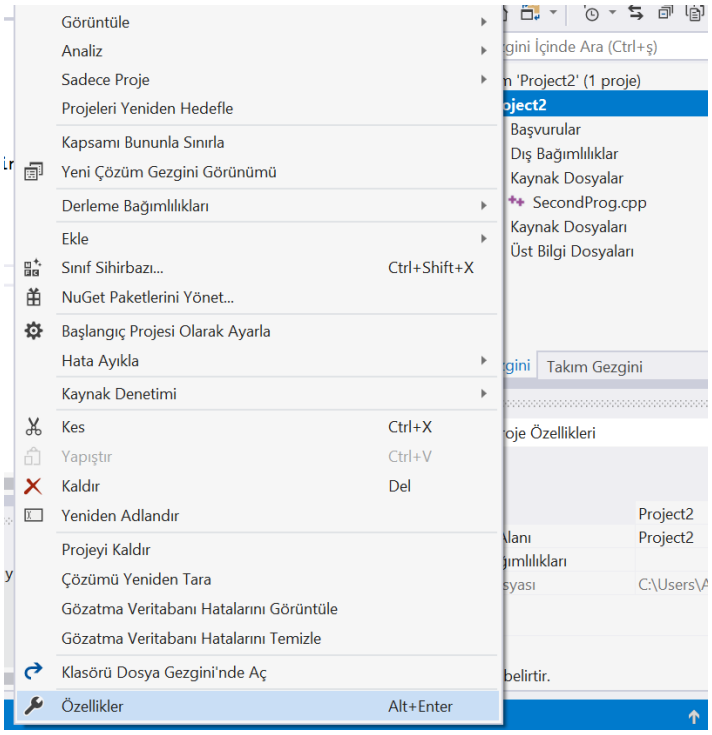
Şekil 4.6. Yeni Proje ekranından Visual C++ ve Boş Proje Seçeneklerinin Belirlenmesi

Şekil 4.6.'da gerçekleştirilen seçimler sonucunda Şekil 4.7'de gösterilen boş proje oluşturulur.



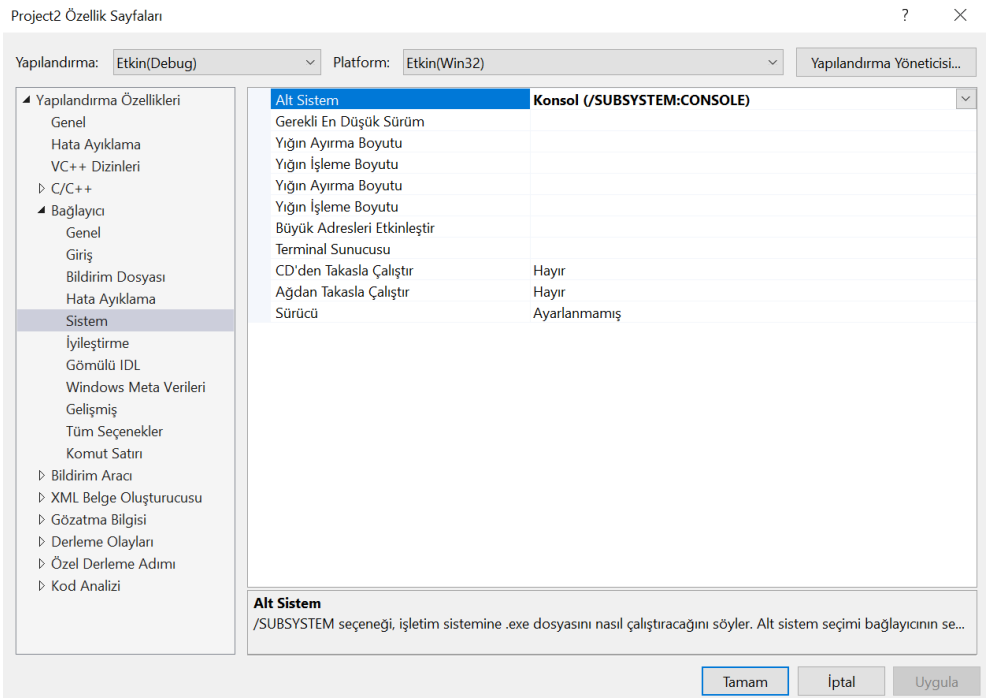
Şekil 4.7. Oluşturulan Boş Visual C++ Projesi

Proje içerisinde yazılan programların çıktı ekranlarının program sonlandığında kaybolmaması için proje üzerinde bir ayar gerçekleştirilmesi gerekmektedir (Bu işlem her bir proje için sadece bir kereliğine gerçekleştirilecek bir işlemdir.). Şekil 4.8.'de gösterildiği gibi proje ismine sağ tıklandığında açılan listeden Özellikler kısmı seçilir. Alt + Enter tuş birleşimi de proje özellikleri sayfasının seçiminde kullanılabilir.



Şekil 4.8. Proje Özellikleri Sayfasının Seçilmesi

Bu seçim sonunda Şekil 4.9.'da gösterilen ekran karşımıza gelir. Bu ekranda sol tarafta *Bağlayıcı* sekmesi altındaki *Sistem* kısmı seçilir. Sonrasında sağ tarafta yer alan *Alt Sistem* Konsol (/SUBSYSTEM:CONSOLE) olarak değiştirilir. Bu seçimler onaylandığında yazılan program *Hata Ayıklama Olmadan Başlat* seçeneği ile çalıştırılırsa çıktı ekranı program sonlandığında aniden kaybolmaz.

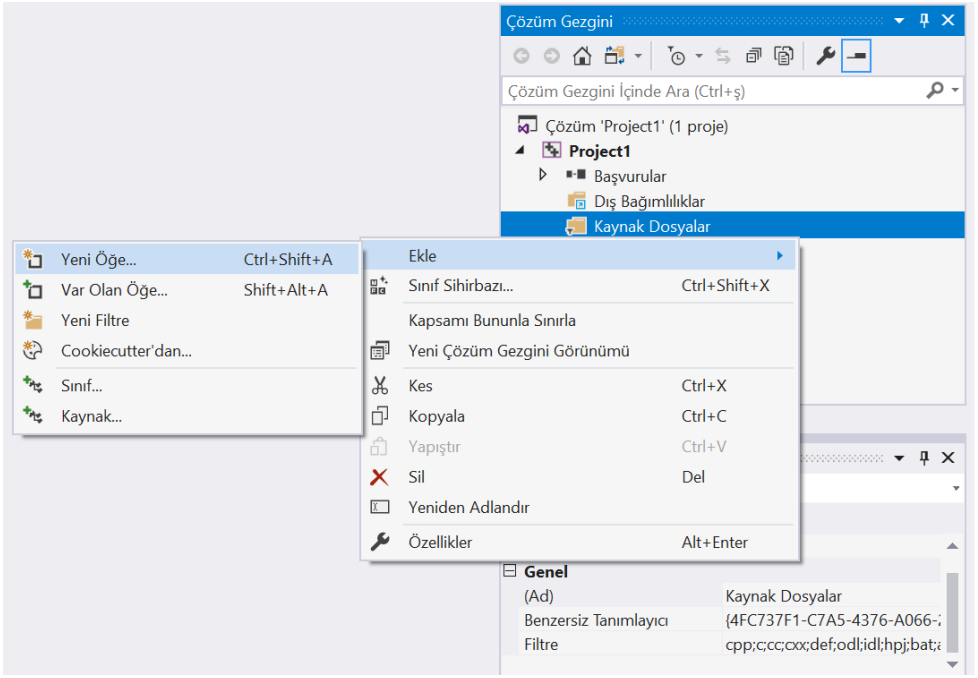


Şekil 4.9. Çıktı Ekranının Kaybolmaması İçin Gerekli Proje Ayarı

DERLEME VE ÇALIŞTIRMA

Şekil 4.10.'da görüldüğü gibi Çözüm Gezgini'nde *Kaynak Dosyalar* sekmesine sağ tıklanarak önce *Yeni Öğe* ve daha sonra *Ekle* seçenekleri kullanılarak proje içerisine yeni bir kaynak dosya eklenebilir.

Projeye yeni öge eklemek için CTRL + SHIFT + A tuş birleşimi kullanılabilir.

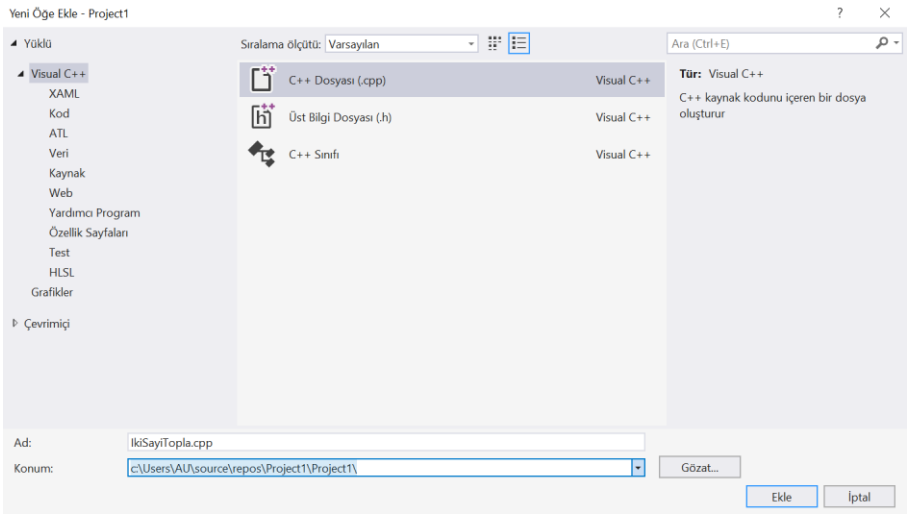


Şekil 4.10. Projeye Yeni Bir Kaynak Dosya Eklenmesi

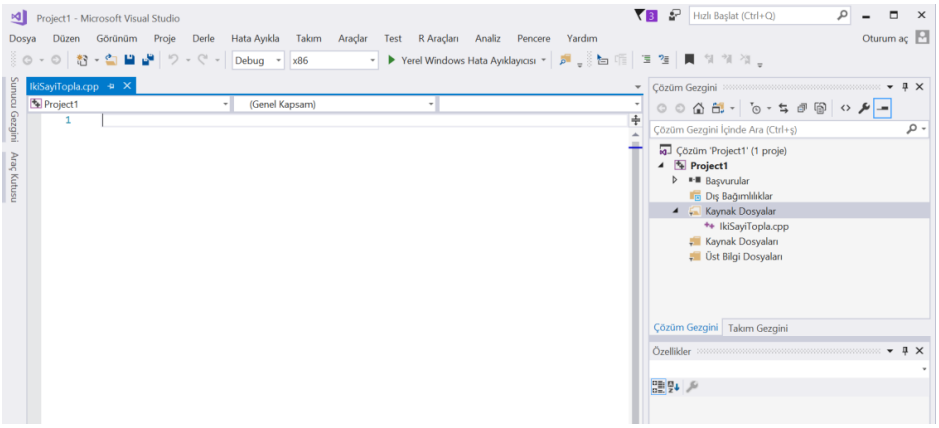
Yeni Öğe Ekle ekranı kullanılarak dosyanın tipi, ismi ve konumu belirlenebilir.

Şekil 4.10.'daki seçimler sonucunda karşımıza Şekil 4.11'de yer alan *Yeni Öğe Ekle* ekranı gelir. Bu ekranda solda *Visual C++* sekmesi seçilir ve sağda *C++ Dosyası (.cpp)* işaretlenir. Daha sonra dosyanın ismi belirlenir. Şekil 4.11.'de oluşturulan dosyaya *IkiSayiTopla.cpp* ismi verilmiştir. Dosyanın konumu c:\Users\AU\source\repos\Project1\Project1\ olarak belirlenmiştir. Burada belirlenen konum aslında Yeni Öğe ekle ekranında karşımıza gelen varsayılan konumdur. Konumda bir değişiklik yapılmamıştır. Ekle butonuyla gerçekleştirilen seçimler onaylanır. Onaylama işleminden sonra proje içerisinde Şekil 4.12.'de yer alan, içi boş *IkiSayiTopla.cpp* isimli C++ kaynak dosyası oluşur.

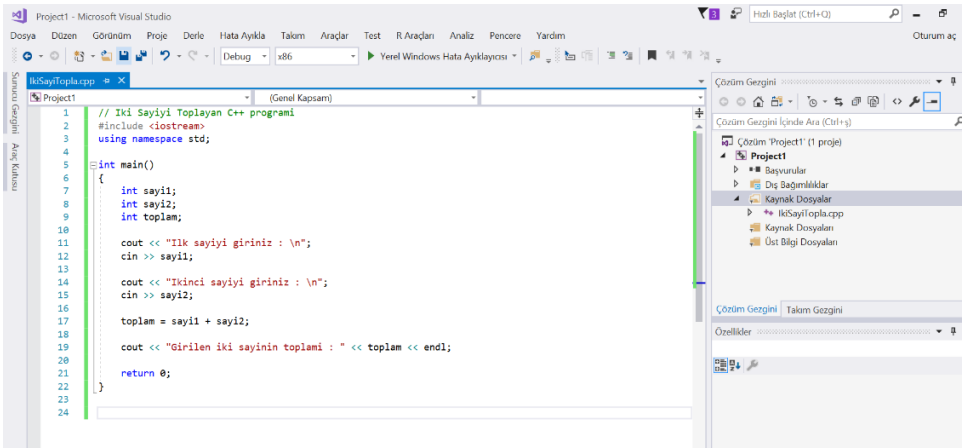
Klavyeden girilen iki sayıyı toplayan C++ programı Şekil 4.13.'de görüldüğü gibi IkiSayiTopla.cpp isimli dosya içerisine yazılmıştır.



Şekil 4.11. Yeni Öğe Ekle Ekranından Projeye C++ Kaynak Dosyası Eklenmesi



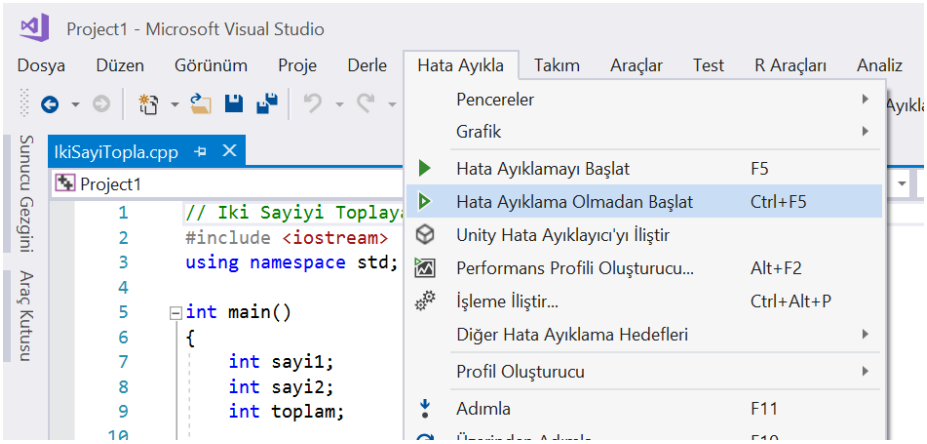
Şekil 4.12. Proje İçerisinde Oluşturulan IkiSayiTopla.cpp isimli Kaynak Dosya



Şekil 4.13. IkiSayiTopla.cpp Dosyasının İçerisinin Editör Aracılığıyla Doldurulması

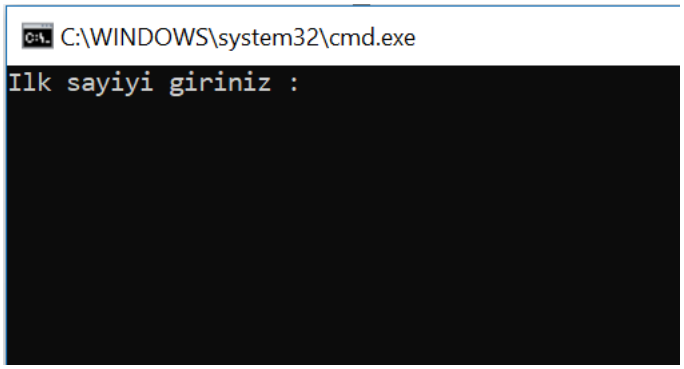
Editör penceresi kullanılarak C++ programını yazma işlemi gerçekleştirilir.

IkiSayiTopla.cpp programını derleyip çalıştırmak için Şekil 4.14.'te verildiği gibi *Hata Ayıkla* menüsünden *Hata Ayıklama Olmadan Başlat* seçeneği kullanılır. Programda herhangi bir hata olmaması durumunda program çalışacaktır ve Şekil 4.15'te görüldüğü gibi program ilk sayıyı kullanıcıdan isteyecektir. Kullanıcı ilk sayıyı girdikten sonra Şekil 4.16.'da yer aldığı gibi ikinci sayı da benzer şekilde kullanıcıdan alınacaktır. Şekil 4.17 kullanıcı ikinci sayıyı girdikten sonra sonucun ekrana yazdırılmasını ve programın çalışmasını tamamlamasını göstermektedir.

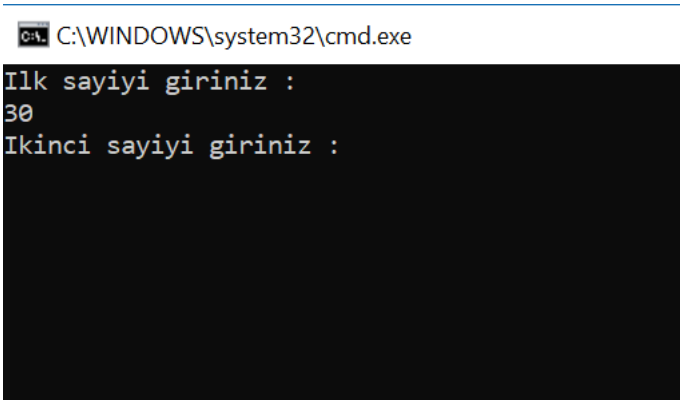


Şekil 4.14. IkiSayiTopla.cpp Programının Çalıştırılması

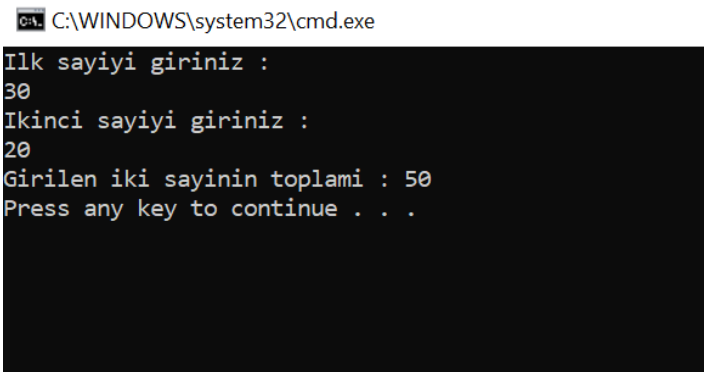
CTRL + F5 tuş birleşimi kullanılarak program çalıştırma işlemi gerçekleştirilebilir.



Şekil 4.15. İlk Sayının Klavyeden Girilmesi



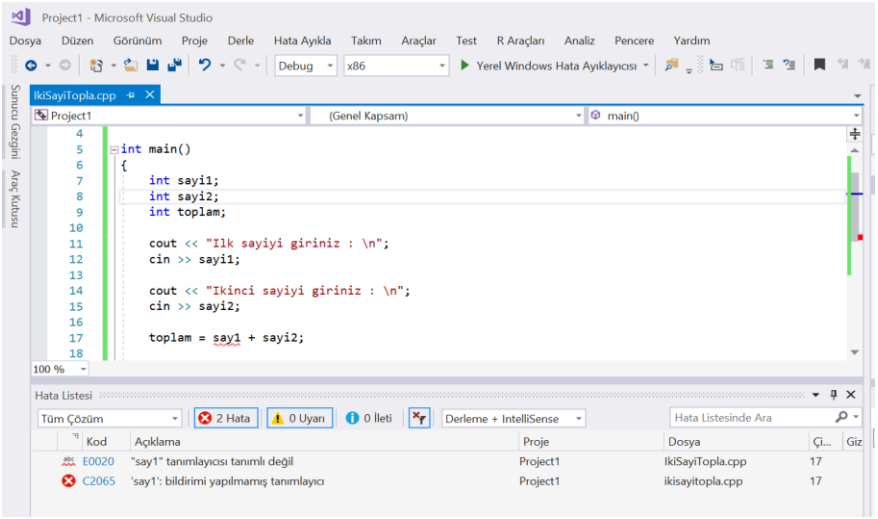
Şekil 4.16. İkinci Sayının Klavyeden Girilmesi



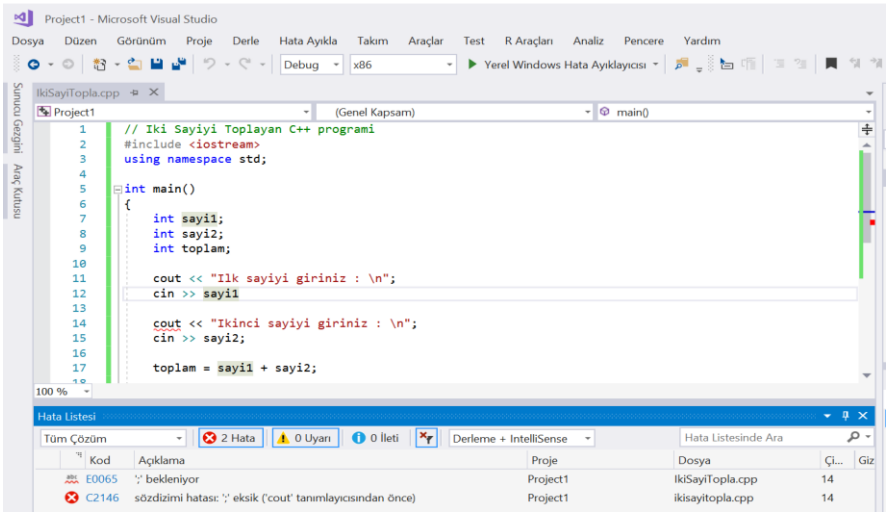
Şekil 4.17. Sonucun Hesaplanması ve Programın Tamamlanması

HATA DÜZELTME

Program yazılırken bir hata yapılmışsa, program çalıştırıldığında derleyici bu hatayı yakalar ve Şekil 4.18'de görüldüğü gibi çıktı penceresinde kullanıcıya gösterir. Derleyicinin bulduğu bu tür hatalar söz dizimi hataları olarak isimlendirilir. Mantıksal hatalar ise derleyici tarafından yakalanamazlar ve doğru olmayan sonuçların üretilmesine neden olurlar. Şekil 4.18.'de görülen hata bir matematiksel deyimde kullanılan ama daha önce tanımlanmamış say1 isimli bir değişken nedeniyle oluşmuştur. Bir başka ifade ile sayi1 değişkeni yanlışlıkla say1 olarak yazılmıştır. Bu hata düzeltildikten sonra program tekrar çalıştırılabilir. Şekil 4.19'da ise başka bir söz dizimi hatası gösterilmektedir. Bir deyimin sonuna eklenmesi gereken ';' karakteri unutulmuştur ve bu durum programın çalışması için engeldir. Derleyici bu hatayı da yakalayıp kullanıcıyı bilgilendirmiştir.



Şekil 4.18. Programda Yapılan Bir Söz Dizimi Hatasının Bulunması ve Kullanıcıya Gösterilmesi (Değişken İsminin Yanlış Yazılması)



Şekil 4.19. Programda Yapılan Bir Söz Dizimi Hatasının Bulunması ve Kullanıcıya Gösterilmesi (Deyim Sonunda ';' Unutulması)

Program yazarken yapılan söz dizimi hataları çıktı ekranında hata listesinden takip edilebilir.

Bütünleşik geliştirme ortamlarının farklı özelliklerini bilmek ve kullanmak program geliştirme sürecinde kolaylıklar sağlar.

Bireysel Etkinlik

- Ünite 3'de çalışılan, kullanıcıdan bir öğrencinin ödev notu, ara sınav notu ve final notunu okuyan, bu değerleri kullanarak öğrencinin dönem sonu notunu hesaplayan programı yeni bir projeye ekleyerek çalıştırınız.

Özet

- Bu ünitede bir programlama dili ile uygulama geliştirmek için kullanılan bütünleşik geliştirme ortamları (BGO) anlatılmıştır. Bu kitapta öğretilen programlama dili C++ olmakla beraber BGO'lar birden fazla programlama dilini destekleyebilirler. Bir BGO sayesinde bir C++ programını oluşturma, derleme, çalıştırma ve hata ayıklama işlemlerinin tümü tek bir ortamda gerçekleştirilebilir.
- Bütünleşik geliştirme ortamları daha hızlı ve verimli bir şekilde kod geliştirme amacıyla kullanılan yazılımlardır. Bu ortamların önceden belirlenmiş belirli bir standartları yoktur ve her bir geliştirme ortamının üstün veya zayıf olduğu yönler olabilir. Bununla birlikte sağlamak zorunda oldukları asgari işlemler vardır. Her bir geliştirme ortamı kod yazmayı sağlayacak bir editöre ve kodun derlenip çalıştırılmasını sağlayan araçlara sahip olmalıdır. Visual Studio temel olarak program geliştirme, derleme, çalıştırma ve hata ayıklama işlemlerini gerçekleştirebilmek için gerekli araçları ve ortamı sağlar. Bu işlemlerin yanı sıra geliştirilen uygulamaların dağıtılması, buluta bırakılması, veri tabanı uygulamaları ve uygulamaların gerçek zamanlı paylaşma ortamlarında birlikte oluşturulması gibi farklı görevler de Visual Studio 2017 tarafından desteklenir.
- Visual Studio Community 2017 geliştirme ortamında kullanılan temel pencereler; çözüm gezgini, editör penceresi, çıktı penceresi ve takım gezginidir. Kod dosyalarının görüntülenmesini ve yönetilmesini çözüm gezgini sağlar. Editör penceresi kaynak kod düzenlemek veya görüntülemek için kullanılır. Çıktı penceresinden hata iletileri ve derleyici uyarıları takip edilebilir. Takım gezgini kod paylaşmak için kullanılır.
- Bir problemi çözmek için yazılan program kaynak dosyayı oluşturmak amacıyla kullandığımız programlama diline özgü bir uzantı ile kaydedilir. Kaynak dosyalar derleyiciler tarafından uygun makine diline çevrilirler. Bağlayıcılar kitaplık dosyalarına bağlantı kurar ve çalıştırılabilir dosyaları üretir. Daha sonra çalıştırılabilir dosya çalıştırılarak istenen işlem gerçekleştirilir.
- İlk adımda Visual Studio 2017 programı Başlat menüsünden açılır. Dosya menüsünden Yeni daha sonra Proje seçenekleri seçilir. Karşımıza gelen Yeni Proje ekranından Visual C++ sekmesi seçilir, Boş Proje seçeneği tıklanır, ekranın alt kısmında yer alan Ad ve Konum alanları kullanılarak projenin ismi ve konumu belirlenir. Tamam butonu yardımıyla seçimler onaylanır ve boş proje oluşturulur.
- Kaynak Dosyalar sekmesine sağ tıklanarak önce Yeni Öğe ve daha sonra Ekle seçenekleri seçilerek proje içerisine yeni bir kaynak dosya eklenir. Karşımıza gelen Yeni Öğe Ekle ekranından Visual C++ sekmesi seçilir ve C++ Dosyası (.cpp) işaretlenir. Yazılacak program iki tam sayıyı kullanıcıdan alan, bu sayıları toplayan ve sonucu ekrana yazdıran bir uygulamadır. Bu nedenle oluşturulan kaynak dosya IkiSayiTopla.cpp olarak isimlendirilmiştir.
- Programımızı oluşturan C++ kodu IkiSayiTopla.cpp isimli dosyanın içerisine yazılır. Bu programı çalıştırmak için Hata Ayıkla menüsünden Hata Ayıklama Olmadan Başlat seçeneği kullanılır. Programda bir söz dizimi hatası olmaması durumunda program çalışacak ve kullanıcıdan ilk sayıyı isteyecektir. İlk sayı girildikten sonra kullanıcıdan ikinci sayı istenecek ve iki sayının toplamı hesaplanarak ekrana yazdırılacaktır.

DEĞERLENDİRME SORULARI

1. Visual Studio 2017 bütünleşik geliştirme ortamında aşağıdaki programlama dillerinden hangisi yer alır?
   a) Pascal
   b) MATLAB
   c) C#
   d) LISP
   e) PERL

2. Yeni Proje oluşturma işleminin klavyeden kısayol tuş birleşimi aşağıdakilerden hangisidir?
   a) CTRL + N
   b) CTRL + SHIFT + N
   c) CTRL + F3
   d) ALT + SHIFT + N
   e) ALT + N

3. C++ kaynak dosyası uzantısı aşağıdakilerden hangisidir?
   a) .c
   b) .c++
   c) .cpp
   d) .m
   e) .php

4. Visual Studio 2017 bütünleşik geliştirme ortamında yer alan Yeni Proje ekranı ile ilgili olarak aşağıdakilerden hangisi söylenemez?
   a) Projenin ismi belirlenebilir.
   b) Projenin konumu belirlenebilir.
   c) Projede kullanılacak kaynak dosyanın ismi belirlenebilir.
   d) Projede kullanılacak programlama dili belirlenebilir.
   e) Projenin çözüm adı belirlenebilir.

5. Visual Studio 2017 bütünleşik geliştirme ortamında hangisi yer alan menülerden değildir?
   a) Düzen
   b) Görünüm
   c) Hata Ayıkla
   d) Proje
   e) Veritabanı

6. Visual Studio 2017 bütünleşik geliştirme ortamında yer alan çözüm gezgininin görevi aşağıdakilerden hangisidir?
   a) Programın yazılmasını sağlar.
   b) Kod dosyalarını bulmayı, görüntülemeyi ve yönetmeyi sağlar.
   c) Hataların ayıklanmasını sağlar.
   d) Çıktıyı takip etmeyi kolaylaştırır.
   e) Kod paylaşımını gerçekleştirir.

7. Visual Studio 2017 bütünleşik geliştirme ortamının hangi alanında kod paylaşma işlemi gerçekleştirilebilir?
   a) Çözüm Gezgini
   b) Editör Penceresi
   c) Çıktı Penceresi
   d) Takım Gezgini
   e) Araç Kutusu

8. Derleyici tarafından yakalanabilen söz dizimi hatalarından birisini aşağıdakilerden hangisi oluşturur?
   a) Cout<<a;
   b) a=b+c;
   c) cin>>a;
   d) return 0;
   e) a=7;

9. Herhangi bir programlama dilinde, programı yazmak, derlemek ve çalıştırmak amacıyla geliştirilen ve bu süreçteki tüm işlemlerin yapılabileceği yazılıma hangi isim verilmektedir?
   a) Bütünleşik Geliştirme Ortamı
   b) Editör
   c) Derleyici
   d) Bağlayıcı
   e) Hata Ayıklayıcı

10. Genel olarak bir bütünleşik geliştirme ortamı hakkında aşağıdakilerden hangisi söylenemez?
    a) Farklı programlama dillerinde uygulama geliştirilmesine olanak sağlar.
    b) Programın yazılması için gerekli araçları sağlar.
    c) Programı derlemek için farklı bir programa ihtiyaç duyar.
    d) Program çalıştırıldığında oluşan hataları gösterir.
    e) Program çıktılarının takibine olanak sağlar.

Cevap Anahtarı

1.c, 2.b, 3.c, 4.c, 5.e, 6.b, 7.d, 8.a, 9.a, 10.c

YARARLANILAN KAYNAKLAR

[1] Visual Studio Documentation. 24 Temmuz 2018 tarihinde
https://docs.microsoft.com/en-us/visualstudio/ adresinden erişildi.

İLİŞKİSEL VE MANTIKSAL OPERATÖRLER VE ŞARTLI DEYİMLER

İÇİNDEKİLER

- Eşittir ve Eşit Değildir Operatörleri
- İlişkisel Operatörler
- Mantıksal Operatörler
- Koşul Kavramı ve if Şartlı Deyimi

HEDEFLER

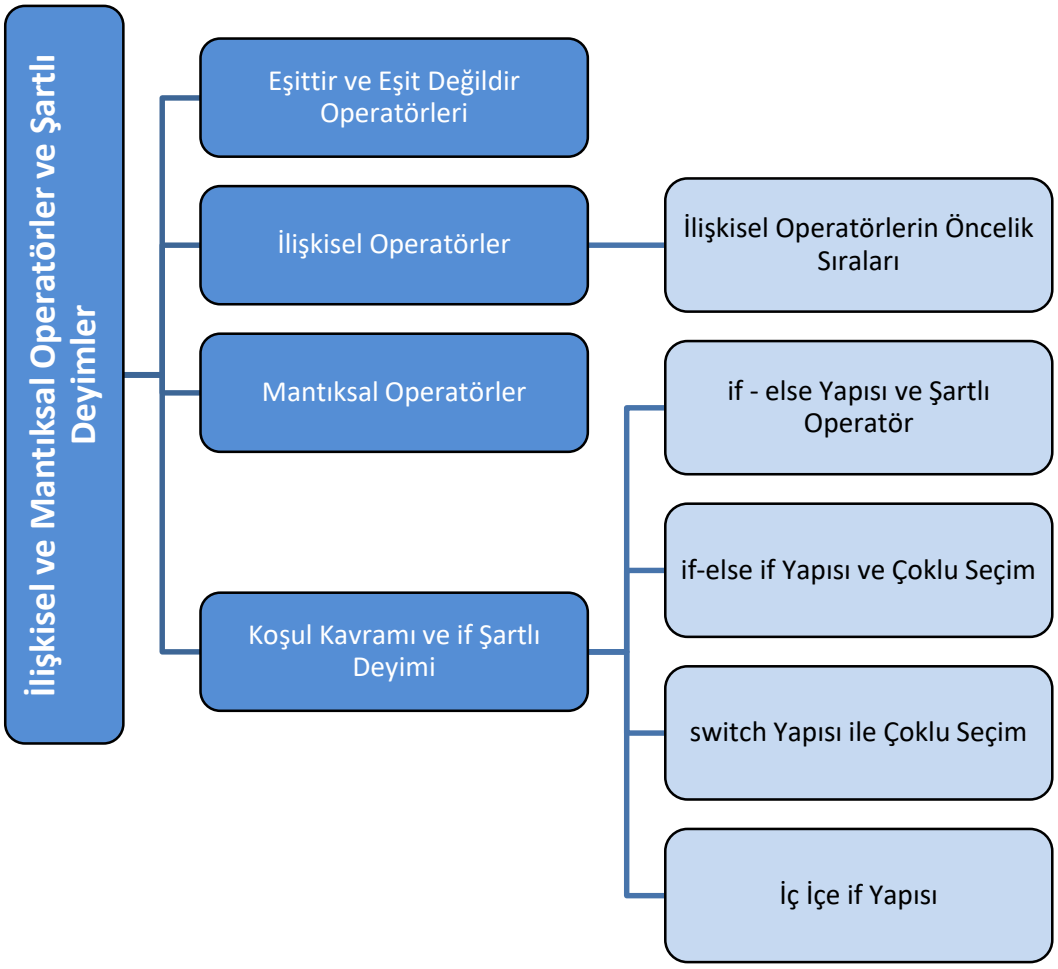
- Bu üniteyi çalıştıktan sonra;
  - Eşitlik ve eşitsizlik operatörünü öğrenebilecek,
  - İlişkisel operatörlerin hangileri olduğunu ve öncelik sıralarını kavrayabilecek,
  - Mantıksal operatörleri ve doğruluk tablolarını anlayabilecek,
  - Koşul kavramını ve C++ programlama dilinde var olan farklı kontrol yapılarını öğrenebileceksiniz.

Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

Dr. Öğr. Üyesi
Bülent TUĞRUL

ÜNİTE
5


İlişkisel ve Mantıksal Operatörler ve Şartlı Deyimler
Eşittir ve Eşit Değildir Operatörleri
İlişkisel Operatörler
İlişkisel Operatörlerin Öncelik Sıraları
Mantıksal Operatörler
if - else Yapısı ve Şartlı Operatör
if-else if Yapısı ve Çoklu Seçim
Koşul Kavramı ve if Şartlı Deyimi
switch Yapısı ile Çoklu Seçim
İç İçe if Yapısı

GİRİŞ

Bilgisayar programları basit olarak hesap, bildirim, komut ve karar yapılarından oluşurlar. Bilgisayara komut verebilecek bir yazılım aracının programlama dili olarak kabul edilebilmesinin temel şartlarından biri, karar yapılarının olmasıdır. Karar yapıları programlama dillerinde bir koşulun doğru veya yanlış olmasına göre programın akış yönünü belirler.

Hızlı ve doğru bir program yazmak için karar yapıları dikkatle kullanılmalıdır.

C++ dilinde karar yapıları, bu ünitede anlatılacak olan if – else yapısı ve 6. ünitede anlatılacak olan döngülerdir. if – else karar mekanizması temel olarak bir karşılaştırma sonucunun doğru olduğu durumda if kod bloğunu, yanlış olduğu durumda ise else kod bloğunu çalıştırır. Bu durumun önemi programdaki komut akış yönünü değiştirmesidir. Böylece iki yoldan sadece birisinin çalıştırılması hedeflenir. Bu ünitede bir C++ programı geliştirmek için gerekli olan eşittir operatörünü, karar mekanizmalarında kullanılan ilişkisel ve mantıksal operatörleri, her iki operatörün öncelik sıralarını, çeşitli if-else ve switch yapılarını ve kullanımlarını inceleyeceğiz.

Bu ünitede ve diğer ünitelerde yer alan C++ kod bloklarının çalıştırılabilmesi için söz konusu kodların kopyalanması ve aşağıda verilen şablon kodun (minimum C++ program iskeletinin) içerisine yerleştirilmesi gerekmektedir. Önceki bölümlerde anlatılan bir C++ programının çalıştırılması için gerekli adımlar dikkate alınmalıdır.

```
#include <iostream>
using namespace std;
int main()
{
  // Kod bloğunun ekleneceği alan
  return 0;
}
```

EŞİTTİR VE EŞİT DEĞİLDİR OPERATÖRLERİ

Değişkenler, bir programlama dilinin en önemli yapı taşlarından biridir. Programcılar yeterli sayıda değişken tanımlayarak ve bu değişkenlere değerler atayarak elde etmek istedikleri sonuca ulaşırlar. Bir değişkene bir değer atamak için atama (=) operatörünün kullanılması gerektiğinden önceki ünitelerde bahsedilmişti. Programcıların aynı zamanda iki farklı değişkenin değerlerinin birbiri ile olan ilişkilerini kontrol etmeleri gerekebilir. Bu kısımda iki farklı değişkenin değerlerinin eşit olup olmama durumunun nasıl kontrol edilebileceği anlatılacaktır. Sonrasında ise değişken değerlerinin birbirlerinden büyük veya küçük olma durumları incelenecektir.

C++ programlama dilinde iki değişkenin değerlerinin birbirine eşit olduğunu kontrol etmek için eşittir (==) veya eşit olmadığını kontrol etmek için eşit değildir (!=) operatörleri kullanılır. Dolayısıyla atama ve eşitlik kontrolü birçok programlama dilinde olduğu gibi C++ dilinde de farklı iki operatör olarak tanımlanmıştır. Her iki operatörün kullanım yerlerine dikkat edilmesi gerekmektedir.

Değişkenlere değer atama ve eşittir operatörleri farklı operatörlerdir. Kullanım yerlerine dikkat edilmelidir.

Tablo 5.1. Eşittir ve Eşit Değildir Operatörleri

Operatör	Ad	Örnek	Anlamı
==	Eşittir	x == y	x değeri y değerine eşit midir?
!=	Eşit Değildir	x != y	x değeri y değerine eşit değil midir?

Eşittir operatörü karşılaştırılan iki sayının/değişkenin değeri birbirine eşit ise geriye sonuç olarak doğru (true), değilse yanlış (false) değer döndürür. Eşit değildir operatörü ise iki sayı/değişken birbirine eşit ise geriye yanlış, değilse doğru değer döndürür. true ve false değerleri bool veri tipine ait değerlerdir (C++ programlama dilinin eski versiyonlarında false için 0, true için 0'dan farklı bir sayı kullanılmaktaydı.).

```
int x = 5, y = 5, z = 6;
if(x == y)
        cout << "x ve y degiskenlerinin degerleri esittir" << endl;
if(x == z)
        cout << "x ve z degiskenlerinin degerleri esit degildir";
```

Yukarıda verilen örnek kod parçasında x, y ve z birer tam sayı değişken olmak üzere tanımlanmışlardır (deklare edilmişlerdir). x ve y değişkenlerine 5, z değişkenine ise 6 değeri ilk değer olarak atanmıştır. Bu durumda x ve y değişkenlerinin değerleri birbirine eşit iken x değişkeni z değerine eşit değildir. Dolayısıyla ilk if şartlı deyimine ait koşul doğru olduğu için ekrana *"x ve y degiskenlerinin degerleri esittir"* ifadesi yazılacaktır. Ama x ve z değişkenleri birbirlerine eşit olmadıkları için ikinci if şartlı deyiminin gövdesi çalıştırılmayacaktır.

```
int x = 5, y = 5, z = 6;
if (x != y)
        cout << "x ve y degiskenlerinin degerleri esittir" << endl;
if (x != z)
        cout << "x ve z degiskenlerinin degerleri esit degildir";
```

Önceki verilen örnekte sadece if şartlı deyimlerine ait koşullarda kullanılan eşittir operatörü yerine eşit değildir operatörü kullanılmıştır. Bu durumda ikinci if şartlı deyimine ait koşul doğru olacağı için *"x ve z degiskenlerinin degerleri esit degildir"* ifadesi ekrana yazdırılacaktır.

Sınır değerler kontrol edilirken dikkat edilmelidir. Eşitlik durumu varsa büyük eşit veya küçük eşit kullanılmalıdır.

İLİŞKİSEL OPERATÖRLER

C++ programlama dili, aritmetik operatörlerin yanında hem ilişkisel hem de mantıksal operatörleri destekler. İlişkisel ifadeler bir koşulun doğruluğunu kontrol etmek için kullanılır. Bir ilişkisel ifade iki işlenen ve bir ilişkisel operatör ile oluşturulur. Bir ilişkisel ifadenin sonucu bool veri tipindedir. Büyüktür, büyük eşittir, küçüktür ve küçük eşittir olmak üzere 4 adet ilişkisel operatör tanımlıdır. Bu operatörlerin türleri, kullanımları ve ne anlama geldikleri aşağıdaki tabloda verilmiştir.

Tablo 5.2. İlişkisel Operatörler

Operatör	Ad	Örnek	Anlamı
>	Büyüktür	x > y	x değeri y değerinden büyüktür
>=	Büyük eşit	x >= y	x değeri y değerinden büyük veya eşittir
<	Küçüktür	x < y	x değeri y değerinden küçüktür
<=	Küçük eşit	x <= y	x değeri y değerinden küçük veya eşittir

```
int x = 5, y = 3, z =5;
if (x > y)
        cout << "x degiskeninin degeri y degiskeninin degerinden buyuktur" << endl;
if (x >= z)
        cout << "x degiskeninin degeri z degiskenin degerinden buyuk veya esittir";
```

Yukarıda verilen örnek kod parçasında x, y ve z birer tam sayı değişken olmak üzere tanımlanmışlardır. x ve z değişkenlerine 5, y değişkenine ise 3 değeri atanmıştır. Bu durumda x değişkeninin değeri y değişkeninin değerinden büyüktür. Böylece ilk if şartlı deyimine ait koşulun sonucu doğru döneceği için *"x degiskeninin degeri y degiskeninin degerinden buyuktur"* ifadesi ekrana yazdırılacaktır. İkinci if şartlı deyimine ait koşul x değişkeninin değerinin z değişkeninin değerinden büyük veya eşit olması durumunu kontrol etmektedir. Değişkenlerin değerlerini incelediğimizde ikinci if şartlı deyimine ait koşulun da geriye doğru değer döndüreceği ve ekrana *"x degiskeninin degeri z degiskenin degerinden buyuk veya esittir"* ifadesinin yazdırılacağı anlaşılacaktır.

İlişkisel Operatörlerin Öncelik Sıraları

C++ programlama dilinde tanımlanmış operatörlerin öncelik sıraları birbirlerinden farklıdır. Doğru bir ilişkisel, mantıksal veya matematiksel ifade yazabilmek için operatörlerin öncelik sıralarının bilinmesi gerekmektedir. İlişkisel operatörlerin atama operatörüne göre öncelikleri vardır. Örneğin; x = y < z ifadesinde öncelikli olarak y < z ifadesi hesap edilir ve elde edilen sonuç x değişkenine atanır. Bu tür kullanımlar bazen kafa karıştırıcı olabilir; ama parantez kullanımı her zaman daha açık ifadeler oluşturmak için fayda sağlar. Böylece kod bloğunu hem yazan hem de kullanan kişi en doğru şekliyle gerçeklemiş olur. Yukarıdaki kod parçasının eşleniği aşağıdaki gibi olacaktır:

```
x = (y < z);
```

Doğru hesaplamalar yapmak için operatörlerin öncelik sıraları bilinmelidir.

Aritmetik operatörler ise ilişkisel operatörlere göre daha yüksek önceliğe sahiptirler. Örneğin; x + 1 > y * 2 ifadesinde önce aritmetik işlemler yapılır, daha sonra büyüktür ifadesinin sol tarafı ile sağ tarafı karşılaştırılır. Aynı ifadenin parantez kullanılarak yazılması durumunda ise eşleniği şu şekilde olacaktır:

```
(x + 1) > (y * 2);
```

MANTIKSAL OPERATÖRLER

Basit koşullar oluşturmak için ilişkisel operatörler yeterlidir; ama daha karmaşık koşulların oluşturulması ancak mantıksal operatörler ile mümkündür.

Örneğin; hastanın şekeri 135'den büyük, tansiyonu 10'dan küçük ve kalp atışı 80'den büyük ise acile yönlendir gibi daha karmaşık durumları mantıksal operatörler kullanarak kolayca gerçekleyebiliriz. C++ programlama dilinde && (VE), || (VEYA), ! (DEĞİL) olmak üzere 3 adet mantıksal operatör vardır. Mantıksal operatörler eşitlik ve ilişkisel operatörlere göre daha düşük ama atama operatörüne göre daha yüksek önceliğe sahiptir.

Tablo 5.3. Mantıksal Operatörler

Operatör	Ad	Örnek	Anlamı
&&	VE	x < 5 && y > 10	x 5'den küçük ve y 10'dan büyük ise doğru
||	VEYA	x < 5 || y > 10	x 5'den küçük veya y'10 dan büyük ise doğru
!	DEĞİL	! (x < 5)	x değeri 5'den küçük değilse doğru

Aşağıda verilen tablo iki farklı koşulun alabileceği bütün olası durumları ve Mantıksal VE operatör sonuçlarını göstermektedir. Tablodan anlaşılacağı üzere iki koşulun aynı anda doğru olması durumunda Mantıksal VE operatörü doğru değer döndürmektedir. Diğer bütün durumlarda yanlış değer döndürmektedir. Örneğin; kullanıcı giriş ekranında kullanıcıdan alacağımız hem kullanıcı adının hem de şifrenin doğru girilmesi durumunu kontrol etmek için Mantıksal VE operatörünü kullanmamız uygun olur. Kullanıcının sisteme giriş yapabilmesi için her iki bilgiyi doğru şekilde bilmesi ve girmesi gerekmektedir. Aksi takdirde kullanıcı adı veya şifresinin biri veya her ikisi birden yanlış ise sisteme giriş izni verilmez.

Tablo 5.4. Mantıksal VE Operatörünün Doğruluk Tablosu

Koşul_1	Koşul_2	Koşul_1 && Koşul_2
Doğru	Doğru	Doğru
Doğru	Yanlış	Yanlış
Yanlış	Doğru	Yanlış
Yanlış	Yanlış	Yanlış

Aşağıda verilen tabloda yine iki farklı koşulun alabileceği bütün durumlar ve Mantıksal VEYA operatörünün sonuçları verilmiştir. Tablodan da anlaşılacağı üzere Mantıksal VEYA operatörü her iki koşulun aynı anda yanlış değere sahip olması durumda yanlış sonuç vermektedir. Diğer bütün durumlarda ise geriye doğru sonuç döndürmektedir. Kullanıcı giriş ekranında VE operatörü yerine VEYA operatörü kullanmak istenmeyen sonuçların oluşmasına neden olabilir. Bir kişiye kredi verilebilmesi için kişinin arabasının veya evinin olmasının yeterli olması durumunda VEYA operatörü kullanılmalıdır. Bu durumda bir kişinin sadece arabaya veya eve veya her ikisine sahip olması durumunda kredi alması sağlanacaktır.

Tablo 5.5. Mantıksal VEYA Operatörünün Doğruluk Tablosu

Koşul_1	Koşul_2	Koşul_1 || Koşul_2
Doğru	Doğru	Doğru
Doğru	Yanlış	Doğru
Yanlış	Doğru	Doğru
Yanlış	Yanlış	Yanlış

Bir sayının belirli bir aralıkta olup olmadığını kontrol etmek için Mantıksal VE operatörü kullanılmalıdır.

Mantıksal VE ve VEYA operatörleri programların daha hızlı çalışması için kısa devre değerlendirme denen bir yöntem ile birden fazla koşul olması durumunda sonraki koşulların değerlerini dikkate almadan karar verebilirler. Mantıksal VE operatörü ile bağlanan birden fazla koşulun yalnız bir tanesinin yanlış olması sonucu yanlış olarak döndürmek için yeterlidir. Böylece VE operatörü koşullardan birinin yanlış değere sahip olduğunu fark ettiğinde sonraki koşulları hesap etmek için zaman kaybetmeden yanlış sonucunu döndürür. Başka bir deyişle iki koşullu bir Mantıksal VE ifadesinde ilk koşul yanlış ise sonuç yanlıştır ve ikinci koşulun değerlendirilmesine gerek yoktur. Benzer durum VEYA operatörü için de geçerlidir. VEYA operatörü ile bağlanan birden fazla koşulun bir tanesinin doğru olması VEYA operatörünün doğru sonuç döndürmesi için yeterlidir. Başka bir ifade ile iki koşul ile bağlanan bir VEYA ifadesinde ilk koşul doğru ise ikinci koşulun değerinin bir önemi yoktur ve VEYA operatörü sonuç olarak geriye doğru değer döndürür.

Kısa devre değerlendirme yöntemi bazen beklenmedik sonuçlara sebep olmaktadır. Dolayısıyla mantıksal operatörleri kullanırken bu durum akıldan çıkarılmamalıdır. Örneğin, programcı kısa devre değerlendirme yöntemini dikkate almadan yazdığı (a < 5) || (b++ < 5) ifadesinde b değişkeninin değerinin her durumda arttırılacağını düşünebilir. Öte yandan a değişkeninin değeri eğer 5'den küçük ise ilk koşul doğru ve VEYA operatörü ikinci koşulu dikkate almayacağı için b değişkeninin değeri arttırılmaz. Dolayısıyla böyle bir durumda b değişkeni açısından yanlış bir değer elde edilir.

Tablo 5.6. Mantıksal Değil Operatörünün Doğruluk Tablosu

Koşul	**! Koşul**
Doğru	Yanlış
Yanlış	Doğru

Mantıksal Değil operatörü koşul değeri doğru ise geriye yanlış, yanlış ise doğru değer döndürür.

```
int x = 3;
if (!(x >= 5))
        cout << "x degiskeninin degeri 5 den kucuktur" << endl;
```

Yukarıda verilen kod parçasında x değişkeni başlangıç değeri 3 olan bir tam sayı olarak belirlenmiştir. if şartlı deyiminde (x >= 5) koşulu x değişkeninin 5'den büyük olup olmadığını kontrol eder. Bu durumda x değişkeninin değeri 5'den küçük olduğu için koşul yanlış değer döndürür. Fakat mantıksal değil operatörü yanlış değeri tam tersine çevirdiği için if şartlı deyiminin koşul değeri doğru olur ve böylece if şartlı ifadesinin gövdesi çalıştırılır ve *"x degiskeninin degeri 5 den kucuktur"* ifadesi ekrana yazdırılır.

KOŞUL KAVRAMI VE IF ŞARTLI DEYİMİ

Programın akışını kontrol etmek bir programcı için çok önemlidir. Programlama dillerinde koşullu ifadeler kullanılarak programın akışı kontrol altına alınabilir. Örneğin, bir öğrencinin bir dersten aldığı notun 50'den çok veya az olmasına bağlı olarak iki farklı aksiyonun uygulanması gerekebilir. Gerçek bir

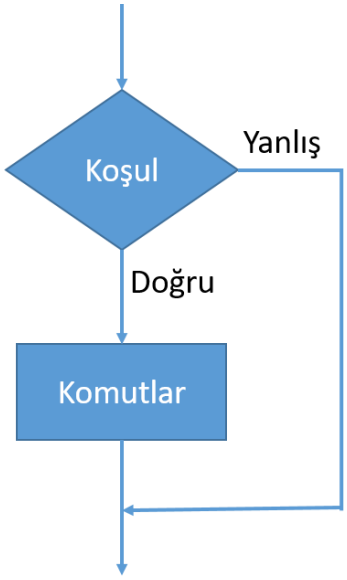
Uygun satır girintisi kullanarak kod yazmak, yazılan kodun okunabilirliğini arttırır.

puanlama sisteminde ise daha çok seçenek söz konusudur. Bu bölümde iki veya daha fazla kararın verilmesi gereken durumlarda C++ programlama dilinde kullanabileceğimiz kontrol yapılarından bahsedilecektir.

Bu yapılardan en önemlisi ve sıklıkla kullanılanı if şartlı deyimidir. if şartlı deyimi bir koşulun doğru olup olmamasına bağlı olarak belirli bir kod bloğunun çalıştırılıp çalıştırılmamasına karar verir. Örneğin, hava sıcaklığı 30 °C'den fazla ise klimayı çalıştır ifadesinde klimanın çalışması her zaman değil ama hava sıcaklığının belirlenen değeri geçmesi durumunda istenmektedir. Bu durumu gerçeklemek için yazılması gereken kod bloğu aşağıdaki gibi olmalıdır:

```
if (sicaklık >= 30)
        cout << "Klimayi ac" << endl;
```

Eğer if şartlı deyiminin koşulu doğru ise (yani sıcaklık değeri 30'a eşit veya büyük ise) hemen sonraki ilk satır çalıştırılır ve ekrana *"Klimayi ac"* yazdırılır. Yanlış ise if şartlı deyimine ait satır çalıştırılmadan program sonraki satırları çalıştırmaya devam eder. if şartlı deyiminin akış diyagramı Şekil 5.1 ile verilmiştir.



Şekil 5.1. If Şartlı Deyiminin Akış Diyagramı

Eğer if şartlı deyimine ait tek bir deyim değil de bir blok ifade oluşturulması isteniyorsa;

```
{
.
.
.
}
```

yapısı kullanılmalıdır.

```
int sicaklık = 35;
if (sicaklık >= 30)
{
```

```
        cout << "Pencereyi kapat" << endl;
        cout << "Klimayi ac" << endl;
}
```

Bu durumda sıcaklık değerinin 30'dan büyük olduğu gözükmektedir ve if şartlı deyimine ait koşul doğru değer döndüreceği için deyimin gövdesine ait olan iki satır çalıştırılacaktır.

if-else Yapısı ve Şartlı Operatör

if şartlı deyimi koşulun doğru olması durumunda belirli bir kod bloğunun çalışmasını sağlar. Program eğer koşul yanlış ise kendisinden sonra gelen satırdan çalışmaya devam eder. Ama bazı durumlarda if şartlı deyiminin yanlış olması durumunda da başka bir kod bloğunun çalıştırılması gerekebilir. Bu durumda if-else yapısı kullanılır. if-else yapısı aşağıdaki formatta kullanılmalıdır:

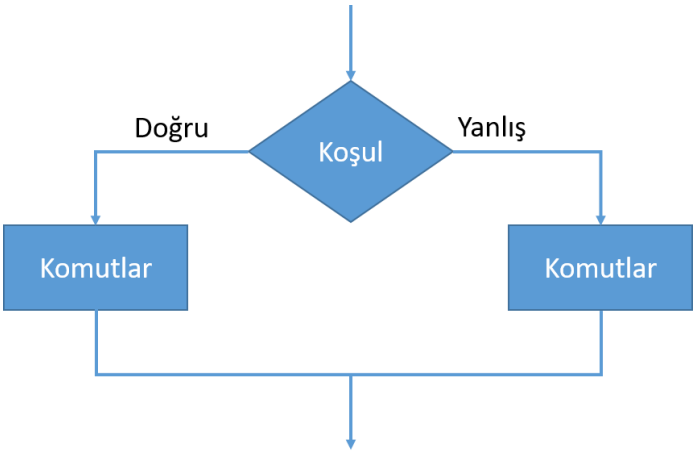
```
if (koşul)
        komut_1;
else
        komut_2;
```

Bu durumda if şartlı deyimine bağlı koşul doğru ise komut_1 çalıştırılır, değilse komut_2 çalıştırılır. Dolayısıyla koşulun doğru veya yanlış olmasına bağlı olarak iki farklı kod bloğundan birinin çalıştırılması sağlanır.

```
if (ortalama >= 50)
        cout << "Dersten gectin" << endl;
else
        cout<<"Dersten kaldin"<<endl;
```

Koşulun doğru olmasında bir yol, olmaması durumunda başka bir yol takip edilmesi gerekiyorsa if-else yapısı kullanılmalıdır.

Yukarıdaki kod parçası bir öğrencinin ortalamasının 50'ye eşit veya daha fazla olması durumda ekrana *"Dersten gectin"* yazdıracaktır. Ama ortalamanın 50'nin altında olması durumunda *"Dersten kaldin"* yazdıracaktır. if-else şartlı deyiminin akış diyagramı Şekil 5.2 ile verilmiştir.



Şekil 5.2. If-Else Yapısının Akış Diyagramı

C++ programlama dilinde if-else yapısı ile aynı işleve sahip bir şartlı deyim tanımlanmıştır. Şartlı deyim if-else yapısı ile yazılmış kodun daha kısa bir şekilde

Şartlı operatör if-else yapısı yerine kullanılabilir.

yazılmasına olanak sağlar. Şartlı operatör ?, : ve ; sembolleri kullanılarak oluşturulur. Genel yapısı aşağıdaki gibidir:

koşul ? komut_1: komut_2;

Burada koşul doğru ise komut_1, yanlış ise komut_2 çalıştırılır. Şartlı deyim ile ikiden fazla yolun takip edilmesi mümkün değildir. Yukarıda if-else kullanılarak oluşturulan örneğin şartlı deyim olan eşleniği aşağıdaki gibidir:

```
cout << ((ortalama >= 50) ? "Dersten gectin" : "Dersten kaldin");
```

Bireysel Etkinlik

- Aşağıda verilen ve if-else yapısını kullanan kod bloğunu şartlı operatör kullanarak yeniden oluşturun.

```
int a = 5, b;
if (a > 5)
        b = 10;
else
        b = 100;
```

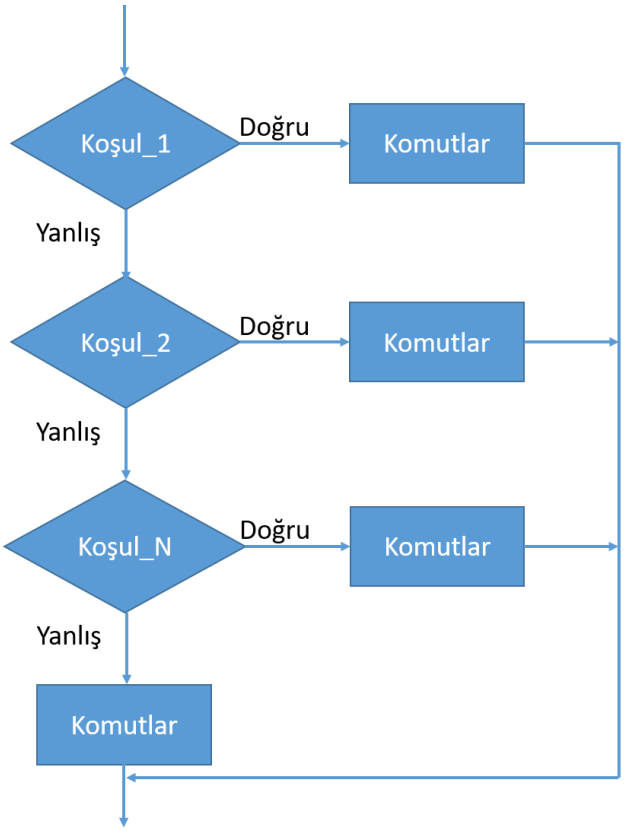
if-else if Yapısı ile Çoklu Seçim

If-else if yapısı ile birden fazla koşul aşağıdaki gibi test edilebilir:

```
if (koşul_1)
        cout << "koşul_1 doğru ise çalıştır" << endl;
else if(koşul_2)
        cout << "koşul_2 doğru ise çalıştır" << endl;
else
        cout << "koşulların hiçbiri doğru değil ise çalıştır";
```

Birden fazla koşulun kontrol edilmesi gerekiyor ise ya if-else if ya da switch çoklu seçim yapıları kullanılabilir.

Yukarıda genel yapısı verilen if-else if yapısında koşul_1 doğru ise hemen altındaki kod bloğu çalıştırılır ve sonraki satırlar dikkate alınmaz. Eğer koşul_1 yanlış ise koşul_2 test edilir. Koşul_2 doğru ise ikinci if yapısından sonraki kod bloğu çalıştırılır ve sonraki satırlar dikkate alınmaz. Eğer hem koşul_1 hem de koşul_2 yanlış ise en son else ifadesine ait kod bloğu çalıştırılır ve if-else if yapısından çıkılır. Bu durumda iki adet koşul test edilmiş ve hiçbiri doğru değil ise en son kod bloğu çalıştırılmıştır. Kullanılacak koşul sayısı test edilmek istenen duruma bağlı olarak arttırılabilir. if-else if şartlı deyiminin akış diyagramı Şekil 5.3 ile verilmiştir.



Şekil 5.3. if-else if Yapısının Akış Diyagramı

Ortalama	Harf Notu
91 - 100	A
81 - 90	B
71-80	C
61-70	D
0 - 60	E

Yukarıda verilen tablo dikkate alınarak öğrencilerin harf notunun belirlenmesi söz konusu ise çözüm if-else if yapısı ile aşağıdaki gibi oluşturulabilir:

Sınır değer testleri özenle yapılmalıdır.

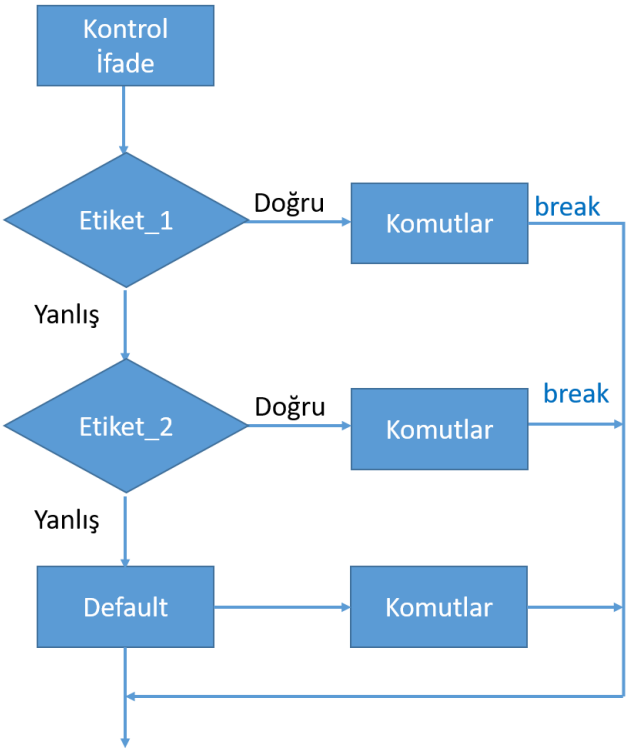
```
if (ortalama > 90)
        cout << "A" << endl;
else if (ortalama > 80)
        cout << "B" << endl;
else if (ortalama > 70)
        cout << "C" << endl;
else if (ortalama > 60)
        cout << "D" << endl;
else
        cout << "E" << endl;
```

Burada dikkat edilmesi gereken husus bir tam sayının 90'dan küçük ve 80'den büyük olması durumunda 80 ile 90 arasında olacağıdır. Benzer durum diğer sayı aralıkları için de geçerlidir.

switch Yapısı ile Çoklu Seçim

switch yapısı ile bir değişkenin veya ifadenin değerine göre birden çok yoldan sadece birinin çalıştırılmasına karar verilir. Bir switch yapısı bir adet kontrol ifadesinden, bir veya daha fazla *case* etiketinden, zorunluluk olmamakla birlikte *default* etiketinden ve *break* ifadesinden oluşur. Bir switch yapısının genel formatı aşağıda verildiği gibidir. switch şartlı deyiminin akış diyagramı ise Şekil 5.4 ile verilmiştir.

```
switch (kontrol_ifadesi)
{
        case etiket_1:
                cout << "Birinci case ifadesi" << endl;
                break;
        case etiket_2:
                cout << "Ikinci case ifadesi" << endl;
                break;
                . . .
        default:
                cout << "default durumu" << endl;
                break;
}
```



Şekil 5.4. Switch Yapısının Akış Diyagramı

Bir program switch yapısına geldiği zaman ilk iş olarak kontrol ifadesinin değerini belirler. Kontrol ifadesi bir tam sayı olabilir. char veri tipinde tanımlanmış bir değişken aynı zamanda tam sayı olarak da kullanılabilir. Bir karakterin tam sayı değer karşılığı aşağıdaki kod parçasının çalıştırılmasıyla belirlenebilir:

Bir karakterin ASCII standardına göre tam sayı karşılığını elde etmek için static_cast<int> kullanılabilir.

```
cout << static_cast< int > ('a') << endl;
```

Yukarıda kod parçası 'a' karakterinin ASCII standardında tam sayı değerini ekrana yazdırır. Sonra kontrol ifadesinin değeri ile ilk case etiketinden başlamak üzere karşılaştırır. Kontrol ifadesine eşit case etiketine ait kod bloğu break ifadesine kadar çalıştırılır. break ifadesinin bulunduğu yerden switch yapısının sonuna kadar olan kod atlanarak switch yapısından çıkılır. Eğer kontrol ifadesi hiçbir case etiketine eşit değilse ve eğer mevcutsa default etiketine ait kod bloğu çalıştırılır. Default etiketi tanımlanmamış ve kontrol değeri hiçbir case etiketine eşit değil ise program switch yapısındaki hiç bir kod bloğunu çalıştırmadan kendisinden sonra gelen ilk satırdan yoluna devam eder.

```
switch (harfnotu)
{
        case 'A':
                cout << "Ortalaman 91 ile 100 arasi"<< endl;
                break;
        case 'B':
                cout << "Ortalaman 81 ile 90 arasi" << endl;
                break;
        case 'C':
                cout << "Ortalaman 71 ile 80 arasi" << endl;
                break;
        case 'D':
                cout << "Ortalaman 61 ile 70 arasi" << endl;
                break;
        case 'E':
                cout << "Ortalaman 0 ile 60 arasi" << endl;
                break;
        default:
                cout << "Harf notu bilinmiyor" << endl;
                break;
}
```

Yukarıdaki kod bloğu bir öğrencin harf notunun bilinmesi durumunda öğrencinin ders ortalamasının hangi değerler arasında olduğunu belirlemek için yazılmıştır. Öğrencinin harf notuna bağlı olarak uygun case veya default etiketlerden birine ait satırların çalıştırılması sağlanmıştır. Örneğin, öğrencinin harf notu 'B' ise ekrana *"Ortalaman 81 ile 90 arasi"* yazdırılacaktır. Eğer yukarıdaki kod parçasında bilerek veya yanlışlıkla break deyimleri yazılmasaydı ve öğrencinin notu 'A' olsaydı aşağıda verilen ekran çıktısı elde edilmiş olacaktı.

```
Ortalaman 91 ile 100 arasi
Ortalaman 81 ile 90 arasi
Ortalaman 71 ile 80 arasi
Ortalaman 61 ile 70 arasi
Ortalaman 0 ile 60 arasi
Harf notu bilinmiyor
```

Şekil 5.5. Yanlış break Kullanımına Ait Örnek

switch yapısında break ifadesinin unutulması beklenmedik sonuçlar doğurabilir.

Bu durumda öğrencinin harf notu 'A' olduğu için ilk case etiketine girilecek ama sonrasında hiçbir break deyimi olmadığı için diğer bütün case ve default etiketlerine ait satırların da çalıştırılması sağlanacaktı.

Bireysel Etkinlik

- Yukarıda verilen örnek switch yapısını gerçekleştirerek elde edeceğiniz ekran çıktısı ile break kullanmadan elde edeceğiniz ekran çıktısını karşılaştırınız.

İç İçe if Yapısı

Bazı durumlarda bir kontrol yapısının içerisinde başka bir kontrol yapısı kullanmamız gerekebilir. Bu tür yapılara iç içe yapılar denir. Eğer bir koşulun sağlandığı durumda başka koşulların da kontrol edilmesi gerekiyor ise iç içe if yapıları kullanılarak istenen amaca ulaşılır. Bu yapılar bir örnek üzerinden anlatılacaktır. Kullanıcıdan alınan üç tam sayının en büyüğünü bulan ve ekrana yazdıran bir program iç içe if yapıları kullanılarak yazılmak istenirse örnek kod aşağıdaki gibi olacaktır:

```
int a, b, c, mak;
cout << "3 tam sayi giriniz: ";
cin >> a >> b >> c;
if (a > b)
{
        if (a > c)
                mak = a;
        else
                mak = c;
}
else
        if (b > c)
                mak = b;
        else
                mak = c;
cout << "En buyuk sayi: " << mak << endl;
```

İç içe if yapılarında else kullanımına dikkat edilmelidir.

Yukarıdaki örnekte ilk olarak kullanıcıdan girilecek tam sayıların değerlerini tutmak üzere 3 adet tam sayı değişkeni ve içerisinde en büyük değerin saklanacağı yine bir tam sayı değişkeni olan mak tanımlanmıştır. Sonrasında kullanıcının klavye aracılığıyla girmiş olduğu 3 tam sayı okunmuş ve her biri sırasıyla a, b ve c değişkenlerine atanmıştır. Daha sonra girilen değerler birbirleriyle karşılaştırılarak girilen en büyük değer tespit edilmiştir. İlk olarak a değeri b değerinden büyük ise a ve c değerlerinin birbiriyle olan durumuna göre mak değişkenine atama yapılmıştır. Eğer a değeri b'den küçük ise en büyük değere b ile c değeri arasındaki

duruma göre karar verilmiştir. Verilen örnekte dikkat edilmesi gereken hangi else ifadesinin hangi if yapısına ait olduğu ve iyi programcılık uygulaması olarak satır girintilere dikkat edilmesidir.

Bireysel Etkinlik

- Yukarıda örnek 3 adet tam sayının en büyüğünü tespit etmektedir. Siz de kod üzerinde gerekli değişiklikleri yaparak bu uygulamayı en küçük sayıyı bulacak hale dönüştürün.

```
if ( a == 0 )
        if ( b == 0 )
                c = 0;
else
        c = 1;
```

Yukarıda verilen kod bloğu incelendiğinde girinti kullanımına göre else ifadesinin ilk if yapısına ait olduğu düşünülebilir; ama gerçekte ikinci if yapısına aittir. Yani else kendisinden bir önceki if yapısı ile ilişkilendirilir. Eğer yukarıdaki örnekte olduğu gibi else ifadesini birinci else ifadesi ile ilişkilendirilmek istenirse "{…}" kullanılmadır. Bu duruma ait örnek aşağıda verilmiştir:

```
if (a == 0)
{
        if (b == 0)
                c = 0;
}
else
        c = 1;
```

Bireysel Etkinlik

- Yukarıda verilen her iki örneği gerçekleyin ve aralarındaki farkı kavrayarak iç içe if yapılarında else kullanımını pekiştirin.

Özet

- Bir program hesap, bildirim, komut ve karar yapıları kullanılarak geliştirilir. Karar yapıları bir programın akış yönüne karar vermek için kullanılır. Bu bölümde if, if-else, if-else if ve switch kontrol yapıları açıklanmıştır. Ama daha öncesinde kontrol yapılarını oluşturabilmek için gerekli ilişkisel ve mantıksal operatörler tanıtılmıştır.
- İlişkisel ifadeler bir koşulun doğrulunu kontrol etmek için kullanılır. Bir ilişkisel ifade, bir ilişkisel operatör ve iki işlenen kullanılarak oluşturulur. Bu operatörler büyüktür (>), büyük eşittir (>=), küçüktür (<) ve küçük eşittir (<=) olmak üzere 4 adettir.
- İlişkisel operatörlere ek olarak eşittir (==) ve eşit değildir (!=) operatörleri iki değerin birbirine eşit olup olmama durumlarını test etmek için kullanılır.
- İlişkisel operatörler ile basit koşullar oluşturulabilir. Daha karmaşık koşullar ise ancak mantıksal operatörlerin yardımı ile elde edilir. C++ programlama dilinde kullanabileceğimiz 3 farklı mantıksal operatör vardır. Bunlar VE (&&), VEYA (||) ve DEĞİL (!) operatörleridir.
- Mantıksal operatörlere ait doğruluk tablolarına bölüm içerisinde yer verilmiştir. İki koşul kullanarak bir mantıksal ifade oluşturduğumuzda eğer her iki koşul da doğru ise Mantıksal VE operatörü doğru sonuç verir; ama koşullardan en az biri yanlış ise yanlış sonuç üretilir. Öte yandan Mantıksal VEYA operatörü her iki koşulun da yanlış olması durumunda yanlış sonuç üretir. Diğer durumlarda geriye her zaman doğru sonuç döndürür. Mantıksal DEĞİL operatörü değeri doğru olan ifadeyi yanlış, yanlış olan ifadeyi doğru yapar. Mantıksal operatörler eşitlik ve ilişkisel operatörlere göre daha düşük ama atama operatörüne göre daha yüksek önceliğe sahiptirler.
- Kontrol yapıları bir programın akış yönüne karar verir ve program geliştirmede önemli bir yere sahiptir. Bir kod bloğunun çalışıp çalışmaması bazen bir koşulun sağlanıp sağlanmamasına bağlıdır. Bu gibi durumlarda if şartlı deyimi kullanılır. Koşulun sağlanması yani doğru olması durumunda if şartlı deyimi içerisinde bulunan satırların çalışması sağlanır. Eğer koşul sağlanmazsa yani yanlış ise if şartlı deyimine ait blok içerisinde bulunan satırlar çalıştırılmadan program akışı if şartlı deyiminden sonraki satıra yönlendirilir. Benzer şekilde koşulun sağlanmadığı durumda farklı komut satırlarının çalışmasını istiyorsak if-else yapısı kullanılmalıdır. Bu durumda koşul sağlanmazsa else yapısına ait komut satırları çalıştırılır. Yani iki yoldan biri tercih edilmiş olunur.
- Bazı karmaşık durumlarda ise koşul sayısı birden fazladır. Eğer koşul sayısı birden fazla ise if-else if ve switch çoklu seçim yapıları kullanılmalıdır.

DEĞERLENDİRME SORULARI

1. C++ programlama dilinde atama operatörü olarak aşağıdakilerden hangisi kullanılır?
   a) ==
   b) !=
   c) =
   d) >=
   e) >=

2. C++ programlama dilinde eşittir operatörü olarak aşağıdakilerden hangisi kullanılır?
   a) ==
   b) !=
   c) =
   d) >=
   e) <=

3. C++ programlama dilinde aşağıdakilerden hangisi tanımlanmış ilişkisel operatörlerden biri değildir?
   a) >
   b) <
   c) >=
   d) <=
   e) =

4. C++ programlama dilinde aşağıdakilerden hangisi "3 < x < 8" matematiksel durumunu kontrol etmek için yazılması gereken mantıksal ifadeye denktir?
   a) 3 < x && x < 8
   b) 3 <= x && x <= 8
   c) 3 < x || x < 8
   d) 3 <= x || x <= 8
   e) 3 < x !! x < 8

5.
```
int a = 5, b =10, c= 20;
if (a >= 5)
        cout<< 1 <<endl;
else if (b <= 10)
        cout<< 2 <<endl;
else if (b > 10)
        cout << 3 << endl;
else if (c < 10)
        cout << 4 << endl;
else
        cout << 5 << endl;
```

Yukarıda verilen kod bloğunun ekran çıktısı aşağıdakilerden hangisidir?

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

6.
```
int a = 10, b =10;
        switch (a + b)
        {
                case 10:
                        cout << 1 << endl;
                        break;
                case 20:
                        cout << 2 << endl;
                        break;
                case 30:
                        cout << 3 << endl;
                        break;
                case 40:
                        cout << 4 << endl;
                        break;
                default:
                        cout << 5 << endl;
                        break;
        }
```

Yukarıda verilen kod bloğunun ekran çıktısı aşağıdakilerden hangisidir?

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

7. C++ programlama dilinde mantıksal VE operatörü olarak aşağıdakilerden hangisi kullanılır?
   a) &&
   b) ||
   c) !
   d) ##
   e) !&

8. Büyük eşittir operatörünün doğru kullanımı aşağıdakilerden hangisinde verilmiştir?
   a) =>
   b) >=
   c) =>=
   d) !>
   e) >!

9. I. ?
   II. :
   III. ;
   IV. ,

   Yukarıdaki sembollerden hangileri şartlı operatör oluşturmak için kullanılır?
   a) I ve III
   b) II ve III
   c) II ve IV
   d) I, II ve III
   e) I, II ve IV

10. Bir koşul değerini yanlış ise doğru, doğru ise yanlış yapmak için hangi mantıksal operatör kullanılır?
    a) VE
    b) VEYA
    c) DEĞİL
    d) EŞİTTİR
    e) EŞİT DEĞİLDİR

Cevap Anahtarı

1.c, 2.a, 3.e, 4.a, 5.a, 6.b, 7.a, 8.b, 9.d, 10.a

YARARLANILAN KAYNAKLAR

[1] Özkan, Y. (2015). C++ Programlama Dili, 3. Baskı, Papatya Bilim.

[2] Deitel, P., & Deitel H. (2014). C++ How to Program, 9. Baskı, Pearson.

DÖNGÜLER

İÇİNDEKİLER

- Arttırma ve Azaltma Operatörleri
- Bileşik Atama Operatörleri
- Döngüler
  - Döngü Oluşturma Kuralları
  - for Döngüsü
  - while Döngüsü
  - break ve continue Deyimleri
  - İç İçe Döngüler

HEDEFLER

- Bu üniteyi çalıştıktan sonra;
  - Arttırma ve azaltma operatörlerini kullanabilecek,
  - Bileşik atama operatörlerini tanıyabilecek,
  - for ve while olmak üzere iki farklı döngü yapısı oluşturabilecek,
  - break ve continue deyimlerini döngüler içinde kullanabilecek,
  - İç içe döngüler oluşturabileceksiniz.

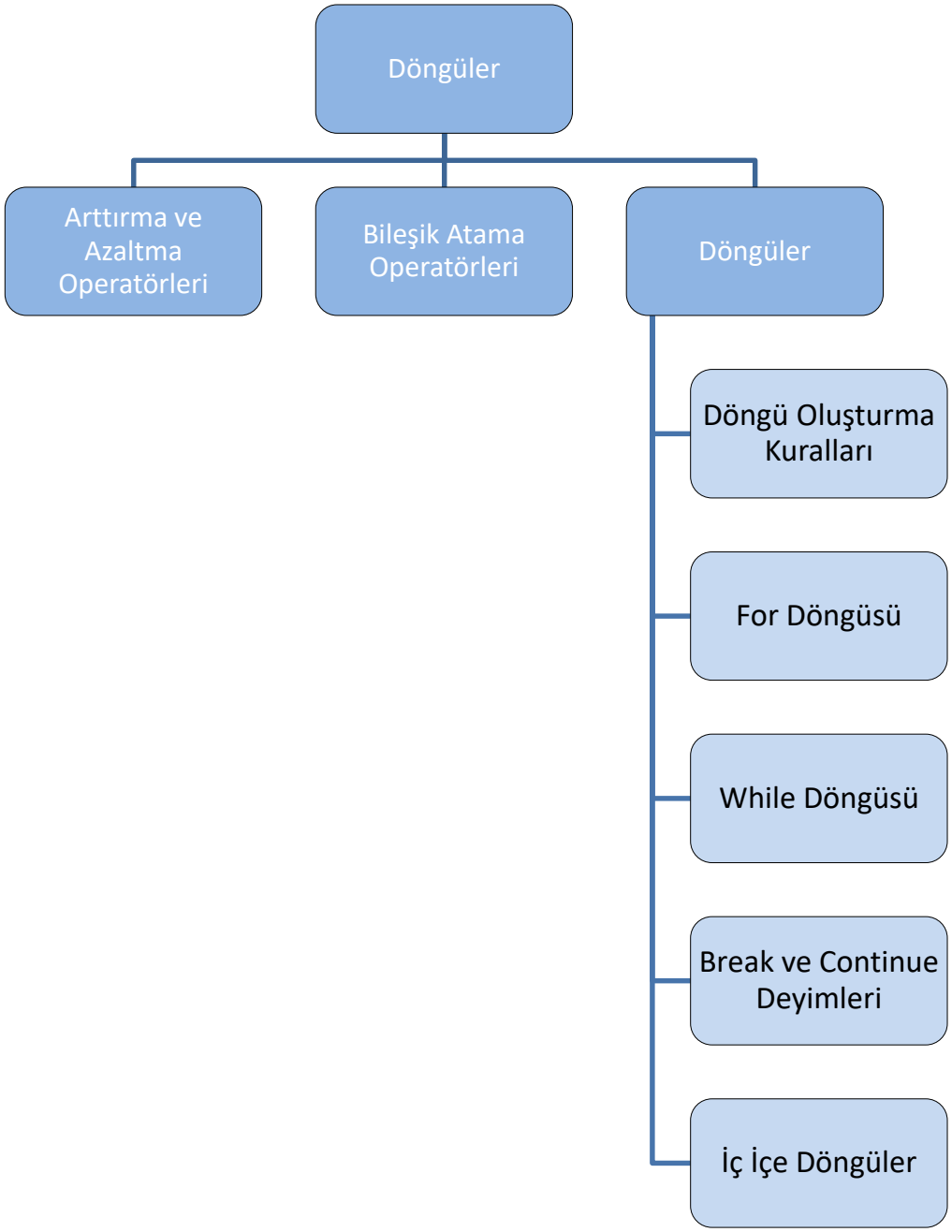
Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

Dr. Öğr. Üyesi
Bülent TUĞRUL

ÜNİTE

6


Döngüler
Arttırma ve Azaltma Operatörleri
Bileşik Atama Operatörleri
Döngüler
Döngü Oluşturma Kuralları
For Döngüsü
While Döngüsü
Break ve Continue Deyimleri
İç İçe Döngüler

GİRİŞ

Bu ünitede ilk olarak döngüleri oluştururken sıklıkla kullanacağımız arttırma/azaltma ve bileşik atama operatörleri anlatılacaktır. Daha sonra bir programlama dilinin sahip olması gereken önemli kavramlarından biri olan döngü yapılarından bahsedilecektir. Döngüler bir komutun veya komut grubunun istenildiği kadar veya belirli bir koşul gerçekleşinceye kadar çalıştırılmasını sağlayan programlama yapılarıdır. Eğer döngü yapıları olmasaydı programcılar aynı kodu tekrar tekrar yazmak zorunda kalacaklardı.

C++ programlama dilinde tanımlanan üç farklı döngü yapısı vardır. Bu döngü yapıları *for*, *while* ve *do… while* olarak bilinir. Bir döngü yapısında dikkat edilmesi gereken en önemli hususlardan birisi, döngünün sonsuza kadar çalıştırılmamasının garanti altına alınması ve belirli şartlar altında sonlandırılması gerektiğidir. Bir döngünün ne zaman sonlandırılacağını döngüye ait kontrol değişkeni belirler. Eğer döngü yapısına girmeden önce döngünün kaç defa çalışacağı önceden biliniyorsa kontrol değişkeni istenen değere ulaşıncaya kadar döngüye ait komutlar çalıştırılır. Bazı durumlarda ise bir koşul gerçekleşinceye kadar döngünün çalışması gerekebilir. Bu durumlarda mantıksal döngü kontrol yapıları kullanılır.

Bazı özel şartların gerçekleşmesi durumunda döngünün tamamlanmasını beklemeden döngüden çıkmak veya çalıştırılan iterasyon içerisindeki bazı satırların dikkate alınmaması gerekebilir. Bu gibi durumlara C++ programlama dilinde tanımlanmış olan *break* ve *continue* deyimleri kullanarak çözüm bulunur.

Bazı karmaşık problemlerin çözümünde birden fazla döngü yapısı oluşturmak gerekebilir. Bir döngünün içerisinde bir başka veya daha fazla döngü oluşturulması durumu iç içe döngüler olarak adlandırılır. Böylece en verimli ve doğru çözüm elde edilmiş olunur. Ama iç içe döngüler dikkatle oluşturulmalıdır. Aksi takdirde kolayca hatalı kod yazımına sebep olabilirler.

ARTTIRMA VE AZALTMA OPERATÖRLERİ

Kod yazarken genel kurallara uymak yazılan kodun okunabilirliğini arttırır.

Operatörler tanımlanmış matematiksel, ilişkisel ve mantıksal işlemleri gerçekleştirebilmek için kullanılan özel karakter veya karakterlerden oluşur. Operatörlerin bir sonuç üretebilmek için işlenenlere (operand) gereksinimleri vardır. C++ programlama dilinde operatörler kullandıkları işlenen sayılarına göre; Tek (Unary), İki (Binary) ve Üç (Ternary) olmak üzere üçe ayrılırlar.

Döngüler kavramı içerisinde en çok kullanacağımız işlemlerden biri, döngü sayacımızı ya bir arttırmak ya da azaltmaktır. Problemin çözümüne bağlı olarak farklı bir değer kullanarak arttırmak ve azaltmak mümkündür. Döngünün kaç defa çalışacağını belirlemek üzere tanımlanmış *sayac* isimli bir değişkenin değerini 1 arttırmak için kullanmamız gereken matematiksel ifade, uzun haliyle *"sayac = sayac + 1;"* şeklinde olacaktır. Aynı sonucu bir sonraki bölümde anlatılacak olan bileşik atama operatörü kullanarak da elde edebiliriz. Ama zaman içerisinde C++ programlama dilini geliştirenler daha kısa bir ifade olan 1 arttırma *(++)* operatörünü dilin yapısına eklemişlerdir. Benzer şekilde bir değişkenin değerini 1 azaltmak istersek azaltma *(--)* operatörünü kullanabiliriz.

Arttırma ve azaltma operatörlerinin değişkenden önce veya sonra kullanılması, istenilmeyen hatalara sebep olabilir.

Arttırma ve azaltma operatörlerinin iki farklı şekilde kullanımları vardır. Eğer arttırma ve azaltma operatörü değişkenin önünde kullanılırsa (*"++sayac"* veya *"--sayac"* gibi) önce arttır (pre-increment) veya azalt (pre-decrement), sonra kullanılırsa (*"sayac++"* veya *"sayac--"* gibi) sonra arttır (post-increment) veya azalt (post-decrement) olarak adlandırılır.

Arttırma ve azaltma operatörlerinin değişkenden önce veya sonra kullanılması farklı sonuçlar üretilmesine sebep olabilmektedirler. Eğer önce arttır veya azalt operatörü kullanılırsa önce değişkenin değeri 1 arttırılır veya azaltılır daha sonra değişken işleme sokulur. Ama sonra arttır veya azalt operatörü kullanılmış ise önce değişkenin mevcut değeri işlemde kullanılır daha sonra değişkenin değerinde arttırma veya azaltma yapılır. Bu durumlar bir örnek üzerinde anlatılacaktır:

```
int a = 10;
cout << a << endl; // Ekranda 10 değeri görüntülenir
cout << "Once arttir operatoru kullanilmasi durumunda" << endl;
cout << ++a << endl; // Önce 1 arttırma işlemi uygulanır ve 11 değeri görüntülenir
cout << a << endl; // Önce arttır operatöründen sonra değişkenin değeri
görüntülenir
cout << "*********************************************" << endl;

a = 10;
cout << a << endl; // Ekranda 10 değeri görüntülenir
cout << "Sonra arttir operatoru kullanilmasi durumunda" << endl;
cout << a++ << endl; // 10 değeri görüntülenir ve sonra 1 arttırma işlemi uygulanır
cout << a << endl; // Önce arttır operatöründen sonra değişkenin değeri
görüntülenir
cout << "*********************************************" << endl;
```

```
10
Once artir operatoru kullanilmasi durumunda
11
11
*********************************************
10
Sonra artir operatoru kullanilmasi durumunda
10
11
*********************************************
```

Şekil 6.1. Yukarıdaki Kod Bloğunun Ekran Çıktısı

- Yukarıdaki kod bloğunda önce ve sonra arttır operatörleri yerine önce ve sonra azalt operatörlerini kullanın ve ekran çıktısını inceleyerek her iki operatörün işlem sonuçlarına olan etkilerini gözlemleyin.

Arttırma ve azaltma operatörlerinin matematiksel ifadelerde kullanımına dikkat edilmelidir.

```
int a, b, c, d, e;
a = 5;
b = 3;
c = 3;
d = a * b++;
cout << d << endl;
e = a * --c;
cout << e << endl;
```

Hem arttırma ve hem de azaltma operatörleri yukarıdaki kod parçası içerisinde görüldüğü gibi bir matematiksel ifade içerisinde kullanılabilir. Ama d = ++(a * b) şeklinde kullanılması durumunda derleyicinin hata mesajı üretmesine sebep olacaktır.

Bireysel Etkinlik

- Yukarıdaki kod bloğunda d = a * b++; ifadesini d = ++(a * b) ifadesi ile değiştirerek oluşacak hata mesajını gözlemleyin.

Benzer şekilde arttırma ve azaltma operatörleri ilişkisel ifade içerisinde kullanılabilir. Arttırma ve azaltma operatörleri ilişkisel operatörlere göre daha yüksek önceliğe sahiptir.

```
a = 5;
if( a++ < 5 )
        cout << a << endl;
```

BİLEŞİK ATAMA OPERATÖRLERİ

Yazılım uzmanları programlarında sıklıkla "a = a + 3;" gibi benzer matematiksel ifadeleri sıklıkla kullanmaktadırlar. Bu ifade ilk bakışta karmaşık gelebilir. Çünkü eşittir (Aslında burada bir değişkene değer atama işlemi yapılmaktadır.) işaretinin hem sağında hem de solunda aynı değişken kullanılmıştır. Fakat kullanılan ifade ile hedeflenen amaç a değişkeninin değerini 3 arttırmaktır. Eğer a değişkeninin değeri atama operatöründen önce 10 ise ifadenin çalışmasını müteakip a değişkeni 13 değerine sahip olacaktır. Toplama operatörüne ek olarak çıkarma, çarpma, bölme ve mod operatörleri de benzer şekilde kullanılabilir. C++ programlama dilinde yazılımcıların yukarıdaki gibi ifadeleri daha kolay bir şekilde yazmaları için bileşik atama operatörleri dilin yapısına eklenmiştir. Aşağıda verilen tabloda bileşik atama operatörlerinin her biri için bir örnek ve eşdeğer ifade verilmiştir:

Tablo 6.1. Bileşik Atama Operatörleri ve Kullanımları

Operatör	Örnek	Eşdeğeri
+=	a +=3;	a = a + 3;
-=	b -=3;	b = b - 3;
*=	c *=3;	c = c * 3;
/=	d /=3;	d= d / 3;
%=	e %=3;	e = e % 3;

Yukarıda verilen tablodan da anlaşılacağı üzere bileşik atama operatörünün kullanılması durumunda yazılımcı değerini değiştirmek istediği değişkeni ifade içerisinde iki defa yazmaktan kurtulmuş olur.

Bireysel Etkinlik

- a = a + 5; matematiksel ifadesi ile a += 5; ifadesinin birbirinin eş değeri olduğunu yazacağınız örnek bir kod üzerinde gözlemleyin.

DÖNGÜ OLUŞTURMA KURALLARI

Döngü yapılarını doğru kullanabilmek için döngü oluşturma kuralları bilinmelidir.

Döngüler bir komutun veya komut grubunun belirli sayıda veya bir şart yerine gelinceye kadar çalışmasını sağlayan yapılardır. Bir döngünün kaç defa çalışması gerektiği önceden biliniyorsa sayaç kontrollü bir kontrol yapısı kullanılır. Eğer döngünün sonlandırılması için örneğin bir girdi verisinin sonuna ulaşılıp ulaşılmadığının bilinmesi gibi bir durum söz konusu ise şartlı kontrol yapısı tercih edilir. C++ programlama dilinde *while*, *do… while* ve *for* olmak üzere 3 farklı döngü oluşturulabilir. for ve while döngüleri aşağıda detaylıca anlatılacaktır. Bir sayaç kontrollü döngü oluşturmak için aşağıda verilen kurallara riayet edilmelidir:

- Döngüyü kontrol edecek bir değişken tanımlanır.
- Kontrol değişkeninin başlangıç ve bitiş değerleri belirlenir.
- Kontrol değişkeninin bitiş değerine ulaşıp ulaşmadığı her bir iterasyonda kontrol edilir.
- Çalıştırılması istenen komut veya komut grubu çalıştırılır.
- Kontrol değişkeninin değeri uygun bir şekilde güncellenir.

Şartlı bir döngü yapısı oluşturmak için aşağıda verilen kurallar takip edilmelidir:

- Döngüyü sonlandıracak şartlı bir ifade oluşturulur.
- Şartlı ifadenin doğru (true) olup olmadığı kontrol edilir.
- Eğer şartlı ifade doğru ise çalıştırılması istenen komut veya komut grubu işletilir.
- Eğer şartlı ifade yanlış (false) ise döngü sonlandırılır.

Şartlı bir döngü oluşturulması durumunda döngünün gövdesinde beklenen durumun gerçekleşmesi halinde şartlı ifadeyi yanlış yapacak bir ifadenin döngü gövdesinde kullanılması gerekir. Aksi halde sonsuz döngü içerisine girilir ve program sonlanmaz.

FOR DÖNGÜSÜ

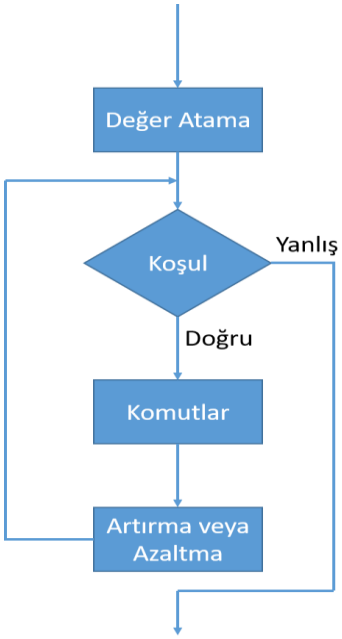
Döngünün başlangıç ve bitiş değerleri dikkatle belirlenmelidir.

Sayaç kontrollü bir döngünün bütün bileşenlerini for döngüsü ile tek bir satırda oluşturmak mümkündür. Bir for döngüsünün genel yapısı aşağıdaki gibidir ve akış diyagramı Şekil 6.2 ile verilmiştir:

```
for (ifade_1; koşul; ifade_2)
        komut_1;
```

Eğer döngü gövdesinde birden fazla ifadenin çalıştırılması gerekiyor ise "{…}" kullanılır. Bu durumda döngü yapısı aşağıdaki gibi oluşturulmalıdır:

```
for (ifade_1; koşul; ifade_2)
{
        komut_1;
        komut_2;
        .
        .
}
```



Şekil 6.2. for Döngüsü Akış Diyagramı

Döngünün kontrol değişkenini tanımlamak ve değer atamak için ifade_1 bölümü kullanılır. Bu bölüm döngüye girmeden önce yalnız bir kez çalıştırılır. Bu bölümde tanımlanan değişkenlere sadece döngü içerisinden erişilebilir. Dolayısıyla döngü dışında erişilmeye çalışırsa hata mesajı ile karşılaşılır. Koşul bölümü döngünün kontrol parçasıdır. Döngüye girmeden önce ve her bir iterasyonda şartın doğru olup olmadığı test edilir. Eğer şart doğru ise döngünün gövdesi çalıştırılır. Daha karmaşık koşullar oluşturmak için koşul bölümünde mantıksal VE,

VEYA ve DEĞİL operatörleri kullanılabilir. Her bir iterasyonda döngünün gövdesi çalıştırıldıktan sonra ifade_2 bölümü çalıştırılır. Bu bölüm genellikle kontrol değişkenini güncellemek için kullanılır.

for döngüsünün ilk bölümünde tanımlanan değişkenlere döngü içerisinde erişilebilir.

```
// HATALI KOD
for(int i = 0; i < 10; i++)
        cout << i <<endl;
cout << i << endl;
```

Eğer kontrol değişkenine hem döngü içerisinden hem de dışarısından erişilmek isteniyorsa değişkenin döngünün dışında tanımlanması gerekir.

```
// HATASIZ KOD
int i;
for(i = 0; i < 10; i++)
        cout << i <<endl;
cout << i << endl;
```

Örnek

- Ekrana alt alta 5 defa "Merhaba" yazdıracak C++ programını for döngüsü kullanarak oluşturunuz.

Çözüm:

```
for(int i = 0; i < 5; i++)
        cout << "Merhaba" << endl;
```

Yukarıdaki çözümde döngünün sayaç değişkeni olarak tam sayı veri tipinde “i” değişkeni tanımlanmış ve bu değişkene başlangıç değeri olarak 0 atanmıştır. Kural olmasa da genel olarak sayaç değişkeni için i, j ve k şeklindeki değişken isimleri kullanılır. Daha sonra çalıştırılan iterasyon içerisindeki sayaç değerinin 5’den küçük olup olmadığı kontrol edilmiştir ve geriye dönen sonucun doğru olması nedeniyle döngünün gövdesi çalıştırılmıştır. Sonrasında döngünün gövdesi "Merhaba" kelimesini ekrana yazdırmış ve imleci bir alt satıra geçirmiştir. Son olarak sayaç değeri 1 arttırılarak döngü koşulunun tekrar test edilmesi sağlanmıştır. Sayaç değeri 5 değerine ulaştığında ise kontrol değeri yanlış sonuç döndüreceği için döngüden çıkılmıştır.

Örnek

- 1 ile 21 arasındaki çift tam sayıları aralarında virgül olacak şekilde ekrana yazdıracak C++ programını for döngüsünü kullanarak oluşturunuz.

Çözüm:

```
for(int i = 2; i < 21; i += 2)
        cout << i <<", " ;
```

Bu çözümde sayaç değeri ikiden başlatılmıştır; çünkü 1 ile 21 arasındaki en küçük çift sayı ikidir. Daha sonra sayaç değerinin her bir iterasyonda 21'den küçük olup olmadığı kontrol edilmiştir ve küçük olduğu sürece sayaç değeri sonrasına virgül eklenerek ekrana yazdırılmıştır. Son olarak sayaç değeri her seferinde 1 değil 2 arttırılmıştır. Çünkü bizden sadece çift sayıların ekrana yazdırılması istenmiştir. Bu kod parçasıyla aynı sonucu verecek farklı çözümler mevcuttur; fakat bu çözümle sayaç değerini problemin doğasına göre uygun şekilde değiştirmenin gerektiği vurgulanmak istenmiştir.

Bireysel Etkinlik

- Yukarıdaki örnekte çift tam sayılar değil de tek sayılar istenmiş olsaydı çözümde hangi değişikliğin yapılması gerekirdi?
- Yukarıdaki örnekte aralığımız 1 ile 21 değil de 1 ile 31 olsaydı çözümde hangi değişikliğin yapılması gerekirdi?
- Yukarıda örnekte çift sayılar değil de ekrana dörde tam bölünebilen sayıların yazdırılması istenseydi hangi değişikliğin yapılması gerekirdi?

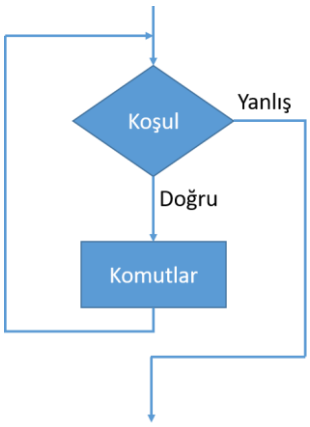
Daha kısa kod yazmak için sayaç kontrollü bir döngü oluşturmak amacıyla for döngüsü tercih edilmelidir.

WHILE DÖNGÜSÜ

Önceki bölümde anlatılan for döngüsü ile sayaç kontrollü bir döngü oluşturmak uygunken, while döngüsü daha çok şartlı bir kontrole sahip döngü yapısı için tercih sebebidir. C++ programlama dilinde her iki döngünün birbirinin eşleniği olarak yazılabileceği unutulmamalıdır. Bir while döngüsünün genel yapısı aşağıdaki gibidir ve akış diyagramı Şekil 6.3 ile verilmiştir:

```
while( koşul )
        komut;
```

Bu döngü yapısında koşul doğru olduğu sürece komut ifadesi çalıştırılır. Eğer döngü gövdesinde birden fazla komutun çalıştırılması gerekiyorsa komutların "{" ve "}" içerisine alınması gerekir.



Şekil 6.3. while Döngüsü Akış Diyagramı

Örnek

- Ekrana alt alta 5 defa "Merhaba" yazdıracak olan C++ programını bu kez de while döngüsünü kullanarak oluşturunuz.

Çözüm:

```
int i = 0;
while (i < 5)
{
        cout << "Merhaba" << endl;
        i++;
}
```

Yukarıdaki çözümde de görüldüğü üzere tek satırda bütün bölümlerini yazabildiğimiz for döngüsü yerine while döngüsü kullanmamız halinde 3 satır daha uzun kod yazmamız gerekmektedir. while döngüsünde ilk olarak sayaç değişkeninin tanımlanması ve bu değişkene bir ilk değer ataması yapılmaktadır. Daha sonra while döngüsünün koşul ifadesinin doğru olup olmadığı kontrol edilmektedir. Sayacın başlangıçta 0 olan değeri 5'den küçük olduğu için döngü gövdesi çalıştırılmakta ve ekrana "Merhaba" yazdırılmaktadır. Sonraki ifade sayaç değerini 1 arttırmaktadır. Döngü gövdesine ait bütün komutlar çalıştırıldıktan sonra döngü koşulu tekrar test edilmektedir. Sayaç değişkenin değeri 5 olunca da döngü koşulu sağlanmadığı için döngüden çıkılmaktadır.

Sonsuz döngü oluşturabilecek durumlara dikkat edilmelidir.

Yukarıda verilen çözüm, koşula bağlı bir döngü kullanılarak oluşturulmak istenseydi bu yeni çözümü aşağıdaki gibi yazmak gerekirdi:

```
int i = 0;
bool kontrol = true;
while (kontrol)
{
        cout << "Merhaba" << endl;
        i++;
        if (i == 5)
                kontrol = false;
}
```

Yukarıdaki kod parçasında dikkat edilmesi gereken husus, döngü koşulunu yanlış yapacak bir durumun olmasıdır. Aksi halse sonsuz döngüye girileceğidir.

Bireysel Etkinlik

- Aşağıda verilen kod parçası sonsuz döngü için bir örnektir. Kod üzerinde gerekli değişiklikleri yaparak çalışır bir kod parçası haline getirin.
- int sayac = 0;
- while (sayac < 10)
  - cout << sayac << endl;

Örnek

- 1 ile 21 arasındaki çift tam sayıları aralarında virgül olacak şekilde ekrana yazdıracak C++ programını while döngüsünü kullanarak oluşturunuz.

Çözüm:

```
int i = 2;
while(i < 21)
{
        cout << i << ", ";
        i += 2;
}
```

Bu çözümde sayaç olarak i değişkeni belirlenmiştir ve başlangıç değeri olarak 2 atanmıştır. Çünkü 1 ile 21 arasındaki en küçük çift sayı 2'dir. Daha sonra sayaç değerinin while döngüsüne ait kontrol yapısı ile 21'den küçük olup olmadığı her bir iterasyonda kontrol edilmiştir. Sayaç değeri 21'den küçük olduğu sürece döngünün gövdesi çalıştırılarak ekrana i değişkenin iterasyon içerisindeki değeri ve hemen arkasına virgül eklenerek ekrana yazdırılmıştır. Son olarak sayaç değeri her iterasyonda 2 arttırılmıştır. Sayaç değerinin 2 arttırılmasının nedeni sadece çift sayıların ekrana yazdırılmasının istenmesidir. Döngü gövdesinde iki farklı komutun çalıştırılması gerektiği için "{…}" kullanılmıştır.

BREAK VE CONTINUE DEYİMLERİ

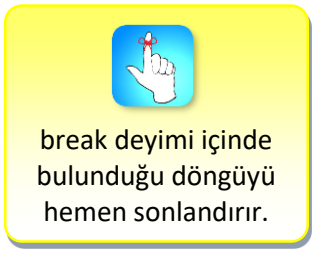
break ve continue deyimlerinin yanlış kullanımları beklenmedik hatalara sebep olabilir.

C++ programlama dilinde döngü akışını kontrol etmek için *break* ve *continue* deyimleri tanımlanmıştır. Belirli şartların gerçekleşmesi halinde döngünün tamamlanmasını beklemeden çıkmak gerekebilir. Böyle durumlarda break deyiminin for veya while döngüsü içerisinde kullanılması durumunda döngünün sonlandırılmasını beklemeden hemen çıkmak mümkün olur. Program döngüden hemen sonraki ilk satırdan başlayarak çalışmasına devam eder.

```
int i;
for (i = 0; i < 5; i++)
{
        if (i == 2)
```

```
                break;
        cout << i << endl;
}
cout << "i degeri \""<< i << "\" iken donguden cikildi";
```

Yukarıdaki kod parçasında sayaç kontrollü bir for döngüsü oluşturulmuştur. Sayacın başlangıç değeri 0 olarak belirlenmiştir ve döngünün 5 defa çalışması amaçlanmaktadır. Öte yandan sayacın değerinin 2 olduğu durumda eğer ifadesinin koşulu doğru olduğu için eğer gövdesine girilmiş ve break deyimi döngünün erkenden sonlanmasını sağlayarak döngüden sonraki ilk satırı çalıştırmıştır. Sayaç değeri 2 iken döngü durduğu için ekrana i değeri olarak 2 yazdırılmıştır.



break deyimi içinde bulunduğu döngüyü hemen sonlandırır.

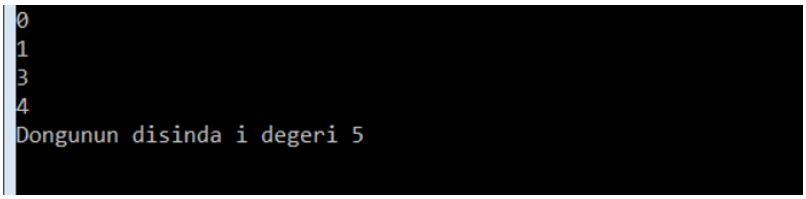
```
0
1
i degeri "2" iken donguden cikildi
```

Şekil 6.4. Yukarıda Verilen Kod Bloğunun Ekran Çıktısı

Bazı durumlarda bir koşulun gerçekleşmesi sonucu döngünün mevcut iterasyonunun yarıda kesilmesi (bazı satırlar çalıştırılmayarak) ve sonraki iterasyonla yola devam edilmesi gerekebilir. Böyle bir durumda continue deyimi for veya while döngüsü içerisinde kullanılabilir. Program akışı continue deyimine ulaştığı zaman bu deyimden sonraki döngü içerisinde bulunan bütün satırlar çalıştırılmayarak döngü gövdesinin sonuna gelinir ve program takip eden bir sonraki iterasyonla çalışmasına devam eder. for veya while döngü yapılarından hangisinin kullanıldığına bağlı olarak iki farklı durum ortaya çıkmaktadır. Eğer continue deyimi for döngüsü içerisinde kullanılmış ise döngünün üçüncü bölümü çalıştırılarak sonrasında koşul ifadesi test edilir. Ama while döngüsü kullanılması durumunda doğrudan döngünün koşulunun test edilmesi adımı gerçekleştirilir. Dolayısıyla sonsuz döngü oluşturmamak için bu husus dikkate alınmalıdır.

```
int i;
for (i = 0; i < 5; i++)
{
        if (i == 2)
                continue;
        cout << i << endl;
}
cout << "Dongunun disinda i degeri "<< i;
```

Yukarıdaki örnekte bir önceki örnekten farklı olarak break deyimi yerine continue deyimi kullanılmıştır. Sayaç değeri 2 olunca eğer ifadesi çalıştırılmıştır ve 2 değeri ekrana yazdırılmadan (atlanarak) for döngüsü kullanıldığı için sayaç değeri 1 arttırılarak for döngüsünün koşulu olan “i < 5” test edilmiştir. Test sonucuna bağlı olarak döngü sonlandırılmadan bir sonraki iterasyondan çalışmasına devam etmiştir. Sonraki iterasyonlarda kontrol değişkeni sırasıyla 3 ve 4 değerlerine ulaşmış ve ekrana yazdırılmıştır. Ama kontrol değişkeni 5 değerine ulaşınca döngü koşulu yanlış sonuç ürettiği için döngüden çıkılmıştır.



Şekil 6.5. Yukarıda Verilen Kod Bloğunun Ekran Çıktısı

break ve continue deyimlerinin dikkatli kullanılması gerekir. Bu deyimlerin özellikle bir sonraki bölümde anlatacağımız iç içe döngüler içerisinde kullanılmaları durumunda hem programcı hem de diğer kullanıcılar açısından hatalara açık ve anlaşılması zor bir programın ortaya çıkmasına sebep olabilmektedirler.

Bireysel Etkinlik

- Kullanıcının klavyeden ard arda girdiği 10 adet tam sayıdan sadece pozitif olanların toplamını bulacak kod parçasını döngü ve continue deyimi kullanarak gerçekleyin.

İÇ İÇE DÖNGÜLER

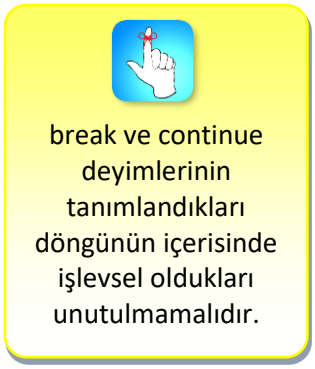
İç içe döngüler karmaşık problemleri sistematik bir şekilde çözmek için kullanılır.

Döngüler iç içe kullanılarak sıralı olmayan bir tam sayı dizisinin küçükten büyüğe sıralanması sağlanabilir veya iki matrisin çarpımı gibi daha karmaşık bir problemin çözümünde kullanılabilir. Hem for hem de while döngüleri birden fazla kez iç içe tanımlanarak iç içe döngüler oluşturulabilir. Bir döngü (dış döngü) gövdesinde bir veya daha fazla döngünün (iç döngü/döngüler) işletilmesi durumu iç içe döngü olarak adlandırılır. Dış döngünün her bir iterasyonunda iç döngü veya döngüler her seferinde yeniden çalıştırılır. Bu çalıştırma işlemi dış döngü tamamlanıncaya kadar devam eder. Bir tam sayı dizisi sıralanırken iki veya bir matris çarpımı işlemi yapılırken üç adet iç içe döngü oluşturmak gerekebilir. Problemin çözümüne uygun olacak şekilde iç içe döngü sayısını arttırmak mümkündür.

```
for (int i = 0; i < 5; i++)
{
        for (int j = 0; j < 5; j++)
                cout << "*";
        cout << endl;
}
```

Yukarıda verilen kod bloğunda iç içe iki adet for döngüsü kullanılmıştır. Her iki döngü de toplam 5 defa çalışmaktadır. Çünkü bu döngülerin başlangıç değerleri 0 ve bitiş değerleri de 5 olarak belirlenmiştir. İlk döngüden hemen sonra ikinci döngüye girilerek ekrana 5 defa "*" sembolü yazdırılmıştır. İç döngü ilk turunu tamamladıktan sonra işaretleyici (imleç) bir sonraki satıra yönlendirilerek dış döngünün ikinci kez çalışması sağlanmıştır ve ekrana yine 5 defa "*" sembolü yazdırılarak sonraki döngüye geçilmiştir. Kod bloğu tamamen çalıştıktan sonra ise aşağıda verilen ekran çıktısı üretilmiştir:

Şekil 6.6. Yukarıda Verilen Kod Bloğunun Ekran Çıktısı



Bireysel Etkinlik

- Siz de aşağıda ekran çıktısı verilen şekli elde etmek için gerekli C++ kodunu iç içe döngüler kullanarak oluşturunuz.

Şekil 6.7. Yukarıda Verilen Bireysel Etkinliğe Ait Ekran Çıktısı

Yukarı bahsedildiği üzere break ve continue deyimleri bazen döngülerden ya erken çıkmak ya da bulunulan iterasyon içerisinde bazı satırları atlamak için kullanılabilir. İç içe döngü yapılarında söz konusu bu iki deyim dikkatli bir şekilde kullanılmalıdır. Zira break ve continue deyimlerinin tanımlandıkları döngünün içerisinde işlevsel oldukları unutulmamalıdır.

Bireysel Etkinlik

- Sıralama yöntemleri önemli bir araştırma konusudur ve çözümlerinde iç içe döngü yapılarını kullanırlar. Siz de araştırma yaparak önemli sıralama yöntemlerinin neler olduğunu bulun ve çözümlerini inceleyerek iç içe döngüler kavramını pekiştirin.

Özet

- Bu bölümde bir programın olmazsa olmazlarından olan döngü yapıları anlatılmıştır. Döngü kavramı anlatılmadan önce ise döngü içerisinde bir programcının sıklıkla kullandığı arttırma/azaltma ve bileşik atama operatörleri incelenmiştir.
- Döngü yapıları içerisinde sıklıkla kontrol değişkeni 1 arttırılır veya 1 azaltılır. Bu matematiksel işlemi klasik yöntemle yapmak mümkündür. Ama programın okunabilirliğini ve yazılabilirliğini arttırmak için C++ dilinde 1 arttırma (++) ve 1 azaltma (--) operatörleri tanımlanmıştır.
- Arttırma ve azaltma operatörlerinin ilgili değişkenin önünde veya arkasında kullanılmasına bağlı olarak matematiksel işlemlerin sonuçlarına etkisi farklı olabilmektedir.
- Arttırma ve azaltma operatörleri değişkenden önce kullanılmış ise değişiklik hemen uygulanır. Ama operatörler sonra kullanılmış ise değişiklik bir sonraki satırda geçerlidir.
- Bileşik atama operatörleri, benzer şekilde daha az kod yazmak için programlama dilinin yapısına eklenmiş yapılardır. "a = a + 3" şeklinde yazabildiğimiz bir ifadeyi kısa olarak "a +=3" şeklinde yazmak mümkündür.
- Sadece toplama operatörü değil aynı zamanda çıkarma, çarpma, bölme ve mod operatörü de benzer şekilde matematiksel ifadelerde kullanılabilir.
- Döngüler bir veya daha fazla işlemin belirli sayıda veya bir koşul sağlanıncaya kadar tekrarlanmasını mümkün kılan programlama yapılarıdır. C++ programlama dilinde for, do...while ve while yapılarıyla 3 farklı şekilde döngü oluşturmak mümkündür. Bir for döngüsünü while kullanarak veya bir while döngüsünü for kullanarak yazmak mümkündür. Ama 3 satır yazarak oluşturduğumuz bir while döngüsünü 1 satır kullanarak for döngüsü olarak yazmak mümkündür.
- Bir döngü döngü oluşturma kurallarına dikkat edilerek oluşturulmalıdır. Döngüye girdikten sonra ister sayaç kontrollü ister bir koşula bağlı olsun bir şekilde kontrol ifadesini yanlış yapacak bir deyimin gövdede yer alması gerekmektedir. Aksi halde sonsuz döngüye girilecektir.
- for döngüsü sayaç kontrollü bir döngü oluşturmak için idealdir. Üç bölümden oluşur. İlk bölüm genellikle sayaç değişkeninin tanımlandığı ve başlangıç değer atamasının yapıldığı bölümdür. İkinci bölüm döngü koşulunun test edildiği bölümdür. Döngü koşulu doğru olduğu sürece döngü gövdesinde tanımlanmış komutlar çalıştırılır. Son bölüm ise sayaç değerinin problem çözümüne uygun bir şekilde güncellendiği bölümdür.
- for döngüsünün ilk bölümünde tanımlanan değişkenlere döngü dışından da erişilmek isteniyorsa değişken tanımlama işleminin döngüden önce yapılması gereklidir. Aksi halde değişkene sadece döngü içerisinden erişilebilir.
- While döngüsü daha çok bir koşula bağlı döngü oluşturmak için idealdir. Döngü için tanımlanmış koşul doğru olduğu sürece döngü gövdesindeki komutlar çalıştırılır. Unutulmaması gereken koşul durumunu yanlış yapacak bir ifadenin döngü gövdesinde muhakkak yer alması gerektiğidir.
- Programın yapısı gereği bazen döngünün sonlandırılmasını beklemeden çıkmak veya bulunulan iterasyonda bazı adımların atlanması gerekebilir. Bu durumlarda break ve continue deyimleri kullanılır. Eğer döngü hemen sonlandırılmak isteniyorsa break deyimi kullanılır. continue deyimi döngü gövdesinde kendisinden sonra yer alan satırların atlanmasını ve döngünün bir sonraki iterasyona başlamasına neden olur.Döngüler iç içe kullanılarak daha karmaşık problemlerin çözülmesi sağlanır. Hem for hem de while döngüleri birden fazla kez iç içe tanımlanarak iç içe döngüler oluşturulabilir. Böyle bir durumda dış döngünün her bir iterasyonunda iç döngü her seferinde yeniden başlatılır.
- İç içe oluşturulan döngü yapılarında break ve continue deyimlerinin nerede kullanıldıkları önemlidir. Çünkü bu deyimler sadece gövdesinde yer aldıkları for veya while döngüsüne aittirler.

DEĞERLENDİRME SORULARI

1.
```
int i;
for (i = 0; i <= 3; i++)
        if (i == 2)
                break;
cout << i << endl;
```
Yukarıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

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

2.
```
int i;
for (i = 0; i <= 3; i++)
        if (i == 2)
                continue;
cout << i << endl;
```
Yukarıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

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

3.
```
int a= 5, b = 3;
cout << (++a) * (b--) <<endl;
```
Yukarıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

a) 10
b) 12
c) 15
d) 18
e) 20

4.
```
int a= 5, b = 3;
cout << (a++) * (b--) <<endl;
```
Yukarıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

a) 10
b) 12
c) 15
d) 18
e) 20

5.
```
int a = 10;
        a /= 2;
        cout << a <<endl;
```
Yukarıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

a) 0
b) 2
c) 5
d) 10
e) 20

6.
```
int a = 10;
a % = 2;
cout << a <<endl;
```
Yukarıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

a) 0
b) 2
c) 5
d) 10
e) 20

7.
```
int a=0, b=0;
for (int i = 0; (a < 2) && (b < 4); i++)
{
        a++;
        b++;
}
cout << a <<", "<<b<<endl;
```
Yukarıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

a) 0, 0
b) 2, 2
c) 4, 2
d) 2, 4
e) 4, 4

8.
```
int a=0, b=0;
for (int i = 0; (a < 2) || (b < 4); i++)
{
        a++;
        b++;
}
cout << a <<", "<<b<<endl;
```

Yukarıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

a) 0, 0
b) 2, 2
c) 4, 2
d) 2, 4
e) 4, 4

9. 0 2 4 6 8 10

Yukarıda verilen ekran çıktısını elde etmek için aşağıdaki döngülerden hangisi kullanılmalıdır?

a)
```
for ( int i = 0; i < 12; i += 2)
cout << i << " ";
```
b)
```
for ( int i = 0; i < 10; i += 2)
cout << i << " ";
```
c)
```
for ( int i = 0; i < 12; i += 1)
cout << i << " ";
```
d)
```
for ( int i = 0; i < 11; i += 1)
cout << i << " ";
```
e)
```
for ( int i = 0; i < 12; i -= 2)
cout << i << " ";
```

10.
```
int i = 0;
bool kontrol = true;
while (kontrol)
{
        cout << "Merhaba" << endl;
        i++;
        if (i == 3)
                kontrol = _____;

}
```

Yukarıda verilen döngüden 3 iterasyon sonra çıkılması için boşluk bırakılan yere aşağıdakilerden hangisinin yazılması uygundur?

a) case
b) for
c) continue
d) false
e) 1

Cevap Anahtarı

1.c, 2.e, 3.d, 4.c, 5.c, 6.a, 7.b, 8.e, 9.a, 10.d

YARARLANILAN KAYNAKLAR

[1] Özkan, Y. (2015). C++ Programlama Dili, 3. Baskı, Papatya Bilim.

[2] Deitel, P., & Deitel H. (2014). C++ How to Program, 9. Baskı, Pearson.

FONKSİYONLAR

İÇİNDEKİLER

- Hazır Fonksiyonlar
  - Matematik Kütüphanesi
- Fonksiyonlar
  - Fonksiyon Tanımlama
  - Fonksiyon Çağırma
  - Fonksiyon Tipleri
  - Fonksiyon Değişkenlerinde Aktif Alan
  - Fonksiyondan Referans ile Parametre Aktarımı
  - Fonksiyondan Fonksiyon Çağırma
  - Rastgele Sayı Üretim Fonksiyonları

HEDEFLER

- Bu üniteyi çalıştıktan sonra;
  - Hazır fonksiyonları kullanmayı öğrenebilecek,
  - Fonksiyonu tanımlayabilecek,
  - Kendi tanımladığınız fonksiyonu çağırabilecek,
  - Farklı tipte ve farklı giriş parametrelerine göre fonksiyonlar kullanabileceksiniz.

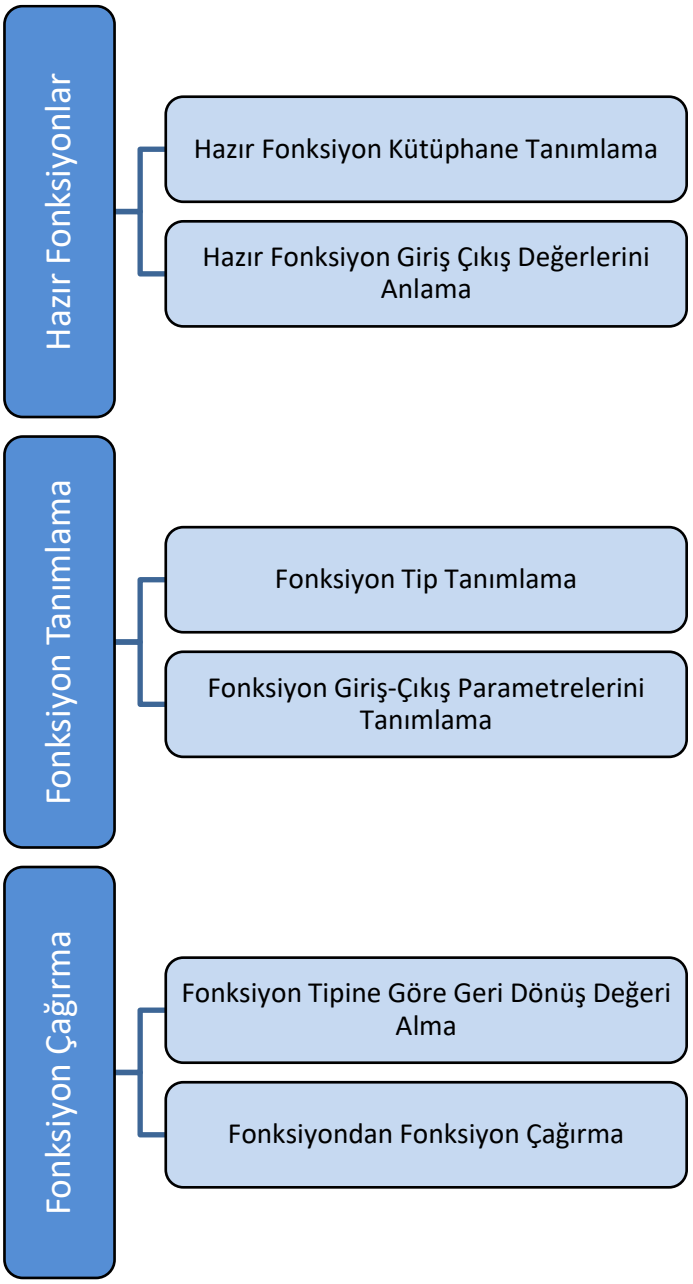
Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

Dr. Onur GÖK

ÜNİTE

7


Hazır Fonksiyonlar
Hazır Fonksiyon Kütüphane Tanımlama
Hazır Fonksiyon Giriş Çıkış Değerlerini Anlama
Fonksiyon Tanımlama
Fonksiyon Tip Tanımlama
Fonksiyon Giriş-Çıkış Parametrelerini Tanımlama
Fonksiyon Çağırma
Fonksiyon Tipine Göre Geri Dönüş Değeri Alma
Fonksiyondan Fonksiyon Çağırma

GİRİŞ

Fonksiyonlar, bir program içerisinde görevi aynı olan alt adımların modüler olarak tasarlanması için kullanılır. Esas ihtiyaç ise kod karmaşasının azaltılmasıdır. Zira kod ne kadar büyürse, hata kontrolü ve yönetim bir o kadar zorlaşır. Bunu engellemenin yolu problemi alt parçalara ayırma, tekrarlı kodları azaltma ve aynı işi tek bir alanda çözmektir. Örnek olarak bir main() fonksiyonu içinde 100000 satırlık bir kodu kontrol etmek, yönetmek, test etmek ve hata ayıklamak uzun bir süreç gerektirir. Bir yazılımın fonksiyonların birleşimi şeklinde tasarlanması her bir bölümün bağımsız olarak test edilebilmesi kolaylığını da sağlar.

Bir fonksiyonun en büyük avantajı, programın herhangi bir noktasından çağrılabilmesi, çalıştırılabilmesi ve ayrıca ilgili fonksiyonun çağrılma sayısının sınırlı olmamasıdır. Bu sayede söz konusu problemin tekrar tekrar çözülmesine gerek kalmaz, programın daha anlaşır olması sağlanır ve problem daha alt adımlara ayrılarak detaylı bir kontrol sağlanır.

C++, nesneye yönelik programlamayı ve fonksiyon kullanımını destekler. C++ programlama dilindeki fonksiyon kullanım şekli matematikteki fonksiyon tanımına benzerdir. Her iki tanımda da fonksiyona giriş uygulanır ve çıkış elde edilir.

Her programlama dilinde olduğu gibi C++ dilinde de fonksiyon tanımlama ve kullanımı için belirlenmiş mantıksal ve simgesel gramer kuralları vardır. Fonksiyonlar ünitesi bu kurallar ile birlikte iki başlık halinde anlatılacaktır:

- Hazır Fonksiyonlar: C++ programlama dilinin desteklediği kütüphanelerden çağrılan, prototip ve işlevsel olarak tanımlanmış ve programcıya kolaylıklar sağlayan fonksiyonlar olarak tanımlanır.
- Programcı tarafından tanımlanan ve geliştirilen fonksiyonlar.

HAZIR FONKSİYONLAR

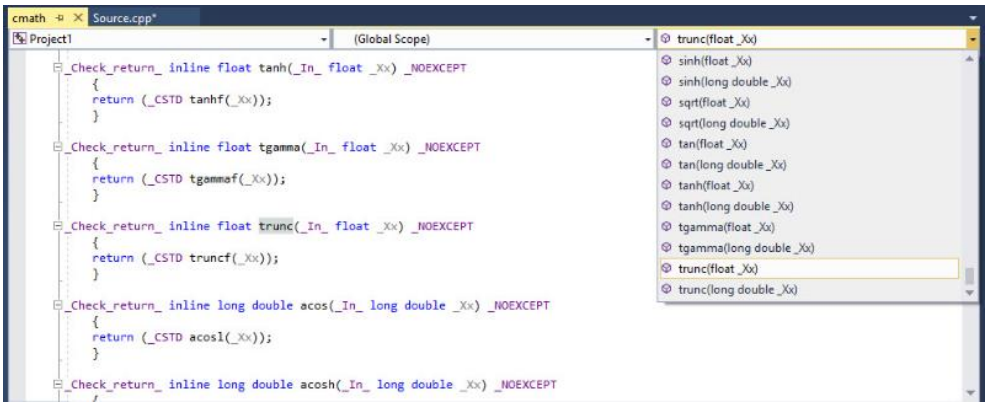
cout ve cin iostream kütüphanesi içerisinde tanımlı hazır fonksiyonlardır.

C++ dilinde program geliştiricilerin kendi fonksiyonlarını yazabilmeleri mümkündür. Öte yandan dil bünyesinde geliştiricilere kolaylık olması açısından hazır kütüphaneler ve o kütüphaneler içerisinde de kullanıma hazır fonksiyonlar bulunmaktadır. Matematik, karakter veya zaman işlemleri gibi çok çeşitli kütüphaneler mevcuttur ve bu kütüphaneler içerisindeki hazır fonksiyonlar geliştiricilere zamandan tasarruf sağlarlar.

Matematik Kütüphanesi

C++ programlama dilinde hazır matematiksel fonksiyonları kullanmak için *cmath* kütüphanesinin *#include <cmath>* şeklinde programa eklenmesi gerekmektedir.

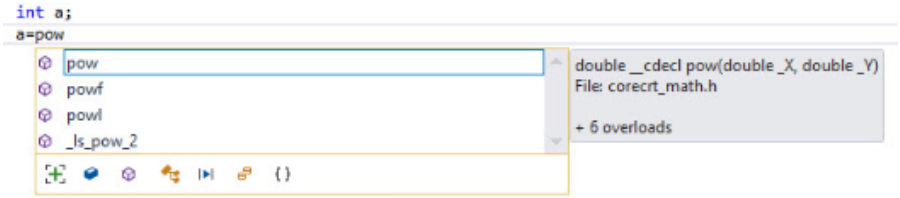
Program içerisinde bu kütüphaneye ait tüm fonksiyonları ve prototiplerini görmek için *cmath* yazısının seçili hale getirilmesi, sonrasında üzerine sağ tıklanması ve "Belge Aç <cmath>" seçeneğinin seçilmesi yeterlidir (Bkz. Şekil 7.1).



Şekil 7.1. cmath Kütüphanesinin İçeriği

Hazır fonksiyon kod içerisinde yazıldığında prototipi sağda BGO tarafından gösterilir.

Herhangi bir fonksiyon program içerisinde kullanılmak istendiğinde, fonksiyonun prototipini görmek gerekmektedir. İlgili prototipi görmek için fonksiyon ismi yazıldığında kod geliştirme ara yüzü bu konuda yardımcı olmaktadır. Bu esnada fonksiyona ait prototip bilgilendirme penceresinde görülmektedir (Bkz. Şekil 7.2). Fonksiyonun hangi tipte sonuç döndürdüğü, fonksiyona hangi değerlerin parametre olarak aktarıldığı gibi bilgiler bu pencere sayesinde öğrenilebilmektedir.



Şekil 7.2. pow Fonksiyonu Bilgilendirme Penceresi

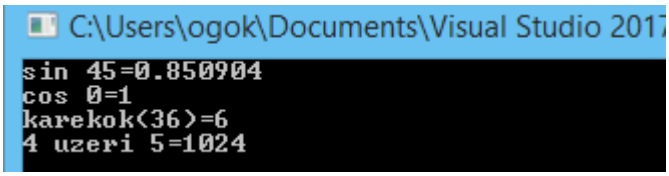
pow fonksiyonunun prototipi *(double_cdecl pow(double_X, double_Y)* bu fonksiyonun 2 adet double veri tipinde parametre aldığını (X ,Y) ve yine double veri tipinde bir değeri geri döndürdüğünü söylemektedir.

Tablo 7.1. Kullanıma Hazır Bazı cmath Kütüphanesi Fonksiyonları

Fonksiyon	Görevi	Örnek	Sonuç
cos(a)	a açısının kosinüsünü hesaplar.	cos(0)	1.0
sin(a)	a açısının sinüsünü hesaplar.	sin(0)	0.0
tan(a)	a açısının tanjantını hesaplar.	tan(45)	1.61978
acos(a)	a açısının arkkosinüsü hesaplar.	acos(1)	0.0
asin(a)	a açısının arksinüsünü hesaplar.	asin(1)	1.5708
atan(a)	a açısının arktanjantını hesaplar.	atan(1)	0.785398
ceil(a)	a'yı yukarı yuvarlar.	ceil(4.3)	5
floor(a)	a'yı aşağı yuvarlar.	floor(4.7)	4
round()	a'yı en yakın tam sayıya yuvarlar.	round(4.5)	5
fabs(a)	a'nın mutlak değerini alır.	fabs(-4.3)	4.3
pow(a,b)	a'nın b. kuvvetini hesaplar (a^b).	pow(2,3)	8.0
sqrt(a)	a'nın karekökünü hesaplar.	sqrt(9)	3.0
fmod(a,b)	a/b işleminin kalanını bulur.	fmod(4.0/3.2)	0.8

Çok kullanılan bazı matematiksel fonksiyonların işlevleri Tablo 7.1' de verilmiştir. Tabloda açıların radyan değerleri sonuç olarak üretilir. Derece olarak görülmesi için radyan-derece dönüşümleri kullanılmalıdır. cmath kütüphanesinde bu tablodaki fonksiyonlar dışında da fonksiyonlar mevcuttur.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
        cout << "sin 45="<<sin(45)<<endl;
        cout << "cos 0="<<cos(0) << endl;
        cout << "karekok(36)=" << sqrt(36) << endl;
        cout << "4 uzeri 5=" << pow(4, 5) << endl;
        cin.get();//Kullanicinin herhangi bir tuşa basması beklenir
        return 0;
}
```



Şekil 7.3. cmath Kütüphanesi Örnekleri

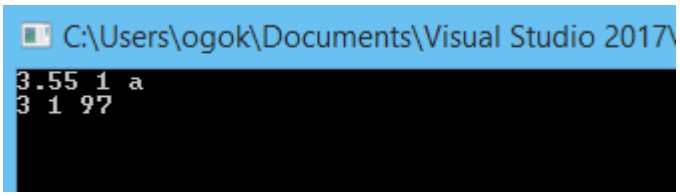
Yukarıdaki örnekte cmath kütüphanesine ait başlık dosyası ilgili fonksiyonlar kullanılabilsin diye C++ programına eklenmiştir. Bu kütüphanede yer alan sin, cos, sqrt ve pow fonksiyonları kullanılarak hesaplamalar kolayca yapılmıştır. *Hazır fonksiyonlar kullanılırken dikkat edilmesi gereken en önemli husus, fonksiyonun hangi veri tipinde değerleri aldığı ve geriye hangi veri tipinde bir sonucu değer olarak geriye döndürdüğü bilgisidir.*

static_cast<int>(a) hazır fonksiyonu a değişkenini int veri tipine çevirir.

Farklı veri tipindeki değerlerin birbirine dönüştürülmesi için *static_cast* isimli fonksiyondan faydalanılabilir. *static_cast* operatör şeklinde kullanılan bir fonksiyondur. Program çalışmaya henüz daha başlamadan derleme anında ilgili veri tipi dönüşümünü gerçekleştiren bir fonksiyondur.

```
#include <iostream>
using namespace std;
int main()
{
        double ondalikSayi = 3.55;
        bool kosul = true;
        char karakter = 'a';
        cout<< ondalikSayi << " " << kosul << " " << karakter;
        int sayi1, sayi2, sayi3;
        sayi1 = static_cast<int>(ondalikSayi);
        sayi2= static_cast<int>(kosul);
        sayi3= static_cast<int>(karakter);
```

```
    cout << endl<<sayi1 << " " << sayi2<< " " << sayi3;
}
```



Şekil 7.4. static_cast Fonksiyonu Uygulamasının Ekran Çıktısı

Farklı veri tipinde değişkenlerin birbirine dönüştürülmesi için kullanılan *static_cast* fonksiyonunun normal fonksiyonlardan farklı olarak operatör şeklinde tanımlandığı görülmektedir. Fonksiyon hangi veri tipinde tanımlanırsa ona uygun değeri geriye gönderecektir.

static_cast<veri tipi> (değisken)

static_cast fonksiyonu değişken değerini giriş olarak alır ve o değişkeni veri tipi boyutuna getirir ve yeni veri tipindeki değeri geriye döndürür.

FONKSİYON

Hazır fonksiyonlar programcı tarafından hızlı kod yazmak için kullanılır. Buna ek olarak geliştiricilerin kendi fonksiyonlarını tanımlaması da mümkündür. Bu alt başlıkta C++ programlama dilinde fonksiyon tanımlaması, fonksiyon tipleri, giriş parametre alım şekli, fonksiyon çağırma yöntemleri ve çıkış üretme şekillerinden bahsedilecektir.

C++ ile programlamada kullanılan çoğu komut kendi kütüphanesinde çağrılan fonksiyondan oluşmuştur. *cout* C++ programlama dilinin *iostream* kütüphanesinde tanımlı bir fonksiyondur. Fonksiyonlar kendi başlarına birer programdırlar. *main* de bir fonksiyondur. Ancak bu fonksiyonun bir başka fonksiyon içinden çağrılmasına gerek yoktur. *main* dışındaki tüm fonksiyonların çalışması için çağrılması gerekmektedir. Alt program olan fonksiyonlar çağrılmadıkları sürece çalışmadıkları gibi, içerisinde tanımlı değişkenler de fonksiyon çalışmadığı sürece bellekte yer tutmazlar. Fonksiyonlar adları kullanılarak çağrılırlar.

Fonksiyon Tanımlama

C++ programlama dilinde fonksiyon tanımlanması için belirli kurallar vardır. Bu tanım aşağıdaki formatta olmalıdır:

```
Veri Tipi FonksiyonAdı(Fonksiyon Giriş Parametreleri)
{
        Deyimler...
        return fonksiyon veri tipinde değer
}
```

Fonksiyonun veri tipi: Fonksiyon gövdesindeki deyimler sonucunda hesaplanan ve diğer fonksiyonlara çıkış olarak döndürülen veri tipi değeridir. Normal değişken tipleri gibi tanımlanır.

Fonksiyon adı tanımlaması: Değişken tanımlanmasında dikkat edilmesi gereken kurallar fonksiyon isimlendirmelerinde de geçerlidir. Değişkenlerde olduğu gibi fonksiyon isimlendirmesinde harfler ve sayılar kullanılabilir, ilk karakter sayı olamaz ve arada boşluk karakteri kullanılamaz. (Bu kitabın tamamında fonksiyon isimlendirmelerinde paskal notasyonu kullanıldığından 3. Ünite'de bahsedilmişti.)

Fonksiyon tanımlanırken veri tipi, ad ve parametre alanı mutlaka tanımlı olmalıdır.

Fonksiyon giriş parametreleri alanı: Fonksiyona gönderilen değerlerin, fonksiyon gövdesindeki değişkenlere atanması için kullanılır.

Çıkış değerini geriye göndermek için return kullanılır. return aynı zamanda fonksiyonun bittiği deyimdir. return çalıştıktan sonra daha alt satırlarda yer alan hiç bir deyim dikkate alınmaz. Başka bir deyişle program akışı fonksiyonun çağrıldığı yere aktarılır.

Örnek

- Fonksiyona gönderilen iki tam sayıdan, birincinin ikinci değere göre üssünü alan C++ fonksiyonunu tanımlayınız.

Yandaki çözümde üs alma işlemi for döngüsü kullanılarak gerçekleştirilmiştir. sonuc değişkeni 1'den kuvvet değerine kadar sayi değişkeni ile çarpılmıştır.

Çözüm:

```
int KuvvetAl(int sayi, int kuvvet)
{
        int sonuc = 1;
         for (int i = 1; i <= kuvvet; i++)
                sonuc = sonuc * sayi;
        return sonuc;
}
```

Yukarıdaki fonksiyonun veri tipi *int* ve fonksiyon ismi de *KuvvetAl* olarak tanımlanmıştır. Bu fonksiyon parametre olarak iki tam sayı değer alır. *sayi* ve *kuvvet* değişkenleri fonksiyona gönderilen 2 tam sayı değerine eşitlenirler. Parametre alanındaki değişkenler fonksiyon içerisinde tanımlı değişkenlerdir ve kullanılmaya başlanırken ilk atanan değere sahiptirler. Birden fazla parametre giriş olarak fonksiyona gönderiliyorsa bunlar birbirlerinden virgül ile ayrılırlar. Değişkenlerin veri tipleri tanımlanmış olmalıdır. Fonksiyonun içerisinde fonksiyon giriş parametreleri ile aynı isimli değişken kullanılmamalıdır.

Üs alma işlemi fonksiyon gövdesinde gerçekleştirilmektedir. Çözüm için sonuç kuvvet defa sayı ile çarpılarak işlem tamamlanmıştır. Program akışı fonksiyon gövdesindeki son deyim olan *return sonuc;* deyimine ulaştığında hesaplanan sonuç fonksiyonun çağrıldığı yere geri döndürülmektedir.

return deyimi fonksiyonun veri tipinde bir değeri geriye döndürmek için kullanılmıştır. *sonuc* değişkeninin veri tipi ile fonksiyonun veri tipinin aynı olduğuna dikkat edilmelidir.

Fonksiyon Çağırma

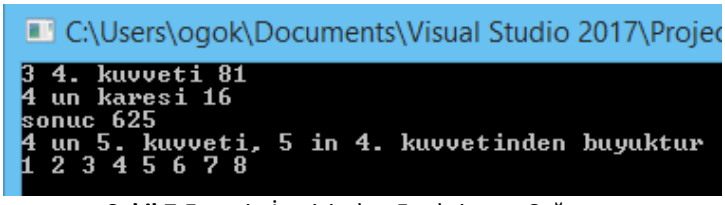
Fonksiyon çağrımı başka bir fonksiyon içerisinden ve hatta ilgili fonksiyon içinden bile yapılabilir. Çağrım şekli normal bir değişkenin kullanımına benzerdir. Bir fonksiyon aritmetik işlemlerin içinde çağrılabilir, while ve for döngüleri içinde bir koşul olarak çağrılabilir, bir ekrana yazdırma işleminin içerisinde çağrılabilir veya bir değişkene başlangıç değeri atamak amacıyla kullanılabilir.

Çözüm:

```
#include <iostream>
using namespace std;
int KuvvetAl(int sayi, int kuvvet)
{
        int sonuc = 1;
        for (int i = 1; i <= kuvvet; i++)
                sonuc = sonuc * sayi;
        return sonuc;
}
int main()
{
        int yazdir1;
        yazdir1 = KuvvetAl(3, 4);
        cout << "3 4. kuvveti " << yazdir1;

        int yazdir2 = KuvvetAl(4, 2);
        cout <<endl << "4 un karesi " << yazdir2;

        int sayi1=5, kuvvet1=4;
        int yazdir3 = KuvvetAl(sayi1, kuvvet1);
        cout << endl << "sonuc " << yazdir3<<endl;

        if (KuvvetAl(4, 5) > KuvvetAl(5, 4))
                cout << "4 un 5. kuvveti, 5 in 4. kuvvetinden buyuktur";
        else
                cout << "4 un 5. kuvveti, 5 in 4. kuvvetinden kucuktur";

        cout << endl;

        int i;
        for (i = 1; i <= KuvvetAl(2, 3); i++)
        {
                cout << i << " ";
        }
        return 0;
```

KuvvetAl() fonksiyonunun döndürdüğü sayısal değer atama veya karşılaştırma için kullanılabilir, ekrana yazdırılabilir.

```
}
```



Şekil 7.5. main İçerisinden Fonksiyonu Çağırma

KuvvetAl fonksiyonu yukarıdaki örnekte ilk önce *yazdir1* değişkenine değer olarak atanacak sonucu geriye döndürmüştür. 3 ve 4 sayıları parametre değerleri olarak fonksiyona gönderilmiştir. Fonksiyonda 3 değeri *sayi* değişkenine, 4 değeri de *kuvvet* değişkenine atanmıştır. Fonksiyon gövdesinde *sonuc* değeri 81 olarak hesaplanmıştır ve *return* deyimi ile geriye döndürülmüştür. Sonrasında bu değer *yazdir1* değişkeninin değeri olarak kullanılmıştır.

Aynı fonksiyon *yazdir2* değişkeninin başlangıç değerini oluşturmak için de kullanılmıştır. *yazdir2* değişkeninin başlangıç değeri bu sayede 16 olmuştur.

Fonksiyona sadece sabit sayılar değil değişkenler de parametre olarak gönderilebilirler. *yazdir3* değişkeninin başlangıç değeri için fonksiyona 2 adet değişken değer gönderilmiştir. *sayi* ve *kuvvet* değişkenlerine sırasıyla değer olarak 5 ve 4 atanmış ve bu değişkenler fonksiyona parametre olarak aktarılmışlardır.

Fonksiyon değeri if koşulu içerisinde ve for döngüsü koşulu olarak da kullanılmışlardır. if koşulunda ve for döngüsü koşulunda sabit sayılar veya değişkenler kullanılabildiği gibi aynı zamanda bir fonksiyonun sonucu da kullanılabilmektedir.

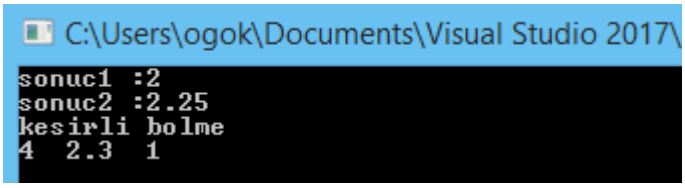
Fonksiyon Tipleri

void aslında bir tip değildir. Fonksiyonun değer döndürmediğini gösteren bir tanımlamadır.

Fonksiyonlardaki tip tanımlaması C++ programlama dilinin desteklediği veri tiplerinin hepsini kapsar. int, double, float, bool veya char veri tipleri gibi fonksiyonun herhangi bir sonuç döndürmesi istenmiyorsa veya çözüm fonksiyon içinde sonuçlandırılacak ise fonksiyonun veri tipi olarak *void* kullanılır ve herhangi bir değer döndürülmeyeceği için de *return* deyiminin kullanılmasına gerek kalmaz.

```
#include <iostream>
using namespace std;
int TamBol(int bolunen, int bolen)
{
        return bolunen/bolen;
}
double KesirliBol(double bolunen, double bolen)
{
        return bolunen/bolen;
}
bool OndalikVarMi(double bolunen, double bolen)
{
        double sonuc = bolunen/bolen;
        double toplam = (int)sonuc;//double toplam=static_cast<int>(sonuc);
```

```
	if (toplam == sonuc)
		return true;
	else
		return false;
}
void FonksiyonaGonderileniYaz(int a, double b, bool c)
{
	cout <<endl<< a << "  " << b << "  " << c;
}

int main()
{
	int yazdir1;
	yazdir1 = TamBol(9, 4);
	cout << "sonuc1 :" << yazdir1<<endl;

	double yazdir2 = KesirliBol(9, 4);
	cout << "sonuc2 :" << yazdir2<<endl;

	int sayi1=8, sayi2=19;
	if (OndalikVarMi(sayi2, sayi1))
		cout << "tam bolme";
	else
		cout << "kesirli bolme";

	int param1 = 4;
	double param2 = 2.3;
	bool param3 = true;
	FonksiyonaGonderileniYaz(param1, param2, param3);
	return 0;
}
```



Şekil 7.6. Fonksiyon Tipleri Ekran Çıktısı

TamBol ve *KesirliBol* fonksiyonlarındaki tek işlem return deyiminin hemen yanında gerçekleştirilen hesaplamanın sonucunun geriye döndürülmesidir. Fonksiyonlar geriye int veya double değer gönderebilirler.

OndalıkVarMi fonksiyonu bool tipinde bir değer geriye döndürmektedir. Bu fonksiyonda dikkat edilmesi gereken en önemli husus return deyiminin fonksiyon gövdesinde birden fazla yerde kullanılabilmesidir. Yine önemli bir başka husus ilgili return deyimi çalıştığında onun altında yer alan diğer deyimlerin dikkate alınmayışıdır.

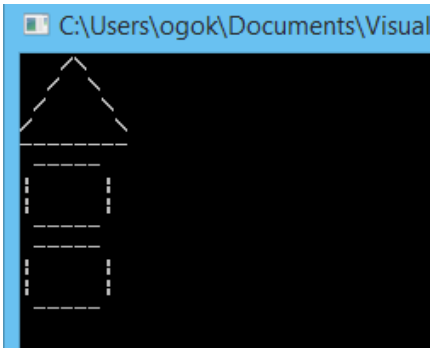
Fonksiyon içerisinde birden fazla return deyimi kullanılabilir. Fonksiyonda return ...; deyiminden sonra başka bir işlem yapılmaz.

FonksiyonaGönderileniYaz fonksiyonu void tipinde bir fonksiyondur. Bunun anlamı, fonksiyon herhangi bir değer çıkışı üretmeyeceği için return deyiminin kullanılmasına gerek olmamasıdır. Diğer önemli bir durum ise main fonksiyonu içerisinde *FonksiyonaGönderileniYaz* fonksiyonu çağrıldığında herhangi bir atama veya karşılaştırma gibi bir işlem yapılmamasıdır.

Fonksiyonlar geriye çıkış değeri göndermiyorlarsa void olarak tanımlandıkları gibi giriş parametresi de almayabilirler. Herhangi bir giriş değeri olmadan fonksiyon içerisinde de çıkış oluşturulabilir.

Fonksiyon parametre almasa bile parantezler () mutlaka kullanılmalıdır.

```
#include <iostream>
using namespace std;
void KareCiz()
{
	cout << " -----" << endl;
	cout << "|     |" << endl;
	cout << "|     |" << endl;
	cout << " -----" << endl;
}
void UcgenCiz()
{
	cout << "   /\\ " << endl;
	cout << "  /  \\ " << endl;
	cout << " /    \\ " << endl;
	cout << "/      \\    " << endl;
	cout << "--------" << endl;
}
int main()
{
	UcgenCiz();
	KareCiz();
	KareCiz();
	return 0;
}
```



Şekil 7.7. Giriş Değeri Almayan Fonksiyonun Ekran Çıktısı

Fonksiyon Değişkenlerinde Aktif Alan

Fonksiyonlardaki değişkenlere erişim değişkenlerin nerede tanımlandığına bağlıdır. Değişkenler hangi fonksiyonda tanımlı ise sadece o fonksiyon içerisinde aktiftir ve değerini korur. Değişkenler tanımlandığı alana göre ikiye ayrılır:

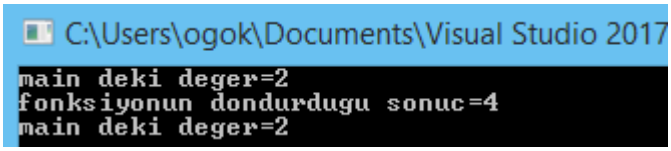
- Yerel Değişkenler
- Global Değişkenler

Yerel değişkenler, program çalıştırıldığında aktif hale gelir ve kendisi için ayrılan bellek alanlarını kullanır. Ancak yer aldıkları blok sona erdiğinde bu bellek alanları iptal olur ve değişkenin içeriği tamamen yok olur. Aynı blok daha sonra tekrar başlasa bile, yerel değişkenler eski değerlerini alamazlar. Program içinde birden fazla fonksiyon varsa, sadece tanımlandığı fonksiyonda geçerli olabilecek değişkenlere yerel değişken adı verilir.

Eğer bir değişkenin program içindeki tüm fonksiyonlar için geçerli olması söz konusu ise, bu kez fonksiyonların dışında bir yerde bildirimi yapılır. Bu tür değişkenlere global (küresel) değişken adı verilir.

main içerisindeki deger değişkeni ile Ornek fonksiyonu içerisindeki deger değişkeni birbirinden farklı değişkenlerdir. Birisinin değişmesi diğerini etkilemez.

```
#include <iostream>
using namespace std;
int Ornek(int deger)
{
        deger = deger * 2;
        return deger;
}
int main()
{
        int deger = 2;
        cout << "main deki deger="<<deger<<endl;
        cout << "fonksiyonun dondurdugu sonuc=" << Ornek(deger)<<endl;
        cout << "main deki deger=" << deger;
        return 0;
}
```



Şekil 7.8. Fonksiyondaki Lokal Değişken Kullanımı Ekran Çıktısı

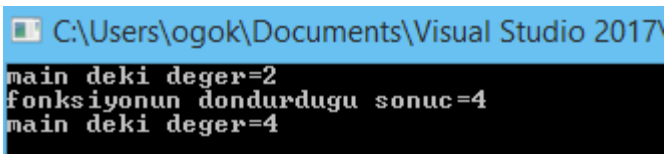
main içerisinde tanımlı deger değişkeninin içeriği fonksiyona gönderilmiştir. Burada dikkat edilmesi gereken, fonksiyon içerisinde tanımlı olan *deger* değişkeni ile *main* içerisindeki *deger* değişkeninin birbirinden farklı alanlar oluşudur. İki değişken de fonksiyonları içerisinde etki alanına sahip yerel değişkenlerdir.

Programın hata vermesinin sebebi main() fonksiyonundaki deger değişkenine Ornek() fonksiyonundan ulaşılamamasıdır. Çünkü deger değişkeni main() içinde tanımlı yerel bir değişkendir.

```
#include <iostream>
using namespace std;
int Ornek(int sonuc)
{
	sonuc = deger * 2;
	return sonuc;
}
int main()
{
	int deger = 2;
	cout << "main deki deger="<<deger<<endl;
	cout << "fonksiyonun dondurdugu sonuc=" << Ornek(deger)<<endl;
	cout << "main deki deger=" << deger;
	return 0;
}
```

Yukarıdaki kod çalıştırılmak istendiğinde derleyici: *'deger': undeclared identifier* hatası verir. Bunun sebebi *main* içerisindeki yerel değişken olan *deger* değişkenine *Ornek* fonksiyonu içerisinden erişilmeye çalışılmasıdır. *deger* değişkeni *main* yerelinde aktif olduğu için erişilemez. Erişilmesi için değişkenin global olarak fonksiyonların dışında tanımlanması gerekmektedir. Aşağıdaki örnek kodda da görüldüğü üzere bir değişken global değişken olarak tanımlanınca her fonksiyon tarafından erişilebilmektedir.

```
#include <iostream>
using namespace std;
int deger; //global değişken
int Ornek(int sonuc)
{
	deger= deger * 2;
	return deger;
}
int main()
{
	deger = 2;
	cout << "main deki deger="<<deger<<endl;
	cout << "fonksiyonun dondurdugu sonuc=" << Ornek(deger)<<endl;
	cout << "main deki deger=" << deger;
	return 0;
}
```



Şekil 7.9. Global Değişken Uygulaması Ekran Çıktısı

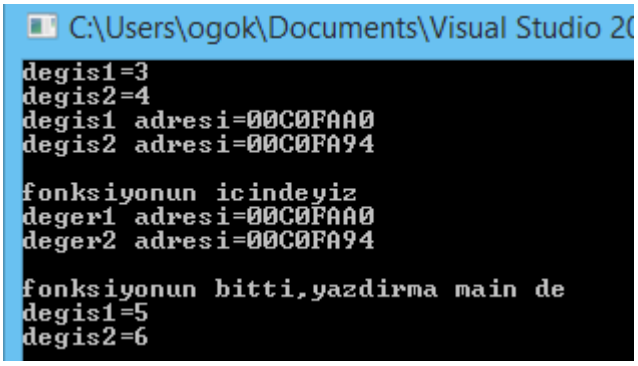
Fonksiyonlara Referans ile Parametre Aktarımı

Bir fonksiyon gövdesinde tanımlı bir değişken sadece ilgili fonksiyon içinde aktif olduğu için diğer fonksiyonlar bu tür değişkenlere normalde erişemezler. Fonksiyondan fonksiyona bilgi aktarımı değişken değerini diğer değişken değerine eşitlemek ile olur. Fakat yerel değişkenin değeri değil de adresi başka bir fonksiyona gönderilirse artık müdahale etme şansı olabilmektedir. Bu işlem işaretçiler yardımı ile yapılabilmektedir. İşaretçiler yardımı ile fonksiyonlar birden fazla değişken üzerinde değişiklik yaparak ve dolaylı olarak birden fazla çıkış üretebilir hale gelebilirler. Bunu yapabilmek için aşağıdaki kuralları bilmek gerekir:

İşaretçi (pointer) bellek adresinin değerini tutan bir değişkendir. İşaretçinin değişken tipi kapsadığı alanı tanımlar.

- & işareti herhangi bir değişkenin başına yazıldığında adresini verir.
- int *a tanımı yapıldığında, a tam sayı değil adres tutar.
- a adresinde tutulan değeri öğrenmek için *a kullanılır.

```
#include <iostream>
using namespace std;
void DegerleriDegistir(int *deger1, int *deger2)
{
	*deger1 = 5;
	*deger2 = 6;
	cout << endl<<"fonksiyonun icindeyiz"<<endl;
	cout << "deger1 adresi=" << deger1 << endl;
	cout << "deger2 adresi=" << deger2 << endl;
}
int main()
{
	int degis1=3, degis2=4;
	cout << "degis1=" << degis1<<endl;
	cout << "degis2=" << degis2<<endl;
	cout << "degis1 adresi=" << &degis1 << endl;
	cout << "degis2 adresi=" << &degis2 << endl;
	DegerleriDegistir(&degis1, &degis2);
	cout << endl << "fonksiyonun bitti,yazdirma main de" << endl;
	cout << "degis1=" << degis1 << endl;
	cout << "degis2=" << degis2 << endl;
}
```



Şekil 7.10. Fonksiyona Referans ile Değer Gönderme Ekran Çıktısı

Yukarıdaki örnekte main içerisinde tanımlı olan *degis1* ve *degis2* değişkenlerinin adresleri *&* işareti kullanılarak fonksiyona gönderilir. Fonksiyon değişkenlerin değerini değil de adreslerini almak için iki adet işaretçi tanımlar *(int *deger1, int *deger2)*. İşaretçiler *degis1* ve *degis2*'nin adreslerini alırlar. Aldıkları adresteki değerleri değiştirmek için **deger1* tanımını kullanırlar.**deger1'* in anlamı *deger1* değişkenin gösterdiği adresteki içeriktir. Yani 3 değerinin tutulduğu alandır. Bu alanın içeriği 5 yapılır. *main* içerisindeki alan ile aynı yer olduğu için *main* fonksiyonunun içindeki değer de değiştirilmiş olur. Bu sayede 2 referans girişi alan fonksiyon 2 adet dolaylı çıkış üretmiştir.

Asal sayı kendisi ve 1 dışında hiç bir sayıya tam bölünemeyen tam sayıdır.

Fonksiyondan Fonksiyon Çağırma

Fonksiyonlar istenen kod satırından veya istenen koşulla başka bir fonksiyonu çağırabilir ve sonuç aldıktan sonra kaldığı yerden işleme devam edebilir. Fonksiyonların başka fonksiyonları tek bir sefer çağırma zorunluluğu yoktur, tekrarlı olarak çağrılıp sonuç alınabilir.

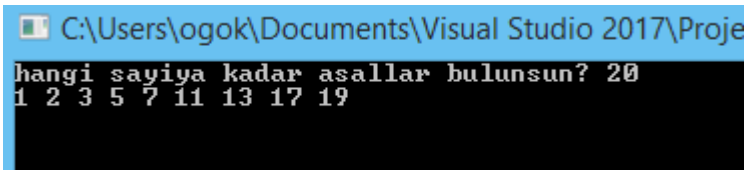
Örnek

- 1'den kullanıcıdan alınan tam sayı değerine kadar olan tüm asal sayıları ekrana basan C++ programını kodlayınız.

Çözüm:

```
#include <iostream>
using namespace std;
bool AsalMi(int kontrol)
{
	int i;
	for (i = 2; i<kontrol; i++)
	{
		if (kontrol%i == 0)
			return false;
	}
	return true;
}
void AsalListele(int sonDeger)
{
	int i;
	for (i = 1; i <= sonDeger; i++)
	{
		if (AsalMi(i)==true)
			cout << i << " ";
	}
```

```
}
int main()
{
	int sayi;
	cout << "hangi sayiya kadar asallar bulunsun? ";
	cin >> sayi;
	AsalListele(sayi);
	return 0;
}
```



Şekil 7.11. Asal Sayıların Listesi

Program kodları yukarıdan aşağıya doğru derlenir. Derleme anında tanımlı olmayan bir fonksiyonun çağrılması hataya sebep olur.

Asal sayıların bulunma işlemi için öncelikle kullanıcıdan sayi değişkeninin değeri alınır. İstenen değer *AsalListele* fonksiyonuna gönderilir.

AsalListele fonksiyonu 1 ile girilen sayıya kadar tüm tam sayıların asal olup olmadığını kontrol eder ve asal olanları ekrana yazdırır. Sayının asal olup olmadığını kontrol etmek için de *AsalMi* isimli alt fonksiyondan faydalanır.

AsalMi fonksiyonuna gönderilen sayının asal olup olmadığını kontrol etmek için 2'den gönderilen kontrol değerine kadar tüm sayılar için mod işlemi gerçekleştirilir. Eğer mod 0'a eşit olursa sayı asal değildir ve *false* değeri geriye döndürülür. Eğer hiçbirinin modu 0'a eşit olmazsa for döngüsünden çıkılır ve sayının asal olduğunu bildirmek için geriye *true* değeri gönderilir.

Fonksiyonları program içerisine yerleştirirken, eğer prototip tanımlaması yapılmadıysa yerleştirme sırası önemlidir. Örneğin; asal sayı örneğinde önce AsalListele fonksiyonu tanımlanırsa ve sonrasında AsalMi fonksiyonu yer alırsa, AsalListele fonksiyonu AsalMi fonksiyonundan haberdar olmadığı için program *"'AsalMi': identifier not found"* hatasını verir.

```
#include <iostream>
using namespace std;
void AsalListele(int sonDeger)
{
	int i;
	for (i = 1; i <= sonDeger; i++)
	{
		if (AsalMi(i)==true)
			cout << i << " ";
	}
}
bool AsalMi(int kontrol)
{
	int i;
```

```
        for (i = 2; i<kontrol; i++)
        {
                if (kontrol%i == 0)
                        return false;
        }
        return true;
}
int main()
{
        int sayi;
        cout << "hangi sayiya kadar asallar bulunsun? ";
        cin >> sayi;
        AsalListele(sayi);
        return 0;
}
```

Programın en başında prototip tanımlaması diğer fonksiyonları haberdar etmek içindir. Böyle bir fonksiyon var, hata verme!

Bu hatayla hiç karşılaşmamak için programın en üstünde prototip tanımlaması yapılabilir. Bu şekilde böyle bir fonksiyonun varlığından tüm fonksiyonların haberi olur. Bu durumda çözüm:

```
#include <iostream>
using namespace std;
void AsalListele(int);//Parametrenin sadece veri tipinin kullanılması yeterlidir
bool AsalMi(int kontrol); //Prototipte değişken adı kullanılsa da olur kullanılmasa da
int main()
{
        int sayi;
        cout << "hangi sayiya kadar asallar bulunsun? ";
        cin >> sayi;
        AsalListele(sayi);
        return 0;
}
void AsalListele(int sonDeger)
{
        int i;
        for (i = 1; i <= sonDeger; i++)
        {
                if (AsalMi(i) == true)
                        cout << i << " ";

        }
}
bool AsalMi(int kontrol)
{
        int i;
        for (i = 2; i<kontrol; i++)
        {
```

```
            if (kontrol%i == 0)
                return false;
        }
        return true;
}
```

Bireysel Etkinlik

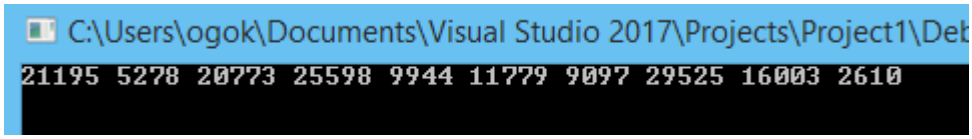
- OBEB ve OKEK değerlerini hesaplayan C++ fonksiyonlarını tanımlayınız.
- Faktoriyel hesabı yapan fonksiyonu tanımlayınız.
- Kullanıcıdan alınan tam sayının basamak değerlerinin toplamını bulan fonksiyonu yazınız (Örneğin, 3456 şeklindeki dört basamaklı sayının basamak değerleri toplamını 3+4+5+6=18 bulan fonksiyon).

Rastgele Sayı Üretim Fonksiyonları

Rastgele sayı üretmek için srand() kullanılmazsa her çalışmada aynı rastgele sayı üretilir.

Rastgele sayı üretimine bilgisayar uygulamalarında, özellikle oyun uygulamalarında sıkça ihtiyaç duyulur. C++ programlama dilinde zaman değişiminden yararlanarak rastgele sayı üretmek mümkündür. Rastgele sayı üretme fonksiyonu olan *rand*, bir çekirdek değer etrafında rastgele sayılar üretir. Çekirdek değer ise *srand* fonksiyonuna parametre olarak aktarılır. Program her çalıştığında farklı bir çekirdek değer kullanılsın diye de bir zaman fonksiyonundan faydalanılır. *rand* ve *srand cstdlib* kütüphanesinde, time fonksiyonu da *ctime* kütüphanesinde tanımlıdır.

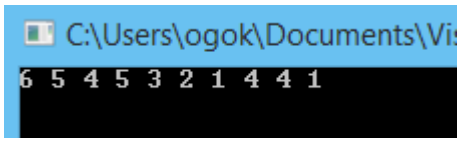
```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int main()
{
        srand(time(NULL));
        int sayi1,i;
        for (i=1; i<11;i++)
        {
                sayi1=  rand();
                cout << sayi1 <<" ";
        }
}
```



Şekil 7.12. Rastgele Üretilen 10 Adet Sayı

Rastgele üretimde sayı alanını daraltmak için rastgele üretilen sayı üzerinde mod alma, toplama veya çıkarma gibi aritmetik işlemler gerçekleştirilebilir.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int main()
{
	srand(time(NULL));
	int sayi1,i;
	for (i=1; i<11;i++)
	{
		sayi1= 1+ rand()%6;
		cout << sayi1 <<" ";
	}
	cin.get();
}
```



Şekil 7.13. Zar Atışı İçin Rastgele Üretilen Sayılar

Bir zar oyununda 1 ile 6 arasında rastgele sayıların üretilmesi gerekmektedir. Üretilen rastgele sayının 6'ya göre modu alınırsa 0 ile 5 arasında rastgele değerler üretilmiş olunur. 0 ile 5 arasında üretilen bu sayılara 1 eklenerek istenilen aralık elde edilebilir.

Özet

- Bu ünite kapsamında fonksiyonların programcıya ne tür faydalar sağlayacağından bahsedilmiş ve C++ programlama dilinde fonksiyonların nasıl kullanılabileceği açıklanmıştır. Bu manada;
  - C++ programlama dilindeki hazır kütüphane fonksiyonlarının kullanılma şekilleri anlatılmıştır.
  - Hazır fonksiyonlar programcıya problem çözümlerinde hız ve kolaylık sağlanması amacıyla tanımlanmıştır. Hazır fonksiyonlar için kütüphane çağrımı ve hazır fonksiyon prototiplerinin yorumlanması ve uygulanma şekli hazır matematiksel fonksiyonlardan örnekler verilerek anlatılmıştır. Matematiksel fonksiyonlar bilgisayardaki veri yapılarına göre farklılıklar gösterebilir. Bunun için C++ dilinde hazır matematiksel fonksiyonları kullanmak için *cmath* kütüphanesinin projeye eklenmesi ve kullanılması anlatılmıştır.
  - Programcı tarafından geliştirilen fonksiyonun dil kuralları içerisinde nasıl tanımlandığı gösterilmiştir. Fonksiyon tanımlama için sembol ve işaret kurallarının yanında giriş parametresi varsa tanımlanma şekli, fonksiyonun döndüreceği değere göre çıkış tipi tanımlaması anlatılmıştır.
  - Tanımlanan bir fonksiyonun diğer fonksiyonlarca nasıl çalıştırılacağı veya çağrılacağı, tanımlamalar yapılırken fonksiyonlardan üretilecek sonuçlara göre tiplerin nasıl tanımlanabileceği örneklendirilerek anlatılmıştır. Fonksiyon çağırma ve sonuç alma çeşitlilikleri üzerine örnekler tanımlanmıştır.
  - Fonksiyonlardaki değişkenlere erişim ilgili değişkenin nerede tanımlandığına bağlıdır. Değişkenler hangi fonksiyonda tanımlı ise sadece o fonksiyon içerisinde aktiftir ve değerini korur. Değişkenlerin tanımlandığı alana göre yerel ve global değişken kavramları, kullanım şekilleri ve tuttuğu değeri kullandığı alan için örnekler verilmiştir.
  - Fonksiyonlar parametrik olarak bir adet çıkış üretirler. Birden fazla çıkış üretmek için dolaylı bir yol izlenmesi gerekmektedir. Giriş parametresi olarak değişken değeri yerine adres kullanılırsa dışarıdaki bir değişkene sonuç döndürebilirler. Bunu ancak adres veya referans üzerinden yapabilirler. Bunun için referans göndererek diğer fonksiyonlardaki yerel değişkenlere erişimin nasıl yapılacağı ve adres tutan değişken tanımlaması anlatılmıştır.
  - Fonksiyonlar istenen kod satırından istenen koşulla başka bir fonksiyonu çağırabilir ve sonuç aldıktan sonra kaldığı yerden işleme devam edebilir. Fonksiyonların başka fonksiyonları tek bir sefer çağırma zorunluluğu yoktur, tekrarlı olarak çağrılıp sonuç alınabilir.Fonksiyonlar için kullanılan değişkenlerin hangi alanlarda aktif olarak değer tuttuğu, fonksiyonlardan diğer fonksiyonlar çağrıldığında sürecin hangi adımlarla işlediği örneklerle anlatılmıştır.
  - Rastgele sayı üretimi için hangi fonksiyonların kullanılabileceği sorusuna cevaplar verilmiştir.

DEĞERLENDİRME SORULARI

1. Fonksiyon prototipi aşağıdakilerden hangisinde yanlış tanımlanmıştır?
   a) int Soru(int a, int b);
   b) void Soru(int a, int b);
   c) double Soru(int a);
   d) void Soru();
   e) double Soru(int a, b);

2. Fonksiyon prototipi aşağıdakilerden hangisinde yanlış tanımlanmıştır?
   a) void Toplama1Yap(int a, int b);
   b) void ToplamaYap1(int a, int b);
   c) void ToplamaYap(int a, int b);
   d) void Toplama Yap(int a, int b);
   e) void ToplamaYap(int a, int c);

3.
```
#include <iostream>
using namespace std;
int Soru(int deg1 ,int deg2 , int deg3)
{
        deg1 = deg1 + deg2+deg3;
        return deg3;
        return deg2;
        return deg1;
        return 4;
}
int main()
{
        cout << Soru(1, 2, 3);
        return 0;
}
```
   Yukarıdaki C++ programı çalıştırıldığında ekrana ne basar?
   a) 1
   b) 2
   c) 3
   d) 4
   e) 6

4. Bir fonksiyonun çıkış değeri olarak 5.25 değeri geri döndürülmek isteniyor ise aşağıdaki tanımlamalardan hangisi kullanılmalıdır?
   a) int Fonksiyon(int a) {return 5.25;}
   b) double Fonksiyon(int a) {return 5.25;}
   c) bool Fonksiyon(int a) {return 5.25;}
   d) double Fonksiyon(int a) { a=5.25; return a;}
   e) void Fonksiyon() {return 5.25;}

5.
```
#include <iostream>
using namespace std;
int Degistir (int deg)
{       deg = 2;
        return 3;
}
int main()
{
        int deg = 1;
        cout << deg << " ";
        cout << Degistir(deg)<<" " ;
        cout << deg;
        return 0;
}
```

Yukarıdaki C++ programı çalıştırıldığında ekrana hangi değerler yazdırılır?

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

6.
```
bool Call (int deg )
{
        if (deg>=0 && deg < 10)
return true;
        else
return false;
}
```

Yukarıdaki Call() fonksiyonu için aşağıdakilerden hangisi doğru fonksiyon çağrımı olur?

a) int a=Call(5);
b) double a=Call(5);
c) if (Call(5)) cout<<"rakam";
d) Call(5);
e) Call(5)=a;

7.
```
#include <iostream>
using namespace std;
int Deneme(int a)
{
        a = a + 5;
        return a;
}
int main()
{
        int b=5;
        b=Deneme(b);
        cout << b;
        return 0;
}
```

Yukarıdaki C++ programı çalıştırıldığında çıktısı aşağıdakilerden hangisi olur?

a) Ekrana 5 yazar.
b) Ekrana 10 yazar.
c) Ekrana 5 10 yazar.
d) Hata verir, fonksiyon main içerisindeki değişken değerini değiştiremez.
e) Ekrana 0 yazar.

8.
```
#include <iostream>
using namespace std;
int F2(int c)
{
        c = c + 1;
        return c;
}
int F1(int b)
{
        b = b + 1;
        b = F2(b);
        return b;
}
int main()
{
        int a=1;
        a=F1(a);
        cout << a;
        return 0;
}
```

Yukarıdaki C++ programı çalıştırıldığında çıktısı aşağıdakilerden hangisi olur?

a) Ekrana 3 yazar.
b) Ekrana 1 yazar.
c) Ekrana 2 yazar.
d) Hata verir, bir fonksiyon bitmeden diğeri çalışmaz.
e) Ekrana 0 yazar.

9.
```
#include <iostream>
#include <cmath>
using namespace std;
double NoktalarArasiUzaklik(int x1, int y1, int x2, int y2)
{
        int sonuc;
        sonuc = pow(x2 - x1, 2) + pow(y2 - y1, 2);
        sonuc = sqrt(sonuc);
        return sonuc;
}
int main()
{
        cout <<NoktalarArasiUzaklik(1,1,5,4);
        return 0;
}
```

Yukarıdaki C++ programı çalıştırıldığında çıktısı aşağıdakilerden hangisi olur?

a) Ekrana 3 yazar.
b) Ekrana 4 yazar.
c) Ekrana 5 yazar.
d) Ekrana 6 yazar.
e) Ekrana 0 yazar.

10.
```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
        cout << sqrt(pow(4, 4));
        return 0;
}
```

Yukarıdaki C++ programı çalıştırıldığında çıktısı aşağıdakilerden hangisi olur?

a) Ekrana 2 yazar.
b) Ekrana 4 yazar.
c) Ekrana 8 yazar.
d) Ekrana 16 yazar.
e) Ekrana 256 yazar.

Cevap Anahtarı

1.e, 2.d, 3.c, 4.b, 5.a, 6.c, 7.b, 8.a, 9.c 10.d

YARARLANILAN KAYNAKLAR

[1] https://msdn.microsoft.com/en-us/library/3bstk3k5.aspx

HATA AYIKLAMA

Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

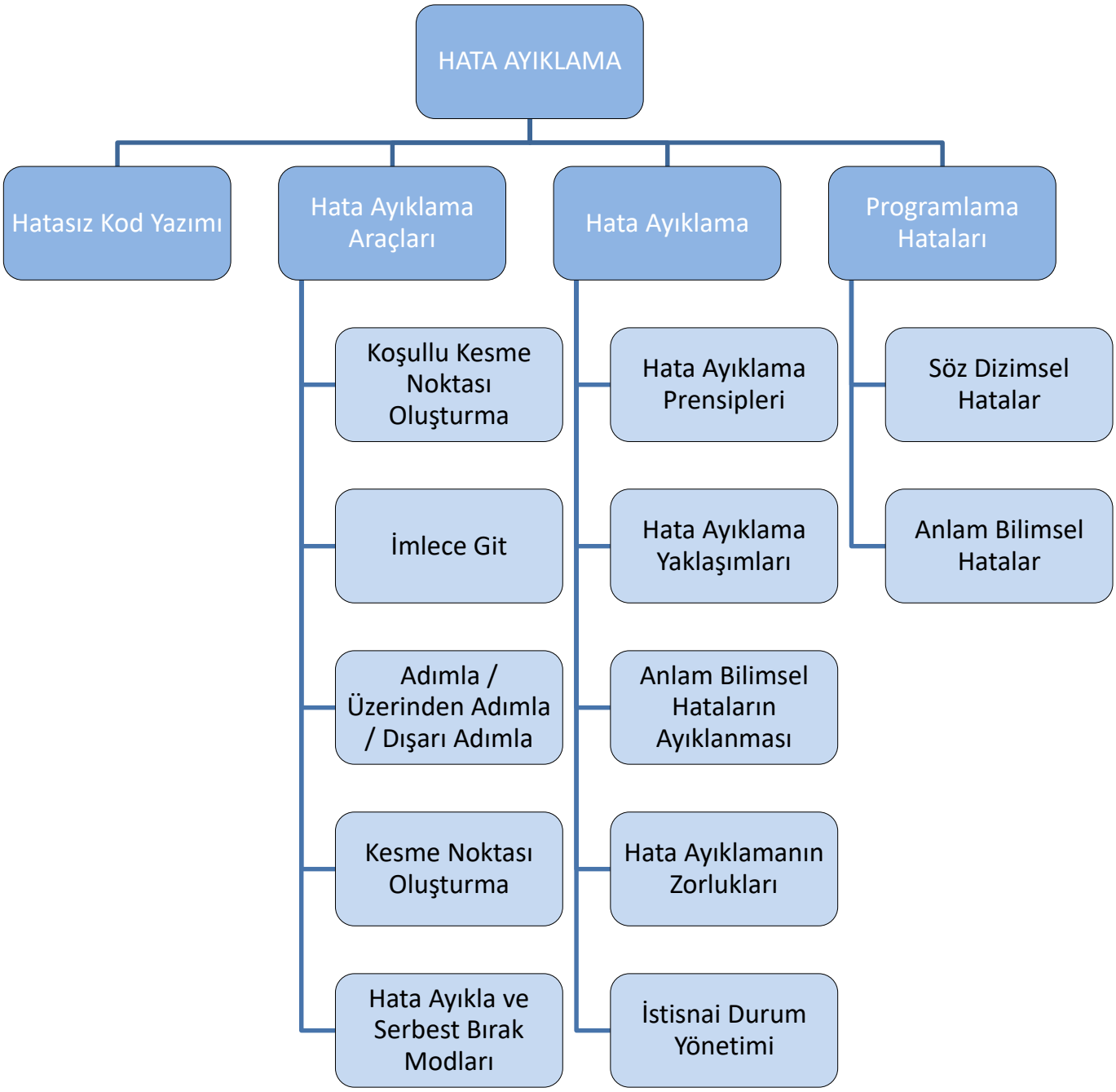
Arş. Gör. Sinan KUL

ÜNİTE 8

İÇİNDEKİLER

HEDEFLER

- Bu üniteyi çalıştıktan sonra;
  - Programlama hatalarının neden kaynaklandığını bilebilecek,
  - Söz dizimsel ve anlam bilimsel hata türlerini bilebilecek,
  - Hataların nasıl tespit edilebileceğini ve ayıklanabileceğini bilebilecek,
  - Hata ayıklama araçlarının nasıl kullanılabileceğini bilebilecek,
  - Hata ayıklama işinin neden zor olduğunu bilebilecek,
  - Hatasız kod yazımı için yapılması gerekenleri öğrenebileceksiniz.


HATA AYIKLAMA
Hatasız Kod Yazımı
Hata Ayıklama Araçları
Hata Ayıklama
Programlama Hataları
Koşullu Kesme Noktası Oluşturma
İmlece Git
Adımla / Üzerinden Adımla / Dışarı Adımla
Kesme Noktası Oluşturma
Hata Ayıkla ve Serbest Bırak Modları
Hata Ayıklama Prensipleri
Hata Ayıklama Yaklaşımları
Anlam Bilimsel Hataların Ayıklanması
Hata Ayıklamanın Zorlukları
İstisnai Durum Yönetimi
Söz Dizimsel Hatalar
Anlam Bilimsel Hatalar

GİRİŞ

Programcılar, programın geliştirilmesine gösterdikleri ilgiyi test aşamasına göstermedikleri için programda karşılaşılan hataların ayıklanması bazen geliştirme aşamasından daha fazla zaman almaktadır. Bu durum bilişim projelerinin bütçe, zaman ve maliyet analizlerini de zorlaştırmaktadır. Çalıştığı sanılan çoğu program beklenmeyen veri girişleriyle karşılaştığında hata vererek çalışmasını durdurmaktadır. Programın hata vermeden çalışmasını yanlış bir biçimde sürdürmesi ise daha vahim sonuçlar doğurabilmektedir.

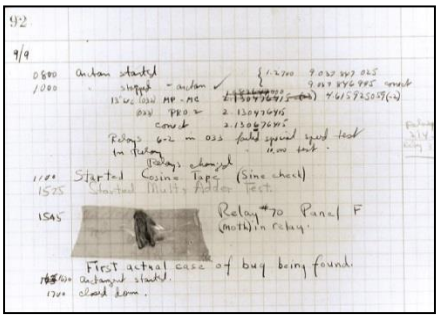
Hatasız kod yazımı kolay olmadığı için hata ayıklama işinin de ciddiye alınması gerekmektedir. Programın geliştirilmesinden imha edilmesine kadarki bütün aşamalarda hata ayıklama hesaba katılmalıdır.

Bu ünitede, hata kavramının ortaya çıkışından başlanarak hata türlerinden örnekleriyle birlikte bahsedilecektir. Hata ayıklama prensiplerinden ve yaklaşımlarından bahsedilecektir. Çalışma zamanında oluşan hataların ayıklanması için geliştirilen istisnai durum yönetimine de değinildikten sonra hata ayıklayıcı programlarda kullanılan araçlar incelenecektir. Son bölümde ise hatasız kod yazımı için neler yapılması gerektiği tartışılacaktır.

PROGRAMLAMA HATALARI

Hata (bug), bir bilgisayar sistemi tarafından üretilen beklenmeyen ya da doğru olmayan sonuçtur. Programın amaçlanan davranışı ile gerçek davranışı arasındaki fark, hatadan kaynaklanmaktadır. Bug (böcek) terimi, ilk olarak 1945 yılında Massachusetts Teknoloji Enstitüsündeki hesaplama laboratuvarında karşılaşılan hatanın, röleler içine sıkışıp kalan bir böcekten kaynaklandığı anlaşılınca "Böceğin ilk vakası bulunmuştur." ifadeleriyle rapor edilmesiyle kullanılmıştır [1]. *Böylece bug (böcek) terimi sistem hatası manasında kullanılmaya başlanmıştır.*

Programlarda karşılaşılan hataların %90'ı programlamadan kaynaklanmaktadır.



Şekil 8.1. İlk bug raporu [2]

Programlarda karşılaşılan hatalar istatistiksel olarak incelendiğinde; hataların %1'inin donanımdan, %9'unun işletim sisteminden ve %90'ının da programlamadan kaynaklandığı ortaya çıkmaktadır [3].

Peki, programcılar neden hata yaparlar?

- *Problemin düzgün tanımlanmaması:* Müşterilerin ilettiği problemin doğru anlaşılmamasından veya önerilen çözümün tanımlanmasındaki belirsizlikten dolayı hata oluşabilmektedir [4].

- *Yanlış algoritma kurulması:* Program için geliştirilen algoritmanın, geliştiricinin kurgusuna ve amacına uygun olsa da doğruyu temsil etmemesi durumudur. Bu durumda, yazılımcı bütün yazılım süreçlerini yinelemek zorunda kalabilir [5].
- *Kodlama hataları:* Kullanılan programlama dilinin kabiliyetlerine tam hâkim olunmaması kaynak kodun yazımında hatalara sebep olabilir. Bu tür hatalar çoğunlukla derleme zamanında ve derleyici tarafından tespit edilir. Geliştiricilerin algısına bağlı olan anlam bilimsel hatalar içinse kaynak kodu ayrıca analiz edilmelidir [4].

Söz Dizimsel Hatalar

Söz dizim (syntax) hatası, programlama dilinin yazım ve gramer kurallarının ihlal edilmesidir. Ölümcül derleme hatası olarak da bilinen söz dizim hatası, derleyici tarafından tespit edilip işaretlenir. Ancak hata, genellikle derleyicinin işaretlediği kod satırının üstündeki satırlardadır. Bir insan kadar iyi analiz edemediği için derleyici, hatayı çoğunlukla takip eden satırda tespit ederek yanlış bir mesaj ile görüntülemektedir. Derleyici tarafından görüntülenen hata sayısı gerçektekinden fazla bile olabilmektedir. Bu durum, özellikle programcılığa yeni başlayanlar için ciddi bir zorluk teşkil etmektedir [6].

Söz dizimsel hatayı derleyici tespit edebilmektedir.

Söz dizimsel hata örnekleri

Noktalı virgül hatası, tanımlanmamış değişken, kapatılmayan parantez, bitmeyen blok (küme parantezi), bitmeyen metinsel ifade, return eksikliği, parametre uyuşmazlığı ve tip uyuşmazlığı söz dizimsel hatalara örnek olarak verilebilir [6].

- *Noktalı virgül hatası:* C dil ailesindeki diğer dillerde olduğu gibi C++ dilindeki kodlar da “;” işaretiyle bitmektedir. Noktalı virgülün unutulması durumunda ise derleyici takip eden satırı işaretler ancak verdiği hata mesajı doğrudur.
- *Tanımlanmamış değişken:* Değişkenler C++ dilinde kullanılmadan önceki satırlarda tanımlanırlar. Böylece hangi tipte ve boyutta veriler için kullanılacağı belirtilerek bellekte yer tahsis edilir. Bir değişken tanımlanmadan kullanıldığında ise derleyici tarafından işaretlenir. Küçük/büyük harf duyarlılığına uyulmaması da C++ dili için hataya sebep olabilmektedir.
- *Kapatılmayan parantez:* Açılan bir parantezin kapatılmadığı tespit edildiğinde derleyici ilgili satırı işaretler. İç içe kullanılan parantezlerde en iç parantezlerden başlanarak dışa doğru gidilerek her bir parantezin kapatıldığı kontrol edilebilir.
- *Bitmeyen blok (küme parantezi):* Başlayan bir kırlangıç (C dili ailesi için) parantezinin (“{ }”) bitirilmemesi durumunda program kodunun son satırı işaretlenir.
- *Bitmeyen metinsel ifade:* C++ dilinde metinsel (string) ifadeler çift tırnak karakterleri (" ") arasında yazılır. İkinci çift tırnak karakterinin unutulması ise metinsel ifadenin sonlandırılmamasına sebep olur.
- *Return eksikliği:* Değer döndüren fonksiyonlar için return ifadesinin konulmaması hataya sebep olmaktadır.

- *Parametre uyuşmazlığı:* Fonksiyon tanımlarında kullanılan parametre sayısı ve veri tipleri ile çağrıldıklarında kullanılan parametrelerin uyuşmaması hataya sebep olmaktadır.
- *Tip uyuşmazlığı:* Değişkenlere değer atamada kullanılan verilerin tiplerinin, değişkenin veri tipiyle aynı olmaması çoğu zaman tespiti zor hatalara sebep olmaktadır. Ondalıklı sayı türünde bir değerin tam sayı türünde bir değişkene aktarılmasıyla değer kaybı oluşabildiği için hesaplama hatalarına sebep olabilmektedir.

"==" yerine "=" kullanımı gibi basit bir hata tespiti ve telafisi zor durumlar doğurabilmektedir.

Yazılım geliştirme ortamı, hata olmamasına rağmen, hata olabileceği düşünülen bazı durumlar için de uyarı vermektedir [6]. Örneğin:

- *Şart ifadesinde atama operatörü:* Diğer bazı dillerde karşılaştırma için kullanılan "=" operatörü, C dil ailesinde atama operatörüdür. Atama işlemi başarılı olduğunda sonuç doğru olacağından şart sağlanacaktır ve çalıştırılmayacağı düşünülen kod satırları yanlışlıkla çalıştırılacaktır. Kısacası "==" yerine "=" kullanımı gibi basit bir hata tespiti ve telafisi zor durumlar doğurabilmektedir.
- *Gövdesiz koşul ifadeleri:* Küme parantezleri kullanılmadan if ve while gibi koşullu deyimlerin kullanımı hataya sebep olabilmektedir. Örneğin, if deyimi kendinden sonraki ilk komutu şarta bağlamaktadır. Eğer birden fazla komutun şarta bağlanması istenirse blok içinde yani küme parantezleri içinde kullanılmalıdır.
- *İlk atama yapılmamış değişkenler:* Bir değişkene değer atarken bir önceki değerinden faydalanılıyorsa ve ilk değer ataması yapılmadıysa ya hataya sebep olacaktır veya hesaplama sonucu beklenmedik bir değer doğuracaktır.

Anlam Bilimsel Hatalar

Anlam bilimsel hatalar, dil bilgisi hatası barındırmadığı için derleyiciler tarafından tespit edilememektedir.

Anlam bilimsel hatalar (semantik), dil bilgisi hatası barındırmadığı için derleyiciler tarafından tespit edilememektedir. Çünkü programcının neyi amaçladığı ve ne yapmak istediği bilgisayar için pek açık ve anlaşılır değildir. *Derleyicinin kontrolünü geçerek, çalışma zamanında ortaya çıkan hatalara, anlam bilimsel hatalar denmektedir* [3].

Anlam bilimsel hatalar barındıran bir program düzgün bir şekilde derlenir ve çalışır. Ancak ya çalışmanın bir bölümünde hata vererek sonlanmaktadır ya da yanlış sonuçlar üretmektedir. Yani *anlam bilimsel hatalar en temelde ikiye ayrılmaktadır: Çalışma zamanı hataları ve mantık hataları* [5].

Çalışma zamanı hataları programın çakılmasına sebep olur; ancak çoğu zaman program çalışmaktadır. Bazen beklenmedik değişken değerleri ile bazen de program akışının yanlış yönlenmesi ile ortaya çıkmaktadır.

Anlam bilimsel hataların diğer bir türü olan mantık hataları ise yanlış sonuçlar veya çıktılar üretilmesine sebep olmaktadır.

Anlam bilimsel hataların tespit edilebilmesi zor ve zahmetlidir. Tespit edilebilmesi için bol test yapılmalıdır. Testlerde kullanılacak veriler ise öyle

Anlam bilimsel hatalar ikiye ayrılmaktadır: Çalışma zamanı hataları ve mantık hataları.

seçilmelidir ki hataları ortaya çıkartabilmelidir. Testlerde ayrıca tekrarlanma eğilimi yüksek olan hataların listesinin tutulması ise faydalı olacaktır. Bir hata ile karşılaşıldığında bu listeye bakılarak programın hangi kod bloğuna odaklanılacağı anlaşılabilir [3].

Anlam bilimsel hata örnekleri:

- *Sonsuz döngü:* Döngünün koşul kısmında kullanılan şart ifadesinin yanlış yazılması veya beklenmeyen bir değişken değerinin oluşması durumunda sonsuz döngü oluşabilmektedir. Yani döngü kaç defa çalışırsa çalışsın döngüden çıkılma şartı hep sağlandığından döngü dışına çıkılamamaktadır. Sonsuz döngü hatasının tespiti için en kolay yöntem, döngünün öncesine ve sonrasına cout gibi deyimlerle döngüye girildiği ve döngüden çıkıldığının yazdırılmasıdır. Şart ifadelerinde kullanılan değişken değerleri de yazdırılırsa hatanın sebebi ortaya çıkmış olacaktır. İkinci yöntem ise kesme noktalarının (breakpoint) eklenmesidir. Böylece program çalışırken, hangi döngüye girildiği ve hangi döngüden çıkılmadığı anlaşılır.

Örnek

•1'den 10'a kadar sayıların ekrana yazdırılmasını sağlayan C++ program kodunu, küme parantezlerini kullanmadan yazınız.

Yukarıdaki örnekte istenen program kodu aşağıdaki gibi yazıldığında, while döngüsündeki kodlar küme parantezleriyle (“{ }”) blok içine alınmadığından, döngüye ait olması gereken “x++;” kodu blok dışında kaldığı için kısır döngüye sebebiyet verilmektedir.

```
x = 1;
while (x < 11)
        cout<<"x\n";
        x++;
```

- *Döngü sayısı:* Döngü bloğunun beklenenden eksik veya fazla icra edilmesi beklenmeyen sonuçlar doğurabilecektir.

Örnek

•İlişkisel operatörlerin yanlış kullanılması dolayısıyla döngünün beklenen sayıda icra edilmemesine bir örnek veriniz.

Döngü kodlarının blok içinde kullanılmaması sonsuz döngüye sebep olabilmektedir.

Aşağıdaki kod örneğinde 10 defa icra edilmesi beklenen for döngüsünün koşul ifadesinde “i<=10” tanımlaması yapılması gerekirken yanlışlıkla “i<10” ifadesi yazılmıştır.

for (int i=1; i<10; i++)

Bu hata aynı zamanda dizi indislerinin 1 den başlatılması hatasıyla da benzerlik göstermektedir. Ancak dizilerle ilgili hatalar çoğu zaman çalışma zamanında dizi sınırlarının aşılmasıyla sonuçlanmaktadır.

- *Koşullu dallanma:* Çalıştırılmaması gereken bir kod bloğunun çalışması veya çalışması gereken bir kod bloğunun da çalışmaması, koşullu dallanmada belirtilen şartın beklenmedik sonuç doğurmasından kaynaklanmaktadır. Bu durum, karşılaştırma operatörlerinin yanlış kullanılmasından da kaynaklanabileceği gibi, değişkenlerin beklenmeyen değerler almasından da kaynaklanabilir.

Örnek

•Koşul deyimleri içinde mantıksal operatörlerin yanlış kullanımına bir örnek veriniz.

Aşağıdaki kod örneğinde "if" deyiminin koşulunda "veya" operatörü (||) kullanılması gerekirken yanlışlıkla "ve" operatörü (&&) kullanılmıştır. Dolayısıyla şart hiçbir zaman sağlanmamaktadır.

if (Cevap =='H' && Cevap =='h')

Aşağıdaki örnekte ise "ve" operatörü yerine "veya" operatörü kullanıldığı için koşul her zaman sağlanmaktadır.

if (str !='H' || str !='h')

Matematiksel operatörlerin önceliklerinin dikkate alınmaması hataya sebep olmaktadır.

- *Matematiksel işlemler:* Matematiksel operatörlerin önceliklerinin dikkate alınmaması veya işlemlerde yapılan otomatik yuvarlamaların bilinmemesi hatalı sonuç üretilmesine sebep olacaktır. Örneğin, "a=8-6/2" ifadesinde yapılmak istenen 8'den 6'nın çıkarılarak, sonucun 2'ye bölünmesidir. Sonucun 1 çıkması beklenirken 5 çıkmaktadır. Bu sorunu parantez kullanarak "(8-6)/2" şeklinde çözmek mümkündür. Benzer bir biçimde ondalıklı sayılarla çalışırken de her bir alt işlemin sonucunun ayrıca kontrol edilmesi gerekmektedir. Örneğin, ondalıklı bir sayının tam sayıya bölünmesi durumunda sonuç yuvarlanacaktır.
- *Sıfıra bölünme sorunu:* Çalışma zamanında ortaya çıkabilen bu hata, varsayılan veya olması gerekenden farklı bir değer girilmesi durumunda sıklıkla karşılaşılabilmektedir.
- *Dosyanın düzgün kapatılmaması:* Dosyalarla çalışılırken açılan her bir dosyanın kapatılması gerekmektedir. Açık olan bir dosyanın yeniden açılmaya çalışılması veya kapalı dosyaya yazılmaya çalışılması gibi durumlar hataya sebep olmaktadır.

- *Global/yerel değişken:* Global değişkenler programın her yerinden erişilebilen değişkenlerdir. Yerel (local) değişkenler ise fonksiyon içindeki değişkenler veya nesne içindeki özelliklerdir. Global değişkenlerin yerel değişkenler ile aynı isimde olması hatalı kullanımlara sebep olabilmektedir.
- *Else kullanımı:* Ardışık kullanılan iki "if" yapısının sonunda kullanılan "else" ifadesinin ilk if bloğuna ait olduğunun zannedilmesi de beklenmeyen sonuçlar doğuracaktır.
- *Dizi boyutu:* Dizi indisinin 1'den başladığının sanılması veya döngüde kullanılan değişkenin alabileceği sınır değerlerinin tespit edilememesi hataya sebep olabilmektedir.

Örnek

- For döngüsü içinde diziye değer atanmak istenirken, dizi sınırlarının aşılmasına bir örnek veriniz.

Aşağıdaki örnekte 5 elemanlı dizinin ilk elemanı diye 1 ile gösterilen indise yani 2. elemanına değer atanmaktadır. Son 5 ile gösterilen indis olmadığı için de hata vermektedir. Dikkat edin burada koşul ifadesi yerinde "i<5" kullanılsaydı hata alınmayacaktı ancak ilk elemana değer atanmadığından program muhtemelen doğru çalışmayacaktı.

```
for (i=1; i <= 5; i++) dizi[i]=i;
```

- *Akış yolu varsayımı:* Program, hata vermemesine rağmen bazı kod bloklarını çalıştırmadan sonlanıyorsa, programın varsayılan akışın dışına çıktığı anlaşılmalıdır. Programa öngörülmeyen veriler girdi olarak yüklendiğinde mesela, program farklı bir akış yolu izleyebilmektedir. Örneğin, bir sınavdan alınan notların tutulduğu değişkene -1 değeri girildiğinde şartlı deyimler beklenenden farklı davranacaktır. Yapılması gerekense, istenen değer aralığı dışında girilen değerlerin reddedilmesi ya da farklı bir dallanma izlenerek kullanıcıya mesaj görüntülenmesi olmalıdır [7].

Bireysel Etkinlik

- Size göre anlam bilimsel hata türlerinden hangisi daha zor tespit edilebilmektedir?

HATA AYIKLAMA

Hata ayıklama, teşhis algoritması ve hata düzeltme algoritmalarının birleşiminden oluşur.

Genel olarak yazılım hatalarının yerini bulma, hataları teşhis etme ve düzeltme işlemi hata ayıklama olarak tanımlanır [4]. *Hata ayıklama, teşhis algoritması (diagnosis algorithm) ve hata düzeltme algoritmalarının (bug-correction algorithm) birleşiminden oluşur.*

Hata Ayıklama Prensipleri

Hata ayıklama prensipleri çoğunlukla sezgiseldir. Uygulamada sıklıkla unutulur ya da dikkatten kaçırılır. Hata ayıklamanın iki temel algoritmanın birleşiminden oluştuğunu söylemiştik: Hatanın yerini bulma (tespit, teşhis) ve düzeltme.

Hatanın teşhis edilmesi

Hatanın teşhis edilmesinde insan marifetiyle neler yapılabileceğine kısaca değinelim [3]:

- *Düşünce:* Hata ayıklamanın etkili bir yöntemi, hata izleriyle sebeplerinin akıldan ilişkilendirilmesidir. Bunun için bir liste oluşturulmasının daha faydalı olacağına ise değinmiştik.
- *Şuuraltı:* Zihin başka işlerle meşgul iken veya dinlenirken şuuraltı mevcut problemin çözümüyle uğraşmaya devam etmektedir. Dolayısıyla hatanın sebebini bulma işi, şuuraltına da bırakılabilmektedir.
- *Başkası:* Karşılaşılan hatanın başkasına anlatılması durumunda sorun açık ve belirgin bir biçimde anlatılacağı için çözüm kendiliğinden bulunabilmektedir. Bu aşamada başkasının tecrübesinden de faydalanılabilir.
- *Deneme yanılma:* Deneme yanılma yoluyla hata yeniden üretilmeye ve hatanın yeri tespit edilmeye çalışılır. Bu yöntemde düşünme devre dışı bırakılır. Hatanın neden kaynaklandığı üzerine çok düşünülmez. Mevcut hata çözülse de çoğu durumda programa yeni hatalar eklenerek, program daha karmaşık bir hale getirilir.

Hatanın düzeltilmesi

Hataların düzeltilmesinde de bazı prensiplerden bahsedilmektedir [3]:

- *Hataların bir arada bulunması:* 10 hatanın 8'i programın en karmaşık olan %20'lik bölümünde bulunmaktadır. Dolayısıyla herhangi bir hata ile karşılaşıldığında, hatanın bulunduğu yerin civarında başka hataların da olması ihtimali akılda bulundurulmalıdır.
- *Hatanın izi, hata değildir:* Hatanın izini, belirtisini kaybetmek, hatayı bütünüyle düzeltmek ve ortadan kaldırmak demek değildir. Hata, bu durumda bazen kendini gizleyebilmektedir. Programın farklı giriş değerleriyle veya farklı çalışmasıyla ortaya çıkabilmektedir.
- *Düzeltmenin doğrulanması:* Hatayı düzeltmek için yapılan işlemler, öncesinde olmayan başka hatalara sebep olabilmektedir. Buna binaen programın tümüyle ve dikkatlice test edilmesi gerekmektedir.

Hata Ayıklama Yaklaşımları

Hatanın izini, belirtisini kaybetmek, hatayı bütünüyle düzeltmek ve ortadan kaldırmak demek değildir.

Brute force yaklaşımı

Kaba kuvvetle hata ayıklama yöntemi olarak da bilinen bu yaklaşımda yürütme anındaki davranışlar izlenir [7]. Pek verimli olmamasına rağmen, fazla düşünce gerektirmediğinden olsa gerek, en yaygın kullanılan yaklaşımdır [3]. Bu yaklaşımın üç uygulaması bulunmaktadır: Birincisi, programın çalışmasının belli bir noktasında program değerlerinin (değişken değerleri, bellek vs.) bir kısmının ya da tamamının ekrana yazdırılmasıdır.

İkinci uygulama kritik kod satırlarına cout ifadelerinin eklenerek ekranda bazı açıklama bilgilerinin görüntülenmesidir. cout ifadeleri eklendikten sonra program çalıştırılır ve çıktılar incelenir. Hatanın yeri tam bulunamazsa yeni cout ifadeleri yazılarak tekrar çalıştırılır. Hata düzeltildikten sonra ise cout ifadeleri silinir.

Üçüncü uygulama ise programın, belli noktalarda durdurulması ve program değerlerinin incelenmesi için hata ayıklama araçlarının kullanılmasıdır.

Kaba kuvvetle hata ayıklama yönteminin en önemli eksikliği, düşünme işini ikinci plana atmasıdır. Hâlbuki hatanın yerini bulmak için çoğu zaman programı geliştiren kişinin dikkatlice düşünmesi tek başına yeterli gelmektedir.

Tümevarım (induction) yöntemi

Tüme varım yöntemi, ilişkili verinin yerinin bulunması, verilerin organize edilmesi, hipotezin kurulması ve savunulması adımlarından oluşmaktadır.

Tümden gelim (deduction) yöntemi

Tümden gelim yönteminde, öncelikle bütün ihtimaller ortaya konmaktadır. Bu ihtimaller içinde elemeler yapıldıktan sonra kalanlar saflaştırılır ve son olarak hangi ihtimalin doğru olduğu tespit edilir. Programı geliştiren kişi bir hataya neyin sebep olabileceğini az çok bildiği için ihtimalleri ortaya koyar ve değerlendirir.

Test aşamasında hataların varlığı tespit edilebilirse de hatasızlık tespit edilemez.

Geriye doğru hata ayıklama

Yanlış sonuçlardan, program mantığının izlenerek geriye gidilmesi ve yanlış mantık yürütülen noktanın bulunması yöntemidir. Küçük programlar için etkili bir yöntemdir.

Bu yöntem, hataların izlerinden geriye doğru giderek hata ile ilişkili olan yerlerin kontrol edilmesi yöntemidir. Hatanın izi ile kendisi nasıl başka başka yerlerde olabilmektedir, açıklayalım. Örneğin, “b=8/a;” ifadesi 5. satırda bulunuyor olsun. Eğer bu satırda sıfıra bölünme hatası alınıyorsa, hatanın yerini tespit edebilmek için a’nın değerinin 0 olduğu satırın bulunması gerekmektedir. 4. satırda ise “a=8-2x;” ifadesi a’nın değerini 0 yapmaktadır. Ancak, esasında x’in değerinin 4 olduğu kod satırına gidilmesi gerekmektedir. Böylelikle hatanın ilk çıkış yeri tespit edilmelidir.

Test ederek hata ayıklama

Program geliştirmenin test ile birlikte yürütülmesi ve her aşamasında bulunması gerektiğini şu söz çok güzel ifade etmektedir: "Test işlemi bug'ların varlığını gösterir ancak yokluğunu asla göstermez [8]."

Test aşamasında program, örnek bir veri seti için kontrollü durumlar altında çalıştırılır. Hata varsa izlerinin belirlenmesini kolaylaştırmaktadır. Hata bulunduğunda hata ayıklama aşamasına geçilmektedir. Hata bulunamasa da sistemin hata vermeyeceğinden emin olunmamalıdır.

Hataların tespiti ve önemli ölçüde azaltılması için üç aşamalı test uygulanabilir [9]:

- *Dâhili Test:* Programı geliştiren kişi tarafından gerçekleştirilir. Bu test aşamasında programın bütün bölümlerinin en az bir kere icra edilmesini sağlayacak veri seti oluşturulur.
- *Harici Test:* Programı geliştiren kişi dışında birileri tarafından gerçekleştirilir. Harici test ile programın amacına uygun olarak çalışıp çalışmadığı test edilir. Kritik bazı işlevlerin gerçekleştirilebildiğinin anlaşılabilmesi için veri seti oluşturulur.
- *Kullanıcı Testi:* Son kullanıcı tarafından programın istek ve beklentileri karşılayıp karşılamadığı test edilir. Böylece ihtiyaç ve gereksinimlerin doğru bir şekilde analiz edilip edilmediği de anlaşılmış olur.

Test aşamalarının başarısını etkileyen en önemli faktör, veri setlerinin doğru seçilmesidir. Veri setlerinin, bütün hata durumlarının tespitini sağlayacak yelpazede olması gerekmektedir. Mümkünse her bir test işlemi birden fazla hata durumunu kontrol edebilmelidir. Hata düzeltme işlemi sonrasında da oluşabilecek yeni hataların tespitine de imkân sağlamalıdır. Yani hata varsa test aşamasında ortaya çıkmalıdır. Test aşamasından başarılı çıkan programın hatasız olduğu da bir noktaya kadar doğrulanabilmelidir [9].

Test aşamalarının başarısını etkileyen en önemli faktör, veri setlerinin doğru seçilmesidir.

Anlam Bilimsel Hataların Ayıklanması

Anlam bilimsel hataların ayıklanmasında üç yöntem kullanılmaktadır [6]:

- *El ile izleme (trace by hand):* Bu yöntemde program çıktıları el yordamı ile adım adım izlenir. Özellikle küçük ölçekli programlardaki hataların tespitinde yaygın olarak kullanılan bu yöntem, programın küçük birimlere ayrılarak her bir birimin ayrıca test edilmesinde de kullanılabilir. Bu yöntemde, programın çalışma mantığı üzerine sistematik bir biçimde düşünülmektedir. Bu yöntemin büyük programların bütününün incelenmesinde kullanılması, insan yeteneklerinin sınırlarını aşabildiği için bilgisayar desteği gerektirebilir.
- *Program izleme (program tracing):* Bu yöntemde, değişken değerlerini izlemek ve program akışını takip etmek için programın bazı kilit noktalarına cout gibi ifadeler eklenir. Böylece programın gidişatının hangi noktada sekteye uğradığı tespit edilebilmektedir. Peki programın kilit

noktaları nerelerdir ve hangi değerlerin görüntülenmesi hata ayıklamaya yardımcı olmaktadır?

- if deyiminden önce koşullarda kullanılan değişken değerlerinin görüntülenmesi
- Döngülere girmeden önce ve girdikten sonra koşullarda kullanılan değişken değerlerinin ve döngüden çıkıldıktan sonra da döngünün icra edilme (iterasyon) sayısının görüntülenmesi
- Fonksiyonlar çağrılmadan önce ve çağrıldıktan sonra parametre olarak kullanılan değişken değerlerinin görüntülenmesi
- Dizi boyutunun ve dizi eleman değerlerinin görüntülenmesi
- Sınıf nesnelerinin tanımlandıklarında, değişkenlere ilk değer ataması yapıldığında ve değişken değerleri değiştiğinde görüntülenmesi
- İşaretçi (pointer) değerlerinin değişmeden önce ve değiştikten sonra görüntülenmesi

- *Sistem etkileşimli hata ayıklama:* Bu yöntemde derleyicinin (compiler) veya hata ayıklayıcının (debugger) kabiliyetlerinden faydalanılır. Program, bazen adım adım çalıştırılır bazen programın kritik noktalarına kesme noktası (breakpoint) eklenerek programın o noktadaki durumu ve değişken değerleri analiz edilir. Bu yöntem hata ayıklama araçları bölümü altında ayrıntılı olarak anlatılacaktır.

Hata Ayıklamanın Zorlukları

Hata ayıklama işi, uzun bir uğraşı gerektirmektedir ve hayat boyu işletilmelidir.

Programın geliştirilmesine kıyasla test aşamasına yeterli zaman tahsis edilmemektedir. Oysaki doğru çalışmayan bir programın hatalarının ayıklanması geliştirme aşamasından daha karmaşık, zaman alıcı ve maliyetlidir. Hatta denilebilir ki hiçbir şey hata yapmak kadar maliyetli değildir. Hatasız kod ise neredeyse yok gibidir. Çalışıyor gibi görünen programların çoğu, önemli veri setleri için hata vererek çalışmayı durdurmaktadır [10]. Dolayısıyla hata ayıklama işi, uzun bir uğraşı gerektirmektedir ve hayat boyu işletilmelidir.

Hata ayıklama işinin zorluğu bazı hataların karakteristik özelliğinden de kaynaklanmaktadır. Hatanın belirtisi programın bir bölümünde iken hatanın sebebi başka bir bölümde olduğunda tespit işlemi zorlaşmaktadır. Programın çalışmasını etkilemeyen hataların tespiti de ayrıca ilgi gerektirmektedir.

Hata ayıklamanın zorluğu programcının yeteneğine, programın algoritmik karmaşıklığına ve geliştirme ortamına bağlıdır.

Hata ayıklama sonrasında da programın hatalardan tamamen temizlendiği düşünülmemelidir. Bir hata düzeltme işi sonrasında, hataların davranışının değişmesi, izlerinin kaybolması veya beklenmeyen hataların ortaya çıkması mümkündür. Hatta başka bir hatanın izinin kaybolması dâhi mümkündür. Bir hatayı düzeltirken başka birinin de görünmez olması durumunda, bu iki hatanın aynı ortak sebeple oluştuğu sanılabilecektir. Bu durumda farklı veri setleriyle yeniden kontrol edilmelidir [7].

Hata ayıklamanın zorluğunun en genel manada programcının yeteneğine, programın algoritmik karmaşıklığına ve geliştirme ortamındaki araçların kabiliyetine bağlı olduğu söylenebilir.

İstisnai Durum Yönetimi

Programın çalışması esnasında bazı kontrol edilmeyen faktörlerin çalışma zamanı hatalarına sebep olduğundan bahsedildi. Çalışma zamanı hatalarını tespit edip müdahale edebilmek için gelişmiş derleyicilerde, istisnai durum yönetimi bulunmaktadır. İstisnai durum yönetiminde her bir hata durumu bir sınıfa karşılık gelmektedir. Çalışma zamanında oluşan hatalar için derleyici tarafından oluşturulan nesnelere ise istisnai durum sınıf nesneleri denilmektedir. İstisnai durum nesneleri özetle çalışma zamanı hataları ile ilgili çeşitli bilgileri tutmak için gerekli özelliklere sahiptir. Yaygın olarak karşılaşılan istisnai durumlar [11]:

- Programın çalışmasını sürdürmesi için yeterli belleğin kalmaması
- Yığın (stack) bölgesinin birden fazla metot tarafından kullanılması
- Null değeri gösteren değişkene erişilmeye çalışılması
- Matematiksel işlem sonrasında değişkenlerin kapasitesinin aşılması (overflow)
- Geçersiz tür dönüşümü yapılması
- Bir dizinin indis sınırlarının aşılması
- Sıfıra bölünme
- Fonksiyonlara yanlış formatta parametre gönderilmesi

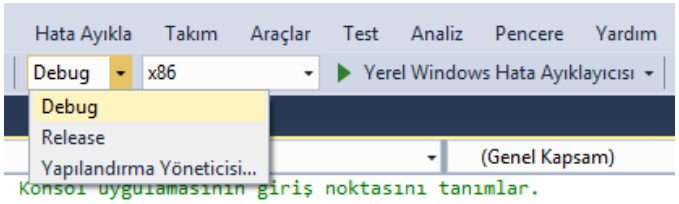
HATA AYIKLAMA ARAÇLARI

Hata Ayıkla (Debug) ve Serbest Bırak (Release) Modları

Program "Debug" modda derlendiğinde dosyalarda hata ayıklama bilgisi yer alır; ancak optimizasyonlar kapalıdır.

Debug mod geliştirme aşamasında sıklıkla kullanılırken, Release mod son kullanıcıya teslim öncesinde kullanılır. Özellikle programın test edilmesi aşamasında satır aralarına eklenen geçici kodlar da program teslim edilmeden önce silinmektedir.

Program "Debug" modda derlendiğinde dosyalarda hata ayıklama bilgisi yer alır; ancak optimizasyonlar kapalıdır. Yani dosya boyutları yüksek olurken performans düşüktür. Release modunda derlendiğinde ise optimizasyonlar açıktır.



Şekil 8.2. Debug ve Release modları

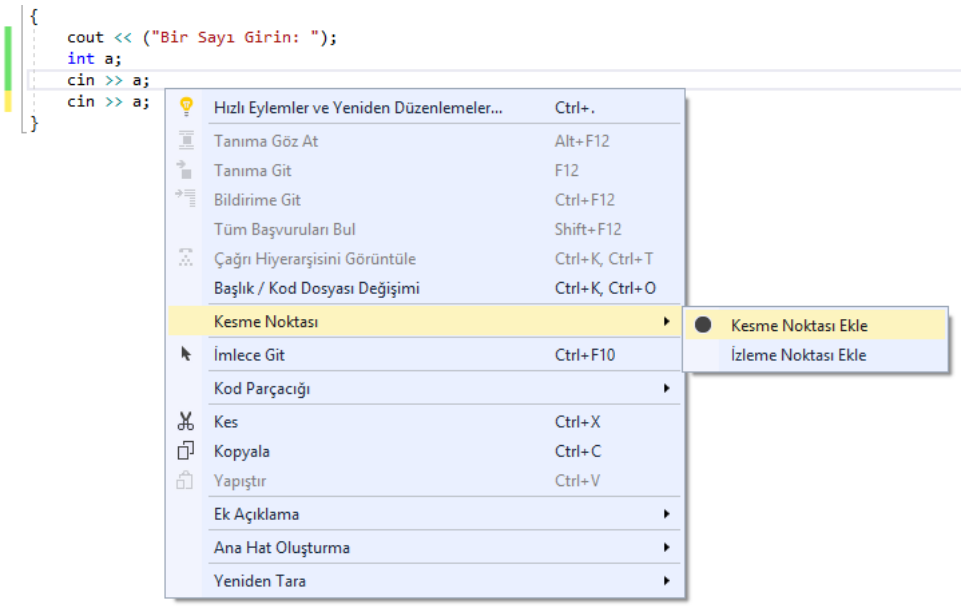
Kesme Noktası Oluşturma (Set a Breakpoint)

Kesme noktası, program çalışırken ilgili satırda çalışmanın durdurulmasını sağlar. Böylece programın akışı takip edilebilmekte ve değişken değerleri anlık olarak takip edilebilmektedir.

Editörde ilgili satırın sol kenarındaki gri bölüme tıklandığında kırmızı bir nokta oluşacaktır ve ilgili satıra kesme noktası eklenecektir. Editör içinde ve ilgili

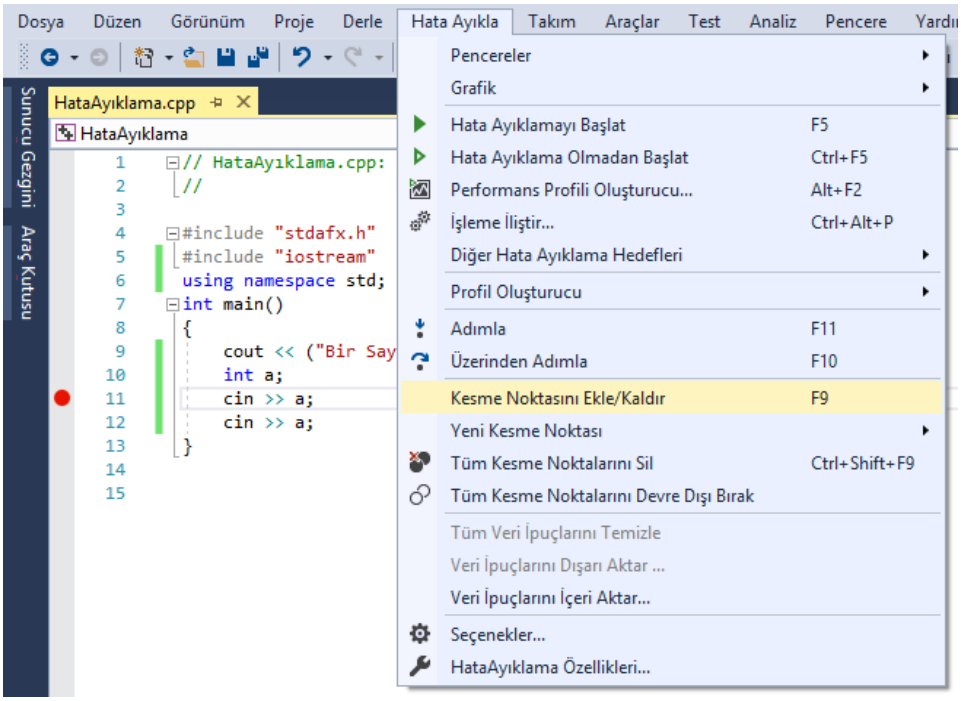
satırda sağ tıklandığında da “Kesme Noktası” ve “Kesme Noktası Ekle” denilerek de kesme noktası eklenebilir.

Kesme noktası, program çalışırken ilgili satırda çalışmanın durdurulmasını sağlar.



Şekil 8.3. Kesme noktası ekle

Benzer bir biçimde “Hata Ayıkla” menüsü altında bulunan Kesme Noktasını Ekle/Kaldır bağlantısına tıklanarak veya *F9 tuşuna basılarak da yeni kesme noktası eklenebilir*, varsa da kaldırılabilir.



Şekil 8.4. Kesme noktası ekle/kaldır

Program çalışması, ilgili satırda durduğunda ekran görüntüsü aşağıdaki gibi olacaktır. Herhangi bir değişkenin üzerine fare işaretçisiyle gelindiğinde ise ilgili değişkenin o andaki değeri görüntülenir.

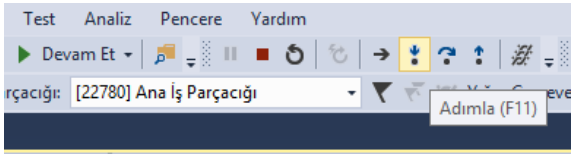
```
4   #include "stdafx.h"
5   #include "iostream"
6   using namespace std;
7   int main()
8   {
9       cout << ("Bir Sayı Girin: ");
10      int a;
11      cin >> a;
12      cin >> a;
13  }
14
```

a 3

Şekil 8.5. Değişken değerlerini görüntüle

Program, hata ayıklamasız olarak derlendiğinde program akışı, kesme noktasında durdurulamamaktadır.

Kesme noktasında durdurulan programın çalışmasını sürdürmesi için klavyeden F5 tuşuna veya araç çubuğu üzerindeki “Devam Et” düğmesine basılabilir. Adımla ve üzerinden adımla da uygulanabilir ki bir sonraki bölümde anlatılacaktır.



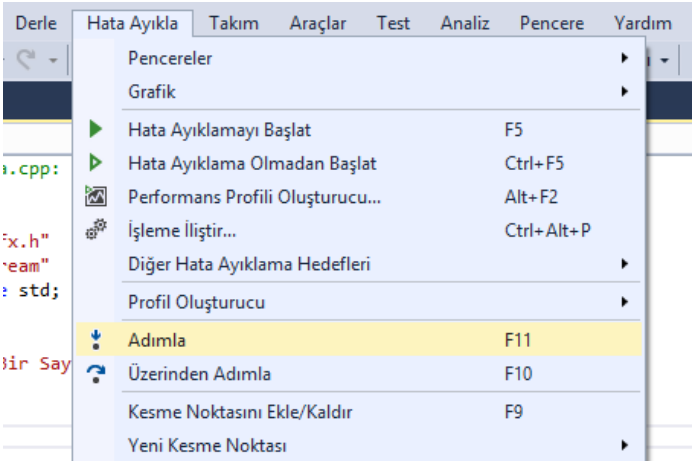
Şekil 8.6. “Devam et” düğmesi

“Hata Ayıkla” menüsü altından “Tüm Kesme Noktalarını” Sil bağlantısı tıklanarak veya Ctrl + Shift + F9 kısayolu ile kesme noktaları kaldırılabilmektedir.

Son olarak belirtmek gerekirse, program, hata ayıklamasız olarak derlendiğinde (Ctrl + F5) program akışı, kesme noktasında durdurulamamaktadır.

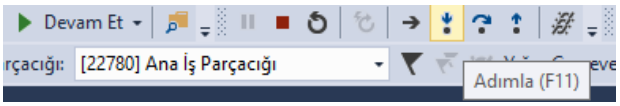
Adımla (Step Into)/ Üzerinden Adımla (Step Over)/ Dışarı Adımla (Step Out)

Program ilk satırdan itibaren adım adım çalıştırılarak hata ayıklanmak istenirse F11 tuşuna basılabilir veya “Hata Ayıkla” menüsü altından “Adımla” bağlantısına tıklanabilir. Böylece bir saat çevriminde (cycle) bir komut işletilerek programın adım adım çalıştırılması sağlanır.



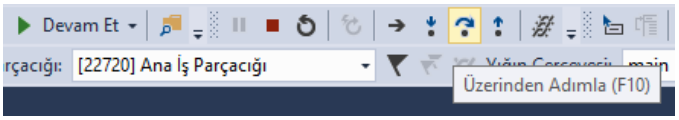
Şekil 8.7. “Adımla” bağlantısı

Program herhangi bir kesme noktasından itibaren de adım adım çalıştırılabilir. Yukarıdaki seçeneklere ek olarak araç çubuğu üzerindeki aşağı ok düğmesinden de "Adımla" uygulanabilmektedir.



Şekil 8.8. "Adımla" düğmesi

Herhangi bir fonksiyonun içindeki kodlar hata ayıklamaya tabi tutulmadan, fonksiyondan bir sonraki satırdan devam edilerek hata ayıklanması istenirse de "Üzerinden Adımla" uygulanabilmektedir. Üzerinden Adımla'nın kısayol tuşu ise F10'dur.



Şekil 8.9. "Üzerinden Adımla" düğmesi

Fonksiyon çağrılması kodunda Adımla uygulandıktan sonra fonksiyonun hemen dışındaki koddan devam etmek istenirse de "Dışarı Adımla" (Shift + F11) uygulanabilir.

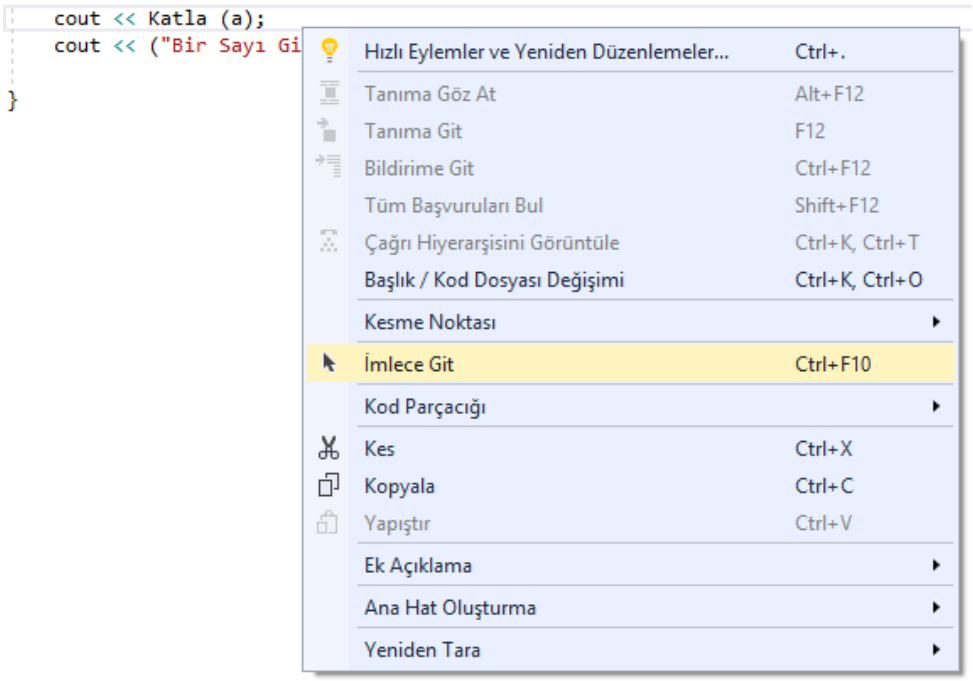
Şekil 8.10. "Dışarı Adımla" düğmesi

Geçici bir kesme noktası eklemek için "İmlece Git" uygulanabilir.

İmlece Git (Run to Cursor)

Geçici bir kesme noktası eklemek için "İmlece Git" uygulanabilir. Böylece imlecin bulunduğu satıra kadar program çalıştırılacak ve imlecin bulunduğu satırda beklenecektir.

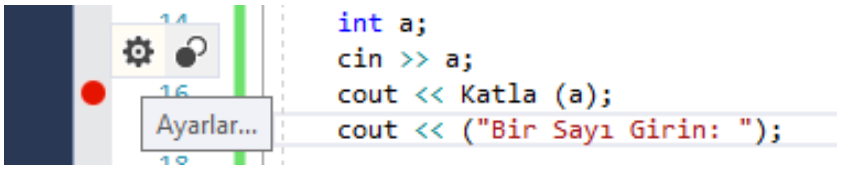
"İmlece Git" uygulamak için ilgili satırda sağ tıklandığında açılan menüden "İmlece Git" bağlantısına tıklanabilir veya Ctrl + F10 tuş kombinasyonu kullanılabilir.



Şekil 8.11. "İmlece Git" bağlantısı

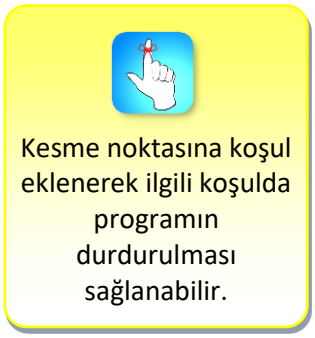
Koşullu Kesme Noktası Oluşturma

Kesme noktasına koşul eklemek için fare işaretçisi kırmızı nokta üzerine geldiğinde görüntülenen ayarlar bağlantısına tıklanmalıdır.



Şekil 8.12. Koşullu kesme noktası oluşturma

Bu esnada aşağıdaki gibi bir pencere görüntülenecektir. Bu pencerede bir koşul ifadesi yazılarak sadece bu koşul sağlandığında program akışının durdurulması sağlanabilmektedir. Örnekte "a==5" ifadesi yazıldığı için dışarıdan girilen değer 5 olduğunda kesme noktası devreye girmektedir.


Kesme noktasına koşul eklenerek ilgili koşulda programın durdurulması sağlanabilir.

Şekil 8.13. Koşullu kesme noktasına koşul tanımlama

Koşullu kesme noktaları içinde artı işareti bulunan kırmızı nokta şeklinde görüntülenmektedir.

HATASIZ KOD YAZIMI

Bir programın hatasızlığı yürütme işinin deterministik olmasına bağlıdır.

Bir programın hatasızlığı ve doğruluğu, kilitlenmelerin bulunmaması, yürütme işinin deterministik olması ve hata töleransı gibi bazı özelliklerin olması ile tanımlanmaktadır [12]. Peki hatasız kod yazabilmek için neler yapılmalıdır? Ünite boyunca zaten dikkate alınması gereken birçok durumdan bahsedildi. Burada başka neler yapılabileceğinden bahsedelim [7]:

Programlama dili özelliklerine hâkim olunması, hatasız kod yazımı için yapılması gerekenlerin başında yer almaktadır.

- *Programlama dili özellikleri:* Herhangi bir programlama diline yeni başlayanlar sıklıkla söz dizim hataları yapmaktadır. Bu bakımdan programlama dili özelliklerine hâkim olunması, hatasız kod yazımı için yapılması gerekenlerin başında yer almaktadır.
- *Girdi (input) değer:* Beklenmeyen bir değerin programa parametre olarak girilmesi, çalışma zamanı hatasına veya beklenmeyen bir kod bloğunun çalıştırılmasına sebep olabilmektedir. Bu durumun kontrol altına alınabilmesi için de kodlama sonrasında test veri setinin kapsamlı bir biçimde hazırlanması gerekmektedir.
- *Varsayılan (default) değer:* Hatasız kod yazımı için varsayılan değer bağımlılığının olmaması gerekmektedir. Örneğin, ilk değer ataması yapılmayan bir tam sayı değişkeninin 0 değerini barındırması varsayılan davranıştır. Varsayılan değerler, farklı makinalar, farklı işletim sistemleri ve farklı platformlar arasında farklılık gösterebildiği için, programın başka ortamlarda çalıştırılması hataya sebep olabilmektedir.
- *Sadelik:* Program kod bloklarının ve fonksiyonların 20 satırdan uzun olması istenmeyen bir durumdur. Onun yerine çoklu işlerin dağıtılarak her bir fonksiyonun bir işi yapacak şekilde tanımlanması, programın anlaşılabilirliğini ve geliştirilebilirliğini, program akışının takibini ve hata ayıklanmasını arttırmaktadır.
- *Adım adım değişiklik:* Kodlamada yaparken bir seferde tek bir değişiklik yapılması, olası hata olması durumunda önceki duruma geri dönülmesini kolaylaştırmaktadır.
- *Anlaşılırlık:* Kodun anlaşılırlığını ve okunurluğunu arttırmak, olası hataların sayısını en aza indirecektir. Yazılımın bütün aşamalarının dökümante edilmesi, kodlama ve isimlendirme standartlarına bağlı kalınması ve kod satırları arasına açıklama satırlarının eklenmesi kodun anlaşılabilirliğini arttırmaktadır.

Ödev

- Programlama hatalarına yol açan beklenmeyen değer girilmesi ve program akışının beklenmeyen sırada icra ettirilmesi durumlarının çözümünde insan bilgisayar etkileşimi perspektifinden neler yapılabilir? Araştırınız.

Özet

- Bu ünitede, hata kavramının ortaya çıkışından başlanarak hata türlerinden örnekleriyle birlikte bahsedilmiştir. Hata ayıklama prensiplerinden ve yaklaşımlarından da söz edilmiştir. Çalışma zamanında oluşan hataların ayıklanması için geliştirilen istisnai durum yönetimine de değinildikten sonra hata ayıklayıcı programlarda kullanılan araçlar incelenmiştir.
- Hata (bug), bir bilgisayar sistemi tarafından üretilen beklenmeyen ya da doğru olmayan sonuçtur.
- Söz dizim (syntax) hatası, programlama dilinin yazım ve gramer kurallarının ihlal edilmesidir. Bu tür hatalar derleyici tarafından tespit edilip işaretlenir.
- Noktalı virgül hatası, tanımlanmamış değişken, kapatılmayan parantez, bitmeyen blok, bitmeyen metinsel ifade, return eksikliği, parametre uyuşmazlığı ve tip uyuşmazlığı söz dizimsel hatalara örnek verilebilir.
- Anlam bilimsel hatalar, dil bilgisi hatası barındırmadığı için derleyiciler tarafından tespit edilememektedir.
- Anlam bilimsel hatalar çalışma zamanı hataları ve mantık hatalarıdır. Çalışma zamanı hataları programın çakılmasına, mantık hataları ise yanlış sonuçlar veya çıktılar üretilmesine sebep olmaktadır.
- Hata ayıklama, teşhis algoritması ve hata düzeltme algoritmalarının birleşiminden oluşur.
- Hatanın teşhis edilmesinde insan düşüncesini ve şuuraltını devreye sokabilir; hatayı başkasına anlatabilir veya deneme yanılma yoluna gidebilir.
- Hatanın izi yani belirtisi hatanın kendisi olmadığından, izini kaybetmekle hata düzeltilmiş olmamaktadır.
- Hata düzeltme sonrasında programın yeniden test edilmesi ve başka hataların ortaya çıkmadığından emin olunması gerekmektedir.
- Hata ayıklama yaklaşımları; brute force yaklaşımı, tümevarım yöntemi, tümden gelim yöntemi, geriye doğru hata ayıklama ve test ederek hata ayıklamadır.
- Test, üç aşamalı olarak uygulanabilir: Dahili test, harici test ve kullanıcı testi
- Test aşamalarının başarısını etkileyen en önemli faktör, veri setlerinin doğru seçilmesidir. Veri setleri, bütün hata durumlarını içermelidir.
- Anlam bilimsel hataların ayıklanmasında üç yöntem kullanılmaktadır: El ile izleme, program izleme ve sistem etkileşimli hata ayıklama
- Çalışma zamanı hatalarını tespit edip müdahale edebilmek için gelişmiş derleyicilerde, istisnai durum yönetimi bulunmaktadır. İstisnai durum yönetiminde her bir hata durumu bir sınıfa karşılık gelmektedir.
- Program “Debug” modda derlendiğinde dosyalarda hata ayıklama bilgisi yer alır; ancak optimizasyonlar kapalıdır.
- Kesme noktası, program çalışırken ilgili satırda çalışmanın durdurulmasını sağlar. Böylece programın akışı takip edilebilmekte ve değişken değerleri anlık olarak takip edilebilmektedir.
- Geçici bir kesme noktası eklemek için “İmlece Git” (Ctrl + F10) uygulanabilir.
- Kesme noktasına koşul eklendiğinde sadece bu koşul sağlandığında program akışının durdurulması sağlanabilmektedir.
- Bir programın hatasızlığı ve doğruluğu, kilitlenmelerin bulunmaması, yürütme işinin deterministik olması ve hata töleransı gibi bazı özelliklerin olması ile tanımlanmaktadır.

DEĞERLENDİRME SORULARI

1. "Programlarda karşılaşılan hataların çoğunluğu ……. 'dan kaynaklanmaktadır?"
   Cümlede boş bırakılan yere aşağıdakilerden hangisi getirilmelidir?
   a) Kullanıcı
   b) Programlama
   c) Donanım
   d) İşletim sistemi
   e) Hesaplama

2. I. Yazım ve gramer kuralların ihlal edilmesidir.
   II. Derleyici tarafından tespit edilebilmektedir.
   III. Program düzgün çalışır ancak beklenmeyen sonuçlar üretir.
   Yukarıda verilen söz dizim (syntax) hatalarıyla ilgili aşağıdakilerden hangisi ya da hangileri yanlıştır?
   a) Yalnız I
   b) Yalnız II
   c) Yalnız III
   d) I ve II
   e) I ve III

3. Söz dizimsel hatalar arasında aşağıdakilerden hangisi bulunmaz?
   a) Komut sonuna noktalı virgülün konulmaması
   b) Değişken tanımlamanın unutulması
   c) Değer döndüren fonksiyonlarda return olmaması
   d) Fonksiyon parametrelerinin sayısının tutmaması
   e) Dosyanın düzgün kapatılamaması

4. Hata ayıklama prensipleriyle ilgili aşağıda verilen bilgilerden hangisi yanlıştır?
   a) Hata izleriyle sebepleri akıldan ilişkilendirilebilir.
   b) Hatanın sebebini bulma işi, şuuraltına bırakılabilir.
   c) Hatanın başkasına anlatılması yoluna gidilebilir.
   d) Deneme yanılma yönüyle hata ayıklama en etkin yöntemdir.
   e) Hataların büyük bölümü bir arada bulunmaktadır.

5. Hata ayıklama yaklaşımları arasında aşağıdakilerden hangisi bulunmaz?
   a) Kaba kuvvetle hata ayıklama
   b) Geriye doğru hata ayıklama
   c) İleriye doğru hata ayıklama
   d) Test ederek hata ayıklama
   e) Tüme varım yöntemi

6. “b = (4 – 2c)/a ” kod satırında sıfıra bölünme hatası alınıyorsa aşağıdaki hata ayıklama yöntemlerinden hangisi uygulanmalıdır?
   a) Kaba kuvvetle hata ayıklama
   b) Tümevarım yöntemi
   c) Tümden gelim yöntemi
   d) Geriye doğru hata ayıklama
   e) Test ederek hata ayıklama

7. İmlece git komutu ne işe yaramaktadır?
   a) İmlecin bulunduğu konum işaretlenir.
   b) Program imlecin bulunduğu konuma kadar çalıştırılır.
   c) İmlecin bulunduğu konumdan sonrası çalıştırılır.
   d) Koşullu kesme noktası oluşturulur.
   e) Program adım adım çalıştırılır.

8. Herhangi bir fonksiyonun içindeki kodlar hata ayıklamaya tabi tutulmadan, fonksiyondan bir sonraki satırdan devam edilerek hata ayıklanması istenirse Üzerinden Adımla uygulanabilmektedir. Üzerinden adımla için aşağıdaki fonksiyon tuşlarından hangisine basılmalıdır?
   a) F8
   b) F9
   c) F10
   d) F11
   e) F12

9. Koşullu kesme noktasının oluşturulması ne işe yaramaktadır?
   a) Koşul sağlanınca ilgili satırda programın çalışması durdurulur.
   b) Koşul sağlanınca ilgili satırda programın çalışması devam eder.
   c) Koşul sağlanınca ilgili fonksiyonun çalışması durdurulur.
   d) Koşul sağlanınca ilgili fonksiyonun çalışması devam eder.
   e) Koşul sağlanınca imlecin bulunduğu satırda çalışma durdurulur.

10. Hatasız kod yazımı için aşağıdakilerden hangisi tavsiye edilmektedir?
   a) Test veri setinin kapsamlı olarak hazırlanması
   b) Varsayılan değer bağımlılığının olması
   c) Program kod bloklarının en az 20 satır olması
   d) Kodlamada bir seferde birden fazla değişiklik yapılması
   e) Kod arasına açıklama satırlarının eklenmemesi

Cevap Anahtarı

1.b, 2.d, 3.e, 4.d, 5.c, 6.d, 7.b, 8.c, 9.a, 10.a

YARARLANILAN KAYNAKLAR

[1] Shapiro, F. R. (1987). Etymology of the Computer Bug: History and Folklore. American Speech, 62(4), 376-378.

[2] Kopplin, J. (2002). An Illustrated History of Computers Part 3. 19 temmuz 2018 tarihinde http://www.computersciencelab.com/ComputerHistory/HistoryPt3.htm adresinden erişildi.

[3] Tahvildari, L. ve Singh A. (1999). Software Bugs. Wiley Encyclopedia of Electrical and Electronics Engineering, 445-456.

[4] Lourenço, J. M. d. S. (2003). *A Debugging Engine for Parallel and Distributed Programs*. Faculdade de Ciências e Tecnologia Departamento de Informática, Doktora Tezi, Lisboa.

[5] Barr, A. (2004). *Find the bug: a book of incorrect programs* (1. baskı). Kanada: Addison-Wesley Professional.

[6] Ford, A. R. and Teorey T. J. (2002). *Practical Debugging in C++* (1. baskı). New Jersey, USA: Prentice Hall.

[7] Sarıdoğan, M. E. (2008). *Yazılım Mühendisliği*. İstanbul: Papatya Yayıncılık.

[8] Dijkstra, E. W. (1972). *Structured programming* (1. baskı). London: Academic Press Ltd.

[9] Lauesen, S. (1979). Debugging techniques. Software: Practice and Experience, 51-63.

[10] O'Connor, M. (2008). Parallel debugging is easy. 21 Temmuz 2018 tarihinde http://www.allinea.com/Portals/90122/docs/allinea-parallel_debugging_is_easy.pdf adresinden erişildi.

[11] Algan, S. (2003). *Her Yönüyle C#* (4. baskı). İstanbul: Pusula Yayıncılık.

[12] Singh, A., Schaeffer J. ve Szafron D. (1998). Experience with parallel programming using code templates. Concurrency: Practice and Experience, 91-120.

TEK BOYUTLU DİZİLER

İÇİNDEKİLER

- Tek Boyutlu Diziler
- Dizi Tanımlama
- Diziye Değer Atama
- Karakter Dizisi
- Dizilerin Fonksiyona Gönderimi
- Dizilerde Arama
- Dizilerde Sıralama

HEDEFLER

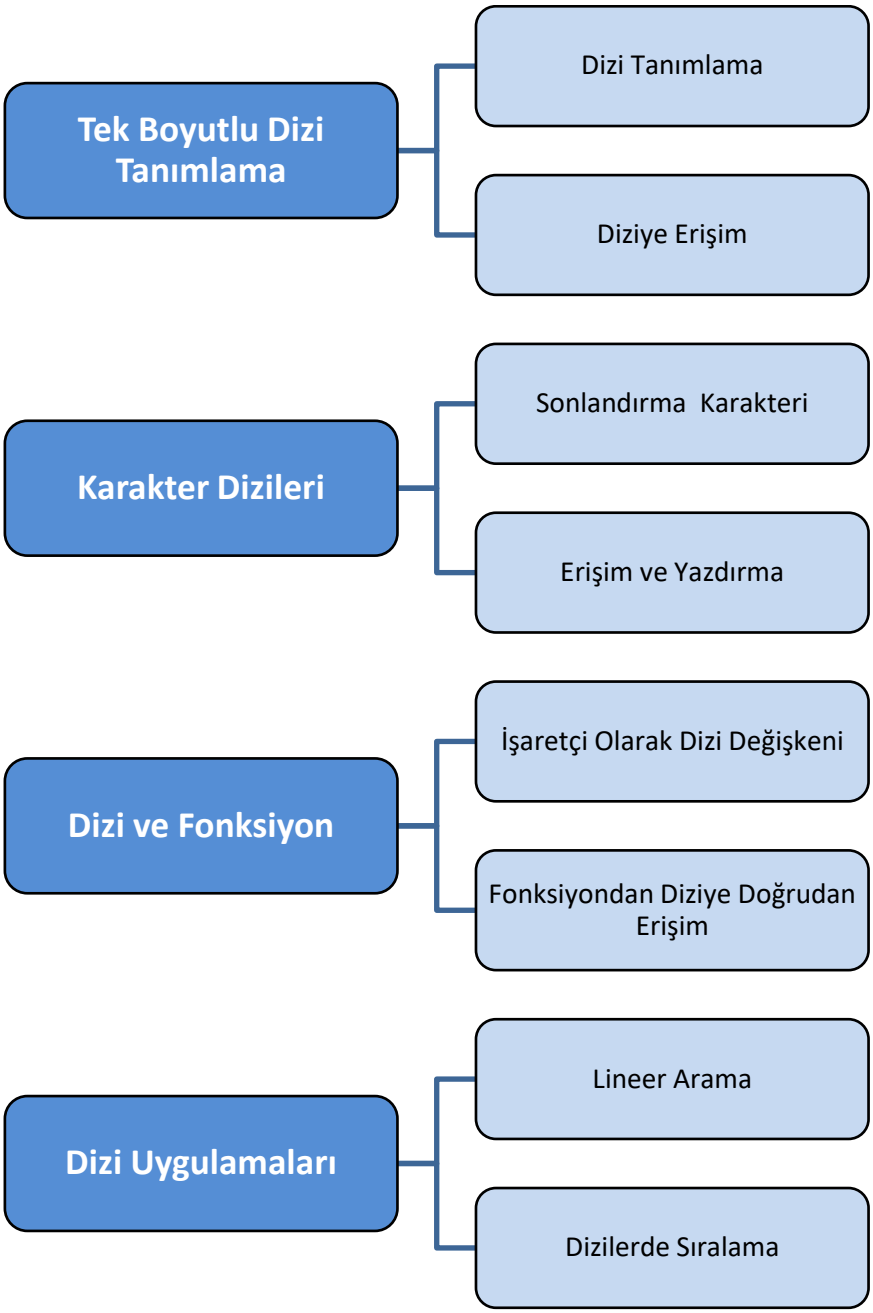
- Bu üniteyi çalıştıktan sonra;
  - Dizilerin kullanım amacını öğrenebilecek,
  - Tek boyutlu dizi tanımlayabilecek,
  - Dizilere değer atama ve erişim yapabilecek,
  - Dizilerin bellekte tutulma şeklini kavrayabilecek,
  - Karakter dizileri üzerinde işlemler yapabilecek,
  - Bir diziyi bir fonksiyona parametre olarak gönderebilecek,
  - Dizilerde arama ve sıralama işlemleri yapabileceksiniz.

Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

Dr. Onur GÖK

ÜNİTE
9


Tek Boyutlu Dizi Tanımlama
Dizi Tanımlama
Diziye Erişim
Karakter Dizileri
Sonlandırma Karakteri
Erişim ve Yazdırma
Dizi ve Fonksiyon
İşaretçi Olarak Dizi Değişkeni
Fonksiyondan Diziye Doğrudan Erişim
Dizi Uygulamaları
Lineer Arama
Dizilerde Sıralama

GİRİŞ

Diziler aynı veri tipinde ve sayısı belli olan birbiri ile ilişkili bir grup veriyi bellekte art arda saklayan indisli değişkenlerdir. Dizilerde veri sayısı sonludur ve tanımlıdır. Başka bir deyişle program çalışmaya başlamadan önce, yani derleme anında dizilerin bellekte kapladığı alan bellidir. Bellekte saklanan verilerin tipi aynı olmak zorundadır. Aynı tipteki verilerin yine bellekte art arda gelmesi verilere erişimi de hızlı ve kolay hale getirmektedir. Erişim işlemi indis veya sıra numarası ile sağlanır. Sıra numarası bellekteki adres değeri ile ilişkilidir.

Dizilerin kullanım amacı basittir ve nettir: Hesaplanacak aynı tipte verinin fazla olması. Aynı tipte bir grup veri üzerinde işlem yapılacaksa eğer, her veri için ayrı bir değişken tanımlamak hem kodlama hem zaman hem de erişim kontrolü açısından yersizdir. Bir grup veriyi saklamak ve işleme sokmak için farklı tanımlamalar mevcuttur; fakat dizi tanımlama bunun en kısa yoludur. Burada dikkat edilmesi gereken husus tıpkı diğer değişken türleri gibi dizilerin de bellekte tutulduğu gerçeğidir. Dizi boyutu, derleyicinin programcıya verdiği toplam alan kadardır. Alan dizinin tipine göre adet olarak büyür veya küçülür. Tam sayı olarak tanımlanan N elemanlı bir dizi, ondalıklı olarak tanımlanan N elemanlı bir başka diziye göre bellekte daha az yer kaplar.

Matematikte ve lineer cebirde kullanılan vektör veya matris tanımlamalarının karşılığı C++ programlamada dizilerdir. C++ dilindeki ayrım da buna benzerdir:

- Tek Boyutlu Diziler
- İki Boyutlu Diziler (Matrisler)

Bu ünite içerisinde tek boyutlu bir dizinin tanımlanması, dizi elemanlarına erişim, sayısal tipteki diziler, karakter tipindeki diziler, dizilerde arama ve sıralama yöntemleri anlatılacaktır.

Tek Boyutlu Diziler

Dizi Tanımlama

Dizi tanımlanırken mutlaka eleman sayısı bilinmelidir. int a[]={1,2,3,4} şeklinde de dizi tanımlanabilir. Bu durumda eleman sayısı 4'tür.

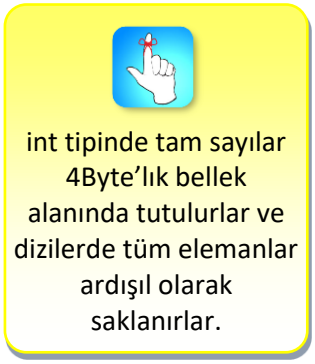
Matematikte ve lineer cebirde kullanılan vektör veya matris tanımlamalarının karşılığı C++ ile programlamada dizilerdir. Vektör şeklindeki tek boyutlu diziler aşağıdaki gibi tanımlanırlar:

Tip diziAdı[elemanSayısı];

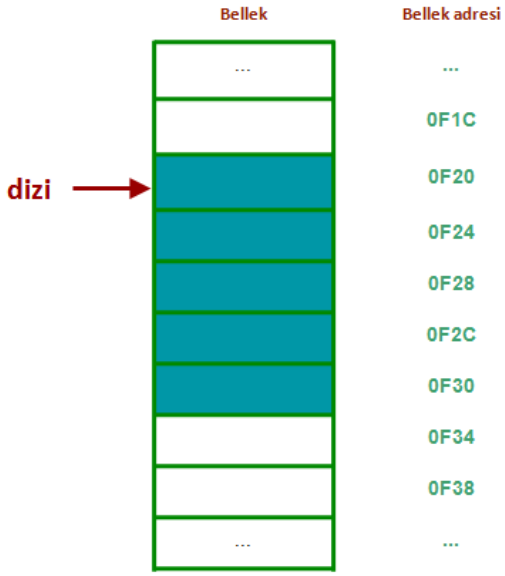
Tip aynı zamanda dizi değişkeninin bellekteki boyutunu etkiler. int, double, float, char olarak tanımlanabilir. Dizi isimlendirmesinde değişken tanımlamasındaki kurallar geçerlidir. Dizinin eleman sayısı köşeli parantez içerisinde verilir. Programın derlenip çalıştığı anda bellekten *eleman sayısı*tip boyutu* kadar alan tahsis edilir. Örnek olarak 5 elemanlı bir tam sayı dizi şu şekilde tanımlanır:

```
int dizi[5];
```

Bu tanımlamada dizi için 5x4 Byte boyutlu bir bellek alanı kullanım için hazırlanır. Dizi değişkeni ile normal değişken davranış yönünden farklıdır. dizi değişkeni bir işaretçidir. Diğer değişkenler gibi sayı veya karakter değeri tutmaz. Dizi değişkeni adres saklar. Sakladığı adres ise bellekte rezerve edilen alanın başlangıç adresidir. (Dizinin ilk elemanının adresi)



int tipinde tam sayılar 4Byte'lık bellek alanında tutulurlar ve dizilerde tüm elemanlar ardışıl olarak saklanırlar.



Şekil 9.1. Tanımlanan Dizinin Varsayımsal Bellek Gösterimi

Yukarıdaki örnekte bellekteki dizi değişkeninin değeri OF1C olur (Bkz. Şekil 9.1). char ve bool diziler için 1 Byte artış, double değer saklayan diziler için 8 byte artış ile adreslerin varsayımsal gösterimi yapılabilir.

Diziye Değer Atama

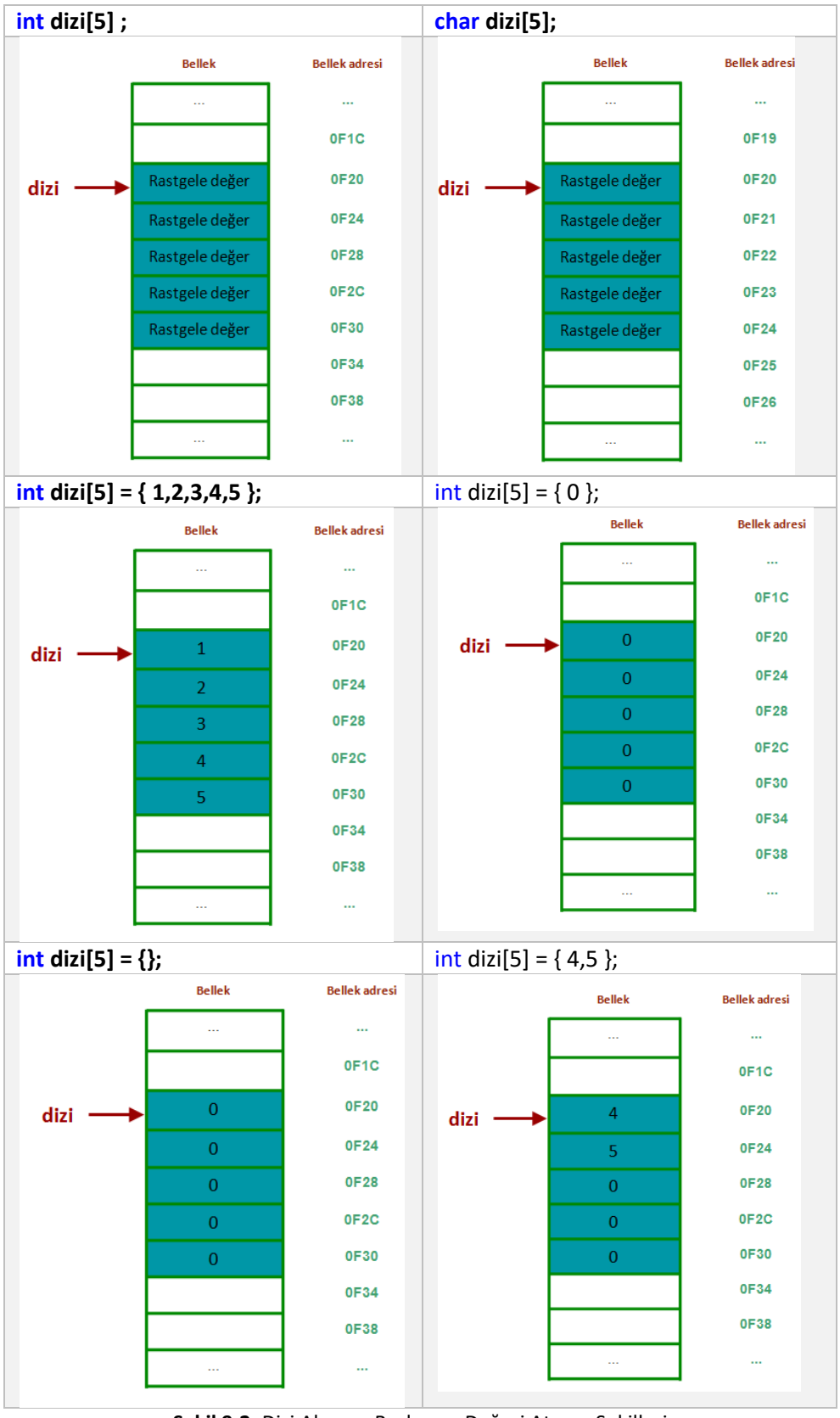
Dizi elemanlarına değer atama işlemleri için ihtiyaca göre farklı metotlar kullanılabilir. *Dizi değerleri ilk baştan atanabilir veya dizi tanımlandıktan sonra kullanıcıdan veya kod içerisinde yönelimli olarak atama yapılabilir.*

Başlangıç değeri atama

Bir diziye başlangıç değeri (ilk değer) atama işlemi için küme parantezi kullanılır. Örnek vermek gerekirse:

```
int dizi[5] ;
char dizi[5];
int dizi[5] = { 1,2,3,4,5 };
int dizi[5] = { 0 };
int dizi[5] = {};
int dizi[5] = { 4,5 };
```

Yukarıda tanımlı dizilerin başlangıç değerleri için bellek gösterimi Şekil 9.2'de yapılmıştır.



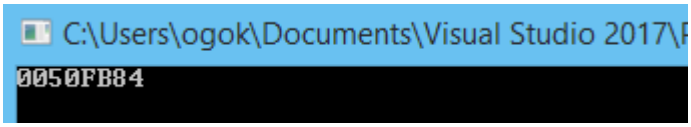
Şekil 9.2. Dizi Alanına Başlangıç Değeri Atama Şekilleri

Diziye sonradan değer atama ve indis

Dizi elemanlarına dizi tanımlandıktan sonra program içerisinde değer atama veya değer değiştirme işlemleri yapılabilir. Bunun için her bir dizi elemanına erişimin nasıl olduğunu bilmek yeterlidir. *Dizinin elemanlarına erişmek için indis kullanılır. İndis dizideki kayıt alanlarının sırasını verir.*

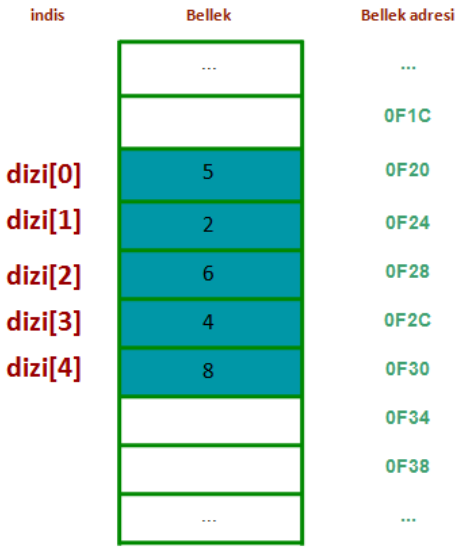
Diziye başlangıç değer atamaları küme parantezi içerisinde yapılır {}.

```
#include <iostream>
using namespace std;
int main()
{
        int dizi[5] = {5,2,6,4,8};
        cout <<dizi;
return 0;
}
```



Şekil 9.3. dizi Değişkeninin Değeri

dizi değişkeninin değeri dizinin ilk elemanının adresidir (Bkz. Şekil 9.3). Dizi elemanlarının içeriğine erişmek için ise köşeli parantez kullanılmalıdır. Dizi tanımlandıktan sonra program içerisinde diziye değer atama veya dizi elemanlarının değerlerini değiştirme işlemleri yapılabilir.

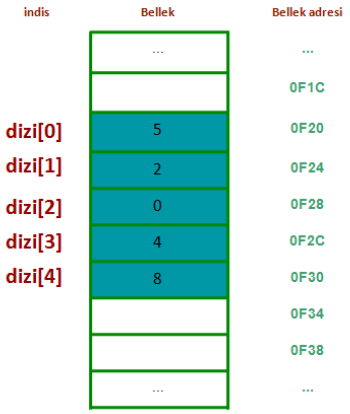


Şekil 9.4. Diziye İndis ile Erişim

dizi değişkeninin ilk elemanının indisi 0'dır. Dizi 5 elemanlıdır ve dizinin ilk elemanın indisi 0, son elemanın indisi de 4'tür. Dizinin x. elemanına ulaşmak için *dizi[x-1]* kullanılır. Yukarıdaki dizinin 3. elemanın değeri 6'dır. 3. elemanın değerini değiştirmek için *dizi[2]* kullanılır.

```
#include <iostream>
using namespace std;
int main()
{
        int dizi[5] = { 5,2,6,4,8 };
        dizi[2] = 0;
        return 0;
}
```

Yukarıdaki program çalıştırıldıktan sonra dizinin bellek haritası Şekil 9.5'te gösterilmiştir:



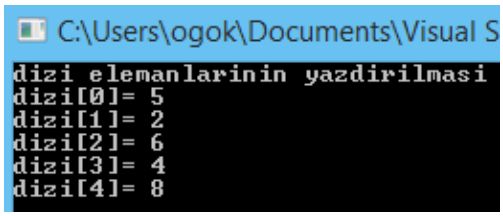
Şekil 9.5. Dizideki Değer Değişimi

Tüm diziye erişim

Dizinin tüm elemanlarına erişmek veya değerlerini değiştirmek için indis ve sayaç kullanılması gerekmektedir. Bu durumda sayaç değeri 0'dan başlar. Sayacın dizinin eleman sayısına kadar arttırılması ile tüm değerlere erişilir. Döngü içinde sayaç arttırımı gereklidir. Bunun için en kolay yol bir *for* döngüsü kullanarak dizide dolaşmaktır.

Tüm dizi içerisinde dolaşmak için kullanılan indis değeri örnekteki döngü ve sayaç ile kontrol edilebilir.

```
#include <iostream>
using namespace std;
int main()
{
        int dizi[5] = { 5,2,6,4,8 };
        int sayac;
        cout << "dizi elemanlarinin yazdirilmasi"<<endl;
        for (sayac = 0; sayac< 5; sayac++)
                cout <<"dizi["<<sayac<<"]= "<<dizi[sayac]<<endl;
        return 0;
}
```

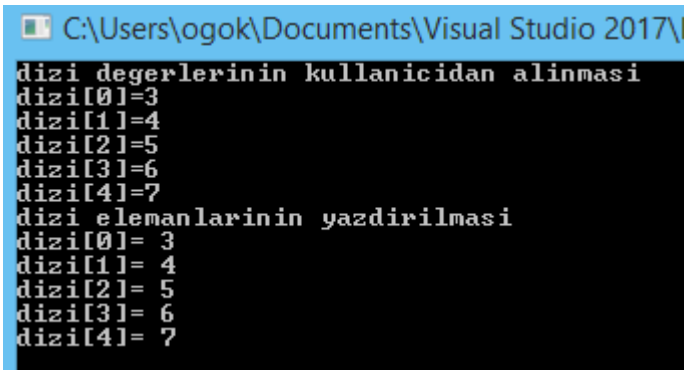


Şekil 9.6. Tüm Dizinin İçeriğinin Ekrana Yazdırılması

for döngüsü içerisinde her *sayac* artışında dizinin farklı bir *indis* değerine ulaşılmaktadır. *sayac sırasıyla* (0,1,2,3,4) değerlerini almaktadır.

Dizinin değerlerini kullanıcıdan almak için yine bir *for* döngüsü kullanılabilir. Aşağıdaki programda dizinin değerleri kullanıcıdan her adımda indis değeri sayaç ile 1 arttırılarak alınmıştır. Sonrasında kullanıcıdan alınan tüm değerler ekrana yazdırılmıştır.

```
#include <iostream>
using namespace std;
int main()
{
	int dizi[5];
	int sayac;
	cout << "dizi degerlerinin kullanicidan alinmasi" << endl;
	for (sayac = 0; sayac< 5; sayac++)
	{
		cout << "dizi["<<sayac<<"]=";
		cin>> dizi[sayac];
	}

	cout << "dizi elemanlarinin yazdirilmasi"<<endl;
	for (sayac = 0; sayac< 5; sayac++)
		cout <<"dizi["<<sayac<<"]= "<<dizi[sayac]<<endl;
	return 0;

}
```



Şekil 9.7. Dizi Değerlerinin Kullanıcıdan Alınması

Karakter Dizisi

Sayısal dizilerden farklı olarak karakter dizilerinde başlangıç değeri atamak için çift tırnak (" ") gerekir.

Karakter dizileri, tanımlama ve işleyiş bakımından sayısal tipteki dizilere benzemek ile beraber belleğe kayıt yapılarında ve yazdırma işlemlerinde ek özellikleri vardır. Karakter dizisi *string* veya karakter katarı olarak da isimlendirilmektedir. String tanımlamaya bir örnek şu şekilde verilebilir:

```
char karakterDizisi[20];
```

Yine tip, değişken ismi ve boyut belirtilmektedir. Karakter dizisine başlangıç değeri çift tırnak içinde atanabilir.

```
char karakterDizisi[20]="deneme";
```

Dizinin herhangi bir karakterine erişim sayısal dizilerde olduğu gibi indis kullanılarak yapılır.

Sonlandırma karakteri

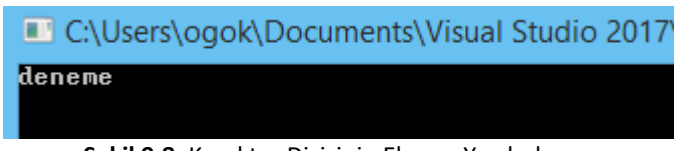
Karakter dizilerinin sayısal tipteki dizilerden en önemli farkı, sonlandırma karakteri bulunmasıdır. Sonlandırma karakteri olarak '\0' (NULL) kullanılmaktadır. Sonlandırma karakteri diziyi yazdırma ve kontrol için kolaylık sağlamaktadır.

Karakter dizilerinde de tüm boyut kullanılmasa dahi başlangıçta tanımlı boyut kadar bellekten yer tahsis edilir.

```
char karakterDizisi[20]="deneme";
```

Program çalışmaya başladığında *karakterDizisi* 20 karakterlik bir alanı bellekten alır. Fakat hepsini kullanmaz. Dizinin nereye kadar kullanıldığını tespit etmek için tüm karakterlerin kontrol edilmesi zorunludur. Bunun yerine kaydedilen karakterlerin bittiği yere işaret koymak bir çözüm oluşturur. Bu işaret C++ programlama dilinde *\0* karakteridir.

```
#include <iostream>
using namespace std;
int main()
{

    char karakterDizisi[20]="deneme";
    cout << karakterDizisi;
return 0;
}
```



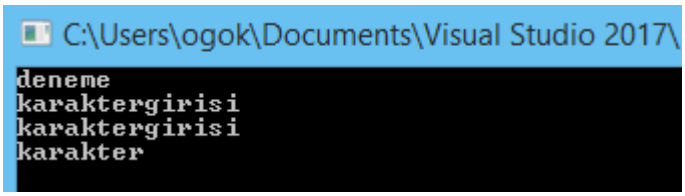
Şekil 9.8. Karakter Dizisinin Ekrana Yazdırılması

Kod ve ekran çıktısına dikkat edilirse bir karakter dizisini ekrana yazdırmak için herhangi bir döngüye gerek olmadığı anlaşılır. *karakterDizisi* ekrana yazdırılmak istendiğinde sayısal tipli dizilerden farklı olarak dizi içeriği *\0* değerine kadar dikkate alınır.

Şekil 9.9 karakterDizisi Değişkeninin Bellek Gösterimi

Sonladırma karakteri (\0) dizinin sonuna değil girilen değerin sonuna konulur.

```
#include <iostream>
using namespace std;
int main()
{
        char karakterDizisi[20]="deneme";
        cout << karakterDizisi<<endl;
        cin >> karakterDizisi;
        cout << karakterDizisi << endl;
        karakterDizisi[8] = '\0';
        cout << karakterDizisi;
        return 0;
}
```

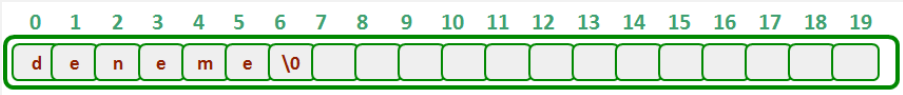


Şekil 9.10 karakterDizisinin Değer Değişimi

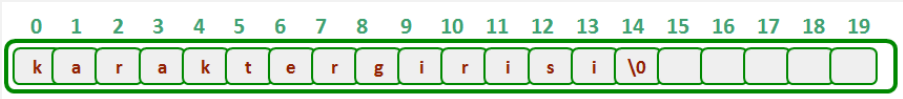
Program çalıştığında her değişim için dizinin bellekteki durumu aşağıdaki şekilde verilmiştir.

Ekran çıktısından da anlaşılacağı gibi cout fonksiyonu karakter dizisinin sonlandırma işaretine kadar olan kısmını ekrana yazdırmıştır. (Bkz. Şekil 9.10)

char karakterDizisi[20]="deneme";



cin >> karakterDizisi; // Kullanıcı giriş olarak "karaktergirisi" değerini girmiştir.



karakterDizisi[8] = '\0'; //sonlandırma karakteri dizinin 9. karakteridir.

Şekil 9.11. karakterDizisi Değer Değişimi Bellek Gösterimi

Dizilerin Fonksiyonlara Gönderimi

Dizi değişkeni bir işaretçidir. Dolayısıyla fonksiyona değeri değil adresi gönderilir.

Dizilerin fonksiyonlara gönderimi için dizi değişkeninin tanımından yararlanılır. Dizi değişkeni belli bir sonlu kayıt listesi için başlangıç adresini işaret eder. Bu işaret değişkenin değer değil adres tutması anlamına gelir. Fonksiyona gönderilen dizi değişkeninin değeri değil adresidir. Fonksiyon konusundan hatırlanacağı üzere referans değeri gönderilerek tanımlanan fonksiyonlarda işaretçi değişken adrese eriştiği için farklı bir fonksiyondaki değişkenin alanına dolaylı olarak erişim sağlanmış olmaktaydı. Bu nedenle fonksiyon içerisinde yapılan her türlü değişiklik dizinin kendisi üzerinde olacaktır. Dolaylı olarak dizi değişkeni fonksiyonlara adres gönderdiği için fonksiyon bitene kadar global değişkenmiş gibi davranacaktır.

```
#include <iostream>
using namespace std;
void DiziyiYazdir(int fonksiyonDizi[],int diziBoyut)
{
        int i;
        for (i = 0; i < diziBoyut; i++)
                cout << fonksiyonDizi[i] << " ";
}
int main()
{
        int dizi[20] = { 0,2,4,6,8,10,12 };
        DiziyiYazdir(dizi, 7);
return 0;
}
```

Dizi içeriğini ekrana yazdırmak için kullanılan fonksiyon yukarıda tanımlandığı üzere iki parametre almaktadır. *fonksiyonDizi* değişkeninin bir dizi olduğunu gösteren köşeli parantez aynı zamanda onun adres tuttuğunu göstermek amaçlı prototipidir. İkinci değişken ise dizinin baştan itibaren kaç elemanının yazdırılacağının bilgisini saklamaktadır. *fonksiyonDizi* değişkeni ve main içerisindeki *dizi* değişkeni bellekte ayrı değişkenlerdir fakat tuttukları değerler aynı alanı işaret etmektedir.

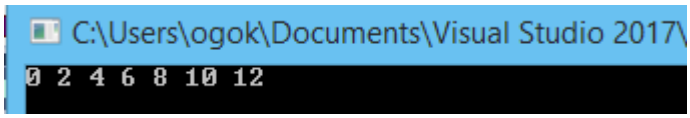
```
#include <iostream>
using namespace std;
void DiziyiYazdir(int *fonksiyonDizi, int diziBoyut)
{
        int i;
        for (i = 0; i < diziBoyut; i++)
                cout << fonksiyonDizi[i] << " ";
}
int main()
{
        int dizi[20] = { 0,2,4,6,8,10,12 };
        DiziyiYazdir(dizi, 7);
```

```
return 0;
}
```

Dizinin adresinin fonksiyona gönderilmesi için tanımlanan *fonksiyonDizi değişkeni işaretçi olduğu için main içerisindeki dizi değişkenini başlangıç adresi olarak kullanır. fonksiyonDizi[] ve *fonksiyonDizi değişkenleri anlam bakımından birbirine eşittir ve her ikisi de adres saklarlar.

Fonksiyona parametre olarak dizi aktarılırken [] veya * işaretleri kullanılabilir. Bu ilgili değişkenin adres değeri aldığını gösterir.

Fonksiyon çağrımında ise fonksiyona dizi değeri yani dizinin başlangıç adresi gönderilmiştir. Her iki program da aynı ekran çıktısını vermektedir (Bkz. Şekil 9.12).



Şekil 9.12 Dizi İçeriğinin Fonksiyonda Yazdırılması

Dizilerde Arama

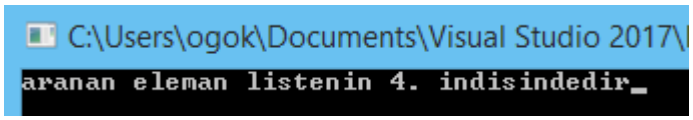
Bir dizi içerisinde arama işlemi için akla ilk gelen sezgisel yaklaşım tüm dizi elemanlarını aranılan değer ile karşılaştırmaktır. Tüm liste elemanlarını tarama işlemi "Lineer Arama" olarak da adlandırılır.

```
int LineerArama(int *diziAra, int diziBoyut, int aranan)
{
        int i;
        for (i = 0; i < diziBoyut; i++)
        {
                if (aranan == diziAra[i])
                        return i;
        }
        return -1;
}
```

Tam sayı dizisi için arama fonksiyonunda arama işlemi aranan bulununcaya kadar devam eder, bulununca da return indis değeri kullanılarak fonksiyondan çıkılır.

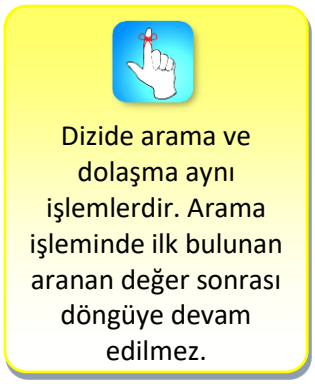
```
int main()
{
        int dizi[10] = { 0,13,5,2,9,10,12,8,15,4};
        int bul;
        int ara;
        ara = 9;
        bul = LineerArama(dizi, 10, ara);
        if (bul != -1)
                cout << "aranan eleman listenin " << bul << ". indisindedir";
        else
                cout << "aranan eleman listede yok";
        return 0;
}
```

Fonksiyonun main içerisinde çağrılması için dizi, boyut ve aranan değer fonksiyona gönderilmiştir. Ekran çıktısı Şekil 9.13'de gösterilmiştir.



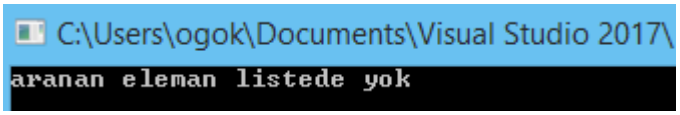
Şekil 9.13. Listede Var Olan Eleman İçin Lineer Arama

Eğer listede olmayan bir eleman dizi içerisinde aranıyorsa for döngüsü biter, fonksiyon return -1 değeri ile son bulur.



Dizide arama ve dolaşma aynı işlemlerdir. Arama işleminde ilk bulunan aranan değer sonrası döngüye devam edilmez.

```
int main()
{
	int dizi[10] = { 0,13,5,2,9,10,12,8,15,4};
	int bul;
	int ara;
	ara = 1;
	bul = LineerArama(dizi, 10, ara);
	if (bul != -1)
		cout << "aranan eleman listenin " << bul << ". indisindedir";
	else
		cout << "aranan eleman listede yok";
	return 0;
}
```



Şekil 9.14 Listede Olmayan Eleman İçin Lineer Arama

Dizilerde Sıralama

Dizilerde sıralama işlemi için çeşitli sıralama algoritmaları mevcuttur. Verinin içeriğine göre, bilgisayarın mimari yapısına göre, verinin boyutuna veya dağılımına göre bu algoritmalar farklılıklar gösterir. Genellikle tekrarlı çalışan tüm sıralama algoritmalarında tüm değerler birbiri ile karşılaştırılır ve bunun için iç içe 2 döngü gerekir.

Sezgisel yaklaşımlardan bir tanesi dizinin en küçük elemanını bularak listenin başına alma fikridir. Bu şekilde geri kalan listede en küçük elemanı en başa alma yöntemi seçmeli sıralama (selection sort) olarak adlandırılır.

Seçmeli sıralama

Seçmeli sıralama algoritması dizi listesindeki en küçük elemanı bularak listenin başındaki elemanla yer değiştirmek ile başlar ve bu adım *eleman sayısı-1* adıma kadar devam eder.

```
int dizi[10] = {13,5,2,9,10,12,8,15,4,7};
```

Yukarıdaki dizide tüm liste tarandığında en küçük eleman 2 olur. 2 ile 13 yer değiştirir.

```
int dizi[10] = {2,    5,13,9,10,12,8,15,4,7};
```

İlk eleman dizide saklanır; fakat en küçük eleman arama işlemi için dizi daraltılır. Artık 0. indisteki değer işlem dışı olur. Geri kalan dizi elemanları içerisinde tekrar en küçük seçilir. 4 değeri daraltılmış listenin başındaki eleman ile yer değiştirir.

Seçmeli sıralama işleminde alt işlem olarak en küçük elemanı bulma işi eleman sayısı-1 kadar yapılmalıdır.

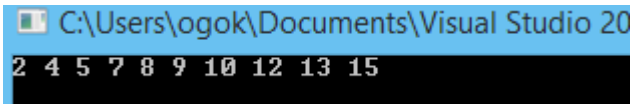
```
int dizi[10] = {2,4,    13,9,10,12,8,15,5,7};
```

Aynı adımlar en küçük son eleman kalıncaya kadar devam eder:

```
int dizi[10] = {2,4,5,    9,10,12,8,15,13,7}; // 3. adım
int dizi[10] = {2,4,5,7,    10,12,8,15,13,9}; // 4. adım
int dizi[10] = {2,4,5,7,8,    12,10,15,13,9}; // 5. adım
int dizi[10] = {2,4,5,7,8,9,    10,15,13,12}; // 6. adım
int dizi[10] = {2,4,5,7,8,9,10,    15,13,12}; // 7. adım
int dizi[10] = {2,4,5,7,8,9,10,12,    13,15}; // 8. adım
int dizi[10] = {2,4,5,7,8,9,10,12,13,15}; // 9. adım
```

```
#include <iostream>
using namespace std;
void SecmeliSirala(int *siraDizi, int boyut)
{
        int i,enKucuk, enKucukYer;

        for (i = 0; i<boyut; i++)
        {
                enKucuk = siraDizi[i];
                enKucukYer =i;

                for (int j = i+1; j<boyut; j++)
                {
                        if (siraDizi[j]<enKucuk)
                        {
                                enKucuk = siraDizi[j];
                                enKucukYer= j;
                        }
                }
                siraDizi[enKucukYer] = siraDizi[i];
                siraDizi[i] = enKucuk;
        }
}
int main()
{
        int dizi[10] = { 13,5,2,9,10,12,8,15,4,7};
        SecmeliSirala(dizi, 10);
         for (i = 0; i < 10; i++)
                cout << dizi[i] << " ";
```

```
        return 0;
}
```



Şekil 9.15 Seçmeli Sıralama Algoritması Ekran Çıktısı

Bireysel Etkinlik

- İkili arama (binary search) algoritmasını araştırınız ve bu algoritmayı bir C++ fonksiyonuna dönüştürmeye çalışınız.
- Kabarcık sıralama (buble sort) algoritmasını araştırınız ve bu algoritmayı bir C++ fonksiyonuna dönüştürmeye çalışınız.

Özet

•Diziler aynı tipte, sonlu ve birbirleriyle ilişkili bir grup veriyi saklamak ve bu veriler üzerinde işlemler yapmak için kullanılırlar. Bu ünite kapsamında dizi kullanmanın programcıya ne gibi bir fayda sağlayabileceğinden bahsedilmiştir ve C++ programlama dilinde dizilerin nasıl kullanılabileceği açıklanmıştır. Bu manada;

- Tek boyutlu dizilerin dil kuralları içerisinde nasıl tanımlandığı ve bellekte sıralı olarak nasıl tutulduğu anlatılmıştır. Dizi tanımlanırken başlangıç değeri atama işlemlerinden bahsedilmiştir ve bellekte gösterimi yapılmıştır.
- Dizi elemanlarına erişim metotları, değer atama ve indis yapısı gösterilmiştir. Dizi elemanlarına değer atama işlemleri için ihtiyaca göre farklı metotlar kullanılabilir. Dizi değerleri ilk baştan atanabilir veya dizi tanımlandıktan sonra kullanıcıdan veya kod içerisinde yönelimli olarak atama yapılabilir. Ünite içerisinde tanımlamadaki bu farklılıkların şematik gösterimi yapılmıştır.
- Dizilerdeki tüm elemanlara ulaşma ve dolaşma işlemleri, dizi içeriğini değiştirme ve sorgulama için döngüler ideal kullanım yöntemidir. Döngü ile tüm dizide dolaşma işlemi ve kullanıcı yönelimli erişim ile değer değişimi örneklendirilmiştir. For döngüsü, indis ve sayaç ile dizide dolaşma uygulaması yapılmıştır.
- Karakter dizilerinin sayısal tipteki dizilere göre tanımlama olarak bir farkı yoktur. Fakat kullanımda kolaylık olması için ek kontroller eklenmiştir. Karakter dizilerinin bellekte tutulma şeklinin sayısal dizilere göre farkı gösterilmiştir. Karakter dizilerinde yazdırma işlemi ve sonlandırma karakteri olan (\0) anlatılmıştır. Sonlandırma karakterinin nasıl bir kolaylık sağladığı örnekler ile anlatılmıştır ve bellek gösterimi yapılmıştır.
- Dizi değişkeninin bir fonksiyona giriş parametresi olarak gönderilmesi normal değişkenlere göre farklılık arz eder. Bu farklılık dizi değişkeninin tipinden kaynaklanır. Dizilerin fonksiyonlara parametre olarak gönderiminin nasıl yapılması gerektiği tanımlanmıştır ve örneklerle anlatılmıştır. Dizi değişkeninin işaretçi olması sebebi ile fonksiyona referans olarak gönderilmesinden ve erişimin nasıl olduğundan bahsedilmiştir.
- Dizilerde arama işlemi ve dolaşma işlemi benzerdir. İndis ve sayaç kullanılarak dizilerde arama işleminden bahsedilmiştir ve baştan sona tarama işlemi olan lineer arama fonksiyonu yazılmıştır.
- Dizilerde sıralama işlemi için tek bir algoritma yoktur. Dizinin içeriğine, boyutuna veya dağılımına göre farklı algoritmalar mevcuttur. Dizilerde sıralama mantığı ve seçmeli sıralamanın algoritması anlatılmıştır. Bu anlatım sonrası seçmeli sıralama algoritması bir fonksiyon şeklinde tanımlanmıştır.

DEĞERLENDİRME SORULARI

1. Dizi aşağıdakilerden hangisinde yanlış tanımlanmıştır?
   a) `int dizi[4] = {4,6,8,10};`
   b) `int dizi = { 4,6,8,10 };`
   c) `int dizi[4] = {};`
   d) `int dizi[4];`
   e) `int dizi[] = {4,6,8,10};`

2. Dizi aşağıdakilerden hangisinde yanlış tanımlanmıştır?
   a) `int dizi[4] = {4,6,8,10};`
   b) `double dizi2[4] = { 1.4, 2.3 };`
   c) `float dizi3[4];`
   d) `bool dizi[4];`
   e) `char dizi1[4] = { a,b,c};`

3.
```
#include <iostream>
using namespace std;
int main()
{
        int dizi[5] = {};
        cout << dizi[3];
        return 0;
}
```
   Yukarıdaki C++ programı çalıştırıldığında ekrana hangi çıktıyı yansıtır?
   a) 0
   b) 1
   c) 2
   d) 3
   e) Rastgele bir sayı

4.
```
#include <iostream>
using namespace std;
int main()
{
        int dizi[4] = {1,2,3,4};
        cout << dizi;
        return 0;
}
```
   Yukarıdaki C++ programı çalıştırıldığında ekrana hangi çıktıyı yansıtır?
   a) 1
   b) 2
   c) 3
   d) 4
   e) Dizinin bellekteki adresi

5.
```
#include <iostream>
using namespace std;
int main()
{
        int dizi[5] = {1,2,3,4,5};
        for (int i = 3; i < 5; i++)
                cout << dizi[i]<<" ";
        return 0;
}
```

Yukarıdaki C++ programı çalıştırıldığında ekrana hangi çıktıyı yansıtır?

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

6.
```
#include <iostream>
using namespace std;
int main()
{
        char dizi[10] = "ornek";
        cout << dizi;
        return 0;
}
```

Yukarıdaki C++ programı çalıştırıldığında ekrana hangi çıktıyı yansıtır?

a) ornek
b) o
c) Dizinin adresini yazar.
d) k
e) 111(Ascii(o))

7.
```
#include <iostream>
using namespace std;
int main()
{
        char dizi[10] = "deneme1234";
        cout << dizi[6];
        return 0;
}
```

Yukarıdaki C++ programı çalıştırıldığında ekrana hangi çıktıyı yansıtır?

a) Ekrana "deneme1234" yazar.
b) Ekrana "deneme" yazar.
c) Ekrana "1234" yazar.
d) Ekrana "1" yazar.
e) Hata verir, çalıştırılamaz.

8.
```
#include <iostream>
using namespace std;
int Fonksiyon(int *A, int n)
{
        int dondur=0;
        for (int i = 0; i < n; i++)
        {
                if (A[i] % 2 == 0)
                        dondur++;
        }
        return dondur;
}
int main()
{
        int dizi[5] = { 12,5,7,4,9 };
        cout << Fonksiyon(dizi, 5);
        return 0;
}
```

Yukarıdaki kod çalıştırılmak istendiğinde sonuç ne olur?

a) Ekrana "3" yazar.
b) Ekrana "2" yazar.
c) Ekrana "12 4" yazar.
d) Ekrana "5 7 9" yazar.
e) Ekrana "12 5 7 4 9" yazar.

9.
```
void Fonksiyon(int *A, int n)
{
        for (int i = 0; i < n; i++)
        {
                cout << A[i]<<" ";
        }
        cout << endl;
}
```
Yukarıdaki fonksiyon için aşağıdaki çağrımlardan hangisi hata verir?

a) Fonksiyon(dizi, 5);
b) Fonksiyon(dizi, 3);
c) Fonksiyon(dizi+2, 2);
d) Fonksiyon(dizi[1], 3);
e) Fonksiyon(&dizi[3], 1);

10.
```
#include <iostream>
using namespace std;
void Fonksiyon(int *A, int n)
{
for (int i = 0; i < n; i++)
        for (int j = 0; j < n-1; j++)
        {
                if (A[j] > A[j + 1])
                {
                        int temp = A[j];
                        A[j] = A[j + 1];
                        A[j + 1] = temp;
                }
        }
}
int main()
{
        int dizi[5] = { 12,5,7,4,9 };
                Fonksiyon(dizi, 5);
        for (int i = 0; i < 5; i++)
{
                cout << dizi[i] << " ";
}
        return 0;
}
```
Yukarıdaki C++ programı çalıştırıldığında ekrana hangi çıktıyı yansıtır?

a) 12 5 7 4 9
b) 12 9 7 5 4
c) 4 5 7 9 12
d) 12
e) 9

Cevap Anahtarı

1.b, 2.e, 3.a, 4.e, 5.c, 6.a, 7.e, 8.b, 9.d

YARARLANILAN KAYNAKLAR

[1] https://msdn.microsoft.com/en-us/library/3bstk3k5.aspx

İKİ BOYUTLU DİZİLER

İÇİNDEKİLER

- İki Boyutlu Diziler
  - İki Boyutlu Dizi Ataması
  - İki Boyutlu Dizi Elemanlarına Erişim
  - İki Boyutlu Dizi Elemanlarıyla Örnekler
- Çok Boyutlu Diziler
  - Üç Boyutlu Dizi Örneği

HEDEFLER

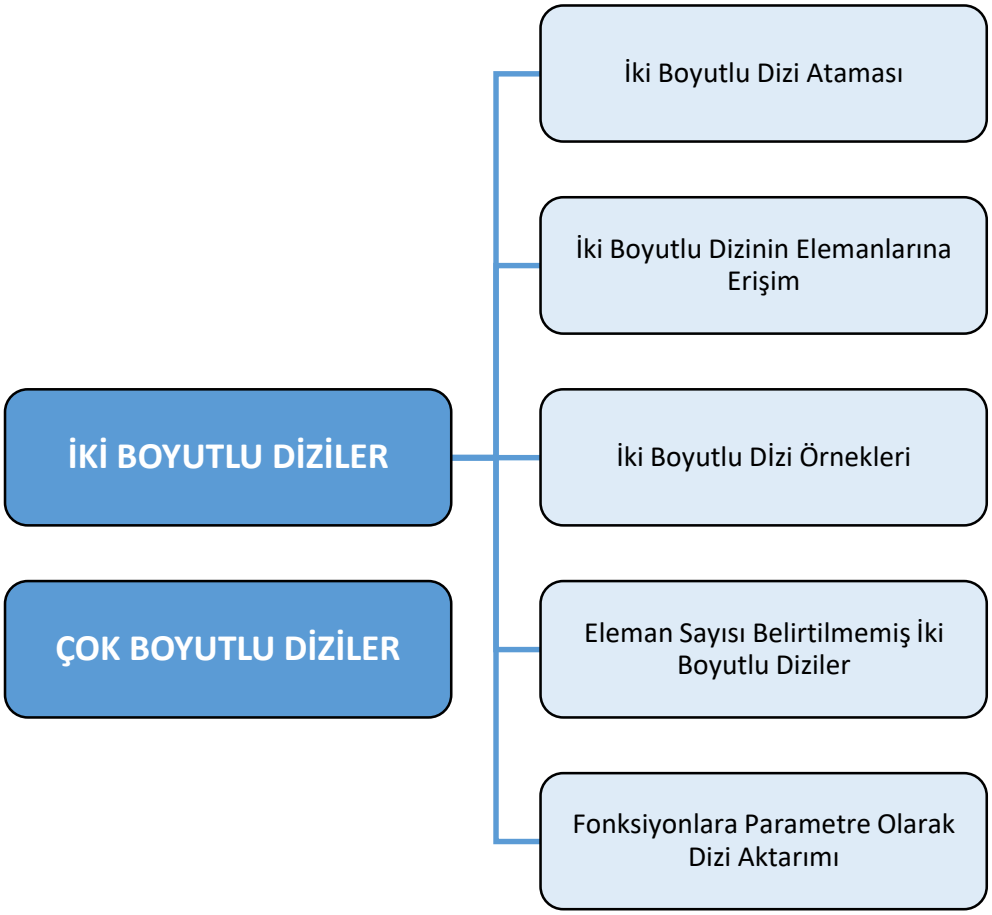
- Bu üniteyi çalıştıktan sonra;
  - İki boyutlu diziyi tanımlayabilecek,
  - İki boyutlu dizilerin elemanlarına atama yapabilecek,
  - İki boyutlu dizi elemanlarını okuyabilecek,
  - İki boyutlu diziler üzerinde matematiksel işlemler gerçekleştirebilecek,
  - Çok boyutlu dizi tanımı yapabileceksiniz.

Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

Dr. Öğr. Üyesi
Serdar AYDIN

ÜNİTE
10


İKİ BOYUTLU DİZİLER
ÇOK BOYUTLU DİZİLER
İki Boyutlu Dizi Ataması
İki Boyutlu Dizinin Elemanlarına Erişim
İki Boyutlu Dizi Örnekleri
Eleman Sayısı Belirtilmemiş İki Boyutlu Diziler
Fonksiyonlara Parametre Olarak Dizi Aktarımı

GİRİŞ

Bu ünitede 9. Ünite'de detayları verilen tek boyutlu dizi uygulamaları genişletilecektir ve iki boyutlu diziler ele alınacaktır. Diziler, belirli sayıda ve aynı türden bir grup ilişkili veriyi bir arada tutmaya imkân sağlayan programlama yapılarıdır. Dizilerde tutulan veri her ne olursa olsun şayet tablo gibi iki boyutluysa veya daha fazla boyuta sahipse bu verinin tek boyutlu dizilerle modellenmesi ve işlenmesi güçtür. Örneğin; bir öğrenci için vize, final ve bütünleme notları tek boyutlu bir dizi içerisinde saklanabilir. Ancak bütün bir sınıf için, her bir öğrencinin vize, final ve bütünleme notlarını saklayan dizileri bir araya getirerek çok boyutlu bir sınıf not listesi dizisi oluşturmak daha mantıklıdır.

Tek boyutlu diziler aşağıdaki gibi tanımlanırlar ve bu tip dizilere vektör denir.

```
int x[2];
```

Dizilerin tek boyutlu olması gerekmez, istenilen boyutta dizi tanımlanabilir. Dizi aşağıdaki gibi tanımlanıyorsa iki boyutludur ve bu tip dizilere de matris adı verilir.

```
int x[2][3];
```

Değişkenleri isimlendirirken dikkat edilmesi gereken kurallar iki boyutlu dizi değişkenleri için de geçerlidir.

İki boyutlu diziler aritmetik işlemlerde ve mühendislik uygulamalarında sıklıkla kullanılan işlemler arasında yer alırlar. İlk paragrafta da belirtildiği üzere iki boyutlu bir dizi bir tablo gibi düşünülebilir. İki boyutlu dizilerin ilk boyutuna satır, ikinci boyutuna da sütun denir. Çok boyutlu diziler ise dizilerin dizileri olarak tanımlanabilir. Çok boyutlu dizilerin en basit hali iki boyutlu dizilerdir. Dizinin tanımlanması aşamasında kullanılan ilk değer satırı ikinci değer de sütunu temsil eder.

Bu ünitede yer alan iki boyutlu dizi örnekleri C++ programlama diliyle hazırlanmıştır, derlenmiştir ve çalıştırılmıştır.

İKİ BOYUTLU DİZİLER

İki boyutlu bir dizi, özünde bir boyutlu dizilerin listesi olarak tanımlanabilir. Birinci boyutunda x eleman, ikinci boyutunda y eleman barındıran iki boyutlu bir dizi;

```
veri_tipi diziAdi[x][y];
```

şeklinde tanımlanır. Bu ifade de "veri_tipi" herhangi bir C++ veri tipi ve diziAdi da geçerli bir C++ tanımlayıcısıdır.

İki boyutlu bir dizi x adet satır ve y adet de sütun ihtiva eden bir tablo gibi düşünülebilir. x ve y sıfırdan büyük birer tam sayı olmalıdır. 3 satırlı ve 4 sütunlu iki boyutlu bir a dizisi aşağıdaki gibidir:

	Sütun 0	Sütun 1	Sütun 2	Sütun 3
Satır 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Satır 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Satır 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Bu tabloda da görüldüğü üzere, dizinin her bir elemanı a[i][j] şeklinde gösterilir. Bu ifadede a dizinin ismi, i ve j ise a dizisinin her bir elemanını eşsiz olarak belirten indislerdir.

Burada yer alan satır ve sütun bilgileri gösterimden ibarettir. Matrisin ilk indisi satır indisi, ikinci indisi ise sütün indisi olarak kabul edilmiştir. Bu matris bilgisayarın belleğinde (RAM, Memory) aslında tek boyutlu bir dizi olarak tutulur. Dolayısıyla bizim satır ve sütün tutan indislerimiz bir kabule dayanır.

	Sütun 0	Sütun 1	Sütun 2
Satır 0	a[0][0]	a[1][0]	a[2][0]
Satır 1	a[0][1]	a[1][1]	a[2][1]
Satır 2	a[0][2]	a[1][2]	a[2][2]
Satır 3	a[0][3]	a[1][3]	a[2][3]

İKİ BOYUTLU DİZİ ATAMASI

Nesne tabanlı programlamayı anlamak için yapısal programlamayı iyi bilmek gerekir.

İki boyutlu diziler tanımlanırken her bir sütunun elemanları "," ile ve her bir satırın elemanları da süslü parantezlerle birbirinden ayrılır.

```
veri_tipi diziAdi[boyut1][boyut2] = {{boyut1 elemanlari},
                                    {boyut2 elemanlari}};
```

Üç satır ve dört sütun içeren bir dizinin ataması aşağıdaki gibi yapılabilir. boyutN elemanları virgül ile ayrılmış dizinin elemanları veri_tipi ile uyumlu veriler olmalıdır.

```
int a[3][4] = {
      { 0, 1, 2, 3 } , // 0 indisli satırın atanması
      { 4, 5, 6, 7 } , // 1 indisli satırın atanması
      { 8, 9, 10, 11 } // 2 indisli satırın atanması
};
```

Her bir satırı birbirinden ayıran süslü parantezler kullanılmasa bile yine de atama işlemi gerçekleştirilebilir. Yukarıdaki atama işleminin süslü parantez kullanılmayan aynı işlevli formu aşağıdaki şekildedir:

```
int a[3][4] = { 0,1,2,3,
                4,5,6,7,
                8,9,10,11 };
```

İki boyutlu dizilerin elemanlarına satır ve sütun indisleri kullanılarak erişilebilir.

İKİ BOYUTLU DİZİNİN ELEMANLARINA ERİŞİM

Bilindiği üzere aslında dizi (array) kavramının varlık sebebi birden fazla değişkeni hafızada bir arada tutmak ve kolayca ulaşmaktır. İki boyutlu dizilerin elemanlarına satır ve sütun indisleri kullanılarak erişilebilir. Örneğin:

```
int deger = a[2][3];
```

Yukarıdaki deyim ile a dizisinin üçüncü satırındaki ve dördüncü sütunundaki dizi elemanının değeri okunabilir.

Çok boyutlu dizilerde dizi elemanlarına erişim genellikle iç içe for döngüleri ile sağlanır.

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

int main()
{

	int a[5][2] = { { 0,0 },{ 1,2 },{ 2,4 },{ 3,6 },{ 4,8
} };
	// dizinin elemanlarinin ekrana yazdirilmasi
	for (int i = 0; i < 5; i++) {
		for (int j = 0; j < 2; j++)
		{
			cout << "a[" << i << "][" << j << "]: ";
			cout << a[i][j] << endl;
		}
	}
	return 0;
}
```

Yukarıdaki kod derlenip çalıştırıldığında çıktısı aşağıdaki gibi olacaktır:

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

Aşağıdaki C++ programı ile 5 satırlı ve 5 sütunlu bir dizi tanımlanmış ve 0, 1, 2, 3 ve 4 değerlerinden oluşan satır değerleri her seferinde 1'er arttırılarak dizi elemanlarına değer verilmiştir. Program incelendiğinde dizi elemanlarına atama yapılan ve dizi elemanlarının okunduğu her iki kısım için de kodlama esnasında döngülerden yararlanıldığı anlaşılır.

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

int main()
{
   //a dizisi tanımlanması
       int a[5][5];
   //birinci boyut üzerinde for döngüsü
       for (int i = 0; i<5; i++) {
      //ikinci boyut üzerinde for döngüsü
              for (int j = 0; j<5; j++) {
              //dizi elemanına değer atama
                     a[i][j] = i + j;
              }
       }
   //birinci boyut üzerinde for döngüsü
       for (int i = 0; i<5; i++) {
      //ikinci boyut üzerinde for döngüsü
              for (int j = 0; j<5; j++) {
              //dizi elemanının ekrana yazdırılması
                     cout << a[i][j];
              }
       //her bir satır sonrası satır sonu eklenmesi
              cout << endl;
       }
       return 0;
}
```

Yukarıdaki kodun ekran çıktısı;

```
01234
12345
23456
34567
45678
```

şeklinde olacaktır.

İKİ BOYUTLU DİZİ ÖRNEKLERİ

Çoklu verilerin girilmesini sağlayan matrisler üzerinde yapabileceğimiz;

- İki boyutlu dizi elemanlarını okuma,
- İki boyutlu dizi elemanlarını okuma ve atama,
- İki boyutlu dizilerin elemanlarını toplama (matris toplamı)
- İki boyutlu dizilerin çarpılması (matris çarpımı)

işlemlerine dair örnekleri bu başlık altında inceleyeceğiz.

ÖRNEK 1: İKİ BOYUTLU DİZİLERDE OKUMA İŞLEMİ

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

int main()
{
	// test dizisi tanımlaması
	int dizi[3][2] =
	{
		{ 2, -5 },
		{ 4, 0 },
		{ 9, 1 }
	};

	// Birinci boyut üzerinde for döngüsü
	for (int i = 0; i < 3; ++i)
	{
		// Birinci boyut üzerinde for döngüsü
		for (int j = 0; j < 2; ++j)
		{
			// Değerlerin ekrana yazdırılması
			cout << "test[" << i << "]["
				<< j << "] = " << test[i][j] <<
endl;
		}
	}

	return 0;
```

Yukarıdaki örneğin ekran çıktısı aşağıdaki gibidir:

```
test[0][0] = 2
test[0][1] = -5
test[1][0] = 4
test[1][1] = 0
test[2][0] = 9
test[2][1] = 1
```

ÖRNEK 2: İKİ BOYUTLU DİZİLERDE ATAMA VE OKUMA İŞLEMLERİ

Bu örneğimizde sicaklik isimli, birinci boyutunda 2 eleman ve ikinci boyutunda ise 7 eleman barındıran iki boyutlu bir dizi tanımlanmıştır. Sonrasında kullanıcının her şehirde her bir gün için yaşanan sıcaklık değerlerini girmesi sağlanmış, ardından girilen değerler tekrar kullanıcıya gösterilmiştir.

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

const int SEHIR = 2;
const int HAFTAGUN = 7;

int main()
{
	int sicaklik[SEHIR][HAFTAGUN];

	cout << "Her bir sehirde haftanin günlerine "
		<< "ait sicaklik degerlerini giriniz. \n";

	// sicaklik dizisine degerler giriliyor
	for (int i = 0; i < SEHIR; ++i)
	{
		for (int j = 0; j < HAFTAGUN; ++j)
		{
			cout << "Sehir " << i + 1 << ", Gun " << j + 1
<< " : ";
			cin >> sicaklik[i][j];
		}
	}
	cout << "\n\nDegerler gosteriliyor:\n";

	// sicaklik dizisinden degerler okunuyor.
	for (int i = 0; i < SEHIR; ++i)
	{
		for (int j = 0; j < HAFTAGUN; ++j)
		{
			cout << "Sehir " << i + 1 << ", "
				<< "Gun " << j + 1 << " = " <<
sicaklik[i][j] << endl;
		}
	}
```

Bu programa ait ekran çıktısı aşağıdaki gibidir:

```
Her bir sehirde de haftanin gunlerinde ait sicaklik degerlerini giriniz giriniz .
Sehir 1, Gun 1 : 24
Sehir 1, Gun 2 : 25
Sehir 1, Gun 3 : 26
Sehir 1, Gun 4 : 24
Sehir 1, Gun 5 : 23
Sehir 1, Gun 6 : 25
Sehir 1, Gun 7 : 22
Sehir 2, Gun 1 : 24
Sehir 2, Gun 2 : 26
Sehir 2, Gun 3 : 28
Sehir 2, Gun 4 : 27
Sehir 2, Gun 5 : 24
Sehir 2, Gun 6 : 23
Sehir 2, Gun 7 : 22


Degerler gosteriliyor:
Sehir 1, Gun 1 = 24
Sehir 1, Gun 2 = 25
Sehir 1, Gun 3 = 26
Sehir 1, Gun 4 = 24
Sehir 1, Gun 5 = 23
Sehir 1, Gun 6 = 25
Sehir 1, Gun 7 = 22
Sehir 2, Gun 1 = 24
Sehir 2, Gun 2 = 26
Sehir 2, Gun 3 = 28
Sehir 2, Gun 4 = 27
Sehir 2, Gun 5 = 24
Sehir 2, Gun 6 = 23
Sehir 2, Gun 7 = 22
```

ÖRNEK 3: İKİ BOYUTLU DİZİLERDE TOPLAMA (MATRİS TOPLAMA)

Klavyeden girilen N*N boyutundaki a ve b matrislerinin elemanlarını toplam matrisinde toplayan C++ programını yazalım.

Çözümde de görüldüğü gibi öncelikle r, c, i, j tam sayı değişkenleri ve her iki boyutunda da 100'er eleman bulunan a, b ve toplam isimli iki boyutlu diziler tanımlanmıştır. Kullanıcının satır sayılarını (r değişkeni) ve sütun sayılarını (c değişkeni) girmesi sağlanmıştır. Sonrasında sırasıyla a ve b dizilerinin eleman değerlerinin kullanıcı tarafından girilmesi sağlanmıştır. Her iki dizinin eleman girişlerinin tamamlanmasının ardından matris toplama işlemi gerçekleştirilerek, sonuç matrisi ekrana yazdırılmaktadır.

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

int main()
{
	int r, c, a[100][100], b[100][100], toplam[100][100], i, j;

	cout << "Satir Sayisini Giriniz (1 ile 100 arasinda): ";
	cin >> r;

	cout << "Sutun Sayisini Giriniz (1 ile 100 arasinda): ";
	cin >> c;

	cout << endl << "1. Matrisin elemanlarini giriniz: " << endl;

	// 1. Dizi doldurulуyor
	for (i = 0; i < r; ++i)
		for (j = 0; j < c; ++j)
		{
			cout << "a dizisinin " << i + 1 << j + 1 << " elemanina
deger girin: ";
			cin >> a[i][j];
		}

	// 2. Dizi doldurulуyor
	cout << endl << "2. Matrisin elemanlarini giriniz: " << endl;
	for (i = 0; i < r; ++i)
		for (j = 0; j < c; ++j)
		{
			cout << "b dizisinin " << i + 1 << j + 1 << " :elemanina
deger girin  ";
			cin >> b[i][j];
		}

	// Matrisler Toplaniyor
	for (i = 0; i < r; ++i)
		for (j = 0; j < c; ++j)
			toplam[i][j] = a[i][j] + b[i][j];

	// Sonuc Matrisi ekrana yaziliyor.
	cout << endl << "Toplam Sonucu: " << endl;
	for (i = 0; i < r; ++i)
		for (j = 0; j < c; ++j)
		{
			cout << toplam[i][j] << "  ";
			if (j == c - 1)
				cout << endl;
		}
```

Bu programın ekran çıktısı aşağıdaki gibi olacaktır:

```
Satir Sayisini Giriniz (1 ile 100 arasinda): 2
Sutun Sayisini Giriniz (1 ile 100 arasinda): 2

1. Matrisin elemanlarini giriniz:
a dizisinin 11 elemanina deger girin: 1
a dizisinin 12 elemanina deger girin: 2
a dizisinin 21 elemanina deger girin: 3
a dizisinin 22 elemanina deger girin: 4

2. Matrisin elemanlarini giriniz:
b dizisinin 11 :elemanina deger girin  2
b dizisinin 12 :elemanina deger girin  3
b dizisinin 21 :elemanina deger girin  1
b dizisinin 22 :elemanina deger girin  2

Toplam Sonucu:
3  5
4  6
```

Örnek 4-1: İKİ BOYUTLU DİZİLERİN ÇARPIMI 1 (KARE MATRİS ÇARPIMI)

Öncelikle 3x3 boyutunda iki adet iki boyutlu kare matrisi ekrana yazdıran, ardından bu dizilerin matris çarpımını yapan C++ programını oluşturalım.

```
Birici Dizi:
 3 2 7
 0 8 1
 9 1 2
İkinci Dizi:
 2 3 1
 7 7 9
 8 6 5
Sonuç Dizisi:
 76 65 56
 64 62 77
 41 46 28
```

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

#define sayi 3

int main()
{
	int birinciDizi[sayi][sayi] = { 3, 2, 7, 0, 8, 1, 9, 1, 2 };
	int ikinciDizi[sayi][sayi] = { 2, 3, 1, 7, 7, 9, 8, 6, 5 };
	int sonuc[sayi][sayi], toplam;
	//Birinci dizi ekrana yazdırılıyor
	cout << "Birinci Dizi: " << endl;
	for (int i = 0; i<sayi; i++)
	{
		for (int j = 0; j<sayi; j++)
			cout << " " << birinciDizi[i][j];
		cout << "\n";
	}
	//İkinci dizi ekrana yazdırılıyor
	cout << "Ikinci Dizi: " << endl;
	for (int i = 0; i<sayi; i++)
	{
		for (int j = 0; j<sayi; j++)
			cout << " " << ikinciDizi[i][j];
		cout << "\n";
	}
	//İki dizi çarpılarak sonuc dizisine yazılıyor
	//ardından ekrana yazdırılıyor
	cout << "Sonuç Dizisi: " << endl;
	for (int i = 0; i<sayi; i++)
	{
		for (int j = 0; j<sayi; j++)
		{
			toplam = 0;
			for (int k = 0; k<sayi; k++)
			{
				toplam += birinciDizi[i][k] *
ikinciDizi[k][j];
			}

			sonuc[i][j] = toplam;
			cout << " " << sonuc[i][j];
		}
		cout << "\n";
	}

	return 0;
```

Yukarıdaki ekran çıktısını üretecek olan C++ uygulaması aşağıdaki gibidir:

Örnek 4-2: İKİ BOYUTLU DİZİLERİN ÇARPIMI 2 (KARE OLMAYAN MATRİSLERİN ÇARPIMI)

Matrislerin çarpımında, çarpılacak olan iki matristen birincisinin sütun sayısı ile ikincisinin satır sayısı birbirine eşit olmak zorundadır. Aksi takdirde çarpma işlemi gerçekleştirilemez.

Çarpılacak olan iki matristen birincisinin sütun sayısı ile ikincisinin satır sayısı birbirine eşit olmak zorundadır.

Aşağıdaki şekilde tanımlanmış aMatris'i ile bMatris'inin çarpımı;

$$aMatris = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

$$bMatris = \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

$$sonuc = \begin{bmatrix} 1*7+4*10 & 1*8+4*11 & 1*9+4*12 \\ 2*7+5*10 & 2*8+5*11 & 2*9+5*12 \\ 3*7+6*10 & 3*8+6*11 & 3*9+6*12 \end{bmatrix} = \begin{bmatrix} 47 & 52 & 57 \\ 64 & 71 & 78 \\ 81 & 90 & 99 \end{bmatrix}$$

şeklinde hesaplanır. Şimdi bu çarpımı bir C++ uygulaması olarak gerçekleyelim. Daha sonra her iki matrisi tanımlayalım ve her bir matris için önce her satır ve her sütun için iç içe 2 döngü ve son olarak da her bir matrisin elemanlarını birbiriyle çarpabilmek için son bir iç döngü kuralım.

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

int main(){
   int aMatris       [3][2] = { { 1, 4 },{ 2, 5 },{ 3, 6 } };
       int bMatris[2][3] = { { 7, 8, 9 },{ 10, 11, 12 } };
       int sonuc[3][3] = { { 0, 0, 0 },{ 0, 0, 0 },{ 0, 0, 0 } };
   //Satır döngüsü
       for (int satir = 0; satir < 3; satir++) {
      //Sütun döngüsü
              for (int sutun = 0; sutun < 3; sutun++) {
     //her iki matrisin elemanlarının birbiriyle çarpımı için iç
     döngü
                     for (int ic = 0; ic < 2; ic++) {
           sonuc[satir][sutun] += aMatris[satir][ic] *
           bMatris[ic][sutun];
                     }
                     cout << sonuc[satir][sutun] << "  ";
              }
              cout << "\n";
       }
       return 0;
}
```

Yukarıdaki programın çıktısı;

```
47  52  57
64  71  78
81  90  99
```

şeklinde olacaktır.

ELEMAN SAYISI BELİRTİLMEMİŞ İKİ BOYUTLU DİZİLER

C++ ile iki boyutlu dizilerin her bir boyutunun barındıracağı eleman sayısı dinamik olarak tanımlanabilir.

```
          char hata_1[] = "Sifir ile bolunme\n";
          char hata_2[] = "Dosya Sonu\n";
          char hata_3[] = "Erisim Reddedildi\n";
```

Bu örneklerde C++, verileri saklayabilmek için gerekli eleman sayısını içeren dizileri otomatik olarak oluşturacaktır. Çok boyutlu dizilerde en soldaki boyut dışındaki boyutların eleman sayıları tanımlanmalıdır.

FONKSİYONLARA İKİ BOYUTLU DİZİLERİN AKTARILMASI

C++ ile iki boyutlu dizilerin her bir boyutunun barındıracağı eleman sayısı dinamik olarak tanımlanabilir.

```
char hatalar[][20] = {
      "Sifir ile bolunme\n",
      "Dosya Sonu\n",
      "Erisim Reddedildi\n"
 };
```

```
#include <iostream>
using namespace std;
void EkranaYazdir(int[][3], int);
int main(void)
{
 // Bir matris oluşturup rastgele değerler atayalım
 int ikiBoyutluDizi[2][3] = {
       { 1, 3, 5, },
       { 3, 8, 9, }
 };

 // bu iki boyutlu diziyi başka bir fonksiyona parametre
 // olarak gönderiyoruz
 EkranaYazdir(ikiBoyutluDizi, 2);

 return 0;
}
void EkranaYazdir(int gelenDizi[][3], int satirSayisi)
{
 int i, j;
 for (i = 0; i < satirSayisi; i++) {
       for (j = 0; j < 4; j++) {
             cout << gelenDizi[i][j] << " ";
       }
       cout << endl;
 }
}
```

İki boyutlu dizi gönderilirken dizinin iki boyutlu olduğu belirtilmeli ve ikinci boyutun eleman sayısı mutlaka yazılmalıdır.

Bazı durumlarda dizilerin fonksiyonlara parametre olarak gönderilmesine ihtiyaç duyulabilir. İki boyutlu dizilerin fonksiyonlara parametre olarak gönderilmesi tek boyutlu dizilerin parametre olarak gönderilmesiyle hemen hemen aynıdır. İki boyutlu dizi gönderilirken dizinin iki boyutlu olduğu belirtilmeli ve ikinci boyutun eleman sayısı mutlaka yazılmalıdır.

İkiden fazla boyuta sahip diziler fonksiyonlara gönderilirken fonksiyon tanımında ilk boyut dışındaki boyutlar için eleman sayısı belirtilmelidir.

Çok boyutlu diziler, iki boyutlu diziler ile sınırlı değildir. Boyutu gerektiği kadar fazla olan dizi üretilebilir.

ÇOK BOYUTLU DİZİLER

Çok boyutlu diziler, iki boyutlu diziler ile sınırlı değildir. Boyutu gerektiği kadar fazla olan dizi üretilebilir. Dizi boyutu arttıkça kullanılan bellek logaritmik olarak artacağı için dikkatli olunması gerekmektedir. Çok boyutlu dizi tanımlaması;

```
veri_tipi diziAdi [boyut1][boyut2][boyut3]…[boyutN];
```

```
char yuzYil[100][365][24][60][60];
```

şeklinde yapılabilir. Örneğin, yukarıdaki dizi tanimi ile bir yüzyılın her bir saniyesini ihtiva eden beş boyutlu bir dizi üretilebilir. Bu deyim sayesinde 3 milyardan fazla eleman ihtiva eden bir dizi üretilmiş olur. Her bir char'ın bellekte 1 byte yer kapladığı göz önünde bulundurulduğunda bu dizi bellekte 3 Gigabyte'dan fazla yer kaplayacaktır.

```
        int a[3][5];
int a[15];
```

Son olarak çok boyutlu diziler programcılar için görsel basitleştirme sağlar.

Yukarıdaki örnekte her iki deyim de aynı sayıda dizi elemanı ihtiva etmektedir. Aşağıdaki her iki örneğin sonucunda da ikiBoyutluDizi aynı değerlere sahip olacaktır:

```
#include <iostream>
#define sutun 5
#define satir 3

int ikiBoyutluDizi[satir][sutun];
int n, m;

int main()
{
	// birinci boyut için for döngüsü
	for (n = 0; n<satir; n++)
   {
		// ikinci boyut için for döngüsü
		for (m = 0; m<sutun; m++)
		{
			// dizi elemanına değer atanması
			ikiBoyutluDizi [n][m] = (n + 1)*(m + 1);
		}
   }
	return 0;
}
```

```
int n, m;

int main()
{
	// birinci boyut için for döngüsü
   for (n = 0; n<satir; n++)
   {
			// ikinci boyut için for döngüsü
		for (m = 0; m<sutun; m++)
		{
				// dizi elemanın değerinin okunması
				ikiBoyutluDizi[n*sutun + m] = (n + 1)*(m
+ 1);
			}
	}
	return 0;
```

ÜÇ BOYUTLU DİZİ ÖRNEĞİ

Çok boyutlu dizilere örnek olarak üç boyutlu bir dizi oluşturalım. Öncelikle birinci boyutunda 2, ikinci boyutunda 3, üçüncü boyutunda ise 2 eleman barındıran test isimli tam sayı tipinde üç boyutlu bir dizi tanımlayalım. Bu dizinin eleman sayısı 2*3*2 = 12 adet olacaktır. Kullanıcının bu 12 değeri girmesi için iç içe 3 for döngüsü oluşturarak döngülerin her bir iterasyonu için test dizisinin elemanlarının doldurulmasını sağlayalım. Değerlerin girilmesinin ardından tekrar iç içe 3 for döngüsü oluşturarak dizinin her bir boyutu üzerinden dizinin elemanlarını okuyalım.

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

int main()
{
	// (2x3x2) 12 eleman iceren dizi tanimlaniyor
	int test[2][3][2];

	cout << "12 adet deger giriniz: \n";

	// birinci boyut için for döngüsü
	for (int i = 0; i < 2; ++i)
	{
		// ikinci boyut için for döngüsü
		for (int j = 0; j < 3; ++j)
		{
			// üçüncü boyut için for döngüsü
			for (int k = 0; k < 2; ++k)
			{
				// dizi elemanına değer atanması
				cin >> test[i][j][k];
			}
		}
	}

	cout << "\nDegerler Gosteriliyor:" << endl;

	// birinci boyut için for döngüsü
   for (int i = 0; i < 2; ++i)
	{
		// ikinci boyut için for döngüsü
		for (int j = 0; j < 3; ++j)
		{
			// üçüncü boyut için for döngüsü
			for (int k = 0; k < 2; ++k)
			{
				// Elemanlarin indis numaralari ile
ekrana yazdirilmasi
				cout << "test[" << i << "][" << j
					<< "][" << k << "] = " <<
test[i][j][k] << endl;
			}
		}
	}
```

Yukarıdaki programın çıktısı aşağıdaki şekildeki gibi olacaktır:

```
12 adet deger giriniz:
1 2 3 4 5 6 7 8 9 10 11 12

Degerler Gosteriliyor:
test[0][0][0] = 1
test[0][0][1] = 2
test[0][1][0] = 3
test[0][1][1] = 4
test[0][2][0] = 5
test[0][2][1] = 6
test[1][0][0] = 7
test[1][0][1] = 8
test[1][1][0] = 9
test[1][1][1] = 10
test[1][2][0] = 11
test[1][2][1] = 12
```

Bireysel Etkinlik

- İki boyutlu bir matrisin satırlarının çarpımı ile sütunlarının toplamının farkını bulan C++ programını yazınız.

Özet

•Diziler, belirli sayıda ve aynı türden bir grup ilişkili veriyi bir arada tutmamıza imkân sağlayan programlama yapılarıdır.
•Dizilerin tek boyutlu olması gerekmez, istenilen boyutta dizi tanımlanabilir.
•Dizi kavramının varlık sebebi birden fazla değişkeni hafızada bir arada tutmak ve böylece kolayca ulaşmaktır.
•Dizilerde tutulan veri her ne olursa olsun şayet tablo gibi iki boyutluysa veya daha fazla boyuta sahipse bu verinin tek boyutlu dizilerle modellenmesi ve işlenmesi zordur.
•İki boyutlu bir dizi, özünde bir boyutlu dizilerin listesi olarak tanımlanabilir.
•İki boyutlu bir dizi x adet satır ve y adet de sütun ihtiva eden bir tablo gibi düşünülebilir. x ve y sıfırdan büyük birer tam sayı olmalıdır.
•İki boyutlu dizilere atama yapılırken her bir sütunun elemanları “,” ile ve her bir satırın elemanları da süslü parantezlerle birbirinden ayrılır.
•Her bir satırı birbirinden ayıran süslü parantezler kullanılmaksızın sadece virgüller ile veriler birbirinden ayrılarak da atama işleme gerçekleştirilebilir.
•İki boyutlu dizilerin elemanlarına satır ve sütun indisleri kullanılarak erişilebilir. Örneğin:

```
int deger = a[2][3];
```

•İki Boyutlu Dizilerin (Matris) çarpımında, çarpılacak olan iki matristen birincisinin sütun sayısı ile ikincisinin satır sayısı birbirine eşit olmak zorundadır. Aksi takdirde çarpma işlemi gerçekleştirilemez.
•C++ ile iki boyutlu dizilerin her bir boyutunun barındıracağı eleman sayısı dinamik olarak tanımlanabilir.
•İki boyutlu dizilerin fonksiyonlara parametre olarak gönderilmesi tek boyutlu dizilerin parametre olarak gönderilmesiyle hemen hemen aynıdır.
•İki boyutlu bir dizi fonksiyonlara parametre olarak gönderilirken dizinin iki boyutlu olduğu belirtilmeli ve ikinci boyutun eleman sayısı mutlaka yazılmalıdır.
•Çok boyutlu diziler, iki boyutlu diziler ile sınırlı değildir. Boyutu gerektiği kadar fazla olan dizi üretilebilir. Dizi boyutu arttıkça kullanılan bellek logaritmik olarak artacağı için dikkatli olunması gerekmektedir.
•Çok boyutlu dizilerde dizi elemanlarına erişim genellikle iç içe for döngüleri ile sağlanır.
•Çok boyutlu diziler fonksiyonlara gönderilirken fonksiyon tanımında ilk boyut dışındaki boyutlar için eleman sayısı belirtilmelidir.

DEĞERLENDİRME SORULARI

1. İki boyutlu bir dizinin toplam eleman sayısı nasıl hesaplanır?
   a) Birinci boyut eleman sayısı ile ikinci boyut eleman sayısı toplanır.
   b) Birinci boyut eleman sayısı ile ikinci boyut eleman sayısı çarpılır.
   c) Birinci boyut eleman sayısı iki ile çarpılır.
   d) İkinci boyut eleman sayısı iki ile çarpılır.
   e) Birinci boyut eleman sayısının karesi ile ikinci boyut eleman sayısının karesi toplanır.

2. int a[3][2] değişkeni ile aynı dizi değişkenini ifade eden dizi aşağıdakilerden hangisidir?
   a) int a[1][2][2];
   b) int a[2][1][2];
   c) int a[6];
   d) int a[15];
   e) int a[24];

3.
```
for (int n = 0; n<i; n++)
 {
            for (int m = 0; m<j; m++)
            {
                   a[n][m] = n+1;
            }
 }
```
   Yukarıdaki kod parçacığının yaptığı işlemi aşağıdakilerden hangisi doğru şekilde anlatmaktadır?
   a) a dizisinin i * j adet elemanı silinir.
   b) a dizisinin j adet satırındaki i adet elemana birinci boyut indisinin bir fazlası yazılır.
   c) a dizisinin i adet satırındaki j adet elemana birinci boyut indisinin bir fazlası yazılır.
   d) a dizisinin j adet satırındaki i adet elemana ikinci boyut indisinin bir fazlası yazılır.
   e) a dizisinin i adet satırındaki j adet elemana ikinci boyut indisinin bir fazlası yazılır.

4. cin >> test[i][j][k]; kod parçasının yaptığı işlemi hangisi açıklar?
   a) test dizisinin i j k indisleri ile belirtilmiş elemanına cin metnini yazar.
   b) cin değişkenine test[i][j][k] ifadesini yazar.
   c) Kullanıcının girdiği değerin test[i][j][k] elemanına yazılmasını sağlar.
   d) test[i][j][k] elemanının değerini kullanıcıya gösterir.
   e) test[i][j][k] elemanını siler.

5. İki boyutlu dizilerin fonksiyonlara parametre olarak gönderilmesi ile ilgili olarak aşağıdakilerden hangisi doğrudur?
   a) Diziler fonksiyonlara parametre olarak gönderilemezler.
   b) Dizinin sadece birinci boyutu parametre olarak gönderilebilir.
   c) Dizinin boyut sayısı belirtilmeli; ancak boyutlarının eleman sayısı belirtilmeksizin fonksiyona parametre olarak gönderilmelidir.
   d) Dizinin iki boyutlu olduğu belirtilmeli ve ikinci boyutun eleman sayısı mutlaka belirtilmelidir.
   e) Dizinin boyutları ve eleman sayıları belirtilmeden fonksiyona parametre olarak gönderilmelidir.

6.
```
for (int i=0; i< 5; i++)
{
   for(int j=0; j<5; j++)
   {
       a[i][j] = i * j;
   }
 }
```
Yukarıdaki kod parçası aşağıdaki işlevlerden hangisini yerine getirmektedir?
   a) Dizinin tüm elemanlarına bulundukları satır ve sütün indis değerlerinin çarpım sonucunu yazar.
   b) Dizinin tüm elemanlarına 5 yazar.
   c) Dizinin tüm elemanlarına 25 yazar.
   d) Dizinin ilk 5 satırının ilk 5 sütununa 25 yazar.
   e) Dizinin ilk 5 satırının ilk 5 sütununa 5 yazar.

7. İki boyutlu 2 dizinin çarpımı için aşağıdakilerden hangisi doğru olmak zorundadır?
   a) Dizilerin satır sayıları eşit olmalıdır.
   b) Dizilerin sütün sayıları eşit olmalıdır.
   c) Birinci dizinin satır sayısı ikinci dizinin sütün sayısına eşit olmalıdır.
   d) Birinci dizinin sütun sayısı ikinci dizinin satır sayısına eşit olmalıdır.
   e) Satır ve sütün sayıları eşit olmak zorunda değildir.

8. Çok boyutlu diziler tanımlanırken aşağıdakilerden hangisine dikkat edilmelidir?
   a) Çok boyutlu dizilerde en soldaki boyut dışındaki boyutların eleman sayıları tanımlanmalıdır.
   b) Çok boyutlu diziler tanımlanırken boyutların eleman sayıları tanımlanmak zorunda değildir.
   c) Çok boyutlu diziler tanımlanırken boyutlar belirtilmek zorunda değildir.
   d) Çok boyutlu diziler tanımlanırken en soldaki eleman 0 girilmelidir.
   e) Çok boyutlu diziler tanımlanırken en soldaki boyutun eleman sayısı tanımlanmak zorundadır.

9. İki boyutlu dizi elemanlarına atama işlemlerinden hangisi doğrudur?
   a) int a[5][2] = {{5,{2}};
   b) int a[5][2] = {{1,2,3,4,5,{2}};
   c) int a[5][2] = {1,2,3,4,5,6,7};
   d) int a[5][2] = {1,2,3,4,5,6,7,8,9,10};
   e) int a[5][2] = {5,2};

10. İki boyutlu dizi elemanlarına atama işlemlerinden hangisi yanlıştır?
   a) int a[2][3] = {1,2,3,4,5,6};
   b) int a[4][3] = {{1,2,3}, {4,5,6}, {7,8,9},{10,11,12};
   c) int a[4][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
   d) int a[2][2] = {{1,2},{3,4}};
   e) int a[2][2] = {{1,2},3,4}};

Cevap Anahtarı

1.b, 2.c, 3.c, 4.c, 5.d, 6.a, 7.d, 8.a, 9.d, 10.e

YARARLANILAN KAYNAKLAR

[1] Pozrikidis, C. , *Introduction to C++ Programming and Graphics*, Springer Natural Amerika Inc., 2007

[2] Aydın, S., *Nesne Tabanlı Programlama, Yayımlanmamış ders notu*, Yönetim Bilişim Sistemleri Bölümü, Atatürk Üniversitesi, 2017

[3] Şeker, Ş. E., *Çok boyutlu diziler (MultiDimensional Arrays), http://bilgisayarkavramlari.sadievrenseker.com/2009/04/30/cok-boyutlu-diziler-multidimensional-arrays/*

STRING SINIFI VE KARAKTER DİZİSİ İŞLEMLERİ

Atatürk Üniversitesi
Açıköğretim Fakültesi

İÇİNDEKİLER

- String Tanımlama
- Bellek Fonksiyonları
- Erişim Yöntemleri
- Metin Düzenleme Fonksiyonları
- Diğer Fonksiyonlar

HEDEFLER

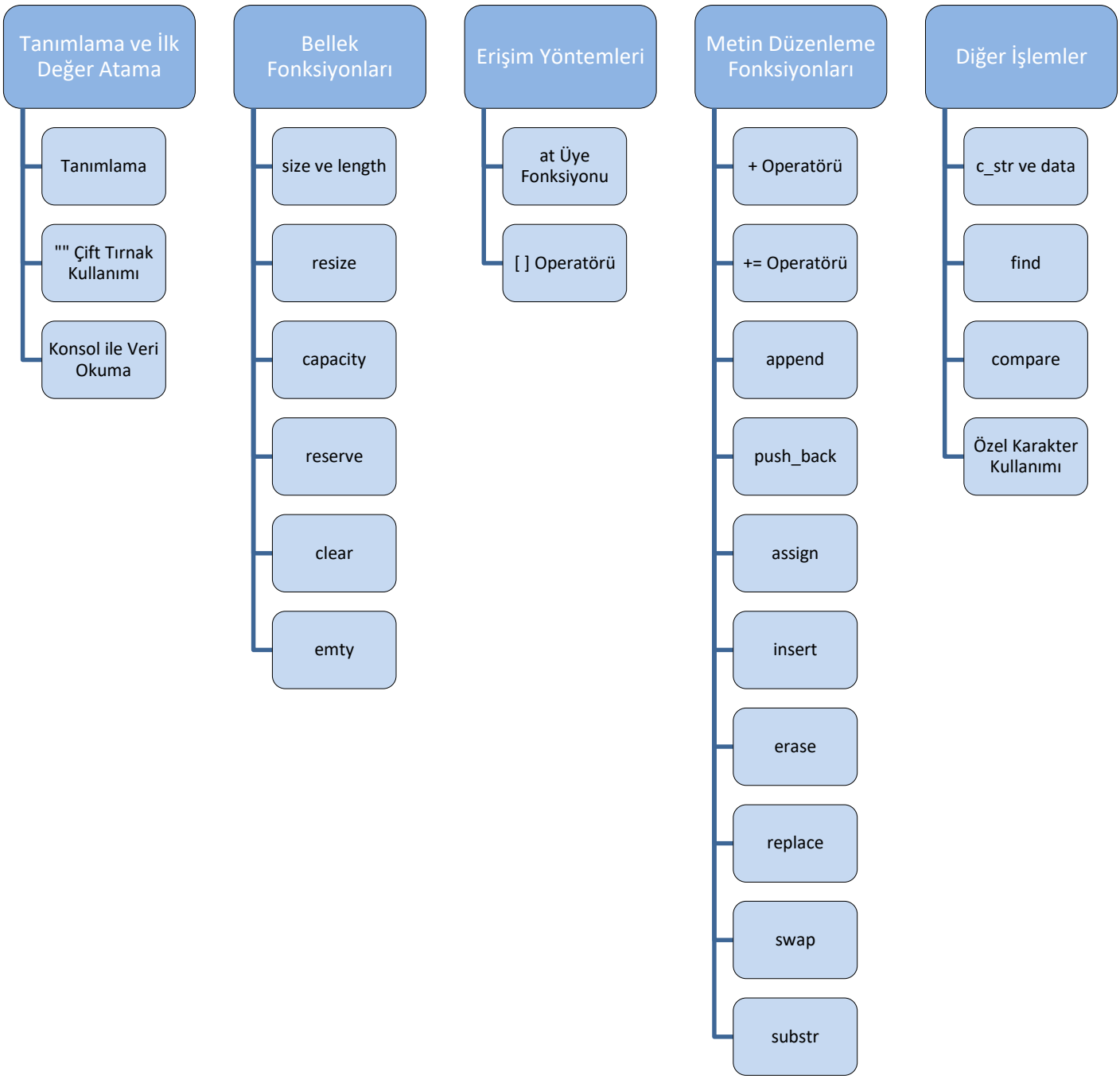
- Bu üniteyi çalıştıktan sonra;
  - String sınıfını ve karakter dizilerini kavrayabilecek,
  - String sınıfı değişkeni oluşturabilecek,
  - String üye fonksiyonlarının işlevlerini kavrayabilecek,
  - String üye fonksiyonlarının kullanımlarını öğrenebilecek,
  - String sınıfı nesnelerinin bellek ile ilgili işlemlerini yapabilecek,
  - String ifadeleri olan metinler üzerinde birleştirme, ekleme, silme, arama gibi işlemler yapabileceksiniz.

PROGRAMLAMA TEMELLERİ

Arş. Gör. Emrah ŞİMŞEK

ÜNİTE

11


Tanımlama ve İlk Değer Atama
Tanımlama
"" Çift Tırnak Kullanımı
Konsol ile Veri Okuma
Bellek Fonksiyonları
size ve length
resize
capacity
reserve
clear
emty
Erişim Yöntemleri
at Üye Fonksiyonu
[ ] Operatörü
Metin Düzenleme Fonksiyonları
+ Operatörü
+= Operatörü
append
push_back
assign
insert
erase
replace
swap
substr
Diğer İşlemler
c_str ve data
find
compare
Özel Karakter Kullanımı

GİRİŞ

String sınıfı, karakter dizileri olarak bildiğimiz veri türüdür. Başka bir ifadeyle string sınıfı değişkenleri, karakter dizilerini temsil eden nesnelerdir. C++ dilinde *string* sınıfı temel (ilkel) veri türlerinden birisi değildir ve karakter (char) veri tipinde değerler saklayan tek boyutlu diziler olarak ifade edilmektedir. String nesneleri, diğer veri tiplerine benzer şekilde kullanılabilmektedir. String nesnelerine atama yapılabilir, *string* değişkenleri birleştirilebilir veya parçalanabilir, *string* değişkenleri arasında karşılaştırma yapılabilir. String sınıfı, metin tipindeki bilgilerin saklanması, kullanılması, değiştirilmesi, kıyaslanması, parçalara ayrılması için kullanılan birçok özellik ve üye fonksiyonu içermektedir. Bu nedenle programlarda metin tipindeki veriler olan dosya isimlerinde, bellek adres bilgilerinde, ekran çıktılarında olmak üzere birçok alanda kullanılmaktadır.

String ifadeler karakterlerden oluşmaktadır ve bir *string* ifadesi çift tırnak arasında tanımlanmış karakterler dizisi şeklinde tanımlanmaktadır. Aşağıdaki örnek bir *string* ifadesidir ve 17 adet karakterden oluşmaktadır:

"String ornegidir."

String sınıfı, metin tipindeki verilerde yapılacak işlemleri kolaylaştırması sebebiyle büyük öneme sahiptir. Bu işlemler ana başlıklar olarak tanımlama, bellek fonksiyonları, erişim ve metin düzenleme işlemleri olarak ifade edilebilir. String sınıfı değişkeni diğer veri tiplerinde olduğu gibi önce değişken tipi, sonrasında değişken adı kullanılarak tanımlanır. *String* sınıfından bir değişkene ilk değer atamak için ise " " çift tırnak kullanılabilir. Bellek fonksiyonları ana başlığında *size, length, resize, capacity, reserve, clear, empty* üye fonksiyonları bulunmaktadır. String sınıfı değişkenlerinin karakter tipindeki elemanlarına erişim amacıyla *at* üye fonksiyonu ve *[ ]* operatörü kullanılmaktadır. String sınıfı değişkenlerde metin düzenleme yapılması amacıyla *+* operatörü, *append, push_back, assign, insert, erase, replace ,swap, find, substr* ve *compare* üye fonksiyonları bulunmaktadır. Ayrıca bir *string* ifadenin *c string* bir ifadeye dönüştürülmesini sağlayan *c_str* ve *data* üye fonksiyonları bulunmaktadır.

STRING TANIMLAMA

String sınıfının kullanılabilmesi için öncelikle programımızda *string* sınıfı kütüphanesinin tanımlanması gerekmektedir. String sınıfının kütüphanesi aşağıdaki şekilde tanımlamaktadır:

String sınıfından bir değişkene değer olarak " " (çift tırnak) arasına yazılan karakterler atanmaktadır.

#include <string>

Yukarıdaki ifade ile artık *string* sınıfı değişkenlerini programımızda kullanabiliriz. String sınıfından bir nesne, diğer veri tiplerindeki değişkenlerle aynı şekilde oluşturulmaktadır. Veri tipi yerine *string* yazılması yeterlidir. Aşağıda ornekString adında bir string nesnesine ilk değer olarak “Ilk string tanimlama.” karakter dizisi atanmıştır.

```
string ornekString = "Ilk string tanimlama.";
```

Yukarıdaki string ifadesinde 21 karakter bulunmaktadır. Çift tırnaklar arasındaki başta rakamlar ve harfler olmak üzere noktalama işaretleri ve matematiksel işaretler gibi bütün karakterler bu string değişkenin bir parçasıdır. Yalnızca farklı işlevleri olan bazı karakterlerin kullanımı için özel karakter kullanımı bölümüne bakmanız gerekmektedir.

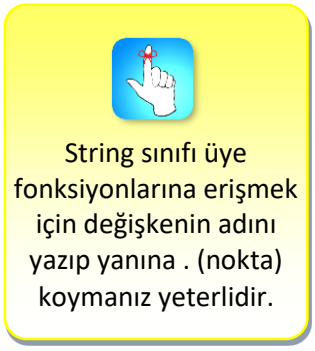
String sınıfının üye fonksiyonlarına erişebilmek için tanımladığımız değişkenin adını yazıp devamına "." koymamız gerekmektedir. Şekil 11.1'de gösterildiği üzere bu şekilde *string* sınıfına ait bütün üye fonksiyonları görülebilmektedir.

```
string ornekString = "Ilk string tanimlama.";
```

String değişkenlere değer atama her zaman çift tırnak arasında yazılarak yapılmayabilir. Mesela aşağıdaki ifadede bir *string* değişkenin bir parçası başka bir *string* değişkene ilk değer olarak atanmıştır.

```
string ornekString = "Ilk string tanimlama.";

string ornekString2(ornekString.begin() + 7, ornekString.end() - 1);
```

Yukarıdaki ifadede ornekString değişkeninin bir parçası olan "tanimlama" kelimesi ornekString2 değişkenine ilk değer olarak atanmıştır. begin ve end üye fonksiyonları *string* değişkeninin başlangıç ve bitiş konumlarını geriye döndürmektedir. Bu fonksiyonların yanlarındaki sayılar kullanılarak bu konumlardaki *string* karakterleri arasında belirli sayıda ileri veya geri gidilmesi mümkündür.



String sınıfı üye fonksiyonlarına erişmek için değişkenin adını yazıp yanına . (nokta) koymanız yeterlidir.

```
#include <iostream>
#include <string>

using namespace std;
// Arş. Gör. Emrah Şimşek

int main()
{

    string ornekString = "İlk string tanımlama.";
    ornekString.
```

append
assign
at
back
begin
capacity
cbegin
cend
clear

Şekil 11.1. String Sınıfı Üye Fonksiyonları

String sınıfı değişkenlerini cin ve cout komutlarıyla doğrudan kullanabilirsiniz.

Bunların dışında *string* sınıfındaki bir değişkene klavyeden veya dosyadan metin okuyarak değer atamak mümkündür.

```
string ornekString;

cin >> ornekString;
```

BELLEK FONKSİYONLARI

Bellek fonksiyonları ana başlığında *string* sınıfı değişkenlerinin boyutlarının ve kapasitelerinin öğrenilmesi, boş olup olmadıklarının kontrol edilmesi, yeniden boyutlandırılması gibi amaçlarla kullanılan üye fonksiyonlar anlatılmaktadır. Bu üye fonksiyonlar aşağıda listelenmiştir:

- *size ve length*
- *resize*
- *capacity*
- *reserve*
- *clear*
- *empty*

size ve length Üye Fonksiyonu

size ve *length* üye fonksiyonları, *string* sınıfı değişkeninin boyutunu öğrenmek amacıyla kullanılmaktadır. Bazı uygulamalarda *string* sınıfı değişkenlerin boyutları, üzerinde çalışacak metin ile ilgili ön bilgi verebilmektedir. Örneğin; klavyeden 5 karakterlik bir metin okunmak istendiğinde, girilen metnin 5 karakter olup olmadığı kolayca kontrol edilebilir. Bu eş anlamlı iki üye fonksiyonun kullanımı aşağıda gösterilmiştir:

```
string ornekString ="String boyut öğrenme.";

cout << ornekString.size() << endl;

cout << ornekString.length() << endl;
```

```
C:\WINDOWS\system32\cmd.exe
21
21
Press any key to continue . . .
```

String sınıfı değişkenlerin boyutunu kontrol altında tutmak, programın performansını arttırır.

resize Üye Fonksiyonu

resize üye fonksiyonu, üstteki başlıkta ifade edilen, *string* sınıfı değişkenin boyutunun yeniden düzenlenmesi amacıyla kullanılmaktadır. Bu üye fonksiyonu ile *string* sınıfı değişkenin boyutu azaltılabilir veya arttırılabilir. Bu üye fonksiyonunda parametre olarak 0 veya pozitif bir tam sayı kullanılmalıdır. Kullanımı şu şekildedir:

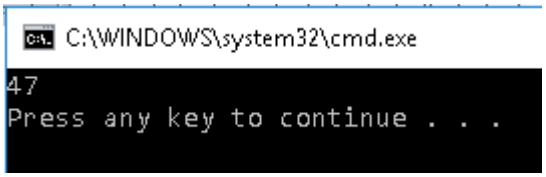
```
string ornekString = "String yeniden boyutlandırma.";

ornekString.resize(14);

cout << ornekString.size() << endl << ornekString << endl;
```

```
C:\WINDOWS\system32\cmd.exe
14
String yeniden
Press any key to continue . . .
```

capacity Üye Fonksiyonu

String sınıfı değişkenin kapasitesi ilk değer atandığı zaman derleyici tarafından belirlenmektedir. *capacity* üye fonksiyonu bu değişkenin kapasitesinin öğrenilebilmesi amacıyla kullanılmaktadır. Program içerisinde *string* sınıfı değişkene daha az sayıda karaktere sahip bir başka değer atanması durumunda bu kapasite değeri değişmemekte olup, daha büyük boyutlu bir değer atandığı zaman bu kapasite değeri, bu büyük değerin karakter sayısı olmaktadır. Aşağıda örnek kullanımı gösterilmiştir:

```
string ornekString = "String ifadesinin kapasitesini öğrenme.";

ornekString = "Deneme";

cout << ornekString.capacity() << endl;
```



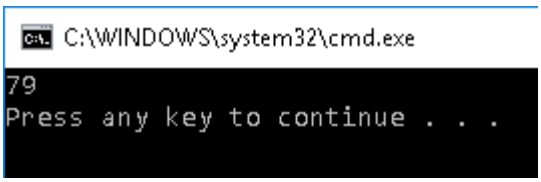
String sınıfı değişkenler için bellekte 15, 31, 47, 63, 79 gibi 16'nın katlarının bir eksiği şekillerde kapasite ayrılmaktadır. 39 karakter bulunan string sınıfı değişkene daha az karaktere sahip başka bir değer atanmasına rağmen kapasitesi 47'dir.

reserve üye fonksiyonu ile string sınıfı değişkene değer atamadan bellekte kapasite ayrılabilmektedir.

reserve Üye Fonksiyonu

reserve üye fonksiyonu, *string* sınıfı değişkeninin kapasitesinin belirlenmesi amacıyla kullanılmaktadır. Bazı uygulamalarda büyük boyutlu olan *string* sınıfı değişkeninin kapasitesinin bu üye fonksiyon ile küçültülmesi, bu sayede programın bellekte gereksiz bellek kullanmasının önüne geçilmesi de sağlanabilir. Bu üye fonksiyon pozitif bir tam sayı değerini parametre olarak almaktadır. Kullanımı aşağıda gösterilmiştir:

```
string ornekString;

ornekString.reserve(64);

cout << ornekString.capacity() << endl;
```



Bellekte 16'nın katlarının bir eksiği şeklinde kapasite ayrıldığından, 64 karakterlik değişken için bellekte 79 karakterlik kapasite ayrılmıştır.

clear Üye Fonksiyonu

clear üye fonksiyonu, *string* sınıfı değişkenin içeriğinin silinmesi amacıyla kullanılmaktadır. Herhangi bir parametre gerektirmemektedir ve bu fonksiyondan

sonra *string* sınıfı değişkenin boyutu 0 olmaktadır. Örnek kullanımı aşağıda verilmiştir:

```
string ornekString = "String içeriği silme.";
ornekString.clear();
cout << ornekString <<" " << ornekString.length() << endl;
```

```
C:\WINDOWS\system32\cmd.exe
0
Press any key to continue . . .
```

emtpy Üye Fonksiyonu

empty üye fonksiyonu, string sınıfı değişkenin boş olup olmadığını kontrol etmek amacıyla kullanılır.

empty üye fonksiyonu, *string* sınıfı değişkenin içeriğinin boş olup olmadığını, yani boyutunun 0 olup olmadığını kontrol etmek amacıyla kullanılır. Geriye *boolean* ifadeler olan *true* veya *false* değeri döndüren bu fonksiyon parametre gerektirmemektedir. Genelde *if* yapısında kontrol amacıyla veya *while* döngülerinde koşul olarak kullanılmaktadır. Örnek kullanımı aşağıda verilmiştir:

```
string ornekString;
cout << ornekString.empty() << endl;
ornekString = "Deneme.";
cout << ornekString.empty() << endl;
```

```
C:\WINDOWS\system32\cmd.exe
1
0
Press any key to continue . . .
```

Örnek

- Kullanıcıdan 3 adet string ifade alınacaktır. Bu işlem için 1 adet string sınıfı değişken tanımlayarak 5 karakter boyutu için bellekte yeterli kapasite ayırınız. Okunan string ifadelerin boyutunu kontrol ederek 5 karakter olanları kabul eden, onun dışında uyarı vererek tekrar veri girilmesini sağlayan C++ kodunu yazınız. Programın sonunda değişkenin içeriğini silmeyi unutmayınız.

```
#include <iostream>
#include <string>
#include <locale>
using namespace std;
// Arş. Gör. Emrah Şimşek
int main()
```

```
{
	setlocale(LC_ALL, "Turkish");
	string ornekString;
	ornekString.reserve(5);
	cout << "5 Karakter Boyutunda 3 String Ifade Giriniz:";
	int degiskenSayisi = 0;
	while (degiskenSayisi<3)
	{
		cout << endl << degiskenSayisi + 1 << ". String Ifadeyi Giriniz:";
		cin >> ornekString;
		if (ornekString.length() == 5)
		{
			cout << "İfade kabul edildi.";
			degiskenSayisi++;
		}
		else
		{
			cout << "Lütfen 5 karakter içeren bir ifade giriniz.";
		}
	}
	cout << endl;
	ornekString.clear();
	return 0;
}
```

ERİŞİM YÖNTEMLERİ

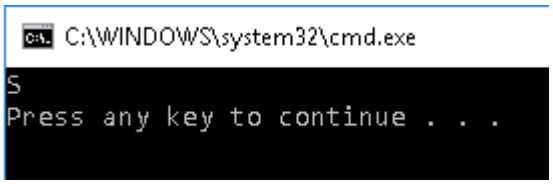
String sınıfı değişkenin karakterlerine erişebilmek amacıyla at üye fonksiyonu ve [ ] operatörü kullanılmaktadır.

String sınıfı değişkenlerinin karakterlerden oluştuğunu önceki başlıklarda ifade etmiştik. Yazdığımız programlarda string sınıfını oluşturan bu karakterlere erişme ihtiyacı duyabilmekteyiz. Böyle durumlarda *string* sınıfı değişkenin karakterlerine erişebilmek amacıyla *at* üye fonksiyonu ve *[ ]* operatörü kullanılmaktadır. String sınıfı değişkenin karakter dizisi anlama geldiğini göz önüne alırsak, *[ ]* operatörü ile dizilerin elemanlarına eriştiğimiz gibi, her bir karaktere, karakterin dizideki konumunu kullanarak erişmemiz mümkündür. Her iki yöntemde de ulaşmak istediğimiz karaktere ait konumu kullanmamız gerekmektedir.

at Üye Fonksiyonu

String sınıfı değişkenin karakterlerine erişmek amacıyla kullanılan *at* üye fonksiyonu, 0'dan büyük ve değişkenin boyut değerinden küçük pozitif tam sayılar almaktadır. Geriye ise ilgili konumda bulunan karakteri döndürmektedir. Örnek kullanımı aşağıda gösterilmiştir:

```
string ornekString = "String karakterlerine erisme.";
cout << ornekString.at(0) << endl;
```

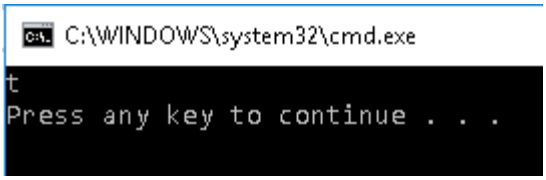


String sınıfı değişkenlerinin elemanı olan karakterlere, diğer veri tiplerindeki dizi elemanları ile aynı şekilde erişilebilir.

[] Operatörü

[ ] operatörü dizilerin elemanlarına erişmek için kullanıldığından, *string* sınıfı değişkenlerde de karakterlere bu şekilde erişmek mümkündür. Örnek kullanımı aşağıda gösterilmiştir:

```
string ornekString = "String karakterlerine erisme.";
cout << ornekString[1] << endl;
```



Örnek

- Kullanıcıdan alınacak, rakamlardan oluşan ve boyutu tek sayı olan bir string ifadenin palindrom olup olmadığını kontrol eden bir C++ kodu yazınız. Sonuç olarak ekrana palindrom veya palindrom değil çıktısı yazdırınız. Konsoldan girilen ifadenin tek sayıda karakter uzunluğunda ve rakamlardan oluştuğunu varsayabilirsiniz. Programın sonunda değişkenin içeriğini silmeyi unutmayınız.

```
#include <iostream>
#include <string>
#include <locale>
using namespace std;
// Arş. Gör. Emrah Şimşek
int main()
{
	setlocale(LC_ALL, "Turkish");
	string ornekString;
	cout << "Tek Sayıda Karakter Uzunluğunda Rakamlardan Oluşan Bir String Ifade Giriniz:";
	cin >> ornekString;
	int stringBoyutu = ornekString.length();
	bool palindromKontrol = true;
	for (int i = 0; i < (stringBoyutu-1)/2; i++)
	{
		if (ornekString.at(i) != ornekString[stringBoyutu - i - 1])
		{
			palindromKontrol = false;
			break;
```

```
                }
        }
        if (palindromKontrol == true)
                cout << "İfade palindromdur." << endl;
        else
                cout << "İfade palindrom değildir." << endl;
        ornekString.clear();
        return 0;
}
```

METİN DÜZENLEME FONKSİYONLARI

Bu başlıkta *string* sınıfı değişkenlerde *düzenleme* yapılması amacıyla kullanılan üye fonksiyonları açıklanmaktadır. Bu düzenleme işlemleri kısaca *string* ifadelerin birleştirilmesi, silinmesi, eklenmesi veya iki değişkenin değerlerinin yer değiştirilmesi olarak açıklanabilir. Yapılan çalışmalarda bu tür metin işlemleri oldukça sıklıkla uygulanmaktadır. Bu amaçla kullanılabilen üye fonksiyonlar aşağıda listelenmiştir:

- + operatörü
- += operatörü
- append
- push_back
- assign
- insert
- erase
- replace
- swap
- substr

+ Operatörü

+ operatörü, string sınıfı değişkenlerinin birbirleri ile veya çift tırnak içinde tanımlanmış string ifadelerin string sınıfı değişkenleri ile *birleştirilmesi* amacıyla kullanılmaktadır. Kullanımı sayısal değişkenlerin kullanımı ile aynıdır. Bu birleştirme işleminde değişkenler veya string ifadeler yazıldıkları sırayla birleştirilirler. Örnek kullanımı aşağıda verilmiştir:

String sınıfı değişkenlerinin toplanması, ilgili değişkenlerin içeriklerinin yan yana birleştirilmesi anlamına gelmektedir.

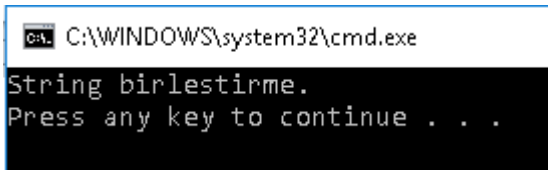
```
string ornekString1 = "String ";
string ornekString2 = "birlestirme.";
string ornekString = ornekString1 + ornekString2;
cout << ornekString << endl;
```

```
C:\WINDOWS\system32\cmd.exe
String birlestirme.
Press any key to continue . . .
```

+= Operatörü

+= ve benzer operatörler, matematiksel işlemlerde bir değişkenin bir değer veya başka bir değişken ile işleme alınarak sonucun yine aynı değişkene atanması gerektiği durumlarda kullanılır. Aynı durum string sınıfı değişkenlerinin birbirleri ile veya çift tırnak içinde tanımlanmış string ifadelerin string sınıfı değişkenleri ile birleştirilmesi amacıyla da kullanılabilmektedir. Kullanımı sayısal değişkenlerin kullanımı ile aynıdır. Örnek kullanımı aşağıda verilmiştir:

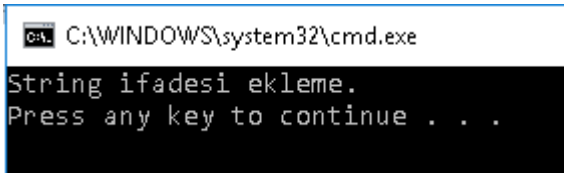
```
string ornekString1 = "String ";
string ornekString2 = "birlestirme.";
ornekString1 += ornekString2;
cout << ornekString1 << endl;
```



append Üye Fonksiyonu

append üye fonksiyonu, bir string sınıfı değişkenin *sonuna* başka bir string ifade *eklemek* amacıyla kullanılır. Eklenecek olan string, ifadenin tamamı olabileceği gibi girilen parametrelerle belirlenebilen bir parçası da olabilir. 3 parametre alan bu fonksiyonda ilk parametre string sınıfı değişken veya çift tırnak içerisinde yazılmış bir string ifadeyi, 2. parametre olarak eklenecek olan string ifadenin hangi karakterden itibaren ekleneceğini, 3. parametre ise 2. parametreden itibaren kaç karakter ekleneceğini belirlemektedir. 2. ve 3. parametreler girilmediği durumda belirlenen string ifadenin tamamı diğer string ifadenin sonuna eklenmektedir. Örnek kullanımı aşağıda gösterilmiştir:

```
string ornekString = "String ";
string ornekString1 = "ifadesi ekleme.";
ornekString.append(ornekString1);
cout << ornekString << endl;
```

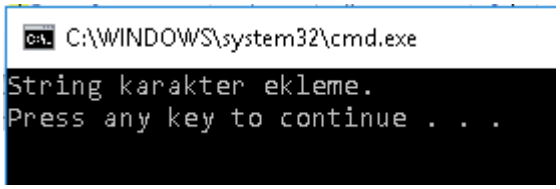


push_back Üye Fonksiyonu

push_back üye fonksiyonu, string sınıfı değişkenin *sonuna* bir karakter *eklenmesi* amacıyla kullanılmaktadır. Karakter eklendikten sonra string sınıfı değişkenin boyutu 1 artar. Bu üye fonksiyon genelde dosyadan karakterler halinde veri okunurken tercih edilir. Kullanımı aşağıda gösterilmiştir:

```
string ornekString = "String karakter ekleme";
ornekString.push_back('.');
```

```
cout << ornekString << endl;
```

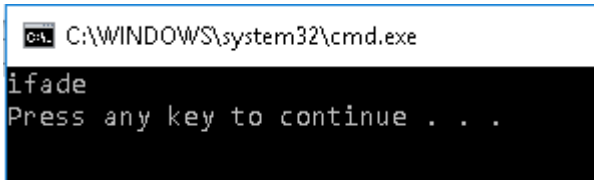


assign Üye Fonksiyonu

assign üye fonksiyonu = (eşittir) işareti ile aynı işlevi olan atama fonksiyonudur.

assign üye fonksiyonu, bir string sınıfı değişkene başka bir string ifadeyi *atamak* amacıyla kullanılır. Bu atama işleminden sonra string sınıfı değişkenin eski değeri tamamen silinir, yerine yeni string ifade atanır. Yeni atanacak karakter dizisi bu string ifadenin tamamı olabileceği gibi girilen parametrelerle belirlenebilen bir parçası da olabilir. 3 parametre alan bu fonksiyonda ilk parametre string sınıfı değişken veya çift tırnak içerisinde yazılmış bir string ifadeyi, 2. parametre olarak eklenecek olan string ifadenin hangi karakterden itibaren ekleneceğini, 3. parametre ise 2. parametreden itibaren kaç karakter ekleneceğini belirlemektedir. 2. ve 3. parametreler girilmediği durumda belirlenen string ifadenin tamamı string sınıfı değişkene atanır. Örnek kullanımı aşağıda gösterilmiştir:

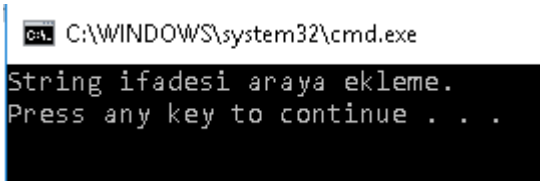
```
string ornekString = "String ifadesi atama.";
string ornekString1 = "ifadesi atama.";
ornekString.assign(ornekString1, 0, 5);
cout << ornekString << endl;
```



insert Üye Fonksiyonu

insert üye fonksiyonu, bir string sınıfı değişkenin *herhangi bir konumuna* başka bir string ifadeyi *eklemek* amacıyla kullanılır. Eklenecek karakter dizisi string ifadenin tamamı olabileceği gibi girilen parametrelerle belirlenmiş bir parçası da olabilir. 4 parametre alan bu fonksiyonda ilk parametre string sınıfı değişkenin hangi konumundan itibaren yeni ifadenin ekleneceğini belirtir. 2. parametre, eklenecek olan string sınıfı değişken veya çift tırnak içerisinde yazılmış bir string ifadedir. 3. parametre olarak eklenecek olan string ifadenin hangi karakterden itibaren ekleneceğini, 4. parametre ise 3. parametreden itibaren kaç karakter ekleneceğini belirlemektedir. 3. ve 4. parametreler girilmediği durumda belirlenen string ifadenin tamamı string sınıfı değişkenin belirtilen konumuna eklenir. Örnek kullanımı aşağıda gösterilmiştir:

```
string ornekString = "String ifadesi ekleme.";
string ornekString1 = " araya";
ornekString.insert(14, ornekString1);
cout << ornekString << endl;
```

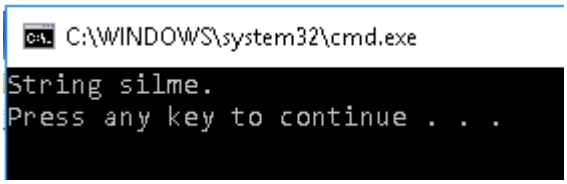


erase Üye Fonksiyonu

String sınıfının üye fonksiyonları program yazarken karakter dizilerine kıyasla, işimizi oldukça kolaylaştırmaktadır.

erase üye fonksiyonu, bir string sınıfı değişkenin belirli bir parçasının *silinmesi* amacıyla kullanılır. 2 parametre alan bu fonksiyonda ilk parametre string ifadenin hangi karakterden itibaren silineceğini, 2. parametre ise 2. parametreden itibaren kaç karakter silineceğini belirlemektedir. Silme işleminden sonra string sınıfı değişkenin boyutu azalmış olur. Örnek kullanımı aşağıda gösterilmiştir:

```
string ornekString = "String ifadesi silme.";
ornekString.erase(7, 8);
cout << ornekString << endl;
```

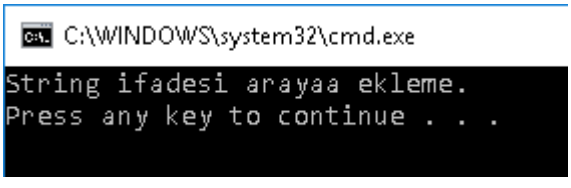


replace üye fonksiyonu, erase ve insert üye fonksiyonlarının işlevlerinin birleştirilmiş halidir.

replace Üye Fonksiyonu

replace üye fonksiyonu, *erase* ve *insert* üye fonksiyonlarının işlevlerinin birleştirilmiş halidir. Kullanılan parametrelerle hem string sınıfı değişkenin bir kısmının *silinmesini* hem de string ifade *eklemeye* imkân sağlamaktadır. 5 parametre alabilen bu üye fonksiyonunda, ilk parametre string sınıfı değişkenin hangi karakterinden itibaren silineceğini ve 2. parametre de 1. parametreden itibaren kaç karakter silineceğini ifade etmektedir. String ifade ekleme işlemi bu silinen konumdan itibaren başlamaktadır. 3. parametre string sınıfı değişkene eklenecek olan değişkenin adını, 4. parametre araya eklenecek olan string ifadenin hangi karakterinden itibaren ekleneceğini, 5. parametre ise 4. karakterden itibaren kaç karakter ekleneceğini ifade etmektedir. 4. ve 5. parametre kullanılmadığı takdirde string ifadenin tamamı diğer string ifadeye eklenmektedir.

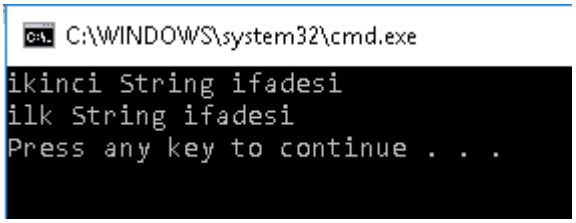
```
string ornekString = "String ifadesi sona ekleme.";
string ornekString1 = " araya";
ornekString.replace(14, 4, ornekString1);
cout << ornekString << endl;
```



swap Üye Fonksiyonu

swap üye fonksiyonu, iki string sınıfı değişkenin değerlerinin *değiştirilmesi* amacıyla kullanılır. Örnek kullanım aşağıda gösterilmiştir:

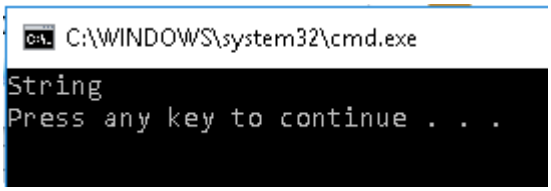
```
string ornekString1 = "ilk String ifadesi";
string ornekString2 = "ikinci String ifadesi";
ornekString1.swap(ornekString2);
cout << ornekString1 << endl << ornekString2 << endl;
```



substr Üye Fonksiyonu

substr üye fonksiyonu, string sınıfı değişkenin değerlerini bir başka değişkene *kopyalamak* amacıyla kullanılır. String ifadenin tamamını kopyalamak için = operatörü kullanılabilir. *substr* üye fonksiyonu genelde string ifadenin bir parçasını kopyalamak gerektiğinde tercih edilmektedir. 2 parametre alan bu üye fonksiyonunun ilk parametresi kopyalanacak string ifadesinin hangi karakterden itibaren kopyalanacağını, 2. parametre ise ilk parametreden itibaren kaç karakter kopyalanacağını ifade etmektedir. Bu üye fonksiyonu kopyalanan string ifadesinin sonuna *c string* ifadesindeki boşluk karakterini eklemez. Örnek kullanım aşağıda gösterilmiştir:

```
string ornekString1 = "String kopyalama.";
string ornekString2;
ornekString2 = ornekString1.substr(0, 6);
cout << ornekString2 << endl;
```



Örnek

•String ifadelerde işlemler yapılmasını sağlayan bir C++ programı yazınız. Program 1 adet string sınıfı değişken içerecektir ve string değişken üzerinde belirli işlevleri yerine getirecektir. Bu program aşağıdaki menüleri içermelidir:

- •Ekrana Yazdır: String sınıfı değişkenin içeriğinin ekrana yazdırılması.
- •Yeni Değer Ata: Değişkene yeni değer atanması.
- •Yeniden Boyutlandır: Değişkenin yeniden boyutlandırılması
- •Ara: Değişkende bir string ifadeyi arama.
- •Araya Metin Ekle: Değişkenin istenen kısmına başka bir metin ekleme.
- •Aradan Metin Sil: Değişkenin istenen kısmını silme.
- •İçeriği Temizle: Değişkenin içeriğini tamamen temizleme.
- •Çıkış

Programlarınızda Türkçe karakterler kullanabilmek için locale kütüphanesini ve setlocale(LC_ALL, "Turkish") komutunu kullanabilirsiniz.

```
#include <iostream>
#include <string>
#include <locale>
using namespace std;
// Arş. Gör. Emrah Şimşek
int main()
{
	setlocale(LC_ALL, "Turkish");
	string ornekString;
	int geciciDeğisken, geciciDegisken2;
	string geciciString;
	int menuKontrol = 1;
	while (menuKontrol > 0)
	{
		if (menuKontrol!=1 && menuKontrol != 4)
			system("cls");
		cout << "Yapmak İstediğiniz İşlemin Sayısını Giriniz:" << endl;
		cout << "1:Ekrana Yazdır\n2:Yeni Değer Ata\n3:Yeniden
Boyutlandır\n4:Ara\n5:Araya Metin Ekle\n6:Aradan Metin Sil\n7:İçeriği
Temizle\n0:Çıkış\n";
		cin >> menuKontrol;
		switch (menuKontrol)
		{
		case 0:
			break;
		case 1:
			cout << ornekString << endl;
			break;
```

```
                case 2:
                        cout << "Yeni Bir String İfade Giriniz(Boşluk Karakteri
İçermeyen Metin):";
                        cin >> ornekString;
                        break;
                case 3:
                        cout << "Yeni Boyut Değerini Giriniz:";
                        cin >> geciciDeğisken;
                        ornekString.resize(geciciDeğisken);
                        break;
                case 4:
                        cout << "Aranacak String İfadeyi Giriniz:" << endl;
                        cin >> geciciString;
                        geciciDeğisken=ornekString.find(geciciString);
                        if (geciciDeğisken > -1)
                                cout << "Aranan String İfade " << geciciDeğisken
<< ". Konumda Bulunmaktadır." << endl;
                        else
                                cout << "Aranan String İfade bulunmamaktadır."
<< endl;
                        break;
                case 5:
                        cout << "Araya Eklenecek String İfadeyi Giriniz:" << endl;
                        cin >> geciciString;
                        cout << "String İfadenin Ekleneceği Konumu Giriniz:" <<
endl;
                        cin >> geciciDeğisken;
                        ornekString.insert(geciciDeğisken,geciciString);
                        break;
                case 6:
                        cout << "String İfadeden Silmek İstediğiniz Metnin
Başlangıç İndisini Giriniz:" << endl;
                        cin >> geciciDeğisken;
                        cout << "Silmek İstediğiniz Metnin Boyutunu Giriniz:" <<
endl;
                        cin >> geciciDegisken2;
                        ornekString.erase(geciciDeğisken, geciciDegisken2);
                        break;
                case 7:
                        ornekString.clear();
                        break;
                default:
                        break;
                }
        }
```

```
        ornekString.clear();
        geciciString.clear();
        return 0;
}
```

DİĞER İŞLEMLER

Yukarıdaki ana başlıklarda tanımlanan üye fonksiyonlar ve yöntemler dışında string ifadenin dönüşümü, string ifadelerde arama ve karşılaştırma amacıyla kullanılan üye fonksiyonlar ve string ifadelerde özel karakter kullanımı da önemli bir yere sahiptir. Bu başlıkta aşağıdaki konular anlatılmaktadır:

c_str ve data üye fonksiyonları ile string bir ifadeyi c string bir ifadeye dönüştürebiliriz.

- c_str ve data
- find
- compare
- Özel Karakter Kullanımı

c_str ve data Üye Fonksiyonu

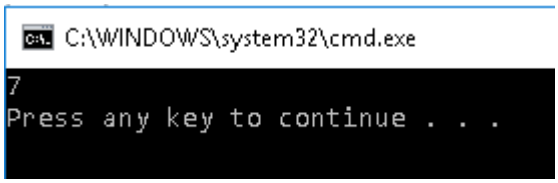
c_str ve *data* üye fonksiyonları, string sınıfı değişkeni *c string* bir ifadeye *dönüştürmekte* ve geriye değer olarak karakter dizisine ait bir *pointer* döndürmektedir. String sınıfı değişken ile *c string* ifadenin tek farkı *c string* ifadenin sonunda karakter dizisinin bittiğini ifade eden boşluk yani "\0" (*null*) karakteri bulunmasıdır. Dosya işlemlerinde dosya adı olarak kullanılan string ifadelerin bazı derleyicilerde *c string* ifadeye dönüştürülmesi gerekmektedir. Kullanımı aşağıda gösterilmiştir:

```
string ornekString = "C string dönüşümü.";
ornekString.c_str();
ornekString.data();
```

find Üye Fonksiyonu

find üye fonksiyonu, string sınıfı değişkenin içerisinde bir string ifadenin *aranması* amacıyla kullanılmaktadır. *find* üye fonksiyonu eğer aranan string ifade bulundu ise geriye eşleşmenin olduğu ilk karakterin indisini döndürmektedir. 3 parametre alan bu üye fonksiyonda ilk parametre string sınıfı değişkende aranacak string ifade, 2. parametre hangi karakterden itibaren aranacağı, 3. parametre ise aranacak string ifadesinin karakter sayısıdır. 2. ve 3. parametre zorunlu değildir. Kullanılmadıklarında string ifadenin tamamı bütün string sınıfı değişkende aranır. Eğer aranan değer string sınıfı değişkende *bulunamazsa* geriye *-1* değeri döndürülmektedir. Örnek kullanım aşağıda gösterilmiştir:

```
string ornekString1 = "String arama.";
string ornekString2 = "arama";
cout << ornekString1.find(ornekString2) << endl;
```

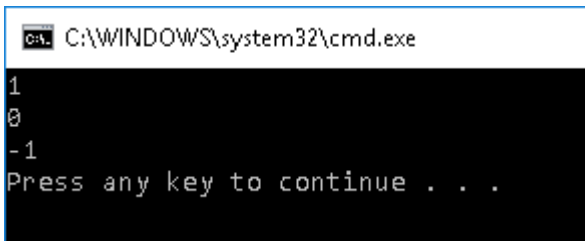


compare Üye Fonksiyonu

compare üye fonksiyonu, karşılaştırma sonucu olumlu ise 0 değerini geri döndürmektedir.

compare üye fonksiyonu, iki string sınıfı değişkenin veya string sınıfı değişken ile çift tırnak içinde yazılmış string ifadenin *karşılaştırılması* amacıyla kullanılmaktadır. string ifadelerin tamamının yerine bir parçasının da karşılaştırılması mümkündür. Bu üye fonksiyon geriye değer olarak doğru anlamında 1 ve yanlış anlamında 0 döndürmektedir. Bu üye fonksiyon tek parametre olarak string ifade kullanıldığında string ifadenin tamamını string sınıfı ifade ile karşılaştırarak sonuç döndürmektedir. Eğer string ifadeden önce 2 adet pozitif tam sayı parametre olarak girilirse, string sınıfı değişkenin bu parametrelerde belirtilen kısmı ile string ifade karşılaştırıp ilgili sonuç geriye döndürülür. Eğer string ifadeden sonra 2 tane pozitif tam sayı parametre olarak girilirse, string sınıfı değişken ile string ifadenin bu parametrelerde belirtilen kısmı karşılaştırılarak geriye sonuç döndürülür. Eğer karşılaştırma sonucu *olumlu* ise geriye *0* değeri, olumsuz veya karşılaştırılan string ifade diğerinden daha uzun ise *1* değeri, karşılaştırılan string ifade diğerinden daha kısa ise *-1* değeri geriye döndürülür. Örnek kullanım aşağıda gösterilmiştir:

```
string ornekString1 = "String karşılaştırma.";
string ornekString2 = "String";
cout << ornekString1.compare(ornekString2);
cout << ornekString1.compare(0, 6, ornekString2);
cout << ornekString1.compare(0, 2, ornekString2, 0, 3);
```



Özel Karakter Kullanımı

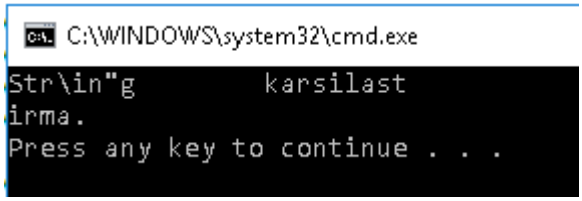
Özel karakterler, string ifadenin içerisinde derleyici için anlamlı olan karakterlerin kullanılmasını sağlamaktadır.

Özel karakterler ekrana veya dosyaya veri yazdırırken sekme atlatmak, özel bir karakter yazdırmak veya bir alt satıra geçmek amacıyla kullanılmaktadır. Bu karakterler "\" işareti ile birlikte yanına bir noktalama işareti veya harf alarak kullanılır. Bu amaçla kullanılabilen özel karakterler aşağıda listelenmiştir:

- \n veya \r: Bir alt satıra geçmek için,
- \t: Bir sekme boşluk bırakmak için,
- \', \", \?, \\: \'den itibaren yazılı olan sırasıyla ', ", ? ve \ karakterlerini yazdırmak için kullanılmaktadır.

Yukarıda gösterilen özel karakterler çift tırnak içerisindeki metin ile birlikte kullanılmaktadır. Örnek kullanım aşağıda gösterilmiştir:

```
string ornekString = " Str\\in\"g \tkarsilastirma.\n ";
cout << ornekString;
```



Bireysel Etkinlik

- Kullanıcıdan konsol aracılığı ile 3 adet string ifade girmesi istenecektir. Birincisi 10-15 kelimeden oluşan bir cümle, ikincisi aranacak kelime ve üçüncüsü ise aranacak kelime tespit edildikten sonra yerine yazılması istenen bir string ifadedir. Bu cümlede aranacak kelime find komutu ile tespit edildikten sonra, bu kelime replace komutu sayesinde yenisiyle değiştirilecektir. Bu problemi çözecek bir C++ kodu yazınız.

Bireysel Etkinlik

- Kullanıcıdan konsol aracılığıyla okunacak bir string ifadeyi, noktalı virgül (;) karakterine göre kelimelere ayıran ve bu kelimeleri tek tek ekrana yazdıran C++ programı yazınız. Bu işlemler için at ve substr üye fonksiyonlarını kullanınız. at üye fonksiyonunu noktalı virgül karakterinin bulunduğu konumları tespit etmek, substr üye fonksiyonunu ise konumları tespit edilmiş string ifadeleri parçalayıp ekrana yazdırmak için kullanınız.

Özet

- String sınıfı, karakter dizileri olarak bildiğimiz veri türüdür. C++ dilinde string sınıfı temel veri türlerinden birisi değildir. Karakter (char) veri tipinden tek boyutlu diziler olarak ifade edilmektedir. String nesneleri, diğer veri tiplerine benzer şekilde kullanılabilmektedir. String nesnelerine atama yapılabilir, string değişkenleri birleştirilebilir veya parçalanabilir, string değişkenleri arasında karşılaştırma yapılabilir. String sınıfı metin tipindeki bilgilerin saklanması, kullanılması, değiştirilmesi, kıyaslanması, parçalara ayrılması gibi birçok özellik ve üye fonksiyonu içermektedir.
- String sınıfı, metin tipindeki verilerde yapılacak işlemleri kolaylaştırması sebebiyle büyük öneme sahiptir. Bu işlemler ana başlıklar olarak tanımlama, bellek fonksiyonları, erişim ve metin düzenleme işlemleri olarak ifade edilebilir. Bir string sınıfı değişkeni diğer veri tiplerinde olduğu gibi, önce değişken tipi, sonrasında değişken adı kullanılarak tanımlanır.
- String sınıfının kullanılabilmesi için öncelikle programımızda string sınıfı kütüphanesinin tanımlanması gerekmektedir. String kütüphanesi #include <string> şeklinde tanımlanır. Kütüphaneyi tanımladıktan sonra programımızda string sınıfı değişkenlerini kullanabiliriz. String sınıfında nesne, diğer veri tiplerindeki değişkenlerle aynı şekilde oluşturulmaktadır. Yalnızca veri tipi yerine string ifadesi yazılması yeterlidir. Oluşturduğumuz string sınıfı değişkene, çift tırnaklar arasındaki başta rakamlar ve harfler olmak üzere noktalama işaretleri ve matematiksel işaretler gibi bütün karakterleri atayabiliriz.
- String sınıfının üye fonksiyonlarına erişebilmek için tanımladığımız değişkenin adını yazıp devamına "." koymamız gerekmektedir.
- String değişkenlere değer atama konsol ekranından veya dosyadan veri okuyarak yapılabileceği gibi, başka string ifadelerin parçaları da tanımladığımız string ifademize değer olarak atanabilir.
- String sınıfının bazı üye fonksiyonları değişkene ait bellek bilgilerinin görüntülenmesini veya düzenlenmesini sağlamaktadır. size ve length üye fonksiyonları string sınıfı değişkenimizin içerdiği karakter sayısını vermektedir. resize üye fonksiyonu değişkenimizin içerdiği karakter sayısının düzenlenmesini sağlamaktadır. Bu üye fonksiyonunu kullanırsak değişkenimizin içerdiği karakterler yeniden belirlenen boyuta göre düzenlenmektedir ve limit dışında olanlar silinmektedir. capacity üye fonksiyonu, string sınıfı değişkenimiz için bellekte ayrılan kapasite bilgisini vermektedir. Kapasite bilgisi boyut bilgisinden farklıdır. Boyut bilgisi değişkenin içerdiği karakter sayısını verirken, kapasite bilgisi 16'nın katlarının bir eksiği olmak üzere bellekte ayrılan yerdir. reserve üye fonksiyonu string sınıfı değişken için bellekte kapasite belirlenmesini sağlamaktadır. Bu üye fonksiyon genelde okunacak karakter sayısının belli olduğu durumlarda önceden yeteri kadar kapasite ayrılması için kullanılmaktadır. clear üye fonksiyonu, string sınıfı değişkenimizin içeriğini tamamen boşaltmakta ve saklanan karakter sayısını sıfırlamaktadır. empty üye fonksiyonu ise string sınıfı değişkenin boş olup olmadığının kontrol edilmesi amacıyla kullanılmaktadır. Eğer değişken karakter içermiyorsa geriye 1 değerini, en az 1 karakter içeriyorsa geriye 0 değerini döndürmektedir.

Özet(devamı)

- String sınıfı değişkenlerin içerdikleri karakterlere tek tek de ulaşmak mümkündür. Bu aynı diğer veri tiplerindeki dizilerin her bir elemanına erişmek gibi köşeli parantez kullanarak mümkün olmaktadır. Bu şekilde 0 indisinden başlamak üzere string sınıfı değişkenin her bir karakteri tek tek kontrol edilebilmektedir. Bu işlem için ayrıca at (et) üye fonksiyonu da kullanılabilmektedir. Bu üye fonksiyon da aynen dizi indislerine erişmek gibi bir adet 0 ve büyük pozitif değer alarak geriye string sınıfı değişkenin içerdiği karakter bilgisini döndürmektedir.
- Metin düzenleme fonksiyonları, string sınıfı değişkenler için en çok ihtiyaç duyduğumuz üye fonksiyonlarıdır. Bu düzenleme işlemleri kısaca string ifadelerin birleştirilmesi, silinmesi, eklenmesi veya iki değişkenin değerlerinin yer değiştirilmesi olarak açıklanabilir. String sınıfı değişkenin ard arda birleştirilmesi için + (artı) veya += (artı eşittir) operatörü kullanılabilir. Aynı şekilde bu işlevi yerine getirebileceğimiz append üye fonksiyonu da bulunmaktadır. Bu 3 işlemde de iki tane string ifade birleştirilmektedir. push_back üye fonksiyonu ise bu işlemlerden farklı olarak string sınıfı değişkenin sonuna 1 karakter eklemek amacıyla kullanılmaktadır. = (eşittir) operatörü ile yapılan atama işlemi, assign üye fonksiyonu ile de yapılabilmektedir. Daha önce atama yapılmış string sınıfı değişkene yeni bir değer atama işlemi yapılırsa, önceki değer tamamen silinmektedir. Ayrıca insert üye fonksiyonu ile string sınıfı değişkenin istediğimiz bir konumuna başka bir string ifade eklememiz mümkündür. erase üye fonksiyonu ile string sınıfı değişkenimizin istediğimiz parçasını silmemiz mümkündür. replace üye fonksiyonumuz ise erase ve insert üye fonksiyonumuzun işlevlerinin birleşmiş halidir. İstediğimiz bir string ifadeyi başka bir string değişkenin istediğimiz bölgesine yazmak amacıyla kullanılabilir. swap üye fonksiyonumuz iki adet string sınıfı değişkenin değerlerinin değiştirilmesi amacıyla kullanılmaktadır. substr üye fonksiyonu ise bir string ifadenin bir parçasının başka bir string değişkene kopyalanması amacına hizmet etmektedir.
- Yukarıdaki ana başlıklarda tanımlanan üye fonksiyonlar ve yöntemler dışında string ifadenin dönüşümü, arama ve karşılaştırma amacıyla kullanılan üye fonksiyonlar ve özel karakter kullanımı da önemli bir yere sahiptir. c_str ve data üye fonksiyonları string ifademizin c string bir ifadeye dönüştürülmesini sağlamaktadır. String ifade ile c string ifade arasındaki fark sadece bellekte c string ifadenin sonunda bir (null) boşluk karakterinin bulunmasıdır. String sınıfı değişkenlerde find komutunu kullanarak arama yapabiliriz. Bu işlemin sonunda aranan metin bulunur ise bu alt metnin başlangıç indisinin bir eksiği geriye değer olarak döndürülmektedir. compare üye fonksiyonu ile iki string ifadenin karşılaştırmasını yapmamız mümkündür. Bu üye fonksiyon klasik karşılaştırma fonksiyonlarından farklı olarak, eğer karşılaştırma işlemi olumlu ise geriye 0 değerini, diğer durumlarda ise karşılaştırılan ifadelerin eksik veya fazla karaktere sahip olmaları durumuna göre -1 veya +1 değerini geriye döndürmektedir. Bunun dışında derleyici için ayrılmış bazı özel karakterlerin string ifadenin içerisinde kullanılabilmesi amacıyla \ (ters slash) karakterinin kaçış karakteri olarak kullanılması gerektiğini bilmek gerekmektedir. Bu karakter ile birlikte n karakteri bir satır alta geçmeyi, t karakteri bir sekme sağa kaymayı, noktalama işaretlerinde ise kullanılan noktalama işaretini yazdırmayı mümkün kılmaktadır.

DEĞERLENDİRME SORULARI

1. String sınıfı nesnelerini kullanabilmek için tanımlanması gereken kütüphane aşağıdakilerin hangisidir?
   a) #include <iostream>
   b) #include <cstring>
   c) #include <string>
   d) string ornekString;
   e) using namespace std;

2. String sınıfının kullanılma amacı aşağıdakilerden hangisinde verilmiştir?
   a) Tam sayılar üzerinde daha kolay işlem yapabilmek
   b) Ondalıklı sayılar üzerinde daha kolay işlem yapabilmek
   c) İleri matematiksel işlemleri daha kolay yapabilmek
   d) Görsel programlamayı daha kolay yapabilmek
   e) Metinler üzerinde daha kolay işlem yapabilmek

3. String sınıfı değişken tanımı aşağıdakilerden hangisinde hatalı yapılmıştır?
   a) String deneme="deneme string ifadesi.";
   b) string kolaySoru123="cevap bu değil\t\n\t\n";
   c) string zor_soru="1. Metin" "2. Metin";//BU DA HATALI DEĞİL Mİ?
   d) string ornekString("bu bir örnek string ifadesidir.");
   e) string hatali_string="abc" + "def";

4. String sınıfı değişkeni c string ifadeye dönüştüren üye fonksiyonu aşağıdakilerin hangisidir?
   a) data
   b) erase
   c) length
   d) reserve
   e) capacity

5. String sınıfı değişkende arama yapmayı sağlayan find üye fonksiyonu aşağıdakilerin hangisinde hatalı kullanılmıştır?
   a) ornekString.find("abc");
   b) ornekString.find(0,1,"abc");
   c) ornekString.find(2,5,"abc",0,2);
   d) ornekString.find(1,2,3,"abc",1,2,3);
   e) ornekString.find(3,"abc",1);

6. "Örnek String." Yandaki string ifadenin boyut bilgisi kaçtır?
   a) 9
   b) 10
   c) 11
   d) 12
   e) 13

7. "Örnek String." Yandaki string ifadenin kapasite bilgisi kaçtır?
   a) 13
   b) 14
   c) 15
   d) 16
   e) 17

8. String ifade yazdırılırken satır atlatmak için hangi özel karakterler kullanılır?
   a) \y
   b) \s
   c) \l
   d) \m
   e) \n

9. "Örnek String." Yandaki string ifade için find("Str") üye fonksiyonunun çıktısı aşağıdakilerden hangisidir?
   a) 0
   b) 2
   c) 4
   d) 6
   e) -1

10. "Örnek String." Yandaki string ifade için at(3) üye fonksiyonunun çıktısı aşağıdakilerden hangisidir?
    a) Ö
    b) r
    c) n
    d) e
    e) k

Cevap Anahtarı

1.c, 2.e, 3.a, 4.a, 5.d, 6.e, 7.c, 8.e, 9.d, 10.d

YARARLANILAN KAYNAKLAR

[1] http://www.cplusplus.com/ adresinden 4 Temmuz 2018 tarihinde erişildi.

[2] http://alikeskin.org adresinden 4 Temmuz 2018 tarihinde erişildi.

[3] https://stackoverflow.com adresinden 4 Temmuz 2018 tarihinde erişildi.

[4] Yrd. Doç. Dr. ESEN YILDIRIM C++ Programlama Ders Notu, 4 Temmuz 2018 tarihinde http://hpss.endustri.cu.edu.tr/ders/ENS255/383_dosya_1341385774.pdf adresinden erişildi.

ŞABLONLAR VE STANDART ŞABLON KÜTÜPHANESİ

İÇİNDEKİLER

- Şablonlar
  - Fonksiyon Şablonları
  - Sınıf Şablonları
- Standart Şablon Kütüphanesi
  - Vector
  - List
  - Map
  - Stack

HEDEFLER

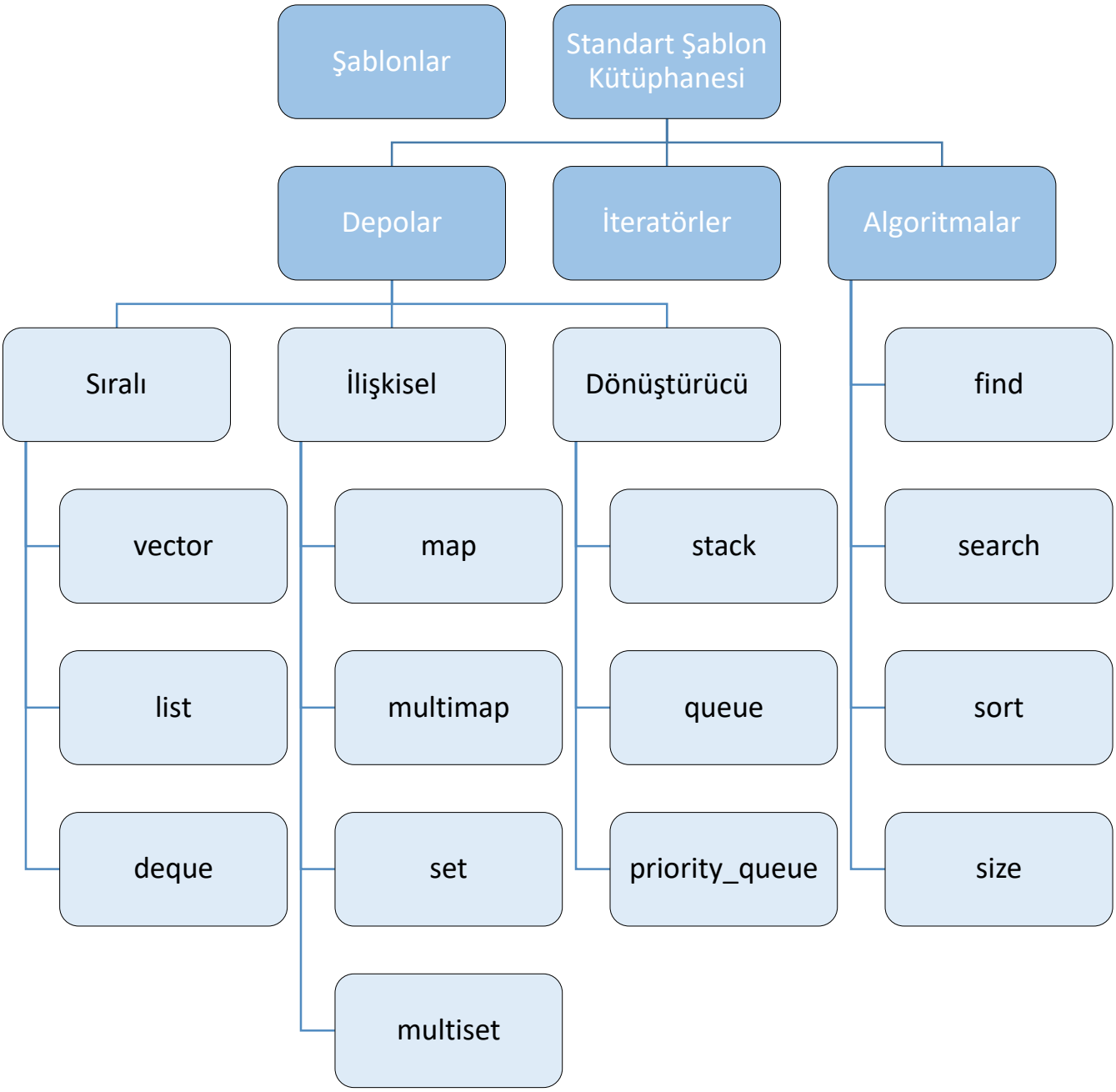
- Bu üniteyi çalıştıktan sonra;
  - Fonksiyon şablonu tanımlayabilecek,
  - C++ dilindeki standart şablon kütüphanesindeki depo, iterator ve algoritma kavramlarını bilebilecek,
  - vector, list, deque sıralı depolarını; map, multimap, set, multiset ilişkisel depolarını ve stack, queue ve priority_queue dönüştürücü depolarını programlarınızda kullanabileceksiniz.

Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

Doç. Dr.
Recep ERYİĞİT

ÜNİTE
12


Şablonlar
Standart Şablon Kütüphanesi
Depolar
İteratörler
Algoritmalar
Sıralı
İlişkisel
Dönüştürücü
find
vector
map
stack
search
list
multimap
queue
sort
deque
set
priority_queue
size
multiset

GİRİŞ

Programlama dillerinin önemli kavramlarından birisi, kod tekrarının en aza indirilmesidir. Kod tekrarını azaltma, fonksiyon, kalıtım ve şablon teknikleri kullanılarak gerçekleştirilebilir. Bu bölümde kod tekrarını azaltma yöntemlerinden biri olan şablon tekniği incelenecektir.

Şablonlar hangi veri tipinin işleneceğini bilmeden sınıf ve fonksiyonların tanımlanmasına izin vererek fonksiyon ve sınıfları soyut hale getirmenin bir yoludur. Bu işlem genel (jenerik) programlama olarak da adlandırılır. Genel programlamada tek bir veri tipine ait işlemlerden ziyade yapılmak istenilen işe odaklanılır. Şablonlar, herhangi bir veri türünü işlemelerine olanak sağlamak için soyut veri tipleri ile birlikte kullanılabilir. Örneğin, verileri bir listeye ekleyip liste üzerinde işlem yapmak istediğinizde farklı veri tipleri için liste programları yazmanız gerekir. Şablon yönteminde liste sınıfı soyut veri tipleri ile tanımlanarak farklı veri tiplerinde çalışabilecek tek bir liste sınıfı hazırlanır. Birkaç farklı veri tipini işleyebilen tek bir fonksiyon veya sınıf kodu yazılacak toplam satır sayısının azaldığı anlamına gelir.

C++ dilinde sınıf ve fonksiyon için şablon kod yazılabilir. Şablon sınıf ve fonksiyonlarda, veri tipleri parametrik tanımlanarak farklı veri tipleri için çağrılmaları sağlanır. Şablon fonksiyon ve sınıf çağrılarında, derleyici çağrı parametrelerinin veri tiplerine uygun fonksiyon ve sınıfları otomatik olarak oluşturur.

Sınıf ve fonksiyonlar için şablon yazılabilir.

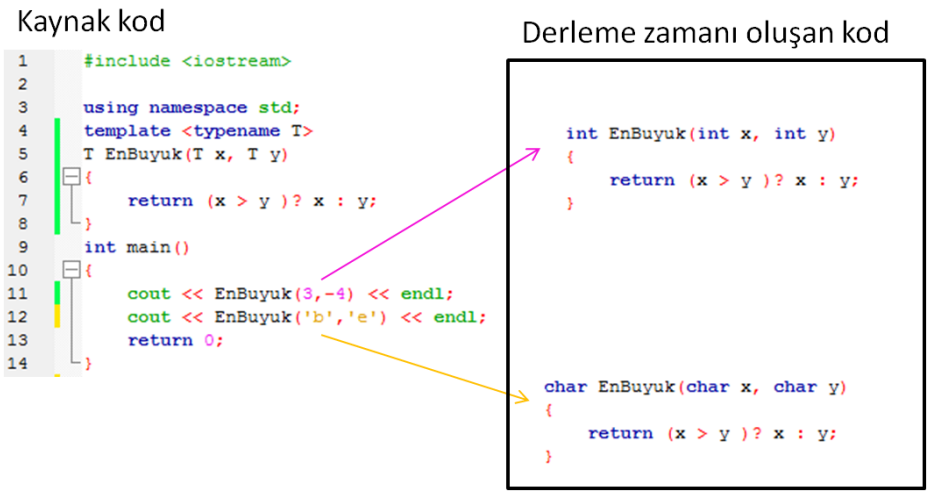
Bu bölümde; şablon kavramı, fonksiyon şablonu oluşturma ve kullanma, C++ dilinin standart şablon kütüphanesinde bulunan depo yapıları, iteratörler ve algoritmalar incelenecektir. Depolar belirli bir türdeki nesneleri bir arada tutan ve yöneten saklayıcı nesnelerdir. Bu bölüm kapsamında ayrıca sıralı ve ilişkisel depolar ile depo dönüştürücülerine ait örnek programlar geliştirilecektir.

ŞABLONLAR

Şablonlar kod tekrarını azaltan basit fakat önemli bir yazılım tekniğidir. Basitliği, veri tipini fonksiyon ve sınıflara parametre olarak iletme fikridir. Bu sayede farklı veri tipleri için aynı kodun yazılması gerekmez. Örneğin, bir yazılımda farklı veri tipleri için sıralama fonksiyonuna ihtiyaç varsa değişik veri tipleri için fonksiyonun yeniden yazılmasını engeller. Her bir veri tipi için ayrı bir kod yazıp saklamak yerine, şablon tekniğinde şablon fonksiyonu yazılarak veri tipi fonksiyona parametre olarak aktarılır.

Şablonlar Nasıl Çalışır?

Şablon kod, fonksiyon çağrılarında kullanılan veri tiplerine uygun fonksiyon kodlarının derleyici tarafından otomatik oluşturulması şeklinde çalışır. Kaynak kodda yalnızca şablon fonksiyon kodu bulunurken, derlenmiş kodda çağrılarda kullanılan farklı veri tipi sayısı kadar fonksiyon kodu vardır.



Şekil 12.1 Şablon fonksiyon örneği

Şekil 12.1'de verilen kod üzerinden şablon oluşturma ve kullanmayı açıklayalım. Şablon oluşturma 4 numaralı satırdaki *template <typename T>* satırı ile başlamaktadır. Şablonun İngilizce karşılığı olan *template*, şablon kodun başlangıcını ifade etmektedir. *<…>* içerisinde şablon parametreleri tanımlanmaktadır. Şablon parametrelerinden *typename* sözcüğü *T* soyut veri tipi değişkeninin tipini ifade etmektedir. Şablon oluşturmada *typename* yerine *class* kelimesi de kullanılabilir. Şablonlarda bu iki kelime arasında fark yoktur. Veri tipi değişkeninin ismi olarak *T* keyfi seçilmiş olup bu amaçla herhangi bir başka isim de kullanılabilir.

Fonksiyonlar, algoritmaların programlama dillerindeki karşılıklarıdır.

Şablon yapıları birden fazla parametre ile *template <typename T, typename U>* şeklinde tanımlanabilir. Şablon parametre sayısının bir üst sınırı yoktur. Parametre sayısını, şablonu geliştirilecek fonksiyon ve sınıfların gereksinim duydukları farklı tipteki veri sayısı belirler.

Şekil 12.1'de 5. satırla 8. satır arasında şablon fonksiyonu tanımlanmıştır. 5. satırdaki EnBuyuk fonksiyonunun girdi parametreleri ve geri döndürdüğü veri tipi şablon parametresi T olarak verilmiştir.

11. satırda, EnBuyuk(3,-4) çağrısı şablon fonksiyona iki tam sayı geçirmektedir. Derleyici bu çağrıya cevap verebilmek için şablon fonksiyonun tam sayı karşılığı fonksiyonu oluşturmaktadır.

12. satırda ise şablona karakter tipi değişkenler geçirilmektedir. Bu durumda da derleyici şablondan karakter tipinde parametrelere sahip bir fonksiyon oluşturacaktır.

Bu örnek programdan görüldüğü üzere şablon kullanılan programlarda, farklı veri tiplerindeki her çağrıya karşılık derleyici, şablon koddan yeni bir fonksiyon kodu oluşturmaktadır.

Şablon oluşturulurken dikkat edilmesi gereken önemli noktalardan birisi, farklı veri tipleri için fonksiyon içerisinde tanımlanan işlemlerin fonksiyon çağrısında kullanılan veri tipleri için anlamlı ve yapılabilir olmasıdır.

C++'da fonksiyon ve sınıflar için şablon oluşturulabilir. Şekilsel olarak birbirlerine benzeseler de anlamsal olarak iki şablon tipi son derece farklıdır.

Fonksiyon Şablonları

Programlama dillerinde fonksiyonlar belirli bir işlevi yerine getiren programlama birimleridir. Bu anlamda fonksiyonlar, algoritmaların programlama dillerinde yazılmış halleri olarak düşünülebilir. C++'da geri dönüşü olmayan (void) ve tek tipte veri döndürebilen fonksiyonlar yazılabilir. Fonksiyonların girdi parametreleri üzerinde bir sınırlama olmayıp işlevin gerektirdiği kadar farklı tipte girdi parametresi tanımlanabilir.

C++ dilinin tip bildirim zorunlu bir dil olması nedeniyle, fonksiyon girdi parametrelerinin ve geri dönüş tipinin kaynak kodda verilmesi gerekmektedir. Yani fonksiyon belirli bir veri tipi için geliştirilmektedir. Fonksiyon çağrılarının da kaynak koddaki veri tipleri dikkate alınarak yapılması gerekmektedir.

Örnek

- C++ programlama dilinde int, double ve string veri tiplerinde değerleri ekrana yazacak void yazdir(veriTipi değişken) fonksiyonunu yazınız.

C++ dilinde sınıf ve fonksiyonlar için şablon geliştirilebilir.

Çözüm:

C++ programlama dilinde bu problemin iki farklı kavrama dayalı olarak çözümü yapılabilir. Birinci çözüm, fonksiyonun aşırı yüklenmesidir. Aşırı yükleme, değişik veri tipleri için fonksiyonun veri tipine uygun olarak yeniden yazılmasıdır. Fonksiyon aşırı yüklemede;

`int` değişkenler için,

```
void Yazdir(int x)
{
    cout <<"değer :"<<x<<endl;
}
```

`double` değişkenler için,

```
void Yazdir(double  x)
{
    cout <<"değer :"<<x<<endl;
}
```

`string` tipindeki değişkenler için,

```
void Yazdir(string x)
{
    cout <<"değer :"<<x<<endl;
}
```

fonksiyonlarının hepsinin yazılması gerekir. Kaynak kodda soruda verilen bütün veri tiplerine ait fonksiyonlar olduğu için tam sayılar Yazdir(4), kesirli sayılar

Yazdir(4.5) ve kelimeler Yazdir("test") çağrıları, kaynak kodda kendi veri tiplerine ait fonksiyonları çalıştıracaklardır. Eğer fonksiyon *char* tipindeki bir değişkenle çağrılsaydı program *char* tipinde girdiye sahip fonksiyon içermediğinden hata üretilecekti.

Diğer bir çözüm ise fonksiyon şablonu kullanılmasıdır. Şablon fonksiyonu;

```
template <typename T>
void Yazdir(T x ){
        cout <<"değer :"<<x<<endl;
}
```

şeklinde verilir.

Şablon kullanıldığı durumda kaynak kodda yalnızca şablon fonksiyonu yazılır. Derleyici, fonksiyon çağrılarına uygun olarak farklı veri tiplerindeki fonksiyonları kendisi oluşturur.

Örnek

- C++ dilinde aynı tipteki iki sayının toplamını geri döndürecek fonksiyon şablonunu yazınız.

Şablon yöntemi ile fonksiyonlara birden fazla veri tipi geçirilebilir.

Çözüm:

Geliştirilecek fonksiyonun tam sayı ve kesirli sayı çiftleri şeklinde verilmiş olan sayıları toplayabilmesi gerekir. Şablon tekniği kullanılmadığı durumda tam sayılar ve kesirli sayılar için veri tipleri farklı iki toplama fonksiyonunun yazılması gereklidir.

Aynı tipteki iki sayının toplamını yapan şablon fonksiyonu C++ dilinde;

```
template <typename T>
T Toplam(T x, T y)
{
        T topla = x + y;
        return topla;
}
```

kodu ile verilir. Toplam fonksiyon şablonu tam sayılar için Toplam(3,4), kesirli sayılar için ise Toplam(-1.3,4.2) şeklinde kullanılacaktır. Derleyici her iki çağrı içinde birer fonksiyon oluşturacaktır.

Örnek

- Farklı veri tiplerine sahip iki sayıyı toplayıp sonucu ilk verilen sayının veri tipinde geri döndürecek fonksiyon şablonunu yazınız.

Çözüm:

Toplanması istenilen sayı çiftlerinin farklı veri tiplerinde olabileceği durumda iki farklı şablon parametresi tanımlamak gerekir. Şablon veri tipi parametrelerinin adlarını keyfi olarak T ve U şeklinde seçersek şablon kodu aşağıdaki gibi olur:

Şablon kullanılarak kaynak kod satır sayısı azaltılır.

```
template <typename T, typename U>
T Toplam(T x, U y)
{
        T topla = x + y;
        return topla;
}
```

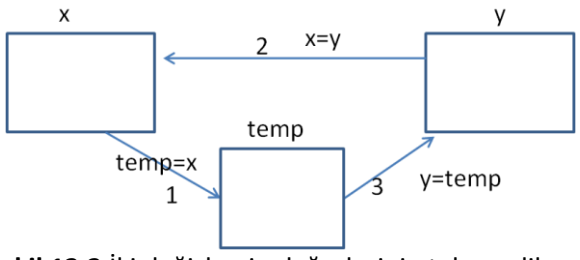
Farklı veri tiplerine sahip iki sayının toplamını bulacak fonksiyonun şablon kodu yukarıda verilmiştir. Fonksiyonun geri dönüş veri tipi birinci şablonun parametresi dikkate alınarak belirlenmiştir. Bunun sonucu olarak, Toplam(3.5, 4) işleminin sonucu birinci girdi parametresinin tipinde 7.5 olacaktır. Toplam(4, 3.5) durumunda ise, fonksiyon çağrısında ilk eleman tam sayı olduğu için, fonksiyon geriye 7 değerini döndürecektir.

Örnek

- C++ programlama dilinde aynı veri tipindeki iki değişkenin değerlerini değiş tokuş edecek şablon fonsiyonunu yazınız.

Çözüm:

Programlama dillerinde iki değişkenin değerlerinin değiş tokuşu, takas işlemi olarak adlandırılır. Farklı yaklaşımlarla problem çözülebilirse de çoğunlukla bir geçici değişken tanımlanıp değerlerden birisinin tanımlanan geçici değişkene aktarılmasıyla problem çözüme kavuşturulur.



Şekil 12.2 İki değişkenin değerlerinin takas edilmesi

Takas işlemi Şekil 12.2 ile görselleştirilmiştir. Görseldeki işlemde veri tipi bildirimleri ve değer atamaları yapılmış x ve y değişkenlerinin değerlerinin değişim aşamaları görüntülenmiştir. İşlem adımlarının sırası 1, 2 ve 3 sayıları ile verilmiştir.

Şekil 12.2'ye uygun şablon kod;

```
template <typename T>
```

```
void Takas(T &x, T &y)
{
    T temp;
    temp = x;
    x = y;
    y = temp;
}
```

ile verilir. Koddaki & işareti değişkenin adresinin fonksiyona gönderileceğini ifade etmektedir. Fonksiyon gövdesinde ise aynı tipteki iki değişkenin değerlerinin değiştirilmesi işlemi 3 adımda gerçekleştirilmektedir.

Standart şablon kütüphanesi, depo sınıflarını barındıran bir kütüphanedir.

Bireysel Etkinlik

- İki farklı veri tipindeki sayının çarpımını yaparak birinci verilen sayının veri tipinde geri döndürecek fonksiyonun şablon kodunu yazınız.

Sınıf Şablonları

C++ dilinde sınıf, değişkenleri ve fonksiyonları bir arada tutan bir programlama yapısıdır. Fonksiyon çoğunlukla bir işlevin gerçekleştirilmesi için yazılırken, sınıflar bir probleme ait birden çok durumun değişkenlerini ve bu durumların her birine karşılık gelen fonksiyonları içermektedir.

Sınıf için şablon kod yazmak söz dizimi olarak fonksiyon şablon yapısına benzemekle beraber bu ünitenin konusu değildir.

Bu ünitenin temel amaçlarından birisi, C++ derleyicilerinin hepsinde bulunan standart şablon kütüphanesinin kullanımının öğretilmesidir.

STANDART ŞABLON KÜTÜPHANESİ

Standart Şablon Kütüphanesi, yaygın olarak bilinen ismiyle STL (Standard Template Library), bütün C++ derleyicilerinde bulunan bir yazılım kütüphanesidir. Depo, algoritma ve iteratör kavramları standart şablon kütüphanesinin temel yapılarıdır.

Standart şablon kütüphanesinde, veriler farklı kurallara göre depolanabilir ve depolanmış olan veriler üzerinde arama, sıralama ve takas gibi işlemler yapılabilir.

Depolar

Depolar belirli bir türdeki nesneleri bir arada tutan ve yöneten saklayıcı nesnelerdir. Bilgisayar bilimlerinde depolar veri yapıları olarak da adlandırılırlar. STL kütüphanesindeki veri yapıları; sıralı, ilişkisel ve depolayıcılar için farklı ara yüzler sunan dönüştürücü depolar olarak üç alt gruba ayrılırlar.

Bir deponun programda başlatılabilmesi için depoya ait başlık dosyasının koda eklenmiş olması gerekir. Tablo 12.1'de C++ dilinde kullanılan depo yapıları ve kullandıkları başlık dosyaları verilmiştir. Başlık dosyasının programa eklenmesi #include <başlıkAdı> şeklinde yapılır. Bir depo değişkeni oluşturma depoAdı<veriTipi> değişkenAdı biçimindedir. veriTipi depoda tutulacak olan verilerin tiplerini belirtmektedir. Verileri bir anahtarla beraber tutan map ve multimap depolar depoAdı<veriTipi1,veriTipi2> değişkenAdı şeklinde başlatılır. veriTipi1 anahtarın, veriTipi2 ise depolanan verinin tipini belirtmektedir.

Sıralı, ilişkisel ve dönüştürücü tiplerinde depolar vardır.

Sıralı Depolar

Sıralı depolar aynı tipteki verileri yerleştirme zamanına ve yerleştirme konumuna göre sıralayan veri yapılarıdır. Sıralanan verinin konumu değerinden bağımsızdır. Sıralı depoların tanımlayıcı bir özelliği verinin bulunacağı sıranın kullanıcı tarafından belirlenmesidir. *vector*, *deque* ve *list* sıralı depolardır.

İlişkisel Depolar

İlişkisel depolarda veriler değerlerine ve deponun sıralama ölçütüne göre saklanır. İlişkisel depolar *set* ve *map* depolar olmak üzere iki grupta incelenebilir. *set* depolar verileri büyüklüklerine göre sıralayan yapılardır. Aynı değere sahip birden fazla veri bu yapılarda olmaz. Aynı değere sahip verilerin bulunabildiği *set* yapıları *multiset* olarak adlandırılır. *map* depolar verileri bir anahtarla ilişkilendirerek tutarlar. Bir anahtara bir değerin bağlandığı yapılar *map* yapısı, bir anahtara birden fazla değerin bağlandığı yapılar ise *multimap* yapısı şeklinde isimlendirilir.

Depo Dönüştürücüleri

Sıralı ve ilişkisel depolara ek olarak STL bünyesinde *stack*, *queue* ve *priority_queue* depo dönüştürücüleri vardır. Depo dönüştürücüler diğer tiplerdeki depoları sarmalayan yapılardır. Depo dönüştürücüler sıralı ve ilişkisel depolardan farklı olarak iteratörleri desteklemezler.

Tablo 12.1 C++ STL kütüphanesinin desteklediği depo yapıları

Depo Adı	Açıklama	Başlık Dosyası
vector	Veriler eklenme sıralarına göre depoda tutulurlar. Ekleme ve silme deponun sonundan olur.	<vector>
deque	Veriler eklenme sıralarına göre depoda tutulurlar. Ekleme ve silme deponun başından ve sonundan olabilir.	< deque>
list	Veriler eklenme sıralarına göre depoda tutulurlar. Deponun herhangi bir konumuna veri ekleme ve herhangi bir konumdan silme yapılabilir.	<list>
set	Verileri değerlerine göre sıralayarak tutar. Bir değer yalnızca bir kez eklenebilir.	<set>
multiset	Verileri değerlerine göre sıralayarak tutar. Depoda bir değerden birden fazla bulunabilir.	<multiset>

map	Verileri bir anahtarla beraber tutar ve anahtar değerine göre sıralama yapar. Bir anahtar için bir değer tutulur.	<map>
multimap	Verileri bir anahtarla beraber tutar ve anahtara göre sıralama yapar. Bir anahtar için birden çok değer tutabilir.	<multimap>
stack	Son giren ilk çıkar (LIFO, Last In First Out).	<stack>
queue	İlk giren ilk çıkar (FIFO, First In First Out).	<queue>
priority_ queue	En yüksek önceliğe sahip eleman ilk çıkar.	<priority_queue>

İteratörler

Algoritmalar, depo verilerine iteratörlerle erişirler.

İteratör, sıralı ve ilişkisel depolardaki verilere erişim için kullanılan nesnelerdir. İteratörler depoların bellek adreslerini işaret ederler. İteratör değişkeninin önüne * işareti konularak iteratörün işaret ettiği değere erişilebilir.

İteratörler, depolar ile algoritmalar arasındaki bağlantıyı sağlayan yapılardır. İteratörler sayesinde algoritmalar farklı depo yapıları üzerinde aynı işlemleri yapabilmektedir.

Örnek bir iteratör değişken bildirimi;

```
vector<int>::iterator it;
```

biçiminde yapılır. Sıralı ve ilişkisel depolarda bulunan elemanlardan ilk sıradakinin adresine iterator fonksiyonlarından begin(), son sıradaki elemandan bir sonrakinin adresine ise end() fonksiyonu ile ulaşılır. Fonksiyon çıktılarının iteratöre atanmaları nedeniyle, bunlar iteratör fonksiyonları olarak da adlandırılırlar.

Algoritmalar

Algoritmalar sıralı ve ilişkisel depolarda arama, sıralama işlemleri yapar.

Standart şablon kütüphanesi algoritmaları depo içeriğini başlatma, sıralama, arama ve dönüştürme işlemlerini gerçekleştiren fonksiyonlardır. C++ standart şablon algoritmaları depolarda tutulan verilere *iteratörler* üzerinden erişerek işlem yaparlar.

Algoritmalar, *iteratörleri* kullanarak farklı kurallara sahip depolarda arama, sıralama ve takas gibi işlemleri gerçekleştirirler.

Örnek

- vector depodaki elemanları iteratör kullanarak ekrana basacak bir C++ programı yazınız.

Çözüm:

vector depolarda veriler yerleştirme sırasına göre tutulurlar. Diğer sıralı ve ilişkisel depolarda olduğu gibi vector elemanlarına iteratör kullanılarak erişilebilir.

vector depoya yeni bir eleman ekleme *push_back* fonksiyonu ile eleman çıkarma ise *pop_back* fonksiyonu ile yapılır.

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

int main()
{
      vector<int> vek;
      for (int i=1; i<=5; i++)
         vek.push_back(i);

      vector<int>::iterator it;
      cout << "Vektör elemanları:" <<endl;
      for (it = vek.begin() ; it != vek.end(); ++it)
         cout  << *it<<endln;

      return 0;
}
```

Programda dikkat edilmesi gereken noktalardan birisi *it ifadesidir. İteratör değişkenleri depo içerisindeki elemanların adreslerini tutarlar. Bu adreslerdeki değerlere iteratör değişkeninin önüne * işareti konularak erişilir.

Bireysel Etkinlik

- Bir vector depoya 12, 3, 4, 5, 6 elemanlarını yerleştrip, sonra vector'den iki veri silecek C++ kodunu yazınız.

Örnek

- deque deposu oluşturaran ve depodan silme işlemi yapan C++ programını yazınız.

Çözüm:

deque depolar verileri yerleştirilme sırasına göre depolayan, öte yandan ekleme ve çıkarma işlemlerinin herhangi bir konumdan yapılabildiği depolardır. deque depolara veri ekleme baştan *push_front*, sondan *push_back* fonksiyonları ile yapılır. Baştan eleman silme *pop_front*, sondan silme ise *pop_back* fonksiyonuyla gerçekleştirilir. *erase* fonksiyonu ile deque depodan bir eleman veya belli bir aralıktaki eleman grubu silinebilir. *insert* fonksiyonu ile deque deponun herhangi bir konumuna veri eklenebilir. Bu durumda yerine veri yerleştirilen eleman bir sonraki sıraya kayar.

```
#include <iostream>
```

```
#include <deque>
using namespace std;
int main()
{
  deque<int> intQue;
  // deque eleman ekleme
  for (int i=1; i<=10; i++)
     intQue.push_back(i);
  // 7. elemanın silinmesi
   intQue.erase(intQue.begin()+6);
  // ilk 3 elemanın silinmesi
  intQue.erase(intQue.begin(),intQue.begin()+3);
   // 4.sıraya 70 sayını yerleştir.
  intQue.insert(intQue.begin()+3,70);
  //iteratör bildirimi
  deque<int>::iterator it;
  cout << "mydeque elemanları"<<endl;
  for(it=intQue.begin(); it!= intQue.end(); ++it)
    cout << *it <<endl;
  return 0;
}
```

Örnek

- map tipi depoya üç adet eleman ekleyen ve eklenen elemanları ekranda gösteren C++ programını yazınız.

map depolarda veriler bir anahtara bağlı olarak tutulur.

Çözüm:

map, verileri bir anahtarla beraber tutan ve anahtar değerine göre sıralayan depodur. İteratör kullanılarak map anahtar değerine iteratörDeğişkeni-> first ile, verilere ise *iteratörDeğişkeni-> second* ile erişilir.

```
#include <iostream>
#include <map>
using namespace std;
int main()
{
  map<char,int> charInt;
  charInt['b'] = 100;
  charInt['a'] = 200;
  charInt['c'] = 300;
  //iteratör bildirimi
  map<char,int>::iterator it;
  //Anahtar ve verilerin gösterilmesi
  for(it=charInt.begin();  it!= charInt.end(); ++it)
  {
      cout << it->first << " => " << it->second << endl;
```

```
  }
  return 0;
    }
```

Örnek

- set tipi depoya üç adet eleman ekleyen ve eklenen elemanları ekranda gösteren C++ programını yazınız.

Çözüm:

set depolar verileri büyüklüklerine göre sıralarlar. set depoya eleman ekleme *insert*, silme *erase*, arama ise *find* fonksiyonları ile yapılır.

Set depolarda veriler büyüklük sırasında tutulurlar.

```
#include <iostream>
#include <algorithm>//find
#include <set>
using namespace std;
int main()
{
  set<int> setl;
  set<int>::iterator it;
  // set'e değer ekleme
  for(int i=1; i<10; i++)
     setl.insert(i*10);// 10 20 30 40 50 60 70 80 90
  // ilk elamanı siler
  setl.erase(setl.begin());
  // 20'yi bulup adresini it'e aktar
  it= setl.find(20);
  //iteratör aracılığıyla silme
  setl.erase(it);
  // 40'ı bul ve sil
  setl.erase(setl.find(40));
  //60'dan sonrasını sil
  it = setl.find (60);
  setl.erase(it, setl.end());
  cout << "Set elemanları";
  for (it= setl.begin(); it!= setl.end(); ++it)
    cout << *it<<endl;
  return 0;
    }
```

Örnek

- İki farklı multiset tipi depoda tutulan verileri takas edecek C++ programını yazınız.

Çözüm:

multiset aynı değere sahip verinin birden fazla olabildiği ve sıralamanın büyüklüğe göre olduğu depolardır. Takas işlemi, iki değişkenin değerlerini karşılıklı olarak değiştikleri durumdur. Algoritmalarda takas işlemi için *swap* fonksiyonu kullanılır.

```
#include <iostream>
#include <algorithm>
#include <multiset>
using namespace std;
int main()
{
  int ar[]={19,72,4,36,20,20};
  multiset<int> bir(ar,ar+3);     // 19,72,4
  multiset<int> iki(ar+3,ar+6);   // 36,20,20

  bir.swap(iki);

  multiset<int>::iterator it;
  cout << "bir in elemanları:";
  for (it =bir.begin(); it!=bir.end(); ++it)
    cout << "  "<<*it; // 20 20 36
  cout <<endl;
  cout << "ikinin elemanları:";
  for (it=iki.begin(); it!=iki.end(); ++it)
    cout << "  "<<*it; // 4 19 72
  return 0;
    }
```

Bireysel Etkinlik

•Bir set depoda bulunan 1.2, 3.4, 5.6, 2.3, 3.2, 5.0,2.1 verilerinden 5.6'yı ve 2.3'ü silecek C++ programını yazınız.

map depoların anahtar değerleri first, büyüklük değerleri second olarak adlandırılır.

Örnek

• İki farklı map tipi depoda tutulan verileri takas edecek C++ programını yazınız.

Çözüm:

map'ler, verileri birer anahtarla beraber tutan ve anahtarın değerine göre sıralayan depolardır. multimap yapıda aynı anahtara birden fazla değer bağlanabilirken map'lerde anahtarla değer bire-bir eşlenmiştir. Aynı anahtara iki

değer atanırsa map bunlardan yalnızca sonra ekleneni dikkate alır. map'in birinci elemanına *first* ikinci elemanına *second* kelimeleri ile erişilir.

```
#include <iostream>
#include <map>
using namespace std;
int main()
{
  map<char,int> bir,iki;
  bir['x']=100;
  bir['y']=200;
  iki['a']=11;
  iki['b']=22;
  iki['c']=33;
  bir.swap(iki);
  map<char,int>::iterator it;
  std::cout << "bir in elemanları"<<endl;
  for (it= bir.begin(); it!= bir.end(); ++it)
    cout << it->first << " => " << it->second << '\n';
  cout << "iki nin elemanları"<<endl;
  for (it= iki.begin(); it!= iki.end(); ++it)
    cout<<it->first<<"=>" << it->second << '\n';
  return 0;
    }
```

Örnek

- Bir list depodaki ilk ve son sayının toplamını bulacak C++ programını yazınız.

Çözüm:

list depoların ilk elemanına front fonksiyonu, son elemanına ise back fonksiyonu ile erişilir.

```
#include <iostream>
#include <list>
using namespace std;
int main()
{
      list<int> liste;
       int toplam;
      liste.push_front(78);
      liste.push_back(16);
      toplam=liste.front()+liste.back();
      cout << toplam << endl;
      return 0;
}
```

Örnek

- stack veri yapısına veri ekleme ve çıkarma işlemi yapan C++ programını yazınız.

Çözüm:

Stack, ilk giren son çıkar veya son giren ilk çıkar mantığıyla tasarlanmış girdileri ve çıktıları kontrol amacıyla kullanılan dönüştürücü depo yapısıdır. Stack veri yapısının fonksiyonları aşağıda listelenmiştir:

pop	Yığın'ın en üst elemanını siler.
push	Yığın'ın en üstüne yeni eleman ekler.
top	Yığın'ın en üst elemanını döndürür.
size	Yığın'ın eleman sayısını döndürür.

```
#include <iostream>
#include <stack>
using namespace std;
int main()
{
  stack<int> stak;
  for(int i=0; i<5; ++i) //0 1 2 3 4
    stak.push(i);
  cout << "Eleman çıkarma ve yazma"<<endl;
  while(!mystack.empty())
  {
     cout << stak.top()<<" "; //4 3 2 1 0
     stak.pop();
  }
  cout << endl;
  return 0;
}
```

Özet

- Şablonlar kod tekrarını azaltan basit fakat önemli bir yazılım tekniğidir. Basitliği, veri tipini fonksiyon ve sınıflara parametre olarak iletme fikridir. Bu sayede farklı veri türleri için aynı kodun yazılması gerekmez. Örneğin, bir yazılımda farklı veri türleri için Sirala() fonksiyonuna ihtiyaç varsa her değişik veri tipi için sıralama fonksiyonunun yeniden yazılmasını engeller. Her bir veri tipi için kodu yazıp saklamak yerine, şablon tekniğinde bir Sirala() şablon fonksiyonu yazılarak veri tipi fonksiyona parametre olarak aktarılır.
- Şablon kullanan programlarda derleme zamanında fonksiyon çağrılarına uygun fonksiyonlar üretilir.
- Sınıf ve fonksiyonlar için şablon yazılabilir.
- Fonksiyon şablon bildirimi; template <typename T> T fonkAdi (typename T) {..} şeklinde verilir.
- Şablon fonksiyonun kullanımı fonkAdi(değer) veya fonkAdi<veriTipi>(değer) şeklinde olur.
- Standart şablon kütüphanesi yaygın olarak bilinen ismiyle STL, bütün C ++ derleyicilerinde bulunan bir yazılım kütüphanesidir. Depolar, algoritmalar ve iteratörler standart şablon kütüphanesinin temel kavramlarıdır.
- Depolar belirli bir türdeki nesneleri bir arada tutan ve yöneten saklayıcı nesnelerdir. Sıralı, ilişkisel ve dönüştürücü depolar olmak üzere üç gruba ayrılırlar.
- Sıralı depolar, aynı tipteki verileri yerleştirme zamanına ve yerleştirme konumuna göre sıralı tutan veri yapılarıdır. Sıralama verinin konumundan ve büyüklüğünden bağımsızdır. Sıralı depoların tanımlayıcı bir özelliği, verinin bulunacağı sıranın kullanıcı tarafından belirlenmesidir. vector, deque ve list sıralı depolardır.
- İlişkisel depolarda veriler büyüklüklerine ve deponun sıralama ölçütüne göre sıralanır. İlişkisel depolar set ve map depolar olmak üzere iki grupta incelenir. Aynı verinin depoda birden fazla bulunabildiği ilişkisel depo yapıları multiset ve multimap depolardır. set depolar verileri büyüklüklerine göre sıralar. map depolar ise verileri anahtar bir değerle beraber tutar.
- Sıralı ve ilişkisel depolara ek olarak C ++ dilinde stack, queue ve priority_ queue depo dönüştürücüleri vardır. Depo dönüştürücüler diğer tiplerdeki depoları sarmalayan yapılardır. Depo dönüştürücüler, sıralı ve ilişkisel depoların aksine iteratörleri desteklemezler.
- İteratörler, sıralı ve ilişkisel depolardaki verilere erişim için kullanılan nesnelerdir. İteratörler depoların bellek adreslerini işaret ederler. Bir iteratör değişkenin önüne * işareti konularak iteratörün işaret ettiği değere erişilir.
- Standart şablon kütüphanesi algoritmaları depo içeriğini başlatma, sıralama, arama ve dönüştürme işlemlerini gerçekleştiren fonksiyon kütüphanesidir. C++ Standart şablon algoritmaları depo elemanlarına iteratörler üzerinden ulaşırlar.

DEĞERLENDİRME SORULARI

1. Veri yapılarının hangisinde ilk giren son çıkar?
   a) vector
   b) list
   c) map
   d) queue
   e) stack

2. Veri yapılarının hangisinde anahtar ve değer birlikte tutulur?
   a) vector
   b) list
   c) map
   d) queue
   e) stack

3. stack veri yapısına aşağıdaki komutlardan hangisi ile yeni bir veri eklenir?
   a) push
   b) pop
   c) add
   d) push_back
   e) set

4. list veri yapısının başına aşağıdaki fonksiyonların hangisi ile yeni bir veri eklenir?
   a) add_front
   b) push
   c) push_front
   d) pop_font
   e) add_back

5. Depolardan hangisi ilişkisel değildir?
   a) map
   b) set
   c) multimap
   d) multiset
   e) deque

6. C++ dilinde kaç tip şablon vardır?
   a) 1
   b) 2
   c) 3
   d) 4
   e) Sayı limiti yoktur.

7. Depolardan hangisinde baştan ve sondan eleman silme fonksiyonları vardır?
   a) vector
   b) list
   c) deque
   d) stack
   e) map

8.
```
#include <iostream>
#include <vector>
using namespace std;
int main ()
{
   vector<int> vek;
   int toplam;
   vek.push_front(12);
   vek.push_back(10);
   toplam= vek.front() + vek.back();
   cout << toplam << endl;
   return 0;
}
```
   Yukarıda verilen programın çıktısı aşağıdakilerden hangisidir?
   a) 2
   b) 10
   c) 12
   d) 22
   e) 24

9. map veri tipinde aşağıdakilerden hangisi ile bir değişken oluşturulur?
   a) map<int,int>
   b) map(int,int)
   c) map[int,int]
   d) map
   e) mapcreate

10.
```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
  vector<int> vek(3);
  for(unsigned i = 0; i < vek.size(); i++)
    vek.at(i) = i;
  for(unsigned i = 0; i <vek.size(); i++)
    cout << " " << vek.at(i);
  return 0;
}
```

Yukarıda verilen programın çıktısı aşağıdakilerden hangisidir?

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

Cevap Anahtarı

1.e, 2.c, 3.a, 4.c, 5.e, 6.b, 7.c, 8.d, 9.a

YARARLANILAN KAYNAKLAR

[1] Vatansever, F. (2007). *Algoritma Geliştirme ve Programlamaya Giriş,* Seçkin Yayınları.

[2] Kirch-Prinz, U., Prinz, P. (2002). *A Complete Guide to Programming in C++,*Jones and Barlett Publishers

[3] http://www.cplusplus.com/reference/stl/

DOSYA İŞLEME

İÇİNDEKİLER

- Akış (Stream) Sınıfları ve Dosyalar
- Dosya Türleri
- Dosya Açma Modları
- Dosya İşleme
  - Dosya Oluşturma
  - Dosyaya Veri Yazma
  - Dosyadan Veri Okuma
- Dosya Okuma İşaretçileri

HEDEFLER

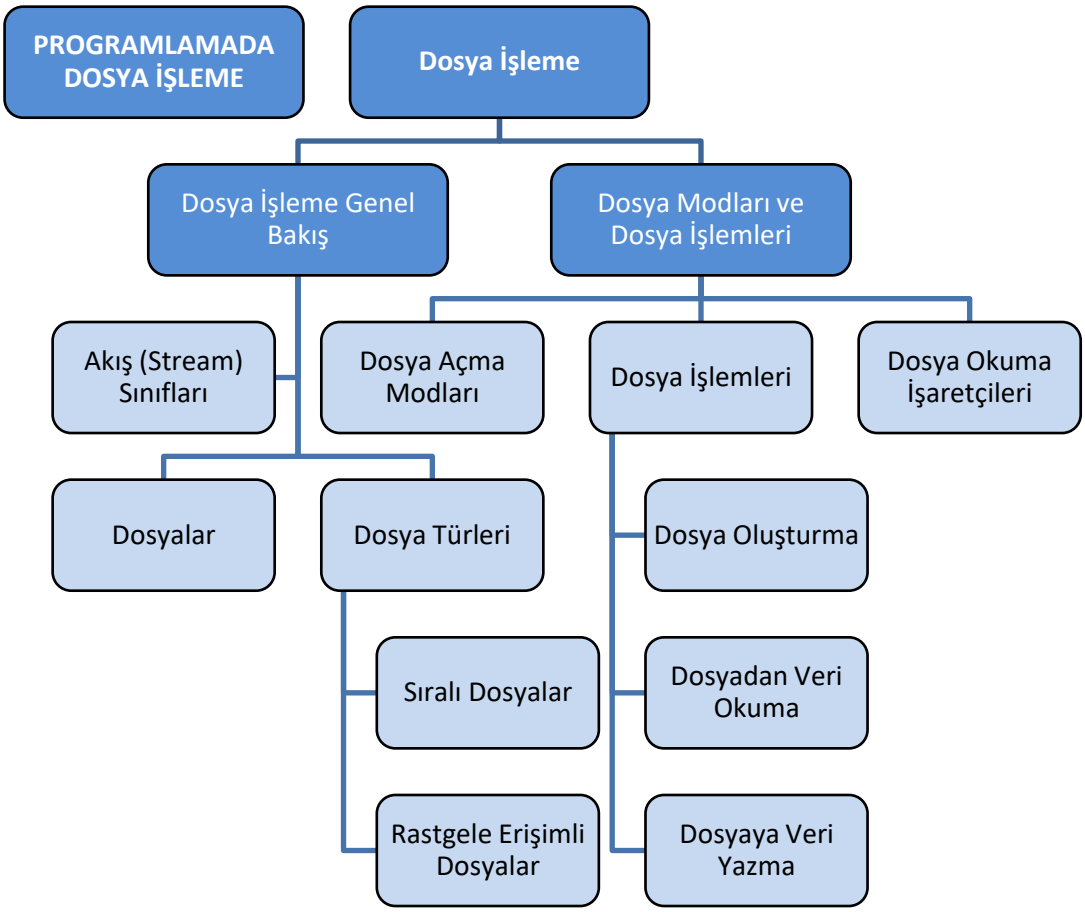
- Bu üniteyi çalıştıktan sonra;
  - Akış (Stream) sınıfları hakkında bilgi sahibi olabilecek,
  - Programlama dillerinde kullanılan dosya işlemlerini öğrenebilecek,
  - Dosya türlerini kavrayabilecek,
  - Sıralı erişimli ve rastgele erişimli dosyalar arasındaki farkı anlayabilecek,
  - Bir programlama dilinde dosya oluşturmayı öğrenebilecek,
  - Bir dosyadan veri okuyabilecek,
  - Bir dosyaya veri yazdırabilecek,
  - Dosya okuma işaretçilerini öğrenebileceksiniz.

Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

Arş. Gör. Mustafa Furkan KESKENLER

ÜNİTE
13


PROGRAMLAMADA DOSYA İŞLEME
Dosya İşleme
Dosya İşleme Genel Bakış
Dosya Modları ve Dosya İşlemleri
Akış (Stream) Sınıfları
Dosya Açma Modları
Dosya İşlemleri
Dosya Okuma İşaretçileri
Dosyalar
Dosya Türleri
Dosya Oluşturma
Sıralı Dosyalar
Dosyadan Veri Okuma
Rastgele Erişimli Dosyalar
Dosyaya Veri Yazma

GİRİŞ

Bir program yazılırken program içerisinde kullanılan değişkenler, diziler ve diğer veri yapıları verileri *geçici olarak* hafızada tutmaktadır. Bu şekilde hafızada tutulan veriler programın çalışması sonlandığında kaybolmaktadır. Bundan dolayı *dosyalar* büyük miktarda verileri kalıcı olarak hafızada tutmak için kullanılırlar.

Bilgisayarlarda veriler kalıcı hafızada yani ikincil depolama (hard disk, SSD, CD, Flash Disk vs.) cihazlarında saklanır. Gerektiği zaman bu veriler kayıtlı oldukları diskten okunarak üzerlerinde işlemler gerçekleştirilir. Disklere kayıt edilecek veriler dosyalar üzerinde depolanabilir. Bu dosyalar düz metin (.txt veya .dat uzantılı) veya veri tabanı dosyası olabilir. Aşağıda, bir program içerisinde kullanılacak verilerin saklanabileceği dosya işlemleri açıklanmıştır.

Dosya işlemleri;

- Yeni bir dosya oluşturma,
- Var olan dosyanın kullanılmak üzere açılması,
- Dosyaya veri yazılması ve/veya dosyadan veri okunması,
- Dosyanın sonlandırılması süreçlerini kapsamaktadır.

Bu bölümde detaylandırılan bir başka ifade olan *akış (stream),* veri akışına genel olarak verilen kavramsal bir terimdir. C++ programlama dilinde bir akış, belirli bir sınıfın nesnesi olarak sembolize edilir. Ekrana veri yazdırma ve kullanıcıdan klavye aracılığıyla veri alma işlemleri için genel olarak *cout* ve *cin* akış nesneleri kullanılmaktadır. Yani farklı türde veri akışlarını temsil etmek için farklı akışlardan faydalanılmaktadır. Örneğin; ifstream sınıfı, diskteki kaynak dosyalardan gelen veri akışını simgelemektedir.

Veri okuma ve yazma gibi işlemleri basitleştirmek amacı ile C++ programlama dili, akış (stream) kavramını kullanmaktadır.

Bu ünite içerisinde akışlar hakkında genel bir bilgi verilerek, verilerin saklanacağı dosyaların nasıl oluşturulduğu, verilerin bu dosyalara nasıl yazılıp okunduğu C++ programlama dili kullanılarak örnek uygulamalar ile açıklanacaktır.

AKIŞ (STREAM) SINIFLARI VE DOSYALAR

Bilgisayarda veriler donanım ve yazılım arasında paylaşılır. Bir program (yazılım) yeni bir veri oluşturur, verileri işler ve siler. Yazılımın üzerinde çalıştığı donanımı, CPU, kısa süreli ve uzun süreli bellek, klavye, ekran, yazıcı, modem gibi aygıtlar oluşturur. Bu donanım parçaları birbirinden farklı gibi görünse de çalışma prensibi bakımından benzerlikler göstermektedir. Örneğin; bir sabit diskten, mikrofondan veya klavyeden veri okunur, yine sabit diske, ekrana, hoparlöre ve yazıcı gibi donanımlara veri yazılabilir. Bu gibi ortak özelliklerin kullanımını basitleştirmek amacı ile C++ programlama dili akış (stream) terimini kullanmaktadır.

Akışlar *iki çeşittir.* Birincisi programın yazabileceği, ikincisi ise okuyabileceği akıştır. Örneğin; yazma olanağı tanıyan *cout, çıkış akışıdır* (ostream), okuma olanağı tanıyan *cin* ise *giriş akışıdır* (istream). Burada adı geçen ostream ve istream sınıflarının her ikisi de iostream sınıfına aittir. Bu ünitenin amacı olan dosyaya veri

yazma ve dosyadan veri okuma işlemlerini gerçekleştirmede kullanılacak olan iostream akış sınıfının alt sınıfları aşağıda açıklanmıştır:

- *ofstream*, dosya *çıkış* işlemlerinin gerçekleştirildiği aşamada kullanılan ve ostream sınıfından türemiş olan bir sınıftır. Dosyaya yazma işlemlerinde kullanılmaktadır.
- *ifstream*, dosya *giriş* işlemlerinin gerçekleştirildiği aşamada kullanılan ve istream sınıfından türemiş olan bir sınıftır. Dosyadan okuma yapma işleminde kullanılmaktadır.
- *fstream*, iostream sınıfından türemiştir. Dosya *giriş-çıkış* işlemlerini birlikte gerçekleştirir. fstream sınıfı hem ofstream hem de ifstream sınıflarını içerdiğinden her iki sınıfın kullanımına ihtiyaç duyulan programlarda yani hem dosyadan okuma hem de dosyaya yazma işlemlerinin yapılmak istendiği programlarda *#include <fstream>* biçiminde programa eklenmelidir.

Dosya, bir bilgisayar programında kullanılan verileri, bilgileri, ayarları veya komutları depolayan bir yapıdır.

Dosya, bir bilgisayar programında kullanılan verileri, bilgileri, ayarları veya komutları depolayan bir yapıdır. Herhangi bir dosya açıldığı zaman dosyaya yönelik bir akış ilişkilendirilir. Bir program çalışmaya başladığı anda üç dosya ve bu dosyalar ile alakalı akışlar görevlendirilir. Bunlar;

- Standart giriş dosyası,
- Standart hata dosyası,
- Standart çıkış dosyasıdır.

Bu dosyalar için görevlendirilen standart giriş akışı, örneğin klavyeden girilen verilerin program tarafından okunmasını sağlamaktadır. Benzer şekilde standart çıkış akışı ise programın ürettiği çıkış verisini ekrana yazdırmaya yaramaktadır. Böylece *akışlar,* dosyalar ve programlar arasında haberleşme kanallarını oluşturmaktadır.

Akışlar, dosyalar ve programlar arasında haberleşme kanallarını oluşturur.

Program yazarken genellikle düz metin dosyaları kullanılarak (örneğin .txt uzantılı) dosyadan okuma ve yazma işlemleri gerçekleştirilebilir.

DOSYA TÜRLERİ

Dosya, aynı türden verilerin belirli bir düzende (yazılış sırası) ilgili kayıt ortamına (disk, CD vs.) yazılmasıyla oluşturulan bir yapıdır. *Dosyalar*, kayıt edildikleri ortama göre, içeriğine erişim yöntemine göre, içerdikleri verinin yapısına göre farklı bakış açılarından sınıflara ayrılabilirler.

Genel olarak dosyalar, *geçici* (iç) dosyalar ve *kalıcı* (dış) dosyalar şeklinde ikiye ayrılabilir. Kalıcı dosyalar disk, bulut, DVD, USB disk biçiminde bir dış kayıt ortamına kaydedilirler. Böylece uzun süre kullanılmak üzere korunabilen ve her kullanılmak istendiğinde yeniden ulaşılan dosyalardır. Geçici dosyalar ise programın çalıştığı esnada sistem tarafından ana bellekte (RAM) oluşturulan ve program kapanınca ana bellekten silinerek yok olan dosyalardır. Programcı ve kullanıcı geçici dosyaların nasıl oluşturulduğu ve niçin kullanıldığı ile ilgilenmek zorunda değildir; ama kalıcı dosyalar hakkında yeterli bilgiye sahip olmalıdır.

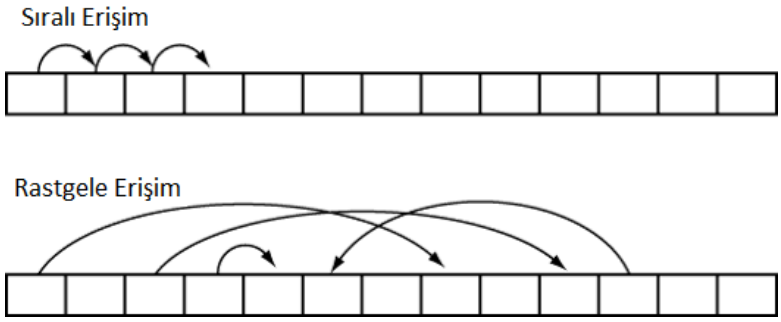
Diğer yandan, kalıcı ve geçici bütün dosyalar, *verilere erişim* yöntemine göre;

- Sıralı Erişimli (sequential) Dosyalar,
- Rastgele (random) Erişimli Dosyalar olmak üzere *ikiye* ayrılmaktadır.

Sıralı erişimli bir dosyada, verilerin bir teyp kasetindeki şerit üzerine kaydedildiği varsayılabilir. Veriler, teyp kasetinin şeridine sırasıyla yazılır ve sırasıyla okunur. Bu nedenle, dosya her açıldığında şeridin en başından başlanarak sırası ile okuma yapılması gerekmektedir.

Doğrudan erişimli yani rastgele erişimli dosyada ise, verilerin bir DVD ya da bir disk (örneğin, hard disk, CD) üzerine yazıldığı varsayılabilir. Okuma/yazma kafası (gramofonlardaki gibi), diskte tasarlanan iz ve sektör üzerinde okuma ve yazma işlemine başlayabilir. Teyp kasetindeki şeritte olduğu gibi, her defasında başa dönülmesi gerekmez (Şekil 13.1). Dönmekte olan disk üzerinde her verinin konumu yani adresi belirlidir. Bu şekilde rastgele erişimli dosyalarda sistem, okuma/yazma kafasını disk üzerinde istenilen yere konumlandırabilir. Böylece disk üzerinde istenilen tüm verilere *doğrudan* erişilebilir.

Dosyalar erişim türlerine göre, sıralı ve rastgele erişimli olmak üzere ikiye ayrılır.



Şekil 13.1. Dosyaya Sıralı Erişim ve Rastgele Erişim Farkı

Ayrıca saklama türüne göre dosyalar *ikili* (binary) ve *metin* (text) olarak ikiye ayrılmaktadır.

DOSYA AÇMA MODLARI

Dosya açılırken dosyanın bulunduğu dizinin, dosya ile nasıl bir işlem yapılacağı bilgisinin ve dosya tipinin belirtilmesi gerekmektedir. Dosya ile yapılabilecek işlemler Tablo 13.1'de gösterilmiştir. Bu işlemlere *dosya açma modları* ismi verilmektedir. Bu modlar stream sınıfların bir üyesi olan *ios* sınıfına aittir. Programın yazımı esnasında bu modlardan birisinin kullanılması durumunda işlemin bu sınıfa ait olduğunu gösterir operatör "::" kullanılmalıdır.

Tablo 13.1. Dosya Açma Modları ve Anlamları

Dosya Açma Modu	Görevi
ios::app	Eğer belirtilen konumda aynı isimde dosya var ise dosya içerisindeki değerler korunur ve çıktılar dosyanın sonundan eklenerek kaydedilmeye başlar. Eğer dosya yoksa oluşturulur ve ilk satırdan verileri kaydetmeye başlar.
ios::in	Giriş modudur. Dosyayı okumak için açar. Eğer belirtilen yerde dosya yoksa program hata verir.
ios::out	Çıkış modudur. Verileri dosyaya yazmak için dosyayı açar. Eğer belirtilen dizinde dosya yoksa oluşturulur, varsa içindeki veriler silinir ve ilk satırdan itibaren tekrar kayıt işlemi yapılır.
ios::ate	Dosya imlecini dosyanın başlangıcına değil, sonuna konumlandırır.
ios::trunc	Belirtilen dizinde aynı isimde dosya varsa o dosyayı siler ve tekrar dosya oluşturur.
ios::binary	Dosyalar binary ve text tipinde kullanılır. Bu mod belirtilmedikçe dosyada text (metin) tipinde (varsayılan tip) işlemler yapılır. Eğer bu mod belirtilirse ikili (binary) dosya formatında işlemler yapılır. Bu modda veriler metin modu yerine ikili formatta yazılır.
ios::nocreate	Eğer belirtilen dizinde dosya var ise yeniden oluşturulmasını engeller. Güncel derleyiciler artık bu modu desteklememektedir. Kullanımında hata alınabilir.
ios::noreplace	Eğer dosya var ise yeniden oluşturulmasını engellemektedir.

En sık kullanılan dosya açma modları; ios::app, ios::in, ios::out dır.

Eğer dosya emniyet seviyesi belirtilmemişse varsayılan olarak 0 değerini alır.

Tablo 13.1'de verilen modların kullanımına dair örnekler dosya açma ve oluşturma başlıkları altında gösterilecektir.

Aşağıda yer alan Tablo 13.2'de program yazarken dosya işlemlerinde kullanılacak olan *dosya emniyet seviyeleri* verilmiştir. Bu değerler program yazımı sırasında parametre olarak kullanılmaktadır. Eğer dosya emniyet seviyesi belirtilmemişse *varsayılan olarak* 0 değeri arka planda belirlenir.

Örnek

- Dosya açma ve dosya emniyet seviyesini belirtmeye yönelik bir C++ kod parçacığı yazınız.

Çözüm:

```
ifstream dosyaOrnek; //ifstream tipinde dosya nesnesi oluşturuldu.
dosyaOrnek.open("C://ProgramlamaTemelleri/Ornek.txt", ios::in, 0);
```

Yukarıdaki örnekte ifstream akışı dosyadan *okuma* yapmak üzere kullanılan sınıftır. dosyaOrnek ise *ifstream* sınıfından oluşturulmuş bir nesnedir. Bu nesne ile dosya açma, okuma ve dosya kapatma işlemleri sınıfa ait fonksiyonlar kullanılarak gerçekleştirilecektir.

ifstream ve ofstream nesnesi ile open() fonksiyonu çağrılarak dosya açma işlemi başlatılmıştır.

Örnekte yazıldığı gibi dosyaOrnek nesnesi ile *open()* fonksiyonu çağrılarak dosya açma işlemi başlatılmıştır. open() fonksiyonu kendisine üç adet parametre almaktadır. Birinci parametre string türünde dosyanın dizinini, ismini ve uzantısını içeren string'dir. İkinci parametre dosya açma modunu belirtmektedir. Bu örnekte ios::in kullanılmıştır. Yani giriş modunda dosya açılarak dosyadan okuma işlemi yapılacağını göstermektedir. Dosya belirtilen dizinde yoksa bu mod dosya oluşturamaz ve program *hata* verir. Üçüncü parametre değeri 0 olarak seçilmiştir. Bu değer dosya emniyet seviyesini belirtmektedir ve *normal* bir dosya üzerinde işlemlerin yapılacağını bildirmektedir. Bu değer olmadan da open() fonksiyonu çağrılabilir. Bu durumda varsayılan olarak 0 değeri üçüncü parametre olarak atanmış olur. Örnek çözüm aşağıda gösterilmiştir:

Çözüm:

```
ifstream dosyaOrnek;
dosyaOrnek.open("C://ProgramlamaTemelleri/Ornek.txt", ios::in);
```

Tablo 13.2. Dosyaların Emniyet Seviyeleri ve Anlamları

Emniyet Seviyeleri	Anlamı
0	Bu parametre normal bir dosya türü oluşturulduğunu gösterir. Varsayılan durum olarak bu değer kullanılır.
1	Salt okunur (Read Only) dosya türüdür. Bu parametre dosya üzerinde değişiklik yapılmasına müsaade etmez ve yalnızca dosyadan okuma işlemi yapmamıza olanak tanır.
2	Parametre olarak bu değer kullanıldığı zaman dosya türünün saklı (hidden) olduğu belirtilmiş olur. Dizin içerisinde dosyanın varlığı görülemez. Gizli dosyaları göster seçeneği kullanılırsa görüntülenebilir.
4	Sistem dosyası olduğunu belirtir.
8	Arşiv dosyası olduğunu belirtir.

DOSYA İŞLEME

Dosya işleme alt başlığında; program içerisinde dosya oluşturma, dosya açma ve kapatma, dosyadan veri okuma ve dosyaya veri yazma adımları anlatılacaktır.

Dosya Oluşturma

Bilgisayar programlarında verilerin program kapanınca kaybolmaması için dosyalar kullanılır. Dosyalar içerisine belirli formatta veriler kaydedilir ve gerektiğinde program tarafından bu veriler okunarak gerekli işlemler yapılır. Bu işlemler var olan bir dosya üzerinde yapılabilir. Fakat belirtilen dizinde eğer dosya yoksa uygun prosedürlere göre yeni bir dosya oluşturulur. Dosya oluşturulurken yukarıda bahsedilen *Dosya Açma Modlarına* da dikkat edilmesi gerekmektedir.

Dosya oluştururken programdaki amaca uygun dosya açma modu seçilmelidir.

Örnek

- Bir dosyaya veri kaydetmek üzere dosya oluşturma işlemini gerçekleştirecek bir C++ programı yazınız.

Çözüm:

```
#include <fstream> //Dosyaya yazmak için ofstream
using namespace std;
int main()
{
        ofstream dosyaOlusturmaOrnegi; //ofstream tipinde dosya nesnesi oluşturuldu.
        dosyaOlusturmaOrnegi.open("Ornek.txt", ios::out);
        return 0;
}
```

Yukarıdaki örnekte ofstream sınıfı kullanılarak dosyaOlusturmaOrnegi adında dosya işlemlerini yürütecek bir *nesne* oluşturulmuştur. Ardından open() fonksiyonu çağrılarak Ornek.txt adında bir *dosya* oluşturulmuştur. Dosyaya veri yazılacağı için dosya oluşturma ve açma işleminde *çıkış modu* (ios::out) tercih edilmiştir. Burada dikkat edilmesi gereken hususlardan bir tanesi oluşturulan Ornek.txt dosyasının C++ kaynak kodunu içeren cpp dosyası ile aynı konumda yer almasıdır. Eğer istenirse farklı adres yazılarak başka bir dizinde de dosya oluşturulabilir.

Dosyaya Veri Yazma

Program içerisinde verilerin kalıcı olarak saklanması için dosyaya yazma işlemi uygulanır. C++ dilinde dosyaya veri yazma işleminde aşağıdaki adımlar takip edilir:

- Programın başlık kısmına fstream kütüphanesi eklenmelidir.
  #include <fstream>
- ofstream sınıfına ait bir nesne oluşturulmalıdır. Nesne ismi dosyaNesnesi olsun;
  ofstream dosyaNesnesi;

- Oluşturulan nesne yardımıyla open() üye fonksiyonu çağrılarak dosya açılmalıdır. Dosya adı DosyaYazma.txt olsun;
  dosyaNesnesi.open("DosyaYazma.txt", ios::out, 0);
- Çıkış işlemi yani dosyaya yazma işlemi için dosya açıldıktan sonra cout<< kullanımındaki gibi << akış sembolü kullanılarak yazma işlemi gerçekleştirilmelidir.
  dosyaNesnesi << deger; //deger değişkeni içerisindeki veri dosyaya yazılır.
- Program sonunda açılan dosya, close() fonksiyonu ile kapatılmalıdır.
  dosyaNesnesi.close();

Örnek

•Bir sınıftaki öğrencilerin bir derse ait sınav puanlarını klavye yolu ile kullanıcıdan alarak bir dosyaya kaydeden C++ programını yazınız. (Sınıf mevcudu 10 olarak kabul edilebilir.)

Dosyaya yazma işlemi için "<<" akış sembolü kullanılmalıdır. (dosyaNesnesi<<deger;)

Çözüm:

```
#include <iostream> //cin ve cout için
#include <fstream> //Dosya okuma yazma için (ofstream)
using namespace std;
int main()
{
        ofstream dosyaOrnek; //ofstream tipinde dosya nesnesi oluşturuldu.
        dosyaOrnek.open("Ornek.txt", ios::out); //dosya açılır
        int puan;
        for (int i = 0; i < 10; i++) //10 iterasyon çalışır (öğrenci sayısı kadar)
        {
                cout << "Ogrenci puanini giriniz: ";
                cin >> puan; //her iterasyonda bir öğrencinin notu girilir
                cout << endl; //konsolda imleci alt satıra kaydırır.
                dosyaOrnek << puan << endl;
        }
        dosyaOrnek.close();//açılan dosya kapatılır.
        return 0;
}
```

Program sonunda açılan dosya, close() fonksiyonu ile kapatılmalıdır.

Bireysel Etkinlik

• Rekürsif (kendi kendini çağırabilen) bir fonksiyon yazarak fibonacci serisinin ilk 20 elemanını hesaplayan bir C++ programı yazınız. Program hesapladığı değerleri bir dosya içerisine alt alta olacak şekilde kaydetmelidir.

Dosyadan Veri Okuma

Dosyadan okuma işlemi için “>>” akış sembolü kullanılır. (dosyaNesnesi>>değer;)

Dosyadan okuma yani programa dosyadan *veri girişi* yapmak için aşağıdaki adımlar takip edilmelidir.

- Programa fstream kütüphanesi eklenmelidir.
  #include <fstream>
- ifstream sınıfına ait bir nesne oluşturulmalıdır. Nesne ismi dosyaNesnesi olsun;
  ifstream dosyaNesnesi;
- Oluşturulan nesne yardımıyla open() üye fonksiyonu çağrılarak *dosya açılmalıdır*.
  dosyaNesnesi.open("DosyaOkuma.txt", ios::in, 0);
- Giriş işlemi yani dosyadan okuma için dosya açıldıktan sonra cin>> kullanımındaki gibi >> *akış sembolü* kullanılarak okuma işlemi gerçekleştirilmelidir.
  dosyaNesnesi>>deger; //deger değişkenine dosyadan okunan değer yazılır.
- Program sonunda açılan dosya close() fonksiyonu ile *kapatılmalıdır*.
  dosyaNesnesi.close();

Örnek

- Bir sınıftaki öğrencilerin sadece bir dersine ait sınav puanlarını içeren dosyadan sırasıyla değerleri okuyup, C++ Standart Şablon Kütüphanesine (STL) ait olan vector sınıfını kullanarak bir vektör içerisine kaydediniz.

Örnek

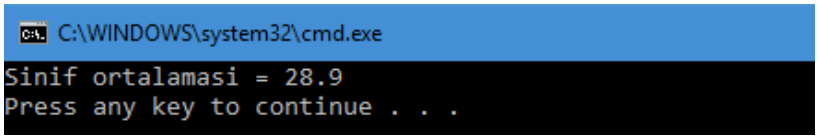
- Ardından vektör içerisindeki değerleri kullanarak derse ait sınıf ortalamasını hesaplayınız ve ekrana yazdırınız (Sınıf mevcudu 10 olarak kabul edilebilir.).
- Bir önceki örnekte oluşturulan dosya kullanılabilir.

Çözüm:

```
#include <iostream> //cin ve cout için
#include <vector> //STL kütüphanesi vector için
#include <fstream> //Dosya okuma yazma için (ifstream)
using namespace std;
int main()
{
```

```
        ifstream dosyaOrnek; //ifstream tipinde dosya nesnesi oluşturuldu.
        dosyaOrnek.open("Ornek.txt", ios::in);
        int puan; //dosyadan okunup bu değişkene atama yapılacak
        vector<int> puanlar;//STL kütüphanesi vector tanımlandı
        for (int i = 0; i < 10; i++) //10 iterasyon çalışır (öğr sayısı)
        {
                dosyaOrnek >> puan; //dosyadan bir değer okunup puana atandı
                puanlar.push_back(puan); //vectorün sonuna bir eleman eklendi
        }

        int toplam = 0; //toplama işleminde etkisiz eleman 0.
        for (int j = 0; j < 10; j++) //10 iterasyon çalışır (öğr sayısı)
        {
                toplam += puanlar[j]; //vektör içerisindeki tüm değerler toplanır
        }

        double ortalama = toplam / 10.0;
        cout << "Sinif ortalamasi = " << ortalama << endl;
        dosyaOrnek.close();//açılan dosya kapatılır.
        return 0;
}
```

Örnek çözümün konsol penceresi çıktısı şekil 13.2'de verilmiştir.



Şekil 13.2. Örnek Çözüm Ekran Çıktısı

DOSYA OKUMA İŞARETÇİLERİ

Rastgele erişimli dosyalarda, dosya içerisindeki kayıtlara doğrudan ulaşılabilir.

Rastgele erişimli dosyalarda dosya içerisindeki kayıtlara *doğrudan* ulaşılabilir olduğu, Dosya Türleri başlığı altında anlatılmıştı. C++ programlama dilinde bu tür dosyalar bir *dizi* gibi düşünülebilir. Yani dosya içerisindeki kayıtların ilk satırdan itibaren dizinin sıfırıncı indeksli elemanından başlanmak üzere depolandığı varsayılabilir. Dizilerde, dizi içerisindeki bir elemana nasıl indeks (indis) aracılığı ile doğrudan ulaşılabiliyorduysa (ornekDizi[25] ile dizinin 26. elemanına ulaşılmakta) bu tür dosyalarda da indeks yerine *seek()* özel fonksiyonu kullanılarak verilere ulaşılmaktadır. Bu fonksiyonun kullanımı şu şekildedir:

dosyaNesnesi.seekg(Belirlenmiş konumdaki byte'ların indeks sayısı);
dosyaNesnesi.seekp(Belirlenmiş konumdaki byte'ların indeks sayısı);

Yukarıda *seek* fonksiyon ismine eklenen *g harfi* (seekg) dosya konum işaretçisi olan dosya indeksinin konumunu okumak (**g**et) için, *p harfi* (seekp) ise dosya işaretçisinin konumuna yazmak (**p**ut) için kullanılmaktadır. seekg ve seekp fonksiyonlarının ilk parametresi olan byte'ların indeks sayısı *long int* veri

tipindedir. Bu parametre dosya işaretçisinin belirtilen konumdan itibaren hareket ettirilmek istenilen byte sayısıdır. Aşağıda okuma yapmak için kullanılan seek fonksiyonu anlatılmıştır:

dosyaNesnesi.seekg(long int, konum);

Örnek kullanımdaki konum, üç farklı *işaretçi değeri* almaktadır. Konum için kullanılan bu değerler ve alternatifleri tablo 13.3'de verilmiştir.

Tablo 13.3. Dosya İşaretçi Konum Değerleri

Konum Değeri	Alternatif Değer	Anlamı
ios::beg	SEEK_SET	İşaretçi konumunu dosyanın başına getirir.
ios::cur	SEEK_CUR	İşaretçi konumunu güncel (mevcut) durumuna getirir.
ios::end	SEEK_END	İşaretçi konumunu dosyanın sonuna getirir.

Program içerisinde rastgele erişimli dosyadan okuma yapmak için kaynak koda <fstream> başlık dosyasının eklenmesi gerekmektedir.

Tablo 13.3'de verilen işaretçi konumu dosyada hangi indeksten başlanılacağını belirtir. Dosya işaretçisinin ileri ya da geriye doğru hareket ettirilmesi için byte sayısı parametre değeri *negatif* ya da *pozitif* olabilir. Pozitif değerler işaretçiyi dosyanın *sonuna doğru* (ileri yönde), negatif değerler ise dosyanın *başına doğru* (geri yönde) taşır.

Program içerisinde rastgele erişimli dosyadan okuma yapmak için kaynak koda *<fstream>* başlık dosyasının eklenmesi gerekmektedir. Aşağıda bu konuyu kavrayabilmek için bütünlük arz eden örneklere yer verilmiştir. İlgili örneklerde kullanılacak olan *L ifadesi* byte'ı ifade etmektedir. Yani dosya işaretçisinin yerinde kalması istenirse 0L ifadesi, 1 byte ileri hareket ettirilmek istenirse 1L, 2 byte geriye hareket ettirilmek istenirse de -2L ifadesi seek fonksiyon parametresinde long int kısmında kullanılmalıdır.

Örnek

- Yeni bir metin dosyası oluşturarak bu dosya içerisine 6 elemanlı bir karakter dizisi (char[]) kaydediniz.

Çözüm:

```
#include <fstream> //Dosya okuma ve yazma için
using namespace std;
int main()
{
        ofstream dosyaIsleme; //dosya yazma için nesne oluşturulur
```

```
        char alfabe[] = "ABCDEF";//char dizisi oluşturulur, elemanlar atanır.
        dosyaIsleme.open("alfabe.txt", ios::out);
        for (int i = 0; i < 6; i++)
                dosyaIsleme << alfabe[i];//Dosyaya 6 adet eleman yazılır.
        dosyaIsleme.close(); //Açılan dosya kapatılır.
        return 0;
}
```

Rastgele erişimli dosyalarda seekp() ve seekg() fonksiyonları ile dosyadan okuma ve yazma işlemleri yapılabilir.

Yukarıdaki örnekte alfabe.txt dosyası program kaynak dosyaları ile *aynı dizinde* oluşturulmuştur ve içerisine sırasıyla A B C D E F harfleri kaydedilmiştir. Şimdi de rastgele erişimli dosya yöntemlerini kullanarak bir örnek uygulama yazalım.

Örnek

- Bir önceki örnekte oluşturulan alfabe.txt dosyası içerisindeki baştan üçüncü elemanın (C) yerine X harfini yazdıran, ardından bulunduğu konumdan geriye giderek baştan ikinci karakteri (B) okuyan ve sondan ikinci elemanın (E) yerine bu karakteri atayan bir C++ programı yazınız.

Çözümü istenilen örnek, seekp ve seekg fonksiyonlarının kullanımını ve ios::beg, ios::cur, ios::end konum işaretçilerinin uygulamasını içermektedir. Burada yapılması gereken işlemler, gerekli kütüphaneleri programa include etmek, ardından dosyadan okuma ve yazma yapmak üzere fstream akış nesnesi oluşturmaktır. Daha sonra rastgele dosya erişimi yöntemiyle bizden istenilenleri gerçekleştirmektir. Söz konusu çözüm aşağıdadır:

Çözüm:

```
#include <fstream> //Dosyadan okuma ve dosyaya yazmak için
using namespace std;
int main()
{
        char karakter;  //dosyadan okuma yapıldığında kullanılacak
        fstream dosyaIsleme; //dosya yazma için nesne oluşturulur.
        dosyaIsleme.open("alfabe.txt");//alfabe.txt dosyası açılır.
        dosyaIsleme.seekp(2L, ios::beg);//dosya işaretçisi ilk satıra konumlandırılır
        dosyaIsleme << 'X'; //Üçüncü karakter yerine X yazılır
        dosyaIsleme.seekg(-2L, ios::cur);//dosya işaretçisi mevcut konumunda tutulur.
        dosyaIsleme >> karakter; //Baştan ikinci karakter dosyadan okunur
        dosyaIsleme.seekp(-2L, SEEK_END); //Alternatif kullanım
        dosyaIsleme << karakter; //Sondan ikinci karakter yerine okunan eleman yazılır
        dosyaIsleme.close(); //Açılan dosya kapatılır.
        return 0;
}
```

Yukarıdaki çözümde rastgele erişim yöntemi ile açılan dosyadan baştan üçüncü harf X olarak değiştirilmiştir. Başlangıçta dosya içeriği ABCDEF'dir. Dosya işaretçisinin konumu *ios::beg* ile *ilk indeks* olarak belirlenmiştir. İndeks | sembolü ile gösterilecek olursa başlangıçta dosya |ABCDEF şeklinde açılarak işaretçi konumlandırılır. İlk kullanılan seekp fonksiyonunda 2L olarak girilen parametre ile *2 byte* kadar *dosya işaretçisi* ileri konumlandırılarak AB|CDEF olarak ayarlanır. Sonra indeksin olduğu yerdeki elemanın yerine (C harfinin) X harfi yazılır. Dosya içeriğinin son hali ABXDEF olarak değişmiş olur. Dosya işareçisi X'in sağında konumlandırılmış olarak bekler (ABX|DEF). Ardından *ios::cur* ile işaretçi konumunun mevcut hali korunur ve -2L ifadesi ile iki byte indeks geriye taşınır. Yani dosya işaretçisinin son hali A|BXDEF olur. Daha sonra *SEEK_END* (ios::end) ifadesi ile dosya işaretçisi en sona taşınır (ABXDEF|). -2L ile iki byte geri taşınarak (ABXD|EF) olarak konumu ayarlanır. Son olarak bir önceki adımda okunan karakter (B) bu konumdaki elemanın yerine yazılır. Böylece dosyanın son hali ABXDBF olur.

ios::beg, ios::cur ve ios:: end dosya işaretçi konumlarının alternatif değerleri sırasıyla SEEK_SET, SEEK_CUR, SEEK_END'dir.

Bireysel Etkinlik

- İçerisinde 10 adet int veri tipinde değer saklayan bir metin dosyası oluşturunuz. Bu değerleri sondan başlayarak ikişer ikişer indis atlayarak dosyadan okuyup ekrana basan bir C++ kodu yazınız.

Özet

- Bir program yazılırken program içerisinde kullanılan değişkenler, diziler ve diğer veri yapıları verileri geçici olarak hafızada tutmaktadır. Bu şekilde hafızada tutulan veriler programın çalışması sonlandığında kaybolmaktadır. Bundan dolayı dosyalar, büyük miktarda verileri kalıcı olarak hafızada tutmak için kullanılırlar.
- Dosya işlemleri; yeni bir dosya oluşturma, var olan dosyanın kullanılmak üzere açılması, dosyaya veri yazılması ve dosyadan veri okunması ve dosyanın sonlandırılması süreçlerini kapsamaktadır.
- Akış (stream), veri akışına genel olarak verilen kavramsal bir terimdir. C++ programlama dilinde bir akış, belirli bir sınıfın nesnesi olarak sembolize edilir.
- Akışlar iki çeşittir. Birincisi programın yazabileceği, ikincisi ise okuyabileceği akıştır.
- Farklı türde veri akışlarını temsil etmek için farklı akışlar kullanılır.
- Dosyaya veri yazma ve dosyadan veri okuma işlemlerini gerçekleştirmede kullanılacak olan iostream akış sınıfının alt sınıfları ofstream, ifstream ve fstream'dir.
- ofstream, dosyaya yazma işlemlerinde kullanılmaktadır.
- ifstream, dosyadan okuma yapma işleminde kullanılmaktadır.
- fstream, dosya giriş-çıkış işlemlerini birlikte gerçekleştirmektedir.
- Dosya, aynı türden verilerin belirli bir düzende kayıt ortamına (disk, CD vs.) yazılmasıyla oluşturulan bir yapıdır.
- Dosyalar, kayıt edildikleri ortama göre, içeriğine erişim yöntemine göre, içerdikleri verinin yapısına göre farklı bakış açılarından sınıflara ayrılabilir.
- Dosya açılırken dosyanın bulunduğu dizinin, dosya ile nasıl bir işlem yapılacağı bilgisinin ve dosya tipinin belirtilmesi gerekmektedir. Bu işlemlere dosya açma modları ismi verilmektedir.
- ios::app, ios::in, ios::out, ios::ate, ios::trunc, ios::binary, ios::nocreate ve ios::noreplace dosya okuma modlarından bazılarıdır.
- Dosya işlemede, program içerisinde dosya oluşturma, dosya açma ve kapatma, dosyadan veri okuma, dosyaya veri yazma adımları yer almaktadır.
- Dosya işlemleri var olan bir dosya üzerinde yapılabilir. Fakat belirtilen dizinde eğer dosya yoksa uygun prosedürlere göre yeni bir dosya oluşturulur.
- Program içerisinde verilerin kalıcı olarak saklanması için dosyaya yazma işlemi uygulanır.
- Rastgele erişimli dosyalarda indeks yerine seek() özel fonksiyonu kullanılarak verilere ulaşılmaktadır.
- seek fonksiyonu seekg ve seekp formlarında kullanılır.
- Dosya işaretçisinin ileri ya da geriye doğru hareket ettirilmesi için byte sayısı parametre değeri negatif ya da pozitif olabilir.
- Pozitif değerler işaretçiyi dosyanın sonuna doğru (ileri yönde), negatif değerler ise işaretçiyi dosyanın başına doğru (geri yönde) taşır.
- Program içerisinde rastgele erişimli dosyadan okuma yapmak için kaynak koda <fstream> başlık dosyasının eklenmesi gerekmektedir.
- Giriş işlemi yani dosyadan okuma için dosya açıldıktan sonra >> akış sembolü kullanılarak okuma işlemi gerçekleştirilmelidir.
- Program sonunda açılan dosya close() fonksiyonu ile kapatılmalıdır.

DEĞERLENDİRME SORULARI

1. Bilgisayarda veriler kalıcı olarak aşağıdaki hangi hafıza türünde depolanamaz?
   a) Hard Disk
   b) RAM Bellek
   c) CD
   d) Flash Disk
   e) SSD Disk

2. C++ programlama dilinde akışlar (stream) ile ilgili aşağıdakilerden hangisi söylenemez?
   a) Farklı türde veri akışlarını temsil etmek için aynı akışlar kullanılır.
   b) Bir akış, belirli bir sınıfın nesnesi şeklinde temsil edilir.
   c) Akış, veri akışına genel olarak verilen kavramsal bir terimdir.
   d) ifstream sınıfı, diskteki kaynak dosyalarından gelen veri akışını simgelemektedir.
   e) cin ve cout bir tür akış nesnesidir.

3. I. Dosya giriş işlemlerini gerçekleştirir.
   II. ostream sınıfından türetilmiştir.
   III. Dosyaya yazma işlemlerinde kullanılır.

   Yukarıdakilerden hangisi ya da hangileri C++ dilinde ofstream sınıfının özelliklerindendir?
   a) Yalnız I
   b) Yalnız II
   c) II ve III
   d) I ve III
   e) I, II ve III

4. Bir bilgisayarda yer alan dosya türleri arasında aşağıdakilerden hangisi bulunmaz?
   a) Rastgele (random) erişimli dosya
   b) Sekizli (oktal) dosya
   c) Geçici (iç) dosya
   d) Sıralı (sequential) erişimli dosya
   e) Kalıcı (dış) dosya

5. Belirtilen dizinde eğer aynı dosyadan var ise dosyanın yeniden oluşturulmasını aşağıdaki dosya açma modlarından hangisi engellemektedir?
   a) ios::nocreate
   b) ios::trunc
   c) ios::app
   d) ios::noreplace
   e) ios::in

6. Dosya açma modlarından birisi olan ios::out ile ilgili özellikler arasında aşağıdakilerden hangisi bulunmaz?
   a) Belirtilen dizinde dosya yoksa oluşturur.
   b) Mevcut dosya açılmışsa içindeki tüm verileri silerek ilk satırdan kayda başlar.
   c) Çıkış modudur.
   d) Verileri dosyaya yazmada görevlidir.
   e) Eğer açılmak istenilen dosya mevcut değilse hata verir.

7. Dosya işlemlerinde kullanılan dosya emniyet seviyelerinden hangisi salt okunur dosya türünü ifade etmektedir?
   a) 0
   b) 1
   c) 2
   d) 4
   e) 8

8. C++ programlama dilinde bir dosyaya veri yazmak için aşağıdaki adımlardan hangisi uygulanmaz?
   a) Programa fstream kütüphanesi eklenir.
   b) ofstream sınıfına ait bir nesne oluşturulur.
   c) Dosya istenilen özelliklerde formatlanır.
   d) open() üye fonksiyonu çağrılarak dosya açılır.
   e) “<<” akış sembolü kullanılarak yazma işlemi gerçekleştirilir.

9. C++ programlama dilinde dosya işlemede kullanılan seekp() fonksiyonunun ikinci parametresi aşağıdakilerden hangisi olamaz?
   a) ios::cur
   b) SEEK_SET
   c) ios::beg
   d) ios::end
   e) SEEK_START

10. fstream dosyaIsleme;
    dosyaIsleme.open("alfabe.txt");
    dosyaIsleme.seekp(-3L, ios::end);

    Yukarıdaki C++ kodunun gerçekleştirdiği işlemler arasında aşağıdakilerden hangisi bulunmaz?

    a) Dosya işlemleri rastgele erişimli bir dosya üzerinde yapılmaktadır.
    b) alfabe.txt adında metin dosyası açılmıştır.
    c) open() üye fonksiyonu fstream sınıfına aittir.
    d) ios::end ile dosya okuma işaretçisi dosyanın sonunda konumlandırılmıştır.
    e) -3L iadesi ile dosya okuma işaretçisi 3 byte ileri doğru hareket etmiştir.

Cevap Anahtarı

1.b, 2.a, 3.c, 4.b, 5.d, 6.e, 7.b, 8.c, 9.e

YARARLANILAN KAYNAKLAR

[1] Akkurt, Prof. Dr. M. (2004). *C++ programlama Dilinin Esasları Ve Uygulamaları.* Birsen Yayınevi, İstanbul.

[2] Deitel, Harvey M & Deitel, Paul J. (2009). *C ve C++*. Sistem Yayıncılık, İstanbul.

[3] Lafore, R. (2002). *Uzmanlar İçin Nesne Yönelimli C++ Programlama Klavuzu.* Alfa Yayınevi, İstanbul.

[4] Johnsonbaugh, R. & Kalin, M. (2000). *Object-oriented programming in C++.* Prentice Hall, Upper Saddle River.

[5] Sarıdoğan, M. E. (1994). *C++ ve nesneye yönelik programlama*. Sistem Yayınevi, İstanbul.

NESNEYE YÖNELİK PROGRAMLAMAYA GİRİŞ

İÇİNDEKİLER

- Nesneye Yönelik Programlama
- Sınıf ve Nesne Yapısı
- Üye Değişkenler ve Üye Fonksiyonlar
- Public ve Private Belirteçler
- Yapıcılar ve Yıkıcılar
- Kapsam Çözünürlük Operatörü
- Nokta Operatörü

HEDEFLER

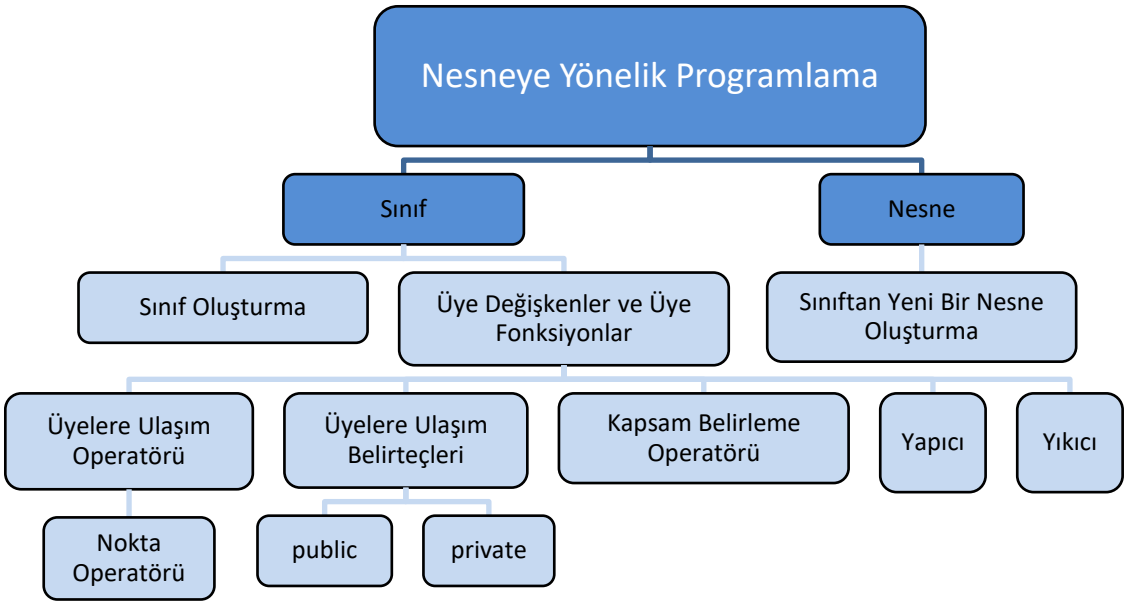
- Bu üniteyi çalıştıktan sonra;
  - Nesneye yönelik programlama kavramlarını öğrenebilecek,
  - Sınıf ve nesne oluşturabilecek,
  - Üye değişkenler ve üye fonksiyonlar tanımlayabilecek,
  - Üyelere ulaşım belirteçlerinden public ve private belirteçlerini öğrenebilecek,
  - Yapıcı ve yıkıcı kavramlarını öğrenebilecek ve bu yapıları programlarınızda kullanabilecek,
  - Kapsam çözünürlük operatörü ve nokta operatörü kullanımını öğrenebileceksiniz.

Atatürk Üniversitesi
Açıköğretim Fakültesi

PROGRAMLAMA TEMELLERİ

Arş. Gör. Dr.
Mete YAĞANOĞLU

ÜNİTE
14


Nesneye Yönelik Programlama
Sınıf
Nesne
Sınıf Oluşturma
Üye Değişkenler ve Üye Fonksiyonlar
Sınıftan Yeni Bir Nesne Oluşturma
Üyelere Ulaşım Operatörü
Üyelere Ulaşım Belirteçleri
Kapsam Belirleme Operatörü
Yapıcı
Yıkıcı
Nokta Operatörü
public
private

GİRİŞ

Nesneye Yönelik Programlama (Object Oriented Programming, OOP), yazılım geliştirme aşamasını oldukça kısaltan ve sistematik hale getiren bir tekniktir. Nesneye yönelik programlamanın en önemli yapı taşları, *sınıf* (class), *nesne* (object), *kalıtım* (inheritance) ve *çok biçimlilik (*polymorphism) olarak söylenebilir [1]. Nesneye Yönelik Programlama, modüler tasarım ile esneklik sağlayan bir yazılım mimarisi oluşturma pratiğidir [2].

Nesneye yönelik programlama, gerçek dünya nesnelerine benzer şekilde, sınıf adı verilen veri türlerinden oluşturulan nesneleri kullanır. Sınıf, gerçek hayattaki bir nesneyi istenen bir şekilde modelleyebilmek için gerekli bütün kodu ve veriyi içeren yazılım birimidir. Sınıf yapısı, üye değişkenler olarak adlandırılan öz niteliklere ve bu üye değişkenleri görüntüleyebilen veya bunlar üzerinde işlem yapabilen ve üye fonksiyonlar adı verilen bileşenlere sahiptir. Sınıf içinde 2 adet üyelere ulaşım belirteci vardır. Bunlar *public* ve *private* olarak adlandırılır. *public* belirteci kullanılmışsa, sınıf içerisinden ve sınıf dışından kolaylıkla bu özelliklere ve fonksiyonlara erişilebilir. *private* belirteci kullanılmışsa sadece sınıf içerisinden özelliklere ve fonksiyonlara erişilebilir.

Nesneye yönelik programlamada nesneler, gönderdikleri veya aldıkları mesajlar açısından birbirleriyle iletişim kurabilir ve nesnelerin ortak özelliklerinden ve fonksiyonlarından faydalanarak yeni sınıflar kolay bir şekilde *kalıtım* adı verilen bir süreç aracılığıyla yaratılabilir. Nesneye yönelik programlamada, programlama ortamındaki her şey bir nesne olarak kabul edilmekte, nesnelerin özellikleri değiştirilerek onlara yeni biçimler verilebilmektedir.

Nesnelerin ilk kullanıma hazırlanması için *yapıcı* (constructor) fonksiyon kullanılmaktadır. *Yıkıcı* (destructor) fonksiyon ise nesneyle işimiz bittiğinde gerekli temizliği yapan fonksiyondur. Yıkıcı fonksiyon çağrıldığında bellekte nesnenin yeri silinmektedir.

NESNEYE YÖNELİK PROGRAMLAMA

Nesneye yönelik programlama, dinamik ve rekabetçi ortamın gereksinimlerini karşılamak için esnek, modüler ve yeniden kullanılabilir yazılım bileşenleri geliştirmek için yeni bir yaklaşımın ve artan bir talebin sonucudur. Programlamanın en doğal yolunu geliştirmek için nesneye yönelik programlama; insanlar, hayvanlar, arabalar, binalar, bilgisayarlar vb. gerçek dünya objelerinden esinlenerek kendi kendini sürdürebilen bir nesne kavramını sunmuştur. Gerçek dünyada nesnelerin boyutu, kütlesi, yüksekliği, rengi gibi özellikleri vardır. Aynı zamanda gerçek dünyadaki nesneler çeşitli işlevler yapabilirler. Örneğin; bir araba hızlanabilir, dönüş yapabilir, yavaşlayabilir veya bir kişi yürümek, koşmak, uyumak ve çalışmak gibi davranışlar sergileyebilir. Dahası, gerçek dünyadaki nesneler birbirleriyle iletişim kurar ve öz niteliklerine ve davranışlarına göre toplanabilirler. Örneğin; arabalar, otobüsler, kamyonlar bazı özellikleri paylaşır, benzer davranışlar sergiler ve araçlar sınıfı altında gruplanabilir.

Nesneye yönelik programlama, programlamadaki nesneleri kullanmayı ve gerçek dünya varlıklarına uygulamayı amaçlamaktadır. Nesneye yönelik programlama gerçek dünyadaki soyut nesneleri somut olarak modeller. Nesneye yönelik programlamanın en temel amacı, prosedürel yaklaşımda karşılaşılan bazı kusurların giderilmesidir. Nesneye yönelik programlama, verileri program geliştirmede kritik bir öğe olarak ele alır ve sistemin etrafında serbestçe akmasına izin vermez. Böylelikle verileri, yanlışlıkla yapılan değişiklikten korur.

Nesneye yönelik programlamanın bazı özellikleri şunlardır [3,4]:

- Programlar nesne olarak bileşenlere ayrılır.
- Veri yapıları, nesneleri karakterize edecek şekilde tasarlanmıştır.
- Bir nesnenin verileri üzerinde çalışan işlevler, veri yapısında birbirine bağlanır.
- Veri gizlidir ve harici işlevle erişilemez.
- Nesneler birbirleriyle iletişim kurabilir.
- Yeni veri ve fonksiyonlar gerektiğinde kolayca eklenebilir.
- Program tasarımında aşağıdan yukarıya yaklaşımı takip eder.
- Kodu yazmaya başlamak zorunda kalmadan, birbiriyle iletişim halinde olan standart çalışma modüllerinden programlar oluşturabiliriz. Bu, geliştirme süresinin ve yüksek verimliliğin korunmasına yol açar.
- Bir nesnenin birden çok örneğine sahip olmak mümkündür.
- Veri gizleme ilkesi, programcının programın diğer bölümlerinde kod tarafından ele geçirilemeyen güvenli bir program oluşturmasına yardımcı olur.
- Çalışmaları nesneler temelinde bir projede bölmek kolaydır.
- Yazılım karmaşıklığı kolayca yönetilebilir.

Nesne Yapısı

Nesneye yönelik programlama tekniğinin en küçük yapı taşı nesnedir. Nesneler veriler ve veriler arası ilişkiyi sağlayan fonksiyonlardan oluşur [1]. Nesneye yönelik programlamada, programlama ortamındaki her şey bir nesne olarak kabul edilmekte, nesnelerin özellikleri değiştirilerek onlara yeni biçimler verilebilmektedir. Hemen hemen her programlama dili, integer, float, double, string vb. gibi standart veri türlerine sahiptir. Nesne kullanıcı tarafından tanımlanan yeni bir veri türüdür ve her şey bir nesne olabilir. Nesneler, kullanıcı tanımlı değişkenlere, başka bir deyişle üye değişkenlere sahiptir. Nesnelerin, kapsüllenmiş veri elemanlarını görüntüleyen veya değiştiren üye fonksiyonları da vardır. Nesneler nesne yönelimli sistemde temel çalışma zamanı öğeleridir ve tanımlanmış kullanıcı tanımlı veri türleri olan bir sınıfın örnekleridir.

Bir program yürütüldüğünde, nesneler birbirlerine mesaj göndererek etkileşime girer. Her nesne, verileri işlemek için veri ve kod içerir. Nesneler, birbirlerinin verilerinin veya kodlarının ayrıntılarını bilmeden etkileşime girebilir, kabul edilen mesajın türünü ve nesnelerin döndürdüğü yanıt türünü bilmek yeterlidir.

Nesne oluşturmak için main içinde sınıf adı ve oluşturmak istediğimiz nesnenin adını yazarız. Sınıf ve nesne tanımlanması aşağıdaki gibidir:

```
#include <iostream>
using namespace std;
class Ornek{//Ornek sınıf oluşturuldu
        //üye değişkenler ve üye fonksiyonlar
};
int main()
{
    Ornek o1;  //önce sınıf adı sonra olusturulan nesne adı
}
```

Nesne tanımlanmasında öncelikli olarak sınıf oluşturulur. *Ornek* sınıfı oluşturulduktan sonra *Ornek* sınıfına ait *o1* nesnesi oluşturulmuştur. Sınıf, ortak özellikleri olan nesnelerin özelliklerini ve davranışlarını barındırır.

Nesne ait olduğu sınıfın bir örneğidir. Nesneler arası ortak özellik ve davranışlar olabilir. Ortak özellikleri barındıran yapıya sınıf, o sınıftan türettiğimiz örneğe ise nesne denir.

Sınıf Yapısı

Sınıf, gerçek hayattaki bir nesneyi istenen bir şekilde modelleyebilmek için gerekli bütün kod ve veriyi içeren yazılım birimidir. Tüm nesneler birer sınıf örneğidir. Sınıflar nesnenin özelliklerini belirler. Nesneye yönelik programlama yapısıyla geliştirilen bir yazılım temel olarak, gerekli görevleri tamamlamak için mesajlar aracılığıyla birbirleriyle iletişim kuran bir sınıf koleksiyonudur. Sınıflar, nesneleri başlatmak için kullanılan kullanıcı tanımlı veri türleridir. Sınıf tanımlanması aşağıdaki yapıdaki gibidir:

Ortak özellikleri barındıran yapıya sınıf, o sınıftan türettiğimiz örneğe ise nesne denir.

```
class SinifAdi
{
        Ulaşım Belirteçleri:  // Üyelere ulaşım belirteçleri public veya private

        Üye Değişkenler;      // Değişkenler bu kısımda tanımlanır

        Üye Fonksiyonlar() {}  //Üye değişkenlere erişim için fonksiyonlar

}; // sınıf ; ile biter
```

Sınıf yapısı, class anahtar kelimesi ile başlar. class anahtar kelimesini sınıf adı takip eder. Sınıf adından sonra süslü parentez ile sınıf içine girilir ve sırasıyla üyelere ulaşım belirteci, üye değişkenler ve üye fonksiyonlar tanımlanır. Sınıf tanımı noktalı virgül (;) ile sonlanır. Ulaşım operatörleri public veya private'dır. Üye değişkenler ve üye fonksiyonlar ise kullanılacak sınıfın değişkenleri ve fonksiyonlarıdır.

Sınıf içerisinde ulaşım belirteçleri, üye değişkenler ve üye fonksiyonlar yer alır.

Sınıf içerisinde ortak özellikler ve davranışlar yer alır. Sınıf, Şekil 14.1'de gösterildiği gibi 3 bölüm ile görselleştirilebilir. Örneğin; insan sınıfı oluşturulacaksa, üye değişkenler yaş, cinsiyet, çalışılan kurum, isim ve ağırlık olabilir. Bu sınıfın üye fonksiyonları ise yaşı, cinsiyeti, çalışılan kurumu, ismi ve ağırlığı geri döndüren fonksiyonlar olabilir. Şekil 14.1'de görüldüğü gibi [5]:

- Sınıf adı: Sınıfı tanımlar.
- Veri üyeleri (Data Members) veya üye değişkenler (Variables): Sınıfın statik özelliklerini (öz nitelikler, durumlar, alanlar) içerir.
- Üye fonksiyonları (Member Functions): Sınıfın dinamik işlemlerini (metot, davranışlar, işlemler) içerir.

Şekil 14.1. Sınıf yapısı

Örnek

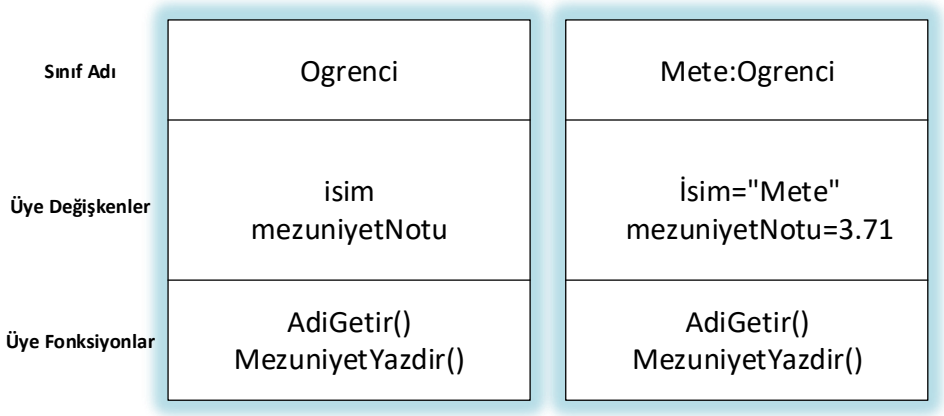
•Öğrenci sınıfını oluşturalım. yas adında üye değişkeni ve öğrencinin yaşını ekrana yazdıracak üye fonksiyonu tanımlayalım.

Çözüm:

```
class Ogrenci
{
public:
        int yas; // Üye değişkeni
        void YasiYazdir()  // Üye fonksiyonu
        {
                cout << "Yas: " << yas;
        }
};
```

Bu örnekte *class Ogrenci* ile öğrenci adında sınıf tanımı yapılmıştır. Daha sonra üyelere ulaşım belirteci olarak *public* kullanılmıştır. Sonrasında yas adında üye değişkeni ve *YasiYazdir* adında üye fonksiyonu tanımlanmıştır.

Sınıf kullanımı için örnek kullanım Şekil 14.2'de görüldüğü gibidir. Ogrenci sınıfına ait bir örnek aşağıda gösterilmiştir [5].



Şekil 14.2. Sınıf örneği

Bir sınıf, gerçek dünyadaki olaylara göre uyarlanan programcı tanımlı, soyut, kendi kendine yeten, yeniden kullanılabilir bir yazılım öğesidir. Bir sınıf 3 öğeden oluşur. Bunlar; sınıf adı, üye değişkenler ve üye fonksiyonlardır.

Üye Değişkenler

Sınıf tanımlaması içinde bildirilen değişkenlere üye değişkenler denir.

Sınıf tanımlaması içinde bildirilen değişkenlere *üye değişkenler (data members)* denir. Bir üye değişkenin bir ismi (tanımlayıcısı) ve bir tipi vardır. Üye değişken belirli bir türün değerini tutar.

Nesne, tanımları içinde değişkenler veya diziler gibi verileri tutan yapılardır. Bir nesne, bu nesneye özgü olan değerler içerebilir. Bunu yapmak için her bir değerin sınıftaki bir üye değişkeni olarak uygun bir tanımına ihtiyacı vardır. Bir üye değişkeni, önceden tanımlanmış sınıflar, herhangi bir türdeki nesneye yönelik işaretçiler veya herhangi bir türdeki nesneler için yapılan referanslar dâhil olmak üzere herhangi bir türde olabilir.

Üye değişkenler *public* veya *private* olabilir; ancak genellikle *private* tutulurlar. Böylece değerler sadece sınıf üye fonksiyonlarının takdirine bağlı olarak değiştirilebilir. Aşağıdaki örnekte C sınıfı, *int* türünde bir *private* ve *float* türünde bir *public* üye değişkeni içerir:

```
class C {
private:
        int x;
public:
        float f;
};
```

Üye Fonksiyonlar

Sınıf tanımlamasında bildirilen fonksiyonlara *üye fonksiyon (member functions)* denir. Sınıfların içinde tanımlanmış yöntemler veya işlevlerdir. Genellikle üye değişkenler ve diğer nesne verileri üzerinde çalışmak için kullanılırlar. Bir sınıfın üye fonksiyonu, sınıf içindeki değişkenlere sahip olan bir metottur. Üye fonksiyon, üyesi olduğu sınıfın herhangi bir nesnesinde çalışır ve bu nesnenin bir sınıfının tüm üyelerine erişim izni vardır. Üye fonksiyonların özellikleri şunlardır:

- Parametre alabilir.
- İşlev gövdesinde tanımlanan işlemleri gerçekleştirir.
- Bir değer döndürebilir.

Üye fonksiyonları sınıf içerisinde tanımlamamız ve sınıfa bağlı tutmamız gerekmektedir. Üye fonksiyonlar ait olduğu nesnenin bütün elemanlarına erişebilirler.

Metotlar bir nesne oluşturulmadan çağrılamazlar. Öncelikli olarak sınıftan bir nesne oluşturmak lazımdır. Nesne oluşturulduktan sonra ise üye fonksiyonlara erişilebilir.

Örnek

•Dikdörtgen prizmanın hacmini sınıf yapısı kullanarak hesaplayınız.

Çözüm:

```
class DikdortgenPrizma {
        double en;
        double boy;
        double yukseklik;

        double HacimHesapla(void)
        {
                return en * boy * yukseklik;
        }
};
```

Üye fonksiyon kullanımı yukarıdaki örnekte görüldüğü gibidir. Dikdörtgen prizmasının hacmini hesaplamak için *en, boy* ve *yukseklik* adında 3 adet üye değişkeni ve *HacimHesapla* adlı üye fonksiyonu kullanılmıştır. Görüldüğü gibi üye fonksiyonu en, boy ve yükseklik değerlerinin çarpımını geriye döndürmektedir.

Üyelere Ulaşım Belirteçleri

Sınıf içindeki üyelere ulaşım belirteçleri public ve private olabilir.

Sınıf içinde üyelere ulaşım belirteçleri *public* ve/veya *private* olabilir. Eğer bir özelliği veya fonksiyonu *public* olarak belirtirsek, bu özelliğe ve fonksiyona sınıf içerisinden ve sınıf dışından kolaylıkla erişebiliriz. *public* üyelere main içerisinden de erişilebilir. *private* üyelere ise sadece sınıf içerisinden erişilebilir. Belirteçler programın herhangi bir yerinde değişiklik yapılmasın diye koruma amaçlı kullanılırlar. Bir değişken veya fonksiyon sınıf içerisinde *public* veya *private* olarak özellikle belirtilmezse, varsayılan (default) olarak *private* değerini alır.

Örnek

- Daire sınıfı oluşturunuz ve sadece sınıf içerisinden erişilebilen yaricap ve renk üye değişkenlerini ve sınıf içinden ve dışından ulaşılabilen YaricapiGetir ve AlanHesapla üye fonksiyonlarını tanımlayınız.

Çözüm:

```
class Daire {       // sınıf adı
private:            //üyelere erişim belirteci (private)
        double yaricap;     // Üye değişkenler
        string renk;
public:             //üyelere erişim belirteci (public)
        double YaricapiGetir(); // Üye fonksiyonlar
        double AlanHesapla();
};
```

Yukarıdaki örnekte Daire sınıfı oluşturulmuş ve *private* olarak *yaricap* ve *renk* üye değişkenleri ve *public olarak YaricapiGetir* ve *AlanHesapla üye fonksiyonları* tanımlanmıştır.

Yapıcı Fonksiyon

Nesnelerin ilk kullanıma hazırlanması için yapıcı fonksiyon kullanılır.

Nesnelerin ilk kullanıma hazırlanması için *yapıcı (constructor)* fonksiyon kullanılır. Yapıcı fonksiyon bir nesne oluştuğunda üye değişkenlere ilk değer ataması yapmak için kullanılır. Yapıcı, *public* ya da *private* olmasına bakmaksızın nesnelere ilk değerini atar. Nesneye yönelik programlamada bir nesnenin oluşması sırasında ilk çalışan fonksiyon yapıcıdır. Yapıcının özellikleri şunlardır:

- Yapıcı fonksiyonun adı sınıf adı ile aynıdır.
- Geriye değer döndürmez.
- Parametre alabilir.

Varsayılan yapıcılar

Varsayılan Yapıcı (Default Constructor), herhangi bir parametre almayan yapıcıdır.

Örnek

•Deneme adında sınıf oluşturup varsayılan yapıcı sayesinde a ve b üye değişkenlerine ilk değer ataması yapıp ekranda gösteriniz.

Çözüm:

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

class Deneme  //Deneme sınıfı
{
public:
        int a, b;
        Deneme()  // Varsayılan Yapıcı
        {
                a = 10;
                b = 20;
        }
};

int main()
{
    Deneme d1;
    cout << "a: " << d1.a << endl << "b: " << d1.b << endl;
    return 0;
}
```

```
Ekran çıktısı:
a : 10
b : 20
```

Örnekte *Deneme* adında bir sınıf oluşturulmuş ve bu sınıfa varsayılan yapıcı kullanılarak sınıf içinde tanımlanan *a* ve *b* üye değişkenlerine ilk değer ataması yapılmıştır. Örnekte görüldüğü gibi yapıcı sınıf adı ile aynıdır ve değişkenlere ilk değer ataması için kullanılır. Varsayılan yapıcı, deneme sınıfından yeni bir nesne oluşturulduğunda otomatik olarak çağrılır [6]. Örnekte görüldüğü gibi *a* ve *b* isimli üye değişkenlere ilk değer ataması sınıf adı ile aynı ada sahip olan *Deneme* yapıcı fonksiyonu ile yapılmıştır.

Parametreli Yapıcılar

Parametreleri yapıcılara iletmek mümkündür. Genellikle bu parametreler oluşturulduğunda bir nesneyi başlatmaya yardımcı olur. Parametreli bir yapıcı oluşturmak için başka herhangi bir fonksiyona yaptığımız gibi yapıcı fonksiyona parametre eklenebilir.

Aşağıdaki örnekte *Koordinat* sınıfı oluşturulduktan sonra *x* ve *y* üye değişkenleri tanımlanmıştır. Parametreli yapıcıya iki adet parametre gönderilerek üye değişkenlere değer ataması yapılmıştır. *XGetir* ve *YGetir* üye fonksiyonları ise *x* ve *y* üye değişkenlerinin değerlerini geri döndürmektedir. *Koordinat* sınıfına ait nesne oluşturulurken parametre olarak iki değer gönderilerek yapıcı fonksiyon çağrılır [6].

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

class Koordinat
{
private:
	int x, y;
public:
	// Parametreli Yapıcı
	Koordinat(int x1, int y1) // Yapıcı parametre alıyor
	{
		x = x1;
		y = y1;
	}
	int XGetir()
	{
		return x;
	}


	int YGetir()
	{
		return y;
	}
};

int main()
{
	//Yapıcı çağrılıyor
	Koordinat k1(10, 15);
	// Yapıcı tarafından atanan erişim değerleri
	cout << "k1.x = " << k1. XGetir() << ", k1.y = " << k1. YGetir ();

	return 0;
  }
```

Ekran Çıktısı:

k1.x = 10, k1.y = 15

Yıkıcı Fonksiyon

Yapıcı fonksiyon bir nesne oluştuğunda otomatik olarak çağrılır ve ilgili nesnenin üye değişkenlerine ilk değerlerini atar. Değer ataması yapıldığı için bellekte buna yer ayrılır. Bellekte ayrılan yeri nesneyle işimiz bittiğinde, yıkıcı fonksiyon serbest bırakır. Yani yapıcı fonksiyon nesne oluştuğunda çağrılmakta, ilgili özelliklere ilk değerlerini atamakta ve böylece onu kullanıma hazırlamaktadır. Yıkıcı (Destructor) fonksiyon ise nesneyle işimiz bittiğinde ona yapıcı ile ayrılan bellek alanını boşaltmaktadır. Yıkıcı fonksiyonun özellikleri ise aşağıdaki gibidir:

Yıkıcı fonksiyonun adı sınıf adı ile aynıdır ve tilda(~) ile başlar.

- Yıkıcı fonksiyon adı yapıcı fonksiyon ve sınıf adı ile aynıdır.
- Yıkıcı fonksiyon tilda (~) ile başlar.
- Geriye değer döndürmez.
- Parametre almaz.
- Bellek alanını boşaltır.

Aşağıda oluşturulan *Test* sınıfında yapıcı ve yıkıcı fonksiyonlar tanımlanmıştır. Öncelikle sınıf adıyla aynı olan *Test* yapıcı fonksiyonu çağrılmış ve *sayi* değişkenine ilk değer ataması yapılmıştır. Yıkıcı fonksiyon ise işlem bittikten sonra çağrılmış ve bellek alanı boşaltılmıştır. Kullanılan *Goruntule* adındaki üye fonksiyon ile de ekrana *sayi* değeri yazdırılmıştır.

```
class Test{
        int sayi;
 public:
        //yapıcı fonksiyon
        Test()
        {
                cout << "Yapıcı fonksiyonun icindesiniz." << endl;
                sayi = 5;
        }
        //yıkıcı fonksiyon
        ~Test()
        {
                cout << "Yıkıcı fonksiyon içindesiniz" << endl;
        }
        //üye fonksiyon
        void Goruntule()
        {
                cout << "sayi degeri:" << sayi << endl;
        }
  };
int main() {
        //Test sınıfından t1 adında yeni bir nesne oluşturuldu
        Test t1;
        t1.Goruntule();
```

```
        return 0;
}
```

Ekran Çıktısı:

```
Yapıcı fonksiyonun icindesiniz.
sayi degeri:5
Yıkıcı fonksiyonun icindesiniz.
```

Kapsam Çözünürlük Operatörü

Kapsam çözünürlük operatörü (::) ile sınıf tanımı dışında üye fonksiyonlara ve üye değişkenlere erişilebilir.

Bazı sınıflar çok sayıda fonksiyon içerebilir ve bu nedenle sınıfın anlaşılırlığı azalır. Böyle bir durumda, fonksiyonları kapsam çözünürlük operatörünü "::" kullanarak sınıf tanımının dışına çıkarma yolu tercih edilir. *Kapsam Çözünürlük Operatörü (Scope Resolution Operator),* bir programcının metotları başka bir yerde tanımlamasına izin verir. Operatörün sembolü "::" dir. Kapsam çözünürlük operatörü başka bir kapsam içerisindeki bir değişkene ya da fonksiyona erişilmesini sağlar.

Kapsam çözünürlük operatörünün kullanımı aşağıdaki örnekte gösterilmiştir. *Programlama* adında bir sınıf oluşturulduktan sonra *Yaz* isimli üye fonksiyonu tanımlanmıştır. Ancak *Yaz* üye fonksiyonunun içeriği sınıf dışında yazılmıştır. *Programlama::Yaz()* kullanılarak kapsam çözünürlük operatörü(::) sayesinde üye fonksiyonu sınıfın dışında tanımlanmıştır.

```
class Programlama {
public:
        void Yaz();  // üye fonksiyon tanımlandı ama içeriği sınıf dışında
yazılacak
  };
  //üye fonksiyonları sınıfın dışında kullanmak için
  //kapsam çözünürlük operatörünü (::) kullanabiliriz.
void Programlama::Yaz()
{
        cout << "Yaz fonksiyonu sınıf dışında kullanıldı\n";
}

int main() {
        Programlama p; // sınıftan yeni bir nesne oluştu
        p.Yaz(); //nokta operatörü ile yaz fonksiyonu çağrıldı

        return 0;
}
```

Kapsam çözünürlük operatörü ile farklı ad uzaylarındaki ifadelere de ulaşılabilir. Örneğin, *std* ad uzayından nesneleri kullanmak için kapsam çözünürlük operatörü kullanılabilir. *using namespace std;* kullanmadan main içerisinde *cout* kullanmak için aşağıdaki örnekte olduğu gibi kapsam çözünürlük operatöründen *(::)* faydalanabiliriz:

```
#include <iostream>

int main() {
	std::cout << "c++";
	return 0;
}
```

Nokta Operatörü

Bir sınıfa ait özelliklere ve metotlara erişebilmek için nokta operatörü (.) kullanılmaktadır. Aşağıda nokta operatörünün kullanımı görülmektedir. *main* içerisinde öncelikle *Ornek* sınıfına ait *o1* nesnesi oluşturulmuştur. *o1* nesnesi ile *nokta operatörü* kullanarak sınıf içerisindeki üye değişkenlere ve üye fonksiyonlara erişilebilmiştir. Örnekte görüldüğü gibi *nokta operatörü* kullanılarak sınıf içerisindeki *sayi1* değişkenine ve *Yaz* fonksiyonuna erişilmiştir:

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

class Ornek{
public: //özelliklere ve fonksiyona erişilebilir
	int sayi1;
	int sayi2;
	//Yaz fonksiyonu
	void Yaz(){
		cout << "Merhaba";
	}

};
int main(){
	Ornek o1;
	o1.sayi1=7; //nokta operatörü kullanılarak sınıf içerisindeki sayi1 değişkenine erişildi
	o1.Yaz();//nokta operatörü kullanılarak yaz fonksiyonuna erişildi
}
```

Uygulamalar

Uygulama 1: Yarıçapı girilen bir dairenin alanını nesneye yönelik programlama kavramları ile çözünüz.

```
#include<iostream>
using namespace std;
// Daire sınıfı oluşturuldu
class Daire {
public:
	const double pi = 3.14; //pi sabiti tanımlandı
	double yaricap;
```

```
        double Alan(double yaricap)
        {
                return pi*yaricap*yaricap; //dairenin alanını hesaplar
        }
};
int main()
{
        Daire d1; // daire sınıfından bir nesne oluşturuldu
        cout << "yaricap giriniz:";
        cin >> d1.yaricap;
        cout << "Alan=" << d1.Alan(d1.yaricap) << endl;

        return 0;
}
```

Ekran Çıktısı:

```
yaricap giriniz:3
Alan=28.26
```

Bu uygulamada öncelikle *Daire* sınıfı oluşturulmuştur. *pi* sabiti 3.14 olarak ve *yaricap* adındaki üye değişkeni de *public* olarak tanımlanmıştır. *Alan* adındaki üye fonksiyonu dairenin alanını hesaplayıp bu değeri geri döndürmektedir. *main* içerisinde Daire sınıfından *d1* adında yeni bir nesne oluşturulmuştur. Yariçap değeri kullanıcıdan istenmiştir ve nokta operatörü sayesinde alan hesaplanıp ekrana yazdırılmıştır.

Uygulama 2: Dikdörtgenin kısa ve uzun kenarını parametre olarak alarak ve sınıf yapısını kullanarak dikdörtgenin alanını hesaplayınız

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

class Dikdortgen {
        int kisaKenar, uzunKenar;
public:
        void SetValues(int, int);
        int Alan()
        {
                return kisaKenar*uzunKenar;
         }
};

void Dikdortgen::SetValues(int x, int y) {
        kisaKenar = x;
        uzunKenar = y;
}
```

```
int main() {
        Dikdortgen dikt;
        dikt.SetValues(3, 4);
        cout << "Alan: " << dikt.Alan() << endl;
        return 0;
}
```

Ekran Çıktısı:

```
Alan:12
```

Bireysel Etkinlik

- Dikdörtgenin çevresini ve alanını sınıf yapısı kullanarak hesaplayıp ekrana yazdıran C++ programını yazınız.
- Silindirin hacmini sınıf yapısını kullanarak hesaplayınız.
- Kullanıcının parametre olarak girdiği alt limit ve üst limit arasındaki sayıların toplamını sınıf yapısını kullanarak hesaplayınız.

Özet

- Bu ünite kapsamında nesneye yönelik programlamadan bahsedilmiş ve C++ kullanılarak nesneye yönelik programlama kavramları tanıtılmıştır.
- Öncelikle nesne ve sınıf kavramları anlatılmış ve bu kavramların kullanımları gösterilmiştir. Nesneye yönelik programlama tekniğinin en küçük yapı taşı olan nesne kavramından bahsedilmiştir. Gerçek hayattaki bir nesneyi istenen bir şekilde modelleyebilmek için gerekli veriyi ve fonksiyonları içeren yazılım birimi olan sınıf yapısından bahsedilmiştir. Tüm nesneler birer sınıf örneğidir. Sınıflar nesnenin özelliklerini belirler. Nesne odaklı yaklaşımla geliştirilen bir yazılım temel olarak, gerekli görevleri tamamlamak için mesajlar aracılığıyla birbirleriyle iletişim kuran bir sınıf koleksiyonudur. Sınıf yapısı, class anahtar kelimesi tanımlanır.
- Bu ünitede sınıf tanımı içerisinde kullanılan üye değişkenlerden ve üye fonksiyonlardan bahsedilmiştir. Sınıf tanımlaması içinde bildirilen değişkenlere üye değişkenler, bildirilen fonksiyonlara da üye fonksiyonlar denir. Üye fonksiyonlar, nesnelerin içinde tanımlanmış yöntemler veya işlevlerdir. Genellikle üye değişkenler üzerinde çalışmak için kullanılırlar. Bir sınıfın üye fonksiyonu, sınıf içindeki üye değişkenlere sahip olan bir metottur. Üye fonksiyon, üyesi olduğu sınıfın herhangi bir nesnesinde çalışır ve bu nesnenin ait olduğu sınıfının tüm üyelerine erişim izni vardır. Üye değişkenler parametre alabilir ve geriye bir sonuç döndürebilir.
- Bu ünitede üyelere ulaşım belirteçleri olan public ve private örneklerle anlatılmıştır. Sınıf içerisinden ve sınıf dışından kolaylıkla erişilebilen belirtecin public, sadece sınıf içerisinden erişilebilen belirtecin de private olduğundan bahsedilmiştir.
- Bu ünitede nesnelerin ilk kullanıma hazırlanması için gerekli yapıcı (constructor) fonksiyonlardan, bu fonksiyonların özelliklerinden ve varsayılan yapıcı ve parametreli yapıcı kavramlarından bahsedilmiştir.Yapıcı fonksiyonun adının sınıf adı ile aynı olduğundan, geriye değer döndürmediğinden ve parametre alabileceğinden bahsedilmiştir. Nesneyle işimiz bittiği zaman yapıcı fonksiyonun kullandığı bellek alanını boşaltmak için kullanılan yıkıcı (destructor) fonksiyondan ve özelliklerinden bahsedilmiştir.Yıkıcı fonksiyonun adı yapıcı fonksiyon ve sınıf adı ile aynı olmalıdır. Yıkıcı fonksiyon tilda (~) ile başlar ve geriye değer döndürmez.
- Bu ünitede sınıfa ait üye fonksiyonlara ve üye değişkenlere sınıf dışından da erişilebileceğinden, bunun kapsam çözünürlük operatörü (::) ile yapılabileceğinden bahsedilmiştir.Kapsam çözünürlük operatörü, bir programcının metotları başka bir yerde tanımlamasına izin vermektedir.
- Bu ünitede son olarak sınıfa ait üye fonksiyonlara ve üye değişkenlere nokta operatörü kullanılarak erişilebileceğinden bahsedilmiştir.

DEĞERLENDİRME SORULARI

1. Nesneye yönelik programlamanın aşağıdakilerden hangisi özelliklerinden değildir?
   a) Nesneler birbirleriyle iletişim kurabilir.
   b) Yeni veri ve fonksiyonlar gerektiğinde kolayca eklenebilir.
   c) Bir nesnenin birden çok örneğine sahip olmak mümkündür.
   d) Sınıf yapısı olmadan nesne oluşturulabilir.
   e) Nesnelerin ilk kullanıma hazırlanması için yapıcı fonksiyon kullanılır.

2. Sınıf ve nesne yapısı için aşağıdakilerden hangisi söylenemez?
   a) Sınıf adı ile nesne adı aynı olmalıdır.
   b) Nesne ait olduğu sınıfın bir örneğidir.
   c) Sınıf yapısı class anahtar kelimesi ile başlar.
   d) Sınıf içerisinde üye değişkenler ve üye fonksiyonlar tanımlanır.
   e) Nesneler arası ortak özellik ve metotlar olabilir.

3. Sınıf içerisinden ve sınıf dışından değişken ve fonksiyonlara ulaşmamızı sağlayan belirteç aşağıdakilerden hangisidir?
   a) class
   b) private
   c) yapıcı
   d) public
   e) iostream

4. Yapıcı fonksiyonlar için aşağıdakilerden hangisi söylenemez?
   a) Yapıcı fonksiyon adı sınıf adı ile aynı olmalıdır.
   b) Yapıcı fonksiyon geriye değer döndürebilir.
   c) Yapıcı fonksiyon parametre alabilir.
   d) Yapıcı fonksiyon bir nesne oluştuğunda özelliklere ilk değer ataması için kullanılır.
   e) Nesnenin oluşması sırasında ilk çalışan fonksiyon yapıcı fonksiyondur.

5. Yıkıcı fonksiyonlar için aşağıdakilerden hangisi söylenemez?
   a) Yıkıcı fonksiyon adı sınıf adı ile aynı olmalıdır.
   b) Yıkıcı fonksiyon tilda (~) ile başlar.
   c) Yıkıcı fonksiyon geriye değer döndürmez.
   d) Yıkıcı fonksiyon parametre alabilir.
   e) Yıkıcı fonksiyon bellek alanını boşaltır.

6.
```
#include <iostream>
using namespace std;

class deneme {
public:
        int sayi;
        void Yaz();
};

void programlama::Yaz() {
        cout << "Yaz fonksiyonu kullanıldı\n";
}

int main() {
        deneme d1;
        d1.Yaz();

        return 0;
}
```

Yukarıdaki C++ programı için aşağıdakilerden hangisi söylenemez?

a) Yaz adında üye fonksiyonu tanımlanmıştır.
b) sayi adında üye değişkeni tanımlanmıştır.
c) Üye fonksiyonun yapacağı işlem sınıf içerisinde tanımlanmıştır.
d) d1 adında nesne oluşturulmuştur.
e) Nokta operatörü kullanılmıştır.

7.
```
#include <iostream>
using namespace std;

class deneme{
public:
	int sayi1;
	int sayi2;
	void Yaz(){
		cout << "c++ ile programlama temelleri";
	}

};

int main(){
	deneme d1;
	d1.sayi1 = 7;
	d1.sayi2 = 8;
	d1.Yaz();
	return 0;
}
```

Yukarıdaki C++ programı çalıştırıldığında aşağıdaki ekran çıktılarından hangisi oluşur?

a) c++ ile programlama temelleri
b) c++ ile programlama temelleri78
c) 7
d) 8
e) Yaz

8.
```
#include <iostream>
using namespace std;

class Calisan{
public:
        int maas;
        Calisan(){
                maas = 1000;
        }
        void ZamYap(int miktar){
                maas += maas;
                cout << maas << endl;
        }
};

int main(){

        Calisan c1;
        c1.ZamYap(1000);
         return 0;
}
```

Yukarıdaki C++ programı çalıştırıldığında aşağıdaki ekran çıktılarından hangisi oluşur?

a) 1000
b) maas = 1000
c) 2000
d) ZamYap(1000)
e) maas

9. Üye fonksiyonlarını sınıf dışında tanımlamak için aşağıdaki operatörlerden hangisi kullanılmalıdır?
   a) #
   b) =
   c) .
   d) ;
   e) ::

10. Nesneye yönelik programlamanın aşağıdakilerden hangisi önemli yapı taşlarından değildir?
   a) Kalıtım
   b) Çok biçimlilik
   c) Sınıf
   d) Nesne
   e) Döngüler

Cevap Anahtarı

1.d, 2.a, 3.d, 4.b, 5.d, 6.c, 7.a, 8.c, 9.e 10.e

YARARLANILAN KAYNAKLAR

[1] Algan, S. (2013). *Her Yönüyle C#*. Pusula Yayıncılık.

[2] Smith, B. (2011). Object-Oriented Programming. In: AdvancED ActionScript 3.0: Design Patterns. Apress

[3] Balagurusamy, E. (1995). *Object Oriented Programming with C++,* Tata McGraw, Hill Publishing Company Ltd, New Delhi, India.

[4] Kamthane, A. (2006). Object-oriented programming with Ansi & Turbo C++. Upper Saddle River, NJ.: Pearson Education.

[5] Java Programming Tutorial, Object-oriented Programming (OOP) Basics, (2018) 10 Temmuz 2018 tarihinde https://www3.ntu.edu.sg/home/ehchua/programming/java/J3a_OOPBasics adresinden erişildi.

[6] GeeksForGeeks, Constructors in C++ (2018), 10 Temmuz 2018 tarihinde https://www.geeksforgeeks.org/constructors-c/ adresinden erişildi.