

# C++ Programlama Dili, Algoritmalar ve Veri Yapıları



Öğr. Gör. Yük. Müh. Gökşin Akdeniz

2024, Eskişehir

Bu kitap Creative Commons BY – NC – SA – 4.0 Lisansı İle Yayınlanmıştır.



# İçindekiler

Giriş.....	7
0.Bilgisayarlar, Programlama ve Programlama Dilleri.....	9
0.1.Bilgisayarın Tarihsel Gelişimine Kısa Bir Bakış.....	9
0.2.Bilgisayarın Temel Bileşenleri.....	14
0.2.1.Donanım.....	14
0.2.1.1.Merkezi İşlem Birimi – CPU.....	14
0.2.1.2.Birincil Bellek.....	14
0.2.1.3.İkincil Depolama.....	15
0.2.1.4.Girdi – Çıktı Aygıtları.....	15
0.2.2.Yazılım.....	15
0.3.Bilgisayar Dili.....	16
0.4.Derleyiciler.....	18
0.5.Geliştirme Araçları.....	18
0.5.C++ Programlama Dili İçin Gerekli Geliştirme Araçları.....	19
0.5.1.Windows için Code::Blocks Kurulumu.....	19
0.5.2.GNU/Linux Dağıtımları İçin C++ Geliştirme Araçları.....	22
0.5.3.*BSD İşletim Sistemi Ailesi İçin C++ Geliştirme Araçları.....	23
0.5.4.Geany İle İnşa Komutları Yapılandırması.....	24
0.5.5.C++ Programlarının İnşa Edilmesi.....	25
0.6.Çözümle – Kodla – Çalıştır Tekniği ile Programlama.....	27
0.7.Programlama Paradigmaları.....	28
0.7.1.Yapısal Programlama – Structured Programming.....	28
0.7.2.Nesne Yönelimli Programlama – Object Oriented Programming.....	28
1.C++ Programlama Dilinin Temel Bileşenleri.....	29
1.1.C++ Programlama Dili İle Yazılan Programların Temel Özellikleri.....	29
1.1.1.Lisans.....	33
1.1.2.Yorumlar.....	33
1.1.3.Özel Karakterler.....	34
1.1.4.Özel Sözcükler.....	35
1.1.5.Tanımlayıcılar.....	36
1.1.6.Boşluk.....	37
1.2.Veri Tipleri.....	37
1.2.1. Basit Veri Tipleri.....	37
1.2.1.1.int Veri Tipi.....	38
1.2.1.1.1.boole Veri Tipi.....	38
1.2.1.1.2.char Veri Tipi.....	38
1.2.1.2.Kayar Noktalı Sayılar Veri Tipi (Floating-Point Data Types).....	39
1.2.1.2.1.float Veri Tipi.....	39
1.2.1.2.2.double Veri Tipi.....	39
1.2.1.2.3.long double Veri Tipi.....	39
1.3.Veri Tipleri, Değişkenler ve Atama İfadeleri.....	43
1.4.Aritmetik İşlemler, İşlem Öncelikleri ve İfadeler.....	44
1.4.1.İşlem Önceliği – İşlem Hiyerarşisi.....	47
1.4.1.İfadeler.....	51
1.4.2. Karma İfadeler.....	51
1.4.3.Veri Tipleri Arasındaki Dönüşümler.....	54
1.5.Karakter Dizisi Veri Tipi – String Type.....	58
1.6.Değişkenler, Atama ve Girdi İfadeleri.....	59

1.6.1.Bellekte Sabitler ve Değişkenler ile Alan Ayrılması.....	60
1.6.1.1.Sabitlerin İsimlendirilmesi.....	60
1.6.1.2.Değişkenler.....	61
1.6.2.Değişkenlere Değer Atanması.....	62
1.6.3.Atama Bildirimleri.....	62
1.6.4.Bir İfadeden Sonucunun Saklanması ve Çağrılması.....	67
1.6.5.Değişkenlerin Tanımlanması ve Etkinleştirilmesi.....	68
1.6.6.Girdi (Okuma) Bildirimleri.....	69
1.6.7.Değişkenlerin Etkinleştirilmesi.....	74
1.7.Arıtma ve Azaltma Operatörleri.....	77
1.8.Çıktı İşlemleri.....	78
1.9.Önişlemci Yönergeleri.....	85
1.9.1.Programlarda namespace, cin ve cout Kullanımı.....	85
1.9.2.Programlarda Karakter Katarı Veri Tipinin Kullanımı.....	86
1.10.C++ Programlama Dili Program Yazım Kuralları.....	86
2.Girdi ve Çıktı İşlemleri.....	89
2.1.G/Ç Akımları ve Standart G/Ç Aygıtları.....	89
2.1.1.cin ve >> Operatörü.....	90
2.2.Ön Tanımlı Fonksiyonlar.....	93
2.2.1.cin ve get.....	95
2.2.2.cin ve ignore Fonksiyonu.....	96
2.2.3.putback ve peek Fonksiyonları.....	100
2.2.4.G/Ç Akım ve G/Ç Fonksiyonları Arasındaki Fark.....	103
2.3.Girdi Hataları.....	103
2.3.1.clear Fonksiyonu.....	104
2.4.Çıktı ve Çıktının Biçimlendirilmesi.....	108
2.4.1.setprecision Biçimlendiricisi.....	108
2.4.2.fixed ve scientific Biçimlendiricisi.....	108
2.4.3.showpoint Biçimlendiricisi.....	111
2.4.4.setw Biçimlendiricisi.....	114
2.4.5.setfill Biçimlendiricisi.....	116
2.4.6.left ve right Biçimlendiricisi.....	119
2.5.Girdi/Çıktı işlemlerinde Karakter Dizileri.....	122
2.6.Dosyaları Kullanarak Girdi/Çıktı İşlemleri.....	124
3.Kontrol Yapıları: Seçimler.....	129
3.1.“if” Ve “if ... else” İle Seçim Yapılması.....	130
3.1.1. Basit Veri Tipleri ve İlişkisel Operatörler.....	130
3.1.2.Karakterlerin Karşılaştırılması.....	131
3.1.3.Tek Seçenekli Bildirimler.....	131
3.1.4.Çift Seçenekli Bildirimler.....	135
3.1.5.Tam Sayı Veri Tipi ve Mantıksal İfadeler.....	139
3.1.6.Mantıksal Veri Tipi ve Mantıksal İfadeler.....	141
3.1.7.Mantıksal Operatörler ve Mantıksal İfadeler.....	144
3.1.8.Mantıksal Operatörlerde İşlem Öncelikleri.....	146
3.2.İlişkisel Operatörler ve Karakter Dizileri.....	152
3.2.1.Bileşik Bildirimler ve İfadeler.....	153
3.2.2.Çoklu Seçimler.....	154
3.2.3.Kayar Noktalı Sayılarda Eşitlik Kontrolü.....	156
3.2.4.Girdi Hatalarının if ile Tespit edilmesi.....	159
3.3.Switch ... Case ... Yapısı.....	170

3.4.Assert ile Programda Hata Yakalama.....	175
4.Kontrol Yapıları: Döngüler.....	180
4.1.Döngüler neden kullanılır?.....	180
4.2.while Döngü Yapısı.....	183
4.2.1.while Döngüsünün Tasarımı.....	189
4.2.1.1.Sayaç ile Kontrol Edilen while() Döngüleri.....	189
4.2.1.2.Bitiş Sözcüğü ile Kontrol Edilen while() Döngüleri.....	192
4.2.1.3.Durum ile Kontrol Edilen while() Döngüleri.....	195
4.2.1.4.Dosya Sonu – EOF Bildirimi ile Kontrol Edilen while() Döngüleri.....	198
4.3.do ... while Döngü Yapısı.....	200
4.4.for Döngü Yapısı.....	208
4.5.break ve continue İfadeleri.....	221
4.6.İç İçé Kurgulanan Döngüler.....	225
5.Programcı Tarafından Tanımlanan Fonksiyonlar.....	251
5.1.Ön Tanımlı – Hazır – Fonksiyonlar.....	252
5.2.Programcı Tarafından Tanımlanan Fonksiyonlar.....	261
5.3.Değer Döndüren Fonksiyonlar.....	261
5.3.1.Değer Döndüren Fonksiyonların Yazım Kuralları.....	262
5.3.2.Formal Parametrelerin Yazım Kuralları.....	262
5.3.3.Fonksiyonların Çağrılması ve Asıl Parametreler.....	263
5.3.4.Fonksiyonların Değer Döndürmesi: return.....	264
5.3.5.Prototip Fonksiyonlar.....	268
5.3.6.Değer Döndüren Fonksiyonlarda Önemli Konular.....	271
5.3.7.Programın Derlenmesi ve Program İşleyışı.....	281
5.4.Void – Değer Döndürmeyen Fonksiyonlar.....	281
5.5.Değer Parametreleri.....	285
5.6.Referans (Değişkenin Bellek Adresi) Parametreleri.....	287
5.7.Değer ve Referans Parametreleri ile Bellek Atamaları.....	293
5.8.Referans Parametreleri ve Değer Döndüren Fonksiyonlar.....	303
5.9.Değişkenlerin ve Tanımlayıcıların Kapsamı.....	303
5.10.Genel Değişkenler, Sabitler ve İstenmeyen Yan Etkiler.....	307
5.11.Statik ve Otomatik Değişkenler.....	317
5.12.Fonksiyonların Ön Tanımlı Parametreleri.....	321
5.13.Fonksiyonlarda Aşırı Yükleme – Overloading.....	326
6.Programcı Tarafından Tanımlanan Basit Veri Tipleri, İsim Uzayları ve string Veri Tipi.....	350
6.1.Sıralama – Enumeration – Veri Tipi.....	350
6.1.1.Değişkenlerin Tanımlanması.....	351
6.1.2.Atamalar.....	352
6.1.3.Sıralama Veri Tipi Üzerinde Gerçekleştirilen İşlemler.....	352
6.1.4.Sıralama Veri Tipi Üzerinde İlişkisel Operatör İşlemleri.....	353
6.1.5.Sıralama Veri Tipi ile Girdi/Çıktı İşlemleri.....	353
6.1.6.Sıralama Veri Tipi ve Fonksiyonlar.....	356
6.1.7.Sıralama Veri Tipi Kullanarak Değişken Tanımlamaları.....	357
6.1.8.Anonim Veri Tipleri.....	357
6.1.9.typedef Bildirimi.....	358
6.2.İsim Uzayları – Namespaces.....	358
6.3.string – Karakter Katarı Veri Tipi.....	361
6.3.1.Karakter Katarı Üzerinde Gerçekleştirilen Diğer İşlemler.....	365
7.Diziler ve string Veri Tipi.....	380
7.1.Diziler.....	381

7.1.1.Dizi Elemanlarına Erişim.....	381
7.1.2.Tek Boyutlu Diziler Üzerinde İşlemler.....	383
7.1.3.Dizilerde Sınır Kontrolü.....	389
7.1.4.Dizilerin Etkinleştirilmesi.....	389
7.1.5.Dizilerin Kısmen Etkinleştirilmesi.....	390
7.1.6.Diziler İle İlgili Bazı Kısıtlamalar.....	391
7.1.7.Fonksiyonlarda Dizilerin Parametre Olarak Kullanımı.....	392
7.1.8.Dizilerin Taban Adresi – Base Address.....	396
7.1.9.Dahili Veri Tipleri ve Dizinin İndisi.....	397
7.1.10.Dizilerin Tanımlanması için Kullanılan Diğer Yöntemler.....	397
7.2.Dizilerde Arama ve Sıralama İşlemleri.....	398
7.2.1.Seçerek Sıralama.....	401
7.3.for Döngülerinde Döngü Sayısının Otomatik Belirlenmesi.....	408
7.4.Karakter Katarı Dizileri – C-Strings.....	409
7.4.1.İki Dizinin Karşılaştırılması.....	412
7.4.2.Karakter Katarı Dizilerinin Okunması ve Yazılması.....	413
7.4.3.Karakter Katarı Dizilerinde Girdi İşlemleri.....	413
7.4.4.Karakter Katarı Dizilerinde Çıktı İşlemleri.....	414
7.4.5.Çalışma Zamanında G/C İşlemleri için Dosya Kullanımı.....	415
7.4.6.Çalışma Zamanında G/C İşlemleri için Dosya İsimlerinde Karakter Katarı Kullanımı	415
7.5.Paralel Diziler.....	416
7.6.İki ve Üzeri Boyutlu Diziler.....	417
7.6.1.İki Boyutlu Dizilerde Elemanlara Erişim.....	419
7.6.2.İki Boyutlu Dizilerin Etkinleştirilmesi.....	420
7.6.3.İki ve Üzeri Boyutlu Dizilerde Sıralama Veri Tipi Kullanımı.....	420
7.6.4.İki Boyutlu Dizilerde Etkinleştirme.....	422
7.6.5.İki Boyutlu Dizilerde Elemanların Yazdırılması.....	423
7.6.6.İki Boyutlu Dizilerde Elemanların Atanması.....	423
7.6.7.İki Boyutlu Dizilerde Satırların Toplanması.....	423
7.6.8.İki Boyutlu Dizilerde Sütunların Toplanması.....	424
7.6.9.İki Boyutlu Dizilerde Satır ve/veya Sütunlardaki En Büyük Değerin Bulunması.....	424
7.6.10.İki Boyutlu Dizilerin Etkinleştirilmesi İçin Diğer Yöntemler.....	426
7.6.11.İki Boyutlu Dizilerin Fonksiyonlara Parametre Olarak Aktarılması.....	427
7.6.12.Karakter Dizileri.....	431
7.6.13.Karakter Dizileri ve Dizi Kullanımı.....	432
7.6.14.Karakter Katarı Dizileri ve Dizi Kullanımı.....	432
7.6.15.Çok Boyutlu Diziler.....	433
8.Birden Çok Çeşitteki Veriyi Barındıran Yapılar (struct).....	435
8.1.struct (record).....	435
8.1.1.struct ile Tanımlanan Yapıdaki Üyelere Erişim.....	438
8.1.2.struct ile Tanımlanan Yapıdaki Üyelere Atama Yapmak.....	439
8.1.3.struct ile Tanımlanan Yapıdaki Üyeler Arasında İlişksel Operatör İşlemleri.....	440
8.1.4.struct ile Tanımlanan Yapıda Girdi/Çıktı İşlemleri.....	441
8.1.5.struct ile Tanımlanan Yapıda Değişkenler ve Fonksiyonlar.....	441
8.1.6.struct ile Tanımlanan Yapı ve Dizilerin Karşılaştırılması.....	443
8.1.7.struct ile Tanımlanan Yapıda Dizilerin Kullanılması.....	443
8.1.8.Dizilerde struct ile Tanımlanan Yapı Kullanımı.....	446
8.1.9.struct ile Tanımlanan Yapıların İçerisinde struct ile Yapı Tanımlanması ve Kullanımı	448
9.Sınıflar ve Veri Soyutlama.....	473
9.1.Sınıflar.....	473

9.1.1.UML İle Sınıf Diagramları.....	477
9.1.2.Nesnelerin (Değişkenlerin) Tanımlanması.....	477
9.1.3.Sınıf Üyelerine Erişim.....	478
9.1.4.Sınıflar Üzerinde İşlem Yapan Hazır Fonksiyonlar.....	479
9.1.5.Sınıflar Üzerinde Hazır Fonksiyonlar İle Atama İşlemleri.....	479
9.1.5.Sınıfin Kapsamı.....	480
9.1.6.Fonksiyonlar ve Sınıflar.....	480
9.1.7.Sınıfların Nesneleri ve Referans Parametreleri.....	480
9.1.8.Üyelik Fonksiyonlarının Uygulanması.....	482
9.1.9.Erişim (Accessor) ve Değişim (Mutator) Fonksiyonları.....	487
9.1.10.Bir Sınıfta Özel ve Genel Üyelerin Sıralanması.....	494
9.1.11.Yapıcı Fonksiyonlar.....	496
9.1.12.Yapıcı Fonksiyonların Çağrılması.....	499
9.1.13.Ön Tanımlı Yapıcı Fonksiyonların Çağrılması.....	499
9.1.14.Yapıcı Fonksiyonların Parametre İle Çağrılması.....	499
9.1.15.Yapıcı Fonksiyonlar ve Ön Tanımlı Parametreler.....	502
9.1.16.Yapıcı Fonksiyonlar ve Sınıflar İçin Uyarılar.....	503
9.1.17.Sınıf Üyelerin Etkinleştirilmesi ve Ön Tanımlı Yapıcı Fonksiyonlar.....	504
9.1.18.Sınıf Üyesi Olarak Diziler ve Yapıcı Fonksiyonlar.....	505
9.1.19.Yıkıcı Fonksiyonlar.....	508
9.2.Veri Soyutlama, Soyut Veri Tipi ve Sınıflar.....	508
9.2.1.Yapılar ve Sınıflar.....	511
9.3.Başlık Dosyaları.....	511
9.4.Sınıfin Statik Olarak Tanımlanan Üyeleri.....	529
10.Kalıtım ve Birleşim.....	556
10.1.Kalıtım.....	556
10.1.1.Temel Sınıfın Üyesi Olan Fonksiyonların Yeniden Tanımlanması.....	559
10.1.2.Türetilmiş ve Temel Sınıfların Yapıcı Fonksiyonları.....	566
EK A: ASCII Tablosu.....	567

# Giriş

Günümüzde çeşitli programlama dilleri kullanılmaktadır. Bunlardan bir tanesi de C++ dilidir. Programlama dilleri arasında C++ bazı özellikleri ile diğerlerinden ayrılmaktadır. C++ öncelikle plana çıkışmasına neden olan özellikleri arasında C programclarının kolaylıkla geçiş yapabilmesi, uluslararası standartlar tarafından tanımlanmış olması yanında nesne yönelimli programlama paradigmاسını da desteklemesidir. Bir çok programlama dilinin temel aldığı bir standartı bulunmamaktadır. Bu eksiklik genellikle programlama dilini geliştiren geliştiriciler ekibinin sorumluluk alması ile giderilmektedir. Bu durum dilin geliştirilmesi sırasında farklı geliştirici ekiplerinin kendi tercihleri doğrultusunda çeşitliliği farklılarından gelişmesine neden olabilmektedir. C++ dili de benzer bir süreçten geçmiş olsa da daha sonra ISO bünyesinde kurulan bir standart belirleme komitesi tarafından hem dilin geliştiricileri de hem de ilgili tarafları temsil eden temsilcilerin bir araya gelerek çalışması ile kabul görmüş bir standardın desteklenmesi ile birlik sağlanmaktadır.

Ayrıca C++ programlama dili aynı zamanda hem C gibi düşük seviyeli çalışabilmeyi sağlarken yüksek seviyeli dillerin desteklediği nesne yönelimli programlama paradigmاسını da desteklemektedir: Böylece programçılara esneklik sağladığı gibi aynı zamanda hızlıca yazılım geliştiricilerini de sağlamaktadır. Bugün bir çok yazılım projesi farklı işletim sistemleri üzerinde C++ ile yazılmış olan tek bir kod ağacı üzerinden yürümektedir. Böylece yazılım geliştirme sürecinde programçının karşı karşıya kaldıkları farklı işletim sistemleri ve/veya platformlar için aynı yazılımın değişik kod ağaçları üzerinden geliştirilmesinin önüne geçilmesini sağlamaktadır.

Bu ders notları Veri Yapıları ve Programlama ile Algoritmalar ve Programlamaya Giriş dersleri için kullanılmak üzere hazırlanmıştır. Ders notlarının Seçilen örnekler C++ dilini bilmeyen birisinin kısa sürede dilin inceliklerini kavrayarak kolaylıkla program yazabilir hale gelmesini amaçlamaktadır. Notlarda yer verilen program kaynak kodları öğrencilerin farklı donanım platformlarında ve işletim sistemlerinde çalışabilecekleri düşünülerek yazılmıştır. Farklı işletim sistemlerinde kullanılabilecek olan geliştirme araçlarının çeşitliliği dikkate alınarak **Code::Blocks** tümleşik geliştirme ortamı tercih edilmiştir. Bu geliştirme ortamı farklı işletim sistemlerinde kolaylıkla kullanılabilmektedir. Kurulum ve yapılandırma için gereken işlemler ilgili bölgelerde ele alınmıştır. Linux dağıtımlarını ve \*BSD işletim sistemi ailesinin üyelerinden irisini kullanan kullanıcılar için Code::Blocks yanında farklı seçenekler bulunmaktadır. Benze şekilde derleyici ve hata ayıklama araçları için de geçerlidir. Bu notlarda Linux dağıtımlarında ise GCC ve LLVM/CLANG, BSD işletim sistemi ailesinde de LLVM/CLANG ve/veya GCC kullanılarak yazılan kodlar test edilmiştir.. Windows kullanıcıları olanların Visual Studio tercih etmeleri durumunda bu ders notlarında yer alan program örneklerinin test edilmemiş olduğunu, ekran görüntülerinde görüldüğü gibi çalışabileceğini garanti etmediğini belirtmek yerinde olacaktır.

Ders notları üç kısımdan oluşmaktadır. İlk kısmı da C++ Programlama Dili öğretilmektedir. Bu kısımda öncelikli amaç, C++ Programlama Dili'ne ait tüm ayrıntıların vermesi yerine öğrenciye kendi kendisine yetebilir bir duruma gelmesini sağlamak için temel kavramların doğru ve eksiksiz olarak öğrenilmesini ve uygulanabilmesini sağlamaktır. Bunun içinde sıkılıkla dikkate alınmayan bilgisayar donanımı konusunda bazı temel bilgilere yer verilmiştir. Ders notlarındaki örnekler basitten karmaşa ve kolaydan zora doğru ilerlemektedir. Örnekler birbirini izleyen bir mantık sırası ile verilmiştir. Ayrıca doğru anlaşılmayan ve öğrenilmeyen bazı kavramların neden olacağı

sorunlara da dikkat çekilmiştir. İlk kısımda yer alan tüm bölümler aynı sıra ile işlenmek durumunda değildir. İlk kısımda yer alan Standart Şablon Kütüphanesi – Standart Template Library ile Nesne Yönlimli Programlama – Object Oriented Programming bölümleri Veri Yapıları konusun geçilmeden önce işlenebilir. Veri Yapıları konusuna girmeden önce ise ikinci bölüm olan Algoritmalar işlenmiş olmalıdır.

İkinci kısımda Algoritmalar konusu, bilgisayara bilimlerindeki çeşitli problemlerin çözümünde kullanılan algoritmaların bir problem özelinde gruplandırıldığı sınıflandırmaya dayalı yaklaşım ile ele alınmamıştır. Bu yaklaşımın üstün yanı belirli bir problem alanı için ortaya konulmuş olan algoritmaların karşılaştırmalı olarak incelenmesini sağlamaktadır. Zayıf yani ise problem çözme becerisini geliştirmek ve algoritma tasarımını dikkate almamakta olmasıdır. Gözlemlerime dayanarak bu yöntemim sadece seçilmiş çok sayıdaki algoritmanın çözümlenerek etkinlik ve verimlilik açısından incelenmesini sağlamak için uygun olduğudur. Programcı veya yazılım geliştiricisi olmak isteyenler için algoritma tasarımını konusundaki eksiklikleri zaman, emek ve kaynak kayıplarına neden olmaktadır. Bunun yerine bir problemin nasıl çözülebileceğinin anlaşılması ve önerilen çözümlerden hangisinin tercih edilmesin uygun olacağına karar verilebilmesi yaklaşımı sadece bilgisayar ile problem çözümü için değil günlük yaşam içinde uygun bir yaklaşımdır. Bu nedenle algoritmalar konusunda bilgisayar bilimlerinde ve mühendislik dallarında sıkılıkla karşılaşılan bazı problem alanlarında çözümün tasarımını üzerinden ele alınmıştır. Bu amaçla problem alanları tanımlanıp, algoritmanın verimliliği dikkate alınarak algoritma tasarım teknikleri C++ Programlama Dili kullanılarak uygulanmıştır.

Üçüncü bölüm ise veri yapılarını ele almaktadır. Veri yapılarının anlaşılması ve doğru bir şekilde uygulanabilmesi için problemde ele alınan verinin, temel veri tiplerinden başlayarak karmaşık veri yapılarına doğru ilk iki bölümde kısmende olsa yer verilmiştir. Bu bölümlerin doğru öğrenilmiş olması Veri Yapıları'nın öğrenilmesi için önemlidir. Örneklerde karmaşık kodlar yazılması yerine STL kullanılarak kısa ve temiz kodlara yer verilmiştir. Ayrıca örneklerde algoritma analizi yapılarak veri yapısının hem tasarım hem de uygulama açısından etkinlik ve verimliliği ön planda tutulmuştur. Veri yapıları sadece belirli algoritmaların kodlanması olarak ele alınmamıştır. Veri yapısının ara yüzü ile uygulaması birlikte ele alınmıştır.

# 0.Bilgisayarlar, Programlama ve Programlama Dilleri

Bilgisayarların günlük yaşamımızda yer almazı bir alan yok gibidir. Günlük işlerimizin büyük bir bölümünde bilgisayarlar farklı şekillerde ve farklı amaçlar için kullanılmaktadır. İşinize veya okula gidip gelirken kullandığınız plastik kart şeklindeki biletleri toplu taşıma araçlarında “okutarak” ücretinizi ödersiniz. Yine bu kartlara “bilet” yüklemesi yapılır. Akıllı telefon ile internette dolaşabilir, interneti kullanarak arkadaşlarınız veya ailiniz ile iletişim kurabilirsiniz. E-posta hesabınıza gelen mesajlara yanıt yazabilir veya bir yenisini oluşturabilirsiniz. Bu işlemlerin benzerlerini da masaüstü bilgisayarı veya bir dizüstü bilgisayarı kullanarak yapabilirsiniz. Evde bulunan “aklıllı televizyon” ile yukarıda sayılanların hemen hemen hepsini yapabilirsiniz.

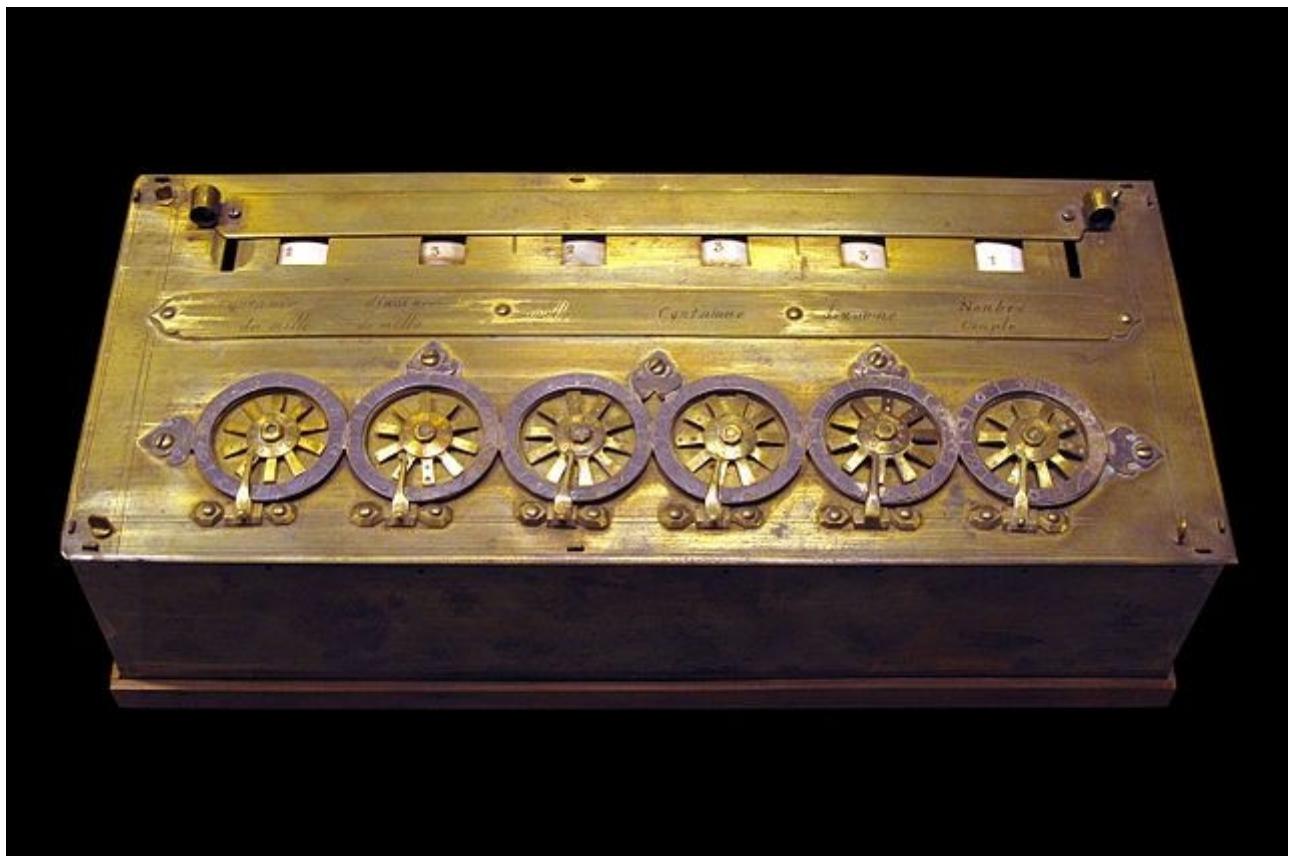
Bu sayılan eylemlerin tümü bilgisayarı oluşturan yazılım ve donanımın birlikte çalışması ile gerçekleşmektedir. Bilgisayar terimi temel olarak bir araya getirilmiş olan birden çok sayıdaki donanımdan oluşur. Bu donanım üzerinde çalışacak olan yazılım olmadan bilgisayar yukarıda sözünü ettiğimizi işlemleri yerine getirmesi söz konusu değildir. Donanım fabrikada üretilirken, yazılım ise programcılar tarafından geliştirilir yani üretilir. Programcının, kullanıcının birkaç tuşa basarak veya dokunarak yaptığı işlemlerin yapılabilmesi için donanım üzerinde çalışacak ve kullanıcının isteklerini yerine getirecek olan yazılımı yazması gereklidir. Bu yazılımların yazılması işlemine programlama adı verilir. Programcının kullanıcının gereksinim duyduğu işlemleri yapabilmesi için hazırlamakta olduğu program/yazılım ise programlama dili ile hazırlanır. C++ bu dillerden bir tanesidir. C++ programlama dilini kullanarak program yazmaya başlamadan önce temel bilgilerin öğrenilmesi gereklidir. Bu amaçla öncelikle bilgisayar teknolojisinin tarihsel gelişimi ile başlıyoruz.

## 0.1.Bilgisayarın Tarihsel Gelişimine Kısa Bir Bakış

Bir bilgisayar teknik olarak bir dizi aritmetik ve mantıksal işlemi yapan gelişmiş bir hesap makinesidir. Tarihte bilinen ilk hesap makinesi abaküsür. Kökeni konusunda farklı görüşler olsada bugün için halen aritmetik işlemleri öğretmek için kullanılan ilk makine olarak varlığını sürdürmektedir. Abaküs, tasarım olarak doğrudan sayıları toplamak ve çıkarmak için kullanılır. Bölme veya çarpma yapmak ise olankı değildir. Abaküsün gelişmiş bir modelini Fransız matematikçi ve filozof Blaise Pascal 1642 yılında geliştirmiştir. Bu makine Pascaline adı ile bilinmektedir. Sekiz karakter uzunluğundaki sayıları toplamak ve çıkarmak için kullanılmıştır. (**Resim 0.0**) 17 yy. Gottfried von Leibniz, toplama, çıkarma, bölme ve çarpma yapan bir hesap makinesi icat etmiştir.<sup>1</sup> (**Resim 0.1**) Joseph Jacquard 1819 yılında kendi adı ile anılan bir dokuma tezgahı icat etmiştir. Bu dokuma tezgahının farklı ise desenlerin önceden hazırlanmış olan delikli kartlar kullanılarak tezgah üzerinde dokunmasını sağlamıştır. Delikli kartların da dokuma başlanmadan önce özel bir makine da istenilen desene ve renge uygun olarak hazırlanmıştır. Dokuma tezgahını çalışuran kişi sırayla okutulan delikli kartlardaki yönergelerin tezgah tarafından okunmasını ve desenin dokunmasını sağlamaktadır. Ayrıca kartların sırasının değiştirilebilmesi ile farklı desen ve renklerde dokumalar yapılmaktadır. Bugün kendi adı ile anılan dokuma tezgahları halen kullanılmaktadır. Joseph Jacquard’ın dokuma tezgahının önemi, bir çok yönergenin belirli bir sıra ile işlenmesi ve aynı zamanda yönergelerin hem bir başka makine kullanılarak hazırlanabilmesi, hemde

<sup>1</sup> Dokuma tezgahı hakkında daha fazla bilgi almak için: [https://en.wikipedia.org/wiki/Jacquard\\_machine](https://en.wikipedia.org/wiki/Jacquard_machine)

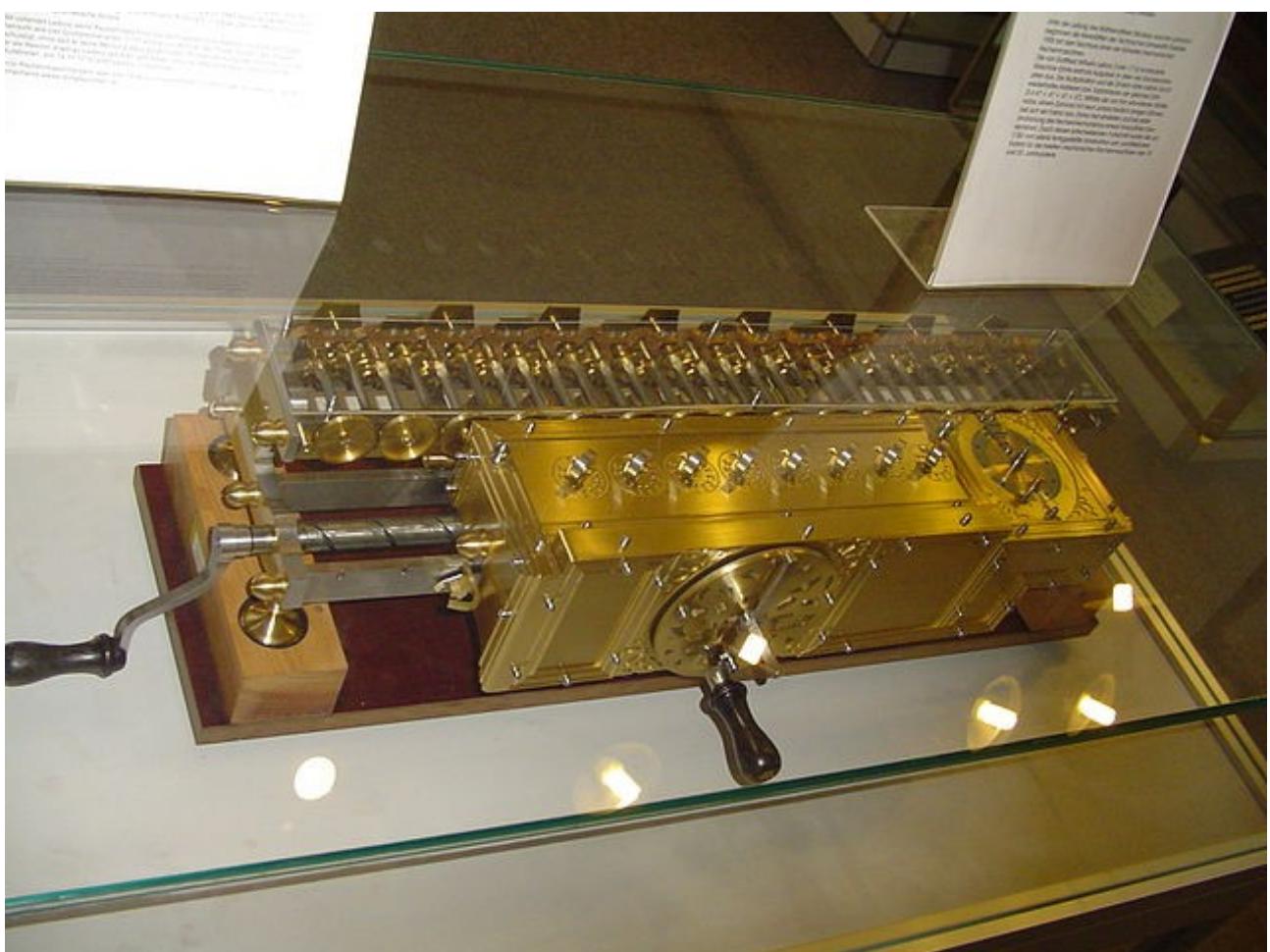
hazırlanmış olan yönergelerin makinaya aktarılarak işlemlerin yapılmasına olanak sağlamasıdır. Bu dokuma tezgahı hem verinin hem de veriyi işlemek için kullanılan yönergelerin saklanabildiği “**stored programme computer**” fikrinin öncülüdür. Bu günde bilgisayarlar bu fikir olmadan bugün var olamazdı.



**Resim 0.0:** Blasie Pascal tarafından imzalanmış olan bir Pascaline Kaynak: By Rama, CC BY-SA 3.0 fr, <https://commons.wikimedia.org/w/index.php?curid=53246694>

1800'lerin ortasında İngiliz bilim insanı Charles Babbage aritmetik işlemleri yapabilen bir mekanik makine tasarlamıştır. İlk tasarladığı makineye **fark makinesi – difference engine** adını vermiş ve bir prototipini üretmiştir. El gücü ile çalışan makinanın yapacağı için önceden bir dizi işlem ile hazırlanması gerekiyordu. Bu makinenin ayırt edici yanı temel aritmetik işlemler yanında sayıların kuvvetini de hesaplayabiliyordu. Fark makinesinin ilk çalışan modeli 2002 yılında ilk tasarımının hazırlanmasından 153 yıl sonra üretilmiş ve Londra'da sergilenmektedir. 8000 parçadan oluşan ve 5 ton ağırlığındaki makine yaklaşık 3,5 m uzunluğundadır. (**Resim 0.2**) Daha sonra ise Babbage **analitik makine – analytical engine** adını verdiği makinayı tasarlamıştır. Bu makine ise fark makenesinin daha büyüğü ve gelişmiş olarak tasarlanmıştır. Oncekinden farklı olarak yönergelerin ve verinin makinaya verilmesi ile makine istenilen işlemleri yapabilecek ve sonuçları da bir kağıda yazabilecek şekilde tasarlanmıştır. Ancak bu makine ise gerçekleştirilememiştir. Bu makineler hakkındaki ayrıntılı bilginin kaynağı Charles Babbage ile Kontes Agusta Ada Lovelace arasındaki mektuplaşmalara dayanmaktadır. Kontes Agusta Ada Lovelace'in yazdığı mektupların aile arşivinde saklanması ve sonrasında ise araştırmacılar tarafından üzerinde çalışılması ile ortaya çıkarılmıştır. Bu nedenle Kontes Agusta Ada Lovelace tarihteki ilk programcı olarak kabul edilir. Trajik olan ise babasının Jacuard'ın dokuma tezgahlarına karşı çıkan dokuma ustaları tarafından yer

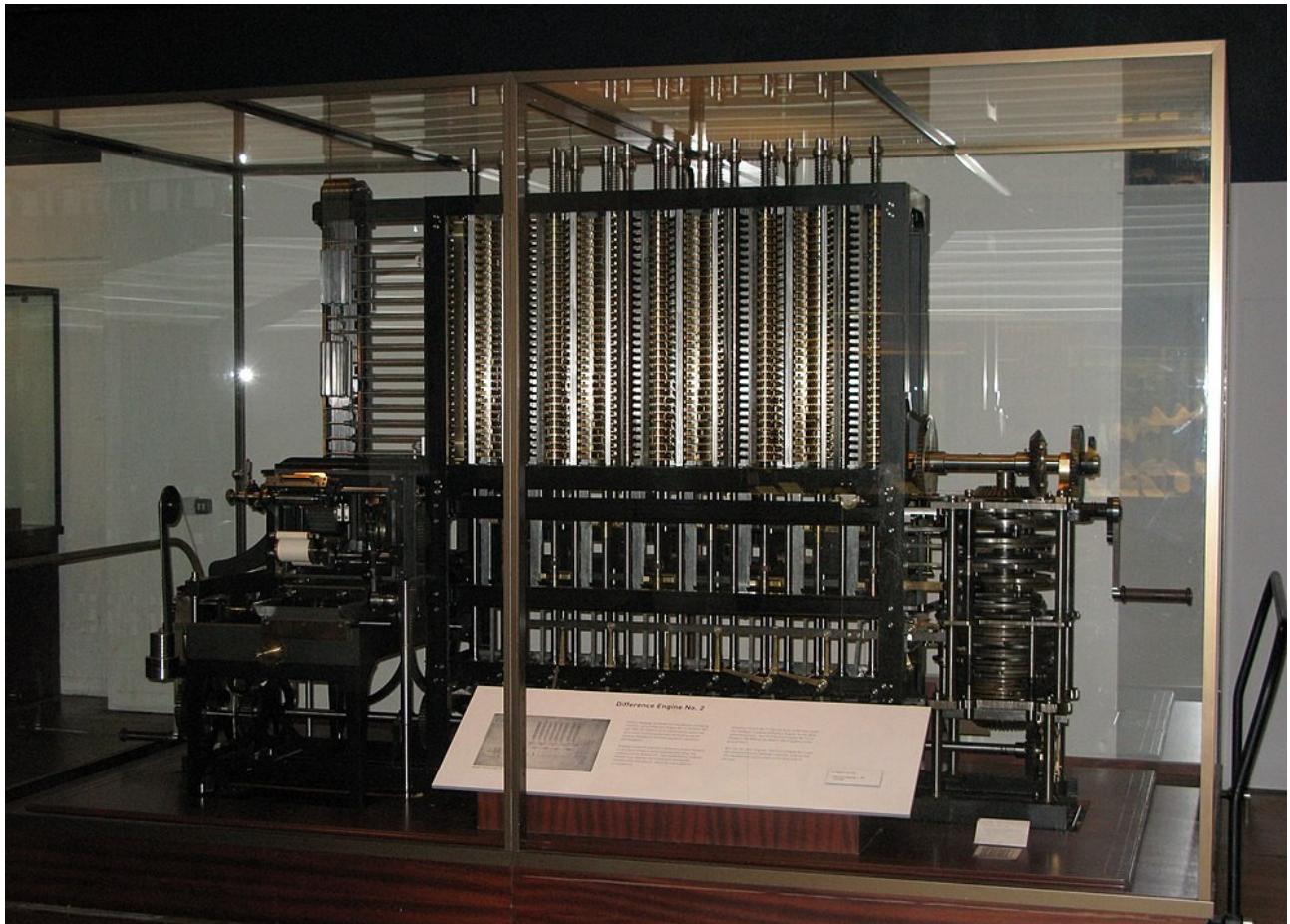
almasına karşılık, Jacquard'ın dokuma tezgahının ışık tuttuğu fikirlerin kızı tarafından geleceğin bilgisayarlarının geliştirilmesinde esas alınacak olan temelin inşa edilmesinde kullanılmış olmalıdır



**Resim 0.1:** Gottfried von Leibniz tarafından geliştirilmiş olan mekanik hesap makinesinin bir kopçası  
Kaynak: By User:Kolossos - recorded by me in de:Technische Sammlungen der Stadt Dresden  
(with photo permission), CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=925505>

1800'lerin ortasında İngiliz bilim insanı Charles Babbage aritmetik işlemleri yapabilen bir mekanik makine tasarlamıştır. İlk tasarladığı makineye **fark makinesi – difference engine** adı vermiş ve bir prototipini üretmiştir. El gücü ile çalışan makinanın yapacağı için önceden bir dizi işlem ile hazırlanması gerekiyordu. Bu makinenin ayırt edici yanı temel aritmetik işlemler yanında sayıların kuvvetini de hesaplayabiliyordu. Fark makinesinin ilk çalışan modeli 2002 yılında ilk tasarımının hazırlanmasından 153 yıl sonra üretilmiş ve Londra'da sergilenmektedir. 8000 parçadan oluşan ve 5 ton ağırlığındaki makine yaklaşık 3,5 m uzunluğundadır. Daha sonra ise Babbage **analitik makine – analytical engine** adını verdiği makinayı tasarlamıştır. Bu makine ise fark makinesinin daha büyüğü ve gelişmiş olarak tasarlanmıştır. Oncekinden farklı olarak yönergelerin ve verinin makinaya verilmesi ile makine istenilen işlemleri yapabilecek ve sonuçları da bir kağıda yazabilecek şekilde tasarlanmıştır. Ancak bu makine ise gerçekleştirilememiştir. Bu makineler hakkında ayrıntılı bilginin kaynağı Charles Babbage ile Kontes Agusta Ada Lovelace arasındaki mektuplaşmalara dayanmaktadır. Kontes Agusta Ada Lovelace'in yazdığı mektupların aile arşivinde saklanması ve sonrasında ise araştırmacılar tarafından üzerinde çalışılması ile ortaya çıkarılmıştır.

Bu nedenle Kontes Ada Lovelace tarihteki ilk programcı olarak kabul edilir. Trajik olan ise babasının Jacuard'ın dokuma tezgahlarına karşı çıkan dokuma ustaları tarafında yer almasına karşılık, Jacquard'ın dokuma tezgahının ışık tuttuğu fikirlerin kızı tarafından geleceğin bilgisayarlarının inşa edilmesinde esas alınacak olan temeli inşa edilmesinde kullanılmış olmalıdır. (Bkz: Kontes Ada Lovelace ve çalışmaları hakkındaki Wikipedia kaydı<sup>2</sup>)



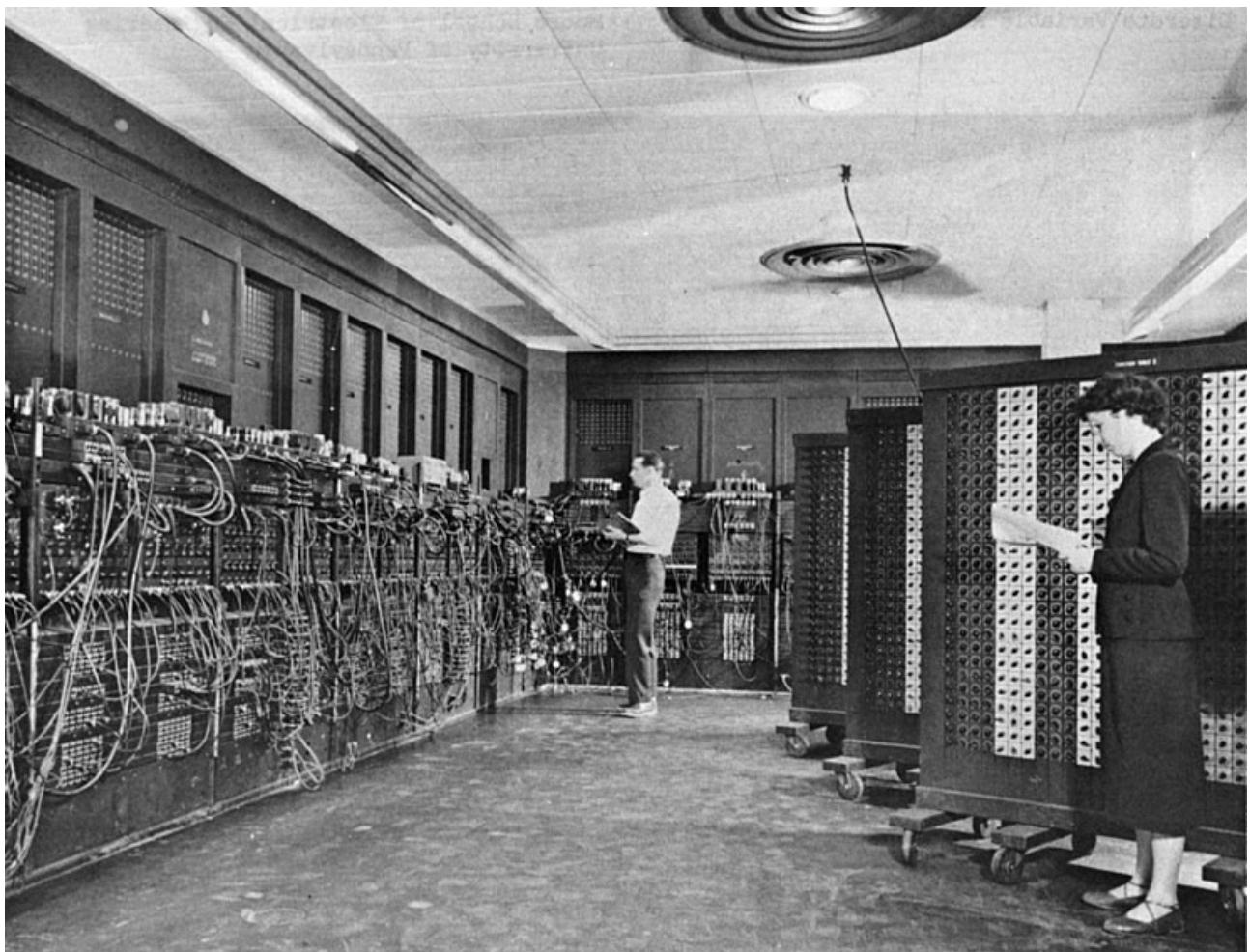
**Resim 0.2:** Babbage'in orijinal tasarımdan inşa edilmiş olan Londra Bilim Müze'sinde sergilenen Fark Makinesi. Kaynak: By User:geni - Photo by User:geni, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=4807331>

19 yy. sonunda ABD'de yapılan nüfus sayımlarında derlenen verilerin daha kısa zamanda işlenmesi ve tasnif edilmesini sağlamak için **Herman Hollerith** tarafından icat edilen ve delikli kartları kullanarak veriyi depolayabilen bir makineyi kullanmışlardır. Hollerith'in icadı büyük bir başarı elde etmiştir. Hollerith, **Tabulating Machine Company** adlı şirketi kurmuştur. Kurduğu şirket ise daha sonra **IBM – International Business Machinery** adını almıştır.

Bilgisayar teknolojisinin gelişimi sürecinde mekanik bilgisayarların üretim ve işletim zorluklarının çözümüne yönelik olarak mekanik sistemlerden yarı mekanik yarı elektronik sistemlere doğru geçiş olmuştur. 1944 yılında **Howard Aiken** öncülüğünde IBM ve Harvard Üniversitesi işbirliği ile bugünkü bilgisayara benzeyen ilk sistem üretilmiştir. Yaklaşık 50 ton ağırlığında ve 750,000 adet parçadan oluşan ve delikli kartları kullanan ilk programlanabilir bilgisayardır. 1946 yılında ile tamamen elektron lambaları kullanılarak Pensilvanya Üniversitesi'nde **Electronic**

<sup>2</sup> [https://en.wikipedia.org/wiki/Ada\\_Lovelace](https://en.wikipedia.org/wiki/Ada_Lovelace)

**Numerical Integrator and Calculator – ENIAC**<sup>3</sup> üretilmiştir. ENIAC 18,000 adet elektron lambasından oluşmakta olup yaklaşık 30 ton ağırlığındadır. (**Resim 0.3**)



**Resim 0.3:** Balistik Araştırmalar Binası, 328'de Glenn A. Beck (arkada) ve Betty Snyder (önde) ENIAC'ın programlanması sırasında çalışırken görülüyor. (Kaynak: <https://en.wikipedia.org/wiki/File:Eniac.jpg>)

Bugün kullanmakta olduğumuz modern anladaki bilgisayarların tasarım prensipleri **John Von Neumann** tarafından 1940'larda ortaya konulmuştur. Von Neumann, tasarımını aritmetik ve mantıksal işlemleri yaptığı bir birim, bellek birimi, girdi ve çıktı birimlerinden oluşmaktadır. Von Neumann tasarım, bilgisayarın işlem yapmak için kullanacağı yönergeler ile üzerinde işlem yapılacak olan verinin donanım üzerinde saklanması, bulunduğu yerden alınarak yüklenmesini, çalıştırılmasını, elde edilen sonuçlarında yine donanım üzerinde özel bir alanda saklanabilmesine olanak vermektedir. Von Neuman prensiplerine uyan ilk elektronik bilgisayar ise ENIAC'ı geliştiren ekip tarafından yapılmıştır. Ekip bu bilgisayara **UNIVAC I (UNIVersal Automatic Computer I)** adını vermiştir. Ticari amaçlı olarak geliştirilen ve kullanılan ilk bilgisayardır. 1952 yılında ABD İstatistik Kurumu'na satılmıştır.

1956 yılında transistör'ün icat edilmesi ile birlikte bilgisayarlara elektron lambalarının kullanılmasından vazgeçilerek boyutları ve enerji tüketimleri azaltılmıştır. Yine bu dönemde programlama dilleri alanında önemli gelişmeler olmuş; **FORTRAN** ve **COBOL** geliştirilmiştir. Bu

3 <https://en.wikipedia.org/wiki/ENIAC>

gelişmeleri izleyen dönemde entegre devre teknolojisinin gelişmesi elektronik devrelerin işlem gücünü artırırken aynı zamanda da yer ve güç tüketimlerini daha da azaltmıştır. 1970 yılında ise mikro işlemcinin icadı bilgisayar teknolojisindeki en önemli gelişme olmuştur. Mikroişlemci sayesinde 1977 yılında **Steve Jobs** ve **Steve Wozniak** ilk **Apple** bilgisayarını geliştirmiştir. IBM 1980 yılında ise kendi adını taşıyan ilk ticari bilgisayarı **IBM – Personal Computer**, **IBM PC** satışa sunmuştur. Donanım teknolojisindeki ve yazılım alanındaki gelişmeler sonucunda 1990'ların ikinci yarısı itibarı ile bilgisayarlar her evde bulunan bir cihaz durumuna gelmiştir. Teknolojideki gelişmeler de hem bilgisayarların işlem gücünü artırırken hem de fiyatlarının düşmesini sağlamıştır.

Bilgisayarlar günümüzde günlük yaşamda her yerde karşımıza çıkmaktır. Bu yaygınlığa bakıldığından bilgisayarların işlevselliklerine ve kullanım amaçlarına göre farklı şekillerde gruplandırılmasını sağlamaktadır. Her ne kadar bilgisayarları farklı gruplar altında incelemek olanaklı olsa da temel olarak çalışma prensipleri aynıdır.

## 0.2.Bilgisayarın Temel Bileşenleri

Bir bilgisayar çeşitli komutları işleyebilen bir elektronik aygıttır. Verilen bilgiyi işler, sonuçlarını görüntüleyebilir, yazabilir veya depolayabilir ve verilen bilgi üzerinde aritmetik ve mantıksal işlemler gerçekleştirir. Bir bilgisayarın temel donanım ve yazılım olmak üzere iki bileşeni vardır.

### 0.2.1.Donanım

Bir bilgisayarın donanım merkezi işlem birimi **CPU – Central Processing Unit**, birincil bellek veya ana bellek olan rastgele erişimli bellek **RAM – Random Access Memory**, giriş ve çıkış aygıtları – **I/O – Input/Output Devices**, ile ikincil depolama aygıtlarından oluşur. Giriş ve çıkış aygıtları olarak klavye, monitör, yazıcı örnek verilebilir. (**Şekil 0.1**)

#### 0.2.1.1.Merkezi İşlem Birimi – CPU

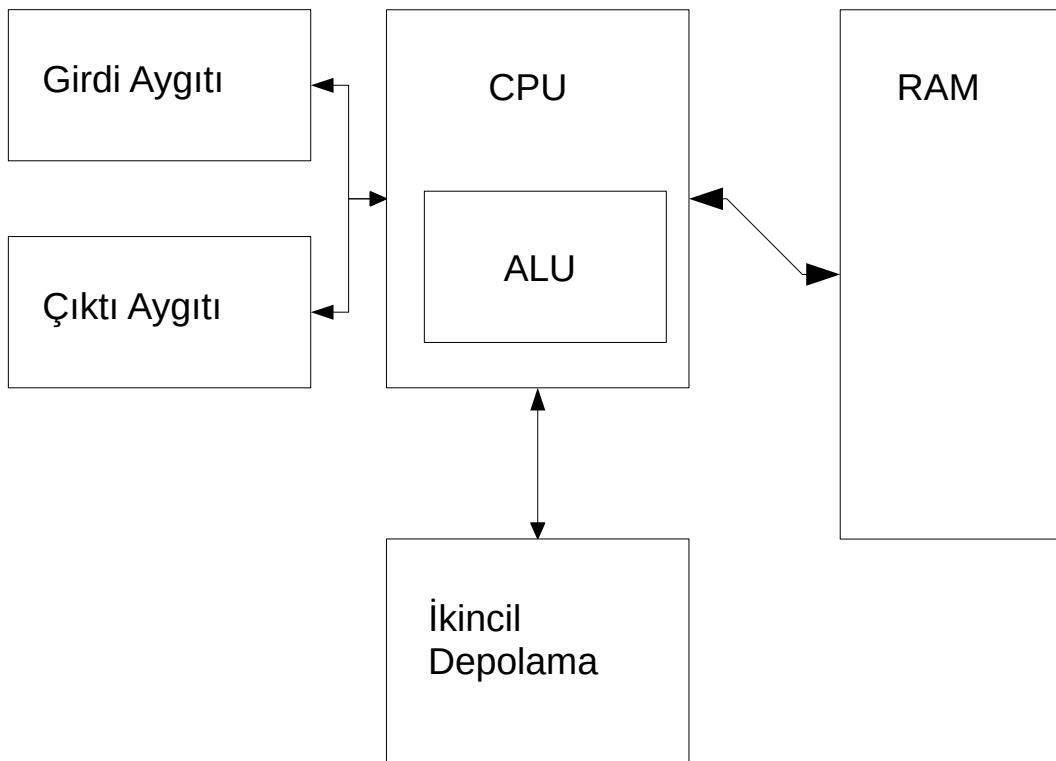
Merkezi işlem birimi bilgisayarın beynidir. Bir bilgisayarın en pahalı parçası da işlemcisi yani merkezi işlem birimidir. Bir işlemcinin birim zamanda yapabildiği işlem sayısı arttıkça işlem gücüde aynı oranda artar. İşlemci aynı zamanda aritmetik ve mantıksal işlemleri de yapan bir birime sahiptir. Bu birim aritmetik ve mantıksal birim **Arithmetic and Logical Unit – ALU** olarak adlandırılır.

#### 0.2.1.2.Birincil Bellek

Merkezi işlem birimi,rastgele erişimli belleğe doğrudan erişebilir. Merkezi işlem biriminin veriyi işleyebilmesi için verinin RAM'a yüklenmiş olması gereklidir. CPU RAM üzerinde bulunan veriyi alır, işler ve sonucunu yine RAM üzerinde önceden belirlenmiş bir bellek alanına yazar. RAM üzerindeki veri sadece CPU tarafından işlenecek olan veri yanında işlemcinin çalıştıracağı komutlar da yer alabilir. RAM üzerinde barındırılan veri kalıcı değil geçici olarak bulunur. Diğer bir deyişle bilgisayarın elektriği kesildiğinde RAM üzerinde bulunan veri tekrardan erişilebilir değildir, kayıp olmuştur.

RAM yapı olarak bir çok küçük depolama biriminden oluşur. Bu depolama birimlerine bellek hücresi de denir. Bir belleği oluşturan her tekil depolama biriminin kolaylıkla erişilebilmesi için kendisini tanımlayan bir adres bilgisi ile tanımlanır. Bu adres bilgileri ise 1 ve 0 oluşan sayı

dizileri şeklinde tanımlanır. Yine aynı şekilde bellek üzerinde yer alan veri de 1 ve 0 oluşan bir dizi şeklinde bulunur.



Şekil 0.1

#### 0.2.1.3. İkincil Depolama

RAM üzerinde bulunan verinin kalıcı olmaması nedeni ile işlenecek olan verinin ve çalıştırılacak olan komutların kalıcı olarak saklanması gereklidir. Bu nedenle saklanacak olan verinin RAM dışında bir depolama biriminde saklanması daha uygun olacaktır. Bu depolama gereksinimini karşılamak için CD-ROM, DVD-ROM, sabit disk ve flash bellek örnek verilebilir.

#### 0.2.1.4. Girdi – Çıktı Aygıtları

Bir bilgisayarın istenen görevleri yerine getirebilmesi için veriyi alması, işlemesi ve sonuçlarını da kullanıcıya sunması gereklidir. Bu sürecin işleyebilmesi için verinin bilgisayara aktarılmasını sağlayan aygıtlara girdi aygıtları adı verilir. Benzer şekilde de bilgisayarda işlenen verinin sunulmasını sağlayan aygıtlara da çıktı aygıtları adı verilir. Klavye, tarayıcı ve fare giriş aygıtlarına, yazıcı, monitör de çıkış aygıtlarına örnek olarak verilebilir.

### 0.2.2. Yazılım

**Yazılım**, bilgisayar üzerinde gerçekleştirilmesi istenen bir işi yerine getirmek için, bir programlama dili ya da program geliştirme araçları kullanılarak, bu görevi yerine getirmek için geliştirilmiş program, veri ve yazılım belgelerinden oluşur. **Program**, belirli bir amaç için tasarlanmış olan ve programlama dili ve araçları kullanılarak hazırlanmış olan bir kod, yazılım parçasıdır. Yazılım bir tasarım olurken, program yazılımının bir parçasıdır. **Veri**, program kodlarının çalıştırılması sırasında üzerinde işlem yapılacak bilgidir.

Bilgisayar üzerinde çalışan yazılım iki gruba ayrılır; **Sistem Yazılımı** ve **Uygulama Yazılımı**. Sistem yazılımı bilgisayarın işleyişini kontrol eden ve yürütten programlardır. Ayrıca programcının geliştireceği yazılımları ve programları hazırlamasını sağlayan programları da kapsar. Uygulama yazılımları ise kullanıcıların gereksinim duydukları ve kullandıkları yazılım ve programlardan oluşur.

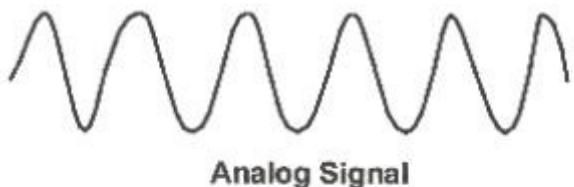
### 0.3.Bilgisayar Dili

Bilgisayar elektronik bir aygittir. Elektronik aygıtlar elektrik sinyallerini işleyebilir. Elektrik sinyalleri de **analog** ve **sayısal – digital** olmak üzere iki gruba ayrılır. Evde bulunan eski plaklar ve kasetler analog sinyalleri barındıran kayıtlardır. Bir plaktan veya kasetten bir diğer kasete veya plağa kayıt yapıldığında dikkatli bir gözlemci kaynak ile kopya arasındaki ses farkını kolaylıkla gözlemeylebilir. Benzer şekilde bir CD kopyaladığınızda ses kalitesinde farkı gözlemelemek olaklı olmayacağından emin olabilirsiniz. Analog sinyaller sürekli bir dalga biçimine sahiptir. Dijital sinyaller ise iki farklı değer alabilen kesikli ve sürekli olabilen dalga biçimine sahiptir.

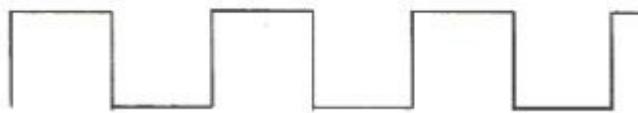
Dijital sinyaller veri iletimi söz konusu olduğunda analog sinyallere göre daha iyi bir performans göstermektedir. Ayrıca bilgisayarlar dijital sinyaller ile çalışacak şekilde tasarlanmıştır. Bir dijital sinyalin alabileceği değerler bilgisayar için yüksek ve düşük voltaj şeklinde olacaktır. Yüksek voltaj 1 ve düşük voltaj ise 0 olarak tanımlanır. Böylece bilgisayar işlenecek olan komutları ve veriyi 1 ve 0 oluşan sayı dizileri olarak okur, işler ve yazar. Bilgisayar bilimlerinde bu gösterime **makine dili – machine language** adı verilir. Makine dili için iki farklı değer tek bir bit ile gösterilir. Bir bit İngilizce dilinde **binary digit** sözcüğünden türetilmiştir. İfade edilen ise ikilik taban aritmetiğindeki tek bir basamağın değerini yani 1 veya 0 göstermektedir. Bit tek başına bilgisayarın işleyeceği komutları ve veriyi temsil etmek için uygun değildir. Öte yandan birden çok sayıdaki bit bir arada kullanılarak hem veri hem de komutlar kolaylıkla ifade edilebilir.

İkilik tabandaki basamaklar esas alındığında verinin ifade edilmesi için harflerin, sayıların ve sembollerin kendilerini temsil edecek olan bir sayı ile gösterilebilmesi söz konusu olur. Bilgisayarlarda yaygın olarak kullanılan temsil yöntemi **American Standard Code for Information Interexchange – ASCII**'dir. ASCII kod sisteminde 127 adet karakter yer alır. Bunun nedeni ikilik tabanda ifade edilen sayı değerlerinin 8 bit büyülüğündeki diziler olarak tasarılmış olmasıdır. ASCII tablosuna bakıldığında “A” tablodaki 66 konumda yer alır. Ancak ikilik tabandaki karşılığı ise 1000001'dir. Bu değer ise 65 karşılık gelir. Bunun nedeni tablodaki ilk karakterin 0000000 ile ifade edilmiş olmasıdır. (**Bkz: Ek A**)

Bilgisayarların işledikleri programlar bir bilgisayar programlama dili ile yazılmaktadır. Yazılan program ise doğrudan bilgisayar tarafından çalıştırılması olaklı değildir. Yazılan programın çalıştırılabilmesi için programın içeriği yönergelerin işlemci tarafından işlenebilecek bir yapıya dönüştürülmesi gereklidir. Bu dönüşüm işlemi sonucunda programcının kolaylıkla anlayabildiği biçimden işlemcinin kolaylıkla işleyebileceği biçimde dönüştürülmüşdür. İşlemcinin doğrudan işleyebildiği yönergelere makine dili adı verilir ve işlemcinin tasarımasına bağlı olarak değişir. Programlama dili ile yazılan programın bütünü bir seferde makine diline dönüştürülüyorrsa bu programlama dillerine derlenen diller adı verilir. Derleme işlemi sonucunda program kodu işlemcinin işleyebileceği yönergeleri barındıran bir dosya üretilir. Bu dosya doğrudan işlemci tarafından işlenebilir.



Analog Signal



Digital Signal

**Şekil 0.2**

Bir bilgisayarda bulunan işlemcinin okuyabildiği ve işleyebildiği yönergelere **komut seti – instruction set** adı verilir. Bu komutlar işlemciye özel olduğundan dolayı farklı işlemciler tarafından aynı şekilde yorumlanamazlar. Bu nedenle farklı işlemciler ve farklı mimariler söz konusudur. Örneğin Von Neuman Mimarisi olarak adlandırılan PC sistemlerinin mimarisi aynı zamanda **Princeton Mimarisi** olarak da bilinir. Bunun yanında da **Harward Mimarisi** de bulunmaktadır. Dijital sinyal işleyiciler ile mikrodenetleyicilerde kullanılır. Benzer şekilde PC kullanılan INTEL uyumlu işlemciler **CISC – Complex Instruction Set Computer** mimarisi ve ARM ve benzeri işlemciler ise **RISC – Reduced Instruction Set Computer** mimarisi ile üretilirler.

Bu tanımlardan da anlaşılacağı üzere farklı mimarilerde geliştirilen donanımların aynı komutlar kullanılarak programlanması söz konusu değildir. Bu işlemcilere uygun olan komut setinin kullanılması gereklidir. Makine dili ile program yazmak düşük seviyeli düşünmeyi ve yazılımın üzerinde çalışacağı donanıma özgü olan mimarisinin çok iyi bilinmesini gerektirir. Bu yazılım geliştirmeyi zorlaştırmaktadır. Makine dili komutlarının öğrenilerek bu dil kullanılarak program yazılması yerine işlemcide yürütülen işlemleri temsil eden **anımsatıcı komutlar – mnemonic commands** olarak tanımlanan **Assembly** dili geliştirilmiştir. Assembly dili ile yazılan bir program doğrudan çalıştırılabilen bir program değildir. Bu programın işlemcinin işleyebileceği şekilde dönüştürülmesini sağlamak için **Assemblers** adı verin derleyici ile makine diline çevrilmesi gereklidir. Assembly dili günümüzde derlenmiş olan programların makine dilinden Assembly diline çevrilerek hata ayıklama işlemlerinde önemli bir yere sahiptir.

Assembly dilinin makine diline göre daha kolay anlaşılmasına karşılık işlemciye özel olmasının gerektirdiği kısıtların aşılması için sonra sırası ile B ve ardından **C programlama dili** geliştirilmiştir. C dili işletim sistemi yazmak için **Brian Kernighan** ve **Dennis Ritchie** tarafından geliştirilmiş UNIX işletim sisteminin geliştirilmesinde önceden kullanılan B dilinin yerine geçmiştir. C programlama dili teknik olarak makine diline yakın olan düşük seviyeli diller ile insanın kolayca anlayabileceği yüksek seviyeli diller arasında ortada kalan bir dildir. İzleyen yıllarda C dilinin gelişen yazılım geliştirme paradigmalarının da etkisi ile nesne yönelimli programlama ve modüler yazılım geliştirme prensiplerinin de kullanıldığı **Bjarne Stroustrup**<sup>4</sup> tarafından **C++ programlama dili** geliştirilmiştir. Bunların yanında farklı alanlardaki gereksinimleri karşılamak ve çeşitli

4 [https://en.wikipedia.org/wiki/Bjarne\\_Stroustrup](https://en.wikipedia.org/wiki/Bjarne_Stroustrup)

problemleri çözmek amacı ile PASCAL, FORTRAN, COBOL, BASIC gibi diller de geliştirilmiştir. C ve C++ programlama dilleri sonraki yıllarda ISO tarafından standartlaştırılmıştır. C programlama dilinin güncel standardı **ISO/IEC 9899:2018** ve C++ programlama dilinin güncel standardı da **ISO/IEC 14882:2020**'dir.

## 0.4.Derleyiciler

Bilgisayarların programları bir bilgisayar dili ile yazılmalıdır. Yazılan program ise doğrudan bilgisayar tarafından çalıştırılması olanaklı değildir. Yazılan programın çalıştırılabilmesi için programın içeriği yönergelerin işlemci tarafından işlenebilecek bir yapıya dönüştürülmesi gereklidir. Bu dönüşüm işlemi sonucunda programının kolaylıkla anlayıldığı biçimden işlemcinin kolaylıkla işleyebileceği biçimde dönüştürülmüşdür. İşlemcinin doğrudan işleyebildiği yönergelere makine dili adı verilir ve işlemcinin tasarımasına bağlı olarak değişir. Programlama dili ile yazılan programın bütünü bir seferde makine diline dönüştürülüyorsa bu programlama dillerine derlenen diller adı verilir. Derleme işlemi sonucunda programın kodu işlemcinin işleyebileceği yönergeleri barındıran bir dosya üretilir. Bu dosya doğrudan işlemci tarafından işlenebilir.

C++ bir derlenen dildir. Yazılan program ilk satırдан başlayarak son satırına kadar bir seferde derleyici tarafından makine diline dönüştürülür. Derleme işlemi sırasında kullanılan ve kaynak kodu makine diline çeviren programa **derleyici – compiler** adı verilir. Derleyici, yazılan programı doğrudan makine diline dönüştürüp bir derlenmiş, ikili dosya veya nesne dosyası oluştururlar. Bu dosya var olan durumu ile doğrudan işlemci tarafından işlenmesi söz konusu değildir. Bu işlemin ardından **bağlayıcı – linker** adı verilen bir diğer program, derleyicinin ürettiği çıktıya gerekli olan diğer program bileşenlerini ekleyerek işlemcinin işleyebileceği bir yapıya dönüştürür. Bu dönüşüm sonucunda elde edilen dosya doğrudan çalıştırılabilir. Bazı durumlarda çalıştırılan program dosyası beklenen sonuçları vermeyeceği gibi bazen de hata mesajı döndürülüp sonlanabilir. Bu durumda programın neden bu şekilde sonlandığını anlayabilmek için programın derlenmiş ve makine diline çevrilmiş olan dosyasını çözümleyen ve programın makine dilindeki karşılığını inceleyerek olası problemleri belirlememiz sağlayan **hata ayıklayıcılar – debugger** kullanılır. Derleyici, bağlayıcı ve hata ayıklayıcılar ve başka yardımcı yazılımlardan oluşan programların bütününe birden **geliştirme araçları – development tools** veya **tool chain** adı verilir. C++ derlenen bir dil olduğu için öğrenilmesini kolaylaştmak için de bir derleyici ile programlama dilinde program yazılmasını sağlayacak bir metin düzenleyici ile diğer araçlara gereksinim duyulacaktır. Kullanılan işletim sistemine ve bu işletim sistemi üzerinde çalışan çeşitli yazılımlar kullanılarak C++ programları yazılabilir, programda bulunan hatalar belirlenip düzeltilebilir ve derlenebilir.

## 0.5.Geliştirme Araçları

Eğer geliştirme araçlarının komut satırı üzerinden gerekli seçeneklerin tanımlanarak kullanılması yerine bunların önceden tanımlanarak ve görsel bir arayüz üzerinde bir dizi diyalog kutusu aracılığı ile kullanılması tercih ediliyorsa bu tür araçlara ve arayzlere ise **tümleşik geliştirme ortamı – Integrated Development Environment** adı verilir. Tümleşik geliştirme ortamları derleyici ve gerekli olan diğer geliştirme araçları için bir arayüz sunarak programının ilgili araçları doğrudan değil dolaylı olarak kullanmasını sağlar. Bu dolaylı erişim ise daha fazla işlemci, bellek

ve sabit disk alanının kullanılmasını gerektirmektedir. Geliştiricinin hazırlayacağı programın veya yazılımın sağlanması gereken özelliklere göre uygun olan araç veya araçlar tercih edilmelidir.

C++ uluslararası standartlar birliği tarafından standartlaştırılmıştır. Çeşitli grupların üzerinde çalıştığı ve geliştirdiği dil, düzenli olarak güncellenen standartlar ile desteklenmektedir. Standartlar dilin özellikleri ile kullanılacak olan derleyicilerin sahip olması gereken özellikleri belirtmektedir. Erişilebilir durumda olan çeşitli derleyiciler ve geliştirme araçlarının tamamı yürürlüğe girmiş olan standartlar ile tam uyumlu olmayıabilmektedir. C++ kullanırken yararlanabileceğiniz ve dilin tüm özelliklerini geniş ölçüde destekleyen derleyiciler ve diğer geliştirme araçları arasından aşağıda belirtilmiş olanlardan uygun olanı kullanmanız yararlı olacaktır.

Kullanabileceğiniz geliştirme araçları ve ortamları aşağıda verilmiştir.

### Tümleşik Geliştirme Ortamı – Integrated Development Environment

Code:blocks      <http://www.codeblocks.org/>

### Geliştirme Ortamı Olarak Metin Düzenleyiciler

Geany      <https://www.geany.org/>

### Derleyiciler ve Araçlar

CLANG      <https://clang.llvm.org/>

GCC      <https://gcc.gnu.org>

Cppcheck      <http://cppcheck.sourceforge.net/>

### Hata Ayıklayıcılar

GDB      <https://www.gnu.org/software/gdb/>

DDD      <https://www.gnu.org/software/ddd/>

Yukarıda adı geçen araçları kullanılmakta olan işletim sisteme göre değişkenlik gösterektir. Bazı işletim sistemlerinde örneğin BSD işletim sistemi ailesinde derleyiciler ve ilgili geliştirme araçları temel sistemin bir parçası olurken, GNU/Linux dağıtımları ve Microsoft'un ticari işletim sistemlerinde sonradan edinilmektedir. Bu nedenle kullandığınız işletim sisteminin sunduğu araçları kullanarak gereksinim duyduğunuz araçları edinmeniz yerinde olacaktır.

## 0.5.C++ Programlama Dili İçin Gerekli Geliştirme Araçları

### 0.5.1.Windows İçin Code::Blocks Kurulumu

Programlama dilleri için gerekli olan geliştirme araçları her işletim sisteminde hazır olarak kullanıcılar sunulmaz. Örneğin Windows kullanıcıları için geliştirme araçları ayrı kullanıcıların satın alabileceği bir ürün olarak sunulur. Bu notların yazıldığı sırada Microsoft firması ticari ürünü olan Visual Studio'nun ücretsiz ve ücretli olan sürümlerini aynı anda sunmaktadır. Bunlar yanında başka ücretsiz araçlarda sunmaktadır. Bu araçları kullanarak da burada verilen kaynak kodlar üzerinde çalışabilir ve derleyebilirsiniz. Microsoft'un sunduğu araçlar Microsoft'un ticari ürünleri ile Linux dağıtımları üzerinde çalışacak olan program kodunun yazılması olanaklıdır. Ancak bu araç-

ların sistem gereksinimleri dikkate alındığında (gerek boş disk alanı, gerek ağ erişimi (aktarılacak verinin miktarının büyülüğu ile bunun için gerekecek bant genişliği dikkate alındığında) daha az boş disk alanına ve bant genişliği ile ap üzerinden aktarılacak olan verinin büyülüğu dikkate alındığında kod yazımı ve derlemesi için Code::Blocks ve GEANY, derleyici olarak CLANG/LLVM veya GCC kullanılmasına karar verilmiştir. Codeblocks ile C/C++ ve wxWidgets projeleri geliştirebilirsiniz. Yazılım içerisinde GCC derleyicisi ve gerekli olan yazılımları hazır olarak sunmaktadır. GCC öncelikle Microsoft'un ticari işletim sistemi ailesi için yazılmadığı ve sonradan aktarıldığı dikkate alınmak durumundadır. Bu nedenle yazılacak olan kaynak kodların derlenmesi ve inşa edilmesi için GCC kullanılırken bazı ek yazılımlara da gereksinim olmaktadır. Code::Blocks bunları hazır olarak sunmakta ve gerekli yapılandırmaları da yaparak kullanılmaya hazır olarak sunmaktadır. Bunun için de kurulum sırasında önerilen ayarların seçilmesi yeterlidir. Code::Blocks GCC sürüm 8.1.0 kullanmaktadır. GCC bu yazılım içerisinde MinGW projesi tarafından hazırlanmış olan GCC'nin Microsoft'un ticari işletim sistemlerinde çalışabilen ikili dosyaları ile (binary, executable) sunulmaktadır. Bu sürüm ders notlarında kullanılmakta olan C++ programlama dilinin 2017 standarı ile tam uyumludur. (Farklı bir derleyici kullanmanız durumunda bu notlarda yer verilen kodlar çalışmaya bilmediği gibi ayrıca bazı ayarlar yapmanız da gerekebilir.)

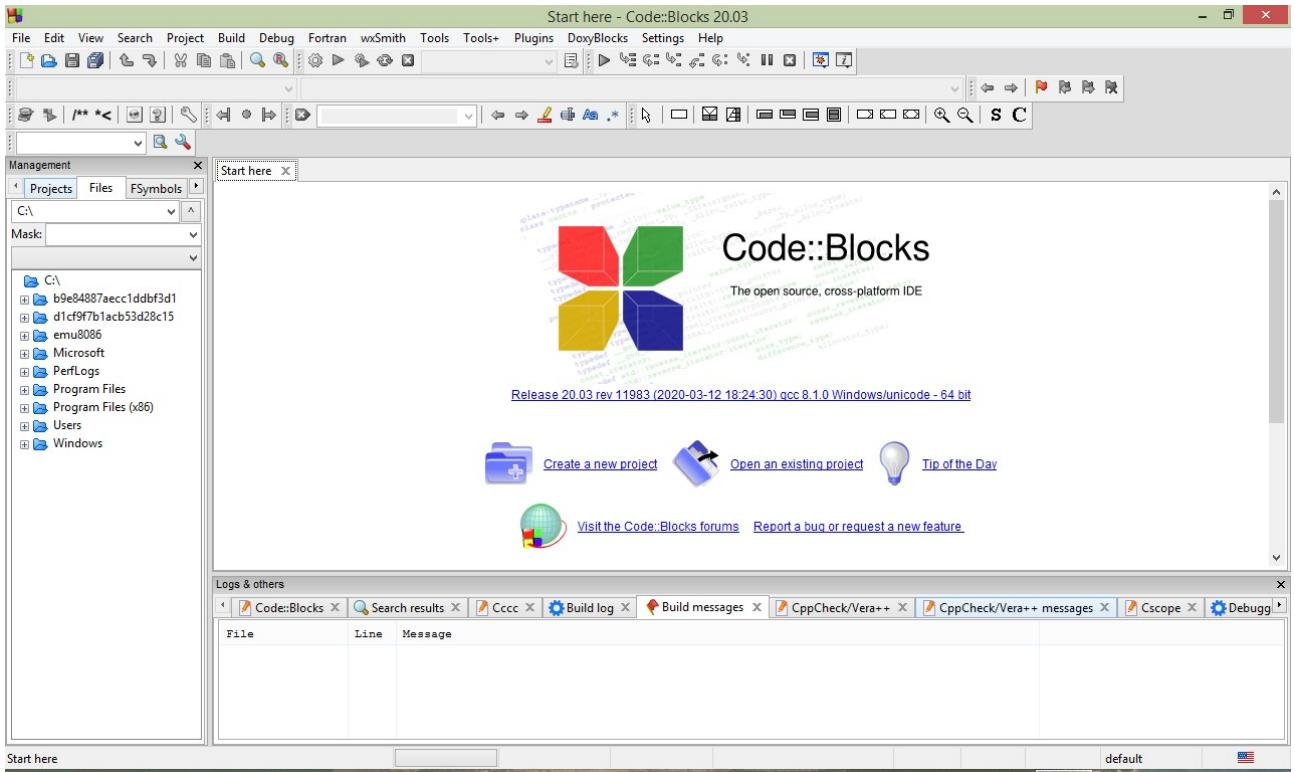
**Resim 0.3**'ten göreceğiniz üzere Code::Blocks temel olarak tüm ayarları ve araçları tek bir ara yüz üzerinde toplamaktadır. Menülerden de tüm ayarlara erişilebilir. Kurulum sırasında işletim sistemi ve işletim sisteminin 32 bit veya 64 bit olmasına göre yazacağınız programların nasıl derleneceği otomatik olarak belirlenmiş olmaktadır. Elinizdeki donanım Core, Core 2 sonrası veya AMD64 serisi işlemcilerden bir tanesi ise ve Windows sürümü de 64 bit ise derlediğiniz programlar 64 bit olarak çalışacaktır. Ama 32 bit derlemek isteseniz bunu derleyici seçeneklerinden değiştirip 32 bit olarak derleyebilirsiniz. Bu kitapta yer alan örneklerde kullanılan işletim sistemlerinin tümü 64 bit olduğu için örnekler de 64 bit olarak derlenmiştir. 32 bit olan bir sistem kullanıyorsanız ortaya çıkabilecek tek fark programın inşa edilip çalıştırılması sonrasında farklı sonuçlar elde edecek olmanızdır.

Bu notlarda yer verilen kodların yazılıp derlenmesinde kullanılacak olan C++ programlama dili standartı ile derleyici ve inşa komutlarının yapılandırılması gereklidir. Code::Blocks her ne kadar varsayılan olarak GCC kullanmak olsa da derleme ve inşa komutlarına aktarılacak olan bazı parametreler ile kaynak kod derlenip inşa edilmeden önce kontrolden geçirilmektedir. Bu kontroller yazılan kodun amaçlandığı gibi çalışmasını sağlamak için önemlidir. Programları yazmada önce menü çubuğundan **Settings → Compiler...** seçerek inşa komutlarının yapılandırılmasına başlanmalıdır.

Derleyici seçenekleri bir arayüz üzerinden tanımlanmaktadır. Burada ilk olarak C++ 2017 ISO standartına göre programların derlenip inşa edilmesini sağlamak için açılan pencerede ilgili standart seçeneklerinin seçilmesi gereklidir. General bölümünde ilgili kutular üzerine tıklanarak Resim 0.4'de görüldüğü gibi gerekli seçimler yapılmalıdır.

Bu aşamanın ardından derleyicinin yazılan programlarda gerekli olan kontrolleri yapması için gerekli komut satırı parametrelerinin tanımlanması gereklidir. Bunun için de **Warnings** bölümune geçilip “**Enable all common compiler warnings (overrides many other settings) [-Wall]**” seçilmesi yeterlidir. Bu seçenek derleyicinin tüm hata mesajlarını göstermesini ve bu hataların

giderilmeden kaynak kodun inşa edilmemesini sağlar. Bunu seçmeyip altında yer alan hata kontrolerinden istediklerinizi işaretleyebilirsiniz.



**Resim 0.3:** Code::Blocks Ara Yüzü

Hata kontrolü seçeneklerine ait ayarların yapılmasıının ardından “**Optimization**” bölümünden derlenen kodların hangi seçeneklere göre optimize edilebileceği seçilir. Derlenip inşa edilmiş olan kodun daha hızlı çalışmasından boyutunun azaltılmasına kadar farklı seçeneklerden istenilenler seçilebilir. (**Resim 0.4**) Benzer şekilde “**CPU architecture tuning**” bölümünden kullanıktır. Bu ve önceki bölümdeki seçenekler yeri geldiğince ve gerektiği yerlerde ilerleyen bölümlerde açıklanacaktır.

Aşağıda verilen seçenekler de C++ standarı olarak 2017 yani C++17 kullanılmaktadır. Kaynak kod tüm hatalara karşı taranmakta ve ardından derlenmekte veya inşa edilmektedir. Hata denetimi için ayrı bir araç olarak **GnuDeBugger – GDB** ve yazılan kodda kullanılan gerekli kütüphanelerin bulunup bulunmadığını kontrol için de **Cppcheck** kullanılmaktadır. Bu program C++ kaynak kodunu hatalara karşı taramakta ve bulduğu hataları raporlamaktadır. Programa hangi dilde kontrol etmesi gereği ve hangi standarı uygulayacağı ve hata mesajlarının yapısı da dahil olmak üzere geniş bir yelpazede çeşitli kontroller tanımlanabilmektedir. Cppcheck ayrı bir proje olduğu için Sourceforge üzerinden indirilip kurulabilir. Cppcheck Microsoft'un 64 bit'lik ticari ürünleri ile kullanılabilmektedir. Programın indirilmesinin ardından cppcheck-2.0-x64-Setup.msi dosyasına çift tıklanarak kurulabilir. Cppcheck aynı zamanda Python programlama diline de gereksinim duymaktadır. Bu nedenle programı kurulurken Python kurulumunun da seçilmesi yerinde olacaktır. Kurulum aşamasının tamamlanmasının ardından Code::Blocks çalıştırılarak menü çubuğundan **Settings → Environment Settings...** ile açılacak olan arayüz üzerinden Cppcheck.exe dosyasının bulunduğu dizinin tam yolu tanımlanmalıdır. Tam yok ise eğer özel bir tanımlama

yapılmadı ise “**C:\Program Files\Cppcheck\cppcheck.exe**” olarak girilebilir. Böylece programlar inşa edilmeden önce hatalara karşı cppcheck ile kontrol edilebilir. Programın tam yolumun belirtilmesi dışında özel bir ayar yapılmasına gerek yoktur.

### 0.5.2.GNU/Linux Dağıtımları İçin C++ Geliştirme Araçları

GNU/Linux dağıtımlarının büyük bir kısmı kurulum sırasında yazılım geliştirme araçlarını hazır olarak sunmaz. Hatta işletim sistemi, yardımcı sistem yazılımları ile uygulama yazılımlarına ait kaynak kodlar de hazır olarak sunulmaz. Bunların yazılım depolarından edinilmesi gereklidir. Bunun için kullandığınız dağıtımın yazılım kaynaklarından gerekli paketleri kurmanız gereklidir.

GNU/Linux dağıtımlarında geliştirme için IDE kullanılabileceği gibi, IDE ile yapılabilen tüm işlemler tekil yazılımlar kullanılarak da yapılabilir. Bunun için öncelikle kaynak kod yazımı için GENAY veya bir diğer metin editörü kullanılabilir. Derleme ve bağlama işlemleri için GCC, CLANG ve hat ayıklama için de GDB kullanılabilir. IDE yapılabilen tüm işlemler için aynı işlevselliği sunan yazılımların kurulması ve bunların gerektiği şekilde ye metin düzenleyici içinde Kaynak kodların yazılması için bir metin editörü yeterli olacaktır. Ancak metin editörünün içerisinde doğrudan derleyici ve diğer araçları kullanmayı tercih ederseniz buna uygun olarak geliştirilmiş geany metin editörü kullanılabilir. (<https://geany.org/>) Gerekli derleyici ve diğer yazılımlarda yukarıda söz edildiği gibi kurulabilir. Dağıtım olarak **Debian** (<https://www.debian.org/>) bu notların yazıldığı sıradaki kararlı sürümü **Debian**, **FreeBSD**, **OpenBSD** tercih edilmiş, yazılan kodlar bu işletim sistemlerinde test edilmiştir.

Kurulum sonrasında geany kurmak için Synaptics Paket Yöneticisi veya komut satırından apt, apt-get-get veya aptitude kullanılarak geany kurulabilir. Geany geniş bir ekleni deposu da bulunmaktadır. Bunlar metin editörünün farklı özellikler ve araçlar ile kullanılmasını sağlamaktadır ancak burada bunlara değinilmeyecektir.

Kurulum için Synaptics Paket Yöneticisi’nde geany aratarak seçim yapabilir ve kurabilirsiniz. Ayrıca derleyici ve hata ayıklayıcılara da gereksinim olacağı için GCC veya LLVM de kurulmalıdır. GCC, FSF tarafından geliştirilen Gnu Compiler Collection adı verilen çeşitli programlama dillerine ait derleyicileri ve ilgili araçları sunan yazılım geliştirme paketidir. Genel Kamu Lisansı – GPL ile sunulmaktadır. LLVM ise GCC benzeri olan ama yapısında farklı kütüphaneler ve derleyiciler bulunduran bağımsız bir projedir. Bu kitapta kullanılan derleyiciler ve diğer araçlar için LLVM projesi araçları kullanılmaktadır. Uygulama ortaya çıkacak olan fark, derleyicilerin aynı kaynak kodu makine diline çevirdiklerinde derlenmiş dosya boyutu ile uygulamanın performansında gözlenmektedir. Buradaki örneklerde dosya boyutu farklarını görmek kolay iken sistem özelliklerine ve kullanıma bağlı olarak performans farklarının herhangi bir araç kullanılmadan fark edilmesi söz konusu değildir. Komut satırından kurulum yapmayı tercih edenler için:

```
apt-get install -y geany geany-common gcc g++ cppcheck
```

veya

```
aptitude install -y geany geany-common gcc g++ cppcheck
```

yazarak GCC ve geany kurulumu gerçekleştirebilir.

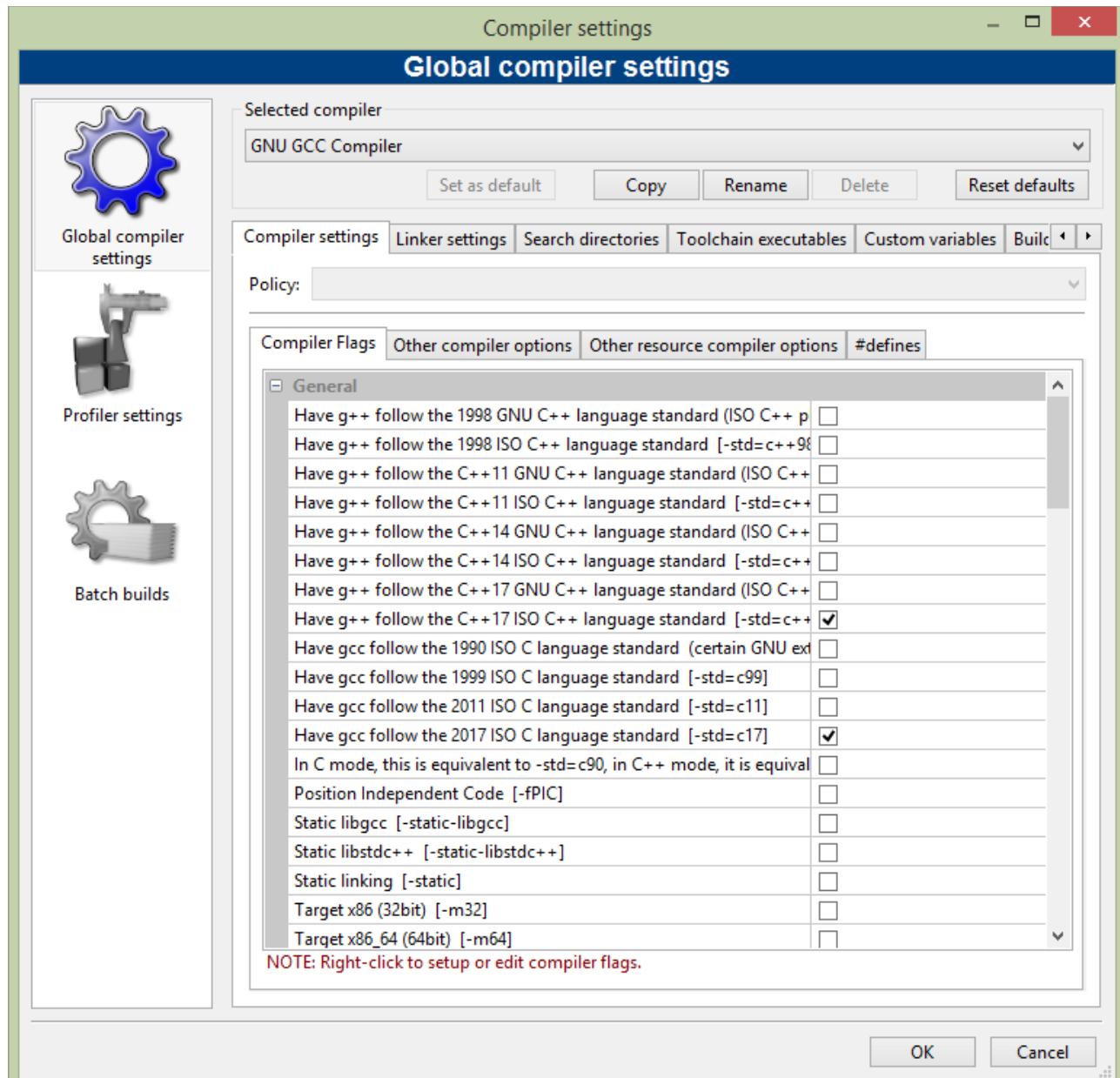
Eğer GCC yerine LLVM kullanmak isterseniz aşağıdaki gibi kurulumu yapabilirsiniz.

```
apt-get install -y geany geany-common llvm cppcheck
```

veya

```
aptitude install -y geany geany-common llvm cppcheck
```

yazarak kurulum yapılabilir.



**Resim 0.4:** Code::Blocks arayüzü üzerinden C++ 2017 ISO standardının seçilmesi ve derleyici seçeneklerinin seçilmesi

### 0.5.3.\*BSD İşletim Sistemi Ailesi İçin C++ Geliştirme Araçları

\*BSD işletim sistemi ailesi GNU/Linux dağıtımlarından farklı olarak işletim sisteminin kurulumu sırasında derleyici ve diğer geliştirme araçlarını temel sistemin bir parçası olarak sunar ve kurulumunu gerçekleştirir. Bu nedenle de ek araçlara gereksinim pek duyulmaz. Bazı yazılımların

kurulabilmesi için farklı derleyiciler veya ana sistemde bulunmayan yeni sürüme gereksinim olabilir. Bu durumda gerekli geliştirme araçları kurularak aynı sistem üzerinde hepsi kullanılabilir. Bu bölümde \*BSD işletim sistemi ailesi olarak FreeBSD ve OpenBSD esas alınmaktadır. NetBSD Türkçe desteği açısından yetersiz olması neden ile tercih edilmemiştir.

GNU/Linux dağıtımları için belirtildiği üzere \*BSD işletim sistemi ailesi üzerinde de tümleşik geliştirme ortamı (IDE – Integrated Development Environment) kullanılmamıştır. Kaynak kodların yazılması için herhangi bir metin editörü yeterli olacaktır. Aynı şekilde metin editörünün içерisinden doğrudan derleyici ve diğer araçları kullanmayı tercih ediyorsanız buna uygun olarak geliştirilmiş geany metin editörünü kullanabilirsiniz. (<https://geany.org/>) Gerekli derleyici ve diğer yazılımlarda FreeBSD için **/usr/ports/** dizini altında yer alan yazılım kurulum kaynakları – **portstree** – kullanılarak derlenenmiş haldeki yazılım paketleri kurularak hazır kullanılmaya başlanabilir.

OpenBSD için kendi yazılım yönetim aracı olan **pkg\_\*** kullanılmalıdır. OpenBSD de benzer şekilde eğer tüm kurulum setleri kullanılarak kurulmuş ise derleyici ve geliştirme araçları da hazır olmak üzere kullanılmaya hazırlıdır. Bir tek gerekli olan ise metin editörü olacaktır.

FreeBSD üzerinde geany portstree üzerinden derleyerek kurmak için şu adımları izleyin:

```
cd /usr/ports/editors/geany && make install && make clean
```

```
cd /usr/ports-devel/cppcheck && make install && make clean
```

FreeBSD üzerinde pkg kullanarak derlenmiş ve hazır yazılım paketlerini kurmak aşağıdaki komut yeterlidir.

```
pkg install geany cppcheck
```

OpenBSD kullanıyorsanız pkg\_add ile kurulum yapabilirsiniz.

```
pkg_add geany cppcheck
```

OpenBSD ve FreeBSD derleyici olarak LLVM/CLANG kullanmaktadır. Bu nedenle ek bir kurulma gereksinim olmayacağındır. Ancak LLVM/CLANG daha güncel sürümünü kullanmak isterseniz yazılım kaynaklarından yararlanabilirsiniz. Yukarıda adı geçen diğer yazılım ise **cppcheck** adlı C/C++ kaynak kodlarını hatalara karşı tarayan ve raporlayan araçtır. Bunun ile ilgili bilgi izleyen bölümde verilmiştir.

#### 0.5.4. Geany ile İnşa Komutları Yapılandırması

Geany, programcılar için geliştirilmiş olan bir metin editöridür. Bu nedenle de bir programcının gereksinim duyacağı özellikler dikkate alınarak geliştirilmiştir. Geany en önemli özelliklerinden bir tanesi geliştirme araçlarının doğrudan editör üzerinde bulunan menüler ile çağrılarak kullanılabilmesi ve aynı zamanda kendi yapısında bulunan terminal bölümü ile kullanılan geliştirme araçlarının döndürdüğü mesajların okunabilmesidir. Böylece kaynak kod içerisinde bulunan hatalar tespit edilebilmekte ve hatta doğrudan ilgili satırda geçiş yapılmaktadır.

Yukarıda belirtildiği üzere gerekli araçların kurulmasının ardından geany menüleri arasında yer alan “**İnşa et**” menüsünün en alt satırındaki “**İnşa Komutlarını Seç**” ile derleyiciler ve çeşitli kaynak kod kontrol araçlarını ve bunlara aktarılacak olan seçenekler doğrudan tanımlanabilir. Diğer

ayarlar için “Düzenle” menüsünden “Seçenekler” tıklayarak gerekli diğer yapılandırma seçenekleri ayarlanabilir. Burada derlenecek olan C++ kodu için “İnşa Komutlarını Seç” menüsünde yer alan “Compile”, “Build”, “Lint” ile “Çalıştır” seçeneklerini yapılandıracağız. “Compile” alanı kaynak kodun derlenerek **nesne koduna (object code)** dönüştürülmesi komutlarını içerir. “Build” alanı inşa aşaması için gerekli komutları tanımlar. Kaynak kodun derlenmesi ve varsa diğer kütüphaneler ile bağlantılıdırılmasını ve işlemin sonucunda doğrudan çalıştırılabilen (**executable**) yazılımın hazırlanmasını sağlar. “Lint” alanı ise kaynak kod üzerinde derleme veya inşa işlemi yapılmadan önce kaynak kodun çeşitli testlerden geçirilmesini, bu testlerin sonucunda bulunan hataların döndürülerek gereken düzeltmelerin yapılmasını sağlayan araçlar ve seçeneklerin tanımlandığı alandır.

Aşağıda verilen seçenekler de C++ standarı olarak 2017 yani C++17 kullanılmaktadır. Kaynak kod tüm hatalara karşı taranmakta ve ardından da derlenmekte veya inşa edilmektedir. Hata denetimi için ayrı bir araç olarak cppcheck (<http://cppcheck.sourceforge.net/>) kullanılmaktadır. Bu program cpp kaynak kodunu hatalara karşı taramakta ve bulduğu hataları raporlamaktadır. Programa hangi dilde kontrol etmesi gereği ve hangi standartı uygulayacağı ve hata mesajlarının yapısı da dahil olmak üzere geniş bir yelpazede çeşitli kontroller tanımlanabilmektedir.

#### Compile:

```
/usr/bin/c++ -std=c++17 -Wall -Weffc++ -Wextra -I/usr/local/include -c "%f"
```

#### Build:

```
/usr/bin/c++ -std=c++17 -Wall -Weffc++ -Wextra -I/usr/local/include -o "%e" "%f"
```

#### Lint:

```
/usr/local/bin/cppcheck --language=c++ --std=c++17 --check-config --enable=warning, style --template=clang++ "%f"
```

Yapılurma seçeneklerinin tanımlanmasının adından “TAMAM” tuşuna basarak gerekli işlemleri sonlandırabiliriz.

### 0.5.5.C++ Programlarının İnşa Edilmesi

Yazılan programlar inşa edilmeden önce dosya olarak kayıt edilmesi gereklidir. Genel kabul görmüş kayıt formatı dosya uzantısının “.cpp” olmasıdır. Windows işletim sistemi ile çalışanlar için dosya uzantılarının önemi büyütür. Çünkü Windows işletim sistemi için dosyanın ne olduğunu belirten ayıt edici etken dosya uzantısıdır. Dosya uzantısının değiştirilmesi durumunda Windows işletim sistemi üzerinde çalışıyorsanız dosya uzantısını neye dönüştürürseniz o uzantılı ile ilişkilendirilmiş olan program çalıştırılır. UNIX ve türevi sistemlerde ise dosya uzantısı önemli değildir. Ancak dosyaların üzerinde işlem yapılrken dosya türünün belirlenmesini dosya uzantısı kolaylaştırdığı için bu kurala uyulur. Ayrıca kaynak kodların dağıtılması ve farklı projeler için aynı isimli dosyalara sahip olması ve dağıtılrken sorun olmaması için bu kural benimsenmiştir. Bir tek farklı UNIX türevi sistemlerde C++ kaynak kodları aynı zamanda cxx uzantısı ile de yer alabilir.

**Kaynak kod** algoritmaların programlama dili ile ifade edilmesidir. Kaynak kod da doğrudan bilgisayarın işleyebilecegi bir program değildir. Bu nedenle de bir dizi işleminden geçirilmesi ve bilgisayarın daha doğrusu CPU tarafından işlenebilecek şekilde dönüştürülmesi gereklidir. Bunun için

**derleyici** olarak anılan bir grup yazılım kullanılır. Derleyiciler sadece kaynak kodu makine diline dönüştürmez. Bu işleminden önce bir dizi işlemi de yerine getirir.

Derleme işleminin ilk aşaması kaynak kod dosyasının içerisinde tanımlanan **önişlemci direktifleri** adı verilen ve programlama diline özgü plan bir dizi özel kütüphane ve fonksiyonların kaynak kod dosyasına dahil edilmesini sağlar. Bunlar kaynak kod içerisinde **#include** ile başlayan satırlarda tanımlanır. Bu aşama **önişlemci – preprocesser** aşaması olarak tanımlanır.

Derleyici sonraki aşamada kaynak kodun son halini programlama dilinin yazım kurallarına göre kontrol eder. Bu “**imla hatası**” kontrolü aşamasında eğer yazılan kaynak kod üzerinde belirlenen tüm hatalar mesaj olarak döndürülür. Hata olması durumunda derleyici işleyişini durdurur. “İmla hataları”nın giderilmesi için kaynak kod belirtilen hata mesajlarına göre incelenir ve hatalar düzelttilir. Düzletmeler tamamlandıktan sonra inşa sürecine yeniden başlanır. Eğer kaynak kod üzerinde bir “imla hatası” bulunmadıysa derleyici kaynak kodu, CPU tarafından anlaşılan **makine diline – machine code** dönüştürür. Bu dönüştürme işlemi sonucunda **nesne dosyası, nesne program – object program** elde edilir.

Derleyicinin burada resmi olarak işi bitmiştir. Bundan sonra ise **bağlayıcı – linker** adı verilen program üretilen nesne dosyasının CPU tarafından işlenebilir hale gelmesini sağlayan diğer programlar yani kütüphaneler ile nesne dosyasının ilişkilendirilmesini sağlar. Böylece artık çalıştırılabilen bir program elde edilmiş olur. Bu işlemin sonucunda elde edilen **çalıştırılabilir – executable** dosya kullanılmaya hazırır.

```
merhaba.cpp - /home/goksin/projeler/C++, Notlar - [C++, Notlar] - Geany
Dosya Düzenle Ara Görünüm Döküman Proje Inşa Et Araçlar Yardım
Semboller Dökümanlar merhaba.cpp x
Fonksiyonlar
  main [41]
Dig Değişkenler
  std [39]
23 * REDDEDİLMİSTİR. HİBİD KOSULUDA TELİF HAKKI SAHİPLERİ VE KATKIDA
24 * BULUNANLARI, HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE KUSURUZ
25 * SORUMLUOLUŞTA, VEYA İZMİT YOKUMLUOLUĞUNDAN OLMAK ÜZERE MANGI
26 * YOKUMLUÜK KURAMINDA PAR ALIRSA, ALŞIN, İSBÜ YAZILMAK KULLANIMITYLA
27 * ORTAYA ÇIKAN DOĞRADAN, DOLANLI, TESADÜFİ, ÖZEN, CEZAI VEYA BİR SEBEP
28 * SONUCUNDA OLUSAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ, KULLANIM, VERİ
29 * VEYA RANDİMMİ KAYBI; YA DA İŞ KESİNTİSİ Dİ DAHİL OLMAK ÜZERE VE
30 * BUNUNLA KİSSİTLİ KALMAKSIZİN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
31 * HASARLARIN OLASILIGI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
32 * SORUMLU DEĞİLLERDİR.
33 */
34
35
36 #include <iostream>
37 #include <locale>
38
39 using namespace std;
40
41 int main()
42 {
43     setlocale(LC_ALL, "");
44
45     cout << "Merhaba Türkiye!" << endl;
46
47     return 0;
48 }
49
50 */

Durum c++ -std=c++17 -Wall -o "merhaba" "merhaba.cpp" (/home/goksin/projeler/C++, Notlar dizinine)
Derleme başarılı.

Derleyici

Mesajlar

Karalama

satır: 35 / 50 kol: 0 seç: 0 INS TAB mode: LF kodlama: UTF-8 dosyaturu: C++ alan: bilinmeye
```

Resim 3: \*BSD, GNU/Linux işletim sisteminde Geany metin düzenleyicisi

Bu dosyalar doğrudan çalıştırılması olanaklı değildir. Çünkü üretilen çalıştırılabilir dosyanın belleğe yüklenmesi ve buradan okunarak işlemci tarafından işlenmesi gereklidir. Bu yükleme işlemi **yükleyici – loader** adı verilen program tarafından yapılır. Belleğe yüklenen program CPU tarafından işlenebilir.

## 0.6.Çözümle – Kodla – Çalıştır Tekniği ile Programlama

Programlama denildiğinde ilk akla gelen “bilgisayar için program yazabilmek” düşüncesidir. Aslında programlama bundan daha farklı olarak bir problemin çözülmesi için uygulanan tekniklerin bütünüdür. İnsanlar aynı veya benzer problemleri çözmek için farklı teknikler kullanmaktadır. Bu çeşitli teknikler inceleneceler olursa bazlarının kavranması ve uygulanmasının diğerlerine göre daha kolay olduğu görülmektedir. Bu kolay olan teknikler çözmek istenen problem veya problemler değişmiş olsa da üzerilerinde ufak tefek değişiklikler yapılarak yeni problemlerin çözümünde kullanılabilmektedir.

İyi bir programcı olabilmenin en iyi yolu iyi bir problem çözücü olmaktan geçmektedir. İyi bir problem çözücü olmanın farklı teknikleri olmakla birlikte yazara göre en iyisi algoritmik düşünmek ve bu düşünce yapısını uygulayabilmektir. Bu düşünce şekline göre bir problemin çözülmesi için şu adımlar izlenmelidir.

1. **Problemin tanımlanması:** Bir problemden söz edebilmek için öncelikle problemin varlığının ortaya konulması gereklidir. Bunu yapabilmenin yolu da problemin tanımlanmasıdır. Problemin tanımlanabilmesi için bulunması istenilen çözümün açıklanması, çözümün uygulanabilmesi için gerekli olan bileşenlerinin neler olduğu tanımlanmalıdır. Ve varsa daha önceden benzer veya aynı problemlere uygulanmış olan çözümler varsa bunlar araştırılır ve uygulanabilirlikleri araştırılır.
2. **Probleme ait çözümün tasarılanması:** Bir problemin çözülebilmesi için uygulanabilir ve problemin istenildiği şekilde çözülmeyi sağlayan bir çözüm önerisinin sunulması gereklidir. Bunun için problem tanımında verilen ve derlenen bilgilerden yararlanılır. Böylece problem ifadesi, verilen ve derlenen bilgilerden yararlanılarak problem çözmek için kullanılabilecek olan bir veya daha çok sayıda çözüm önerisi hazırlanır. Bu önerilerin bu aşamada uygulanabilir olması önemlidir. Uygulanamayacak olan çözüm önerileri elenir. Kalan öneriler belirlenmiş ölçütlerle göre değerlendirilir ve bunlar arasından en iyi ve uygulanabilir olan çözüm önerisi olarak seçilir.
3. **Problem ait çözüm önerisinin uygulanması:** Üzerinde uzlaşılan çözüm önerisinin tasarılandığı gibi problemin çözümü aşamasında uygulanmasına geçilir. Bu aşamada önceki süreçte öngörülemeyen veya koşulların değişmesi neden ile ortaya çıkan değişiklik gereksinimleri dikkate alınarak gerekli düzenlemeler yapılır ve uygulanır.
4. **Bakım aşaması:** Geliştirilen ve uygulanan problem çözümleri günlük yaşamın akışı içerisinde çeşitli etkiler nedeni ile yeni durumlara uyarlanması durumundadır. Bu nedenle problemin çözülmesi için geliştirilen öneriler ile bunların uygulamalarının var olan yeni duruma göre ele alınması ve gerektiğinde üzerinde düzenlemeler yapılması gereklidir. Bu çalışmalar ise bakım aşamasını ortaya çıkarır ve uygulanan çözümün öngörülen yaşam döngüsü süreci boyunca düzenli olarak aynı titizlikle yürütülmesini gerektirir.

Bu düşünce yapısının programcının bakış açısı ile bir programın yazılmasında şu aşamalardan oluşan yapı olarak ifade edilmesi söz konusudur. (Şekil 1)

**1. Adım: Problem**

**2. Adım: Çözümleme**

<b>3.Adım: Algoritma Tasarımı ve Seçimi</b>	
<b>4.Adım: Kodlama</b>	
<b>5.Adım: Önişlemci</b>	
<b>6.Adım: Derleme</b>	<b>(Hata durumunda 4. adıma geri dönülür)</b>
<b>7.Adım: Bağlayıcı</b>	<b>(Kütüphaneler)</b>
<b>8.Adım: Yükleyici</b>	
<b>9. Adım: Çalıştırma</b>	<b>(Hata durumunda 4, 3 ve 2 adıma geri dönülür)</b>
<b>10. Adım: Sonuç</b>	

## 0.7.Programlama Paradigmaları

Programlama paradigmaları önceki bölümde anlatılan bir problemin bilgisayar programı olarak çözülmesi aşamalarının farklı büyüklükteki problemler için nasıl uygulandığını belirten bir çerçevedir. Bu ders notlarında yapılandırılmış programlama ile nesne yönelimli programlama paradigmalarına yer verilmiştir.

### 0.7.1.Yapısal Programlama – Structured Programming

Yapısal programlama,bir problemin bütün olarak değil, daha küçük problemlere bölünmesi yolu ile çözülmesi yaklaşımını benimser. Bu yaklaşımda problemler olabildiğinde küçük parçalara ayrılır. Her bir parça kendi özelinde analiz edilir, algoritması tasarlanır ve ardından kodlanır. Alt problemlerin tamamı çözümlendikten sonra elde edilen çözümler birleştirilir ve asıl problemin çözümü elde edilmiş olur. Bu yönteme yapısal tasarım ve yapısal programlama adı verilir. Bazı kaynaklarda sistem yaklaşımı, şelale modeli, yukarıdan aşağıya veya aşağıdan yukarıya yaklaşımı olarak da ifade edilirken çeşitli kaynaklarda iteratif yaklaşım olarak da yer verilir.

### 0.7.2.Nesne Yönelimli Programlama – Object Oriented Programming

Nesne yönelik programlama yönteminde bir problemin çözülebilmesi için nesne adı verilen bileşenlerin belirlenmesi ile başlanır. Bir nesne problemin çözülmesi için önemli bir ögedir. Nesneler birbiri ile etkileşim içinde olduklarından programlama aşamasında nesneler arası ilişkiler esas alınır. Bir nesne tek başına bir çıktı üretmez. Bunun için de veriler üzerinde işlemler yapması gereklidir. Nesne yönelik programla konu ilerleyen bölümlerde ele alınacaktır.

# 1.C++ Programlama Dilinin Temel Bileşenleri

Programlama dilleri, bilgisayar kullanılarak bir problemin çözülebilmesi için kullanılacak olan yönteme ait başlangıç adımından başlayarak sona kadar işlenen ve sonlu sayıdaki işlem basamaklarının bilgisayara aktarılarak, çalıştırılmasını sağlayan yönergelerdir. Bu yönergeler tek başına bir işe yaramazlar. Tersine, bu yönergelerin doğru bir şekilde bir araya getirilmesi ile biz insanların aramızda iletişim kurmak için kullandığımız dilimiz gibi belirli bir kurala göre yazılması gereklidir. Programlama da bu benzetmeden anlaşıldığı gibi hem bilgisayarlar ile iletişim kurmak için kullanılan hem de aynı zamanda bazı problemlerin bilgisayar kullanılarak çözülebilmesi için problemin nasıl işlenerek sonuç elde edileceğini belirten sıralı ve sonlu sayıdaki yönergelerin kümesidir. Bu yapılan tanımın eksik olduğu ileri sürülebilir, bu doğrudur. Programlama dilleri ile programlama konusunda konuyu tüm yönleri ile alan ve bunu eksiksiz bir şekilde tanımlamak için yazılmış olan çok sayıda kitap bulunmaktadır. Ancak bu notların kapsam C++ Programlama dilini öğretmek ve bu dili kullanarak bilgisayar programları yazmak olduğu dikkate alındığında yeterli bulunacaktır.

Programlama, basit olarak yemek yapmak ile oldukça benzerdir. Yemek yapmak için var olan tariflerden farklı olarak aynı yemek için yeni bir tarif hazırlamak kolay değildir. Bazı tarifleri kullanarak yemek yapmak kolaydır, bazıları ise daha fazla ustalık gerektirir. Her yemek tarifi lezizli yemekler yapacağınızı garanti etmez. Tamamen yeni bir yemek ve buna ait bir tarif hazırlamak hem malzemeyi hem de pişirme yöntemleri konusunda çok bilgili olmayı gerektirir.

Yemek tarifleri le programlama arasında doğrudan bir ilişki olmasa da yukarıda belirtilen konular, programlama ve programcılık için de geçerlidir. İyi bir şef olmak için sadece yemek tarifleri içeren bir kitap veya bu konu ile ilgili olarak yayınlanan çeşitli dergileri ve dergi bölümlerini okumakta yeterli değildir. Programlama öğrenmek ve programcı olmakta pek farklı değildir. Program yazabilmek için temel düzeyde programlama dili bilgisi ve kullanılan geliştirme araçlarını kullanabilme becerisine sahip olunması da gereklidir. Bunlara ek olarak yazılan programların test edilerek amaçlandığı gibi işlediğinden ve beklenen sonuçları ürettiğinden de program teslim edilmeden önce kontrol edilmelidir.

## 1.1.C++ Programlama Dili İle Yazılan Programların Temel Özellikleri

Programlama kitaplarında klasik olduğu üzere ilk yazılan program ekrana “Merhaba Dünya” yazdırın bir program yazarak başlamaktır. Bu geleneğin çıkış noktası ile C programlama dilini işletim sistemi yazmak için icat eden **Dennis Ritchie** ve daha sonra yakın çalışma arkadaşı olan **Ken Thompson** ile birlikte yazdığı “C programlama Dili” adlı kitaptan gelmektedir.

```
/*
* merhaba.cxx
*
*
```

\* Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde

\* kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki

\* ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki

\* ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

#include <iostream> // Girdi çıktı işlemleri için

#include <locale> // İşletim sisteminden dil ayarları alınır.

using namespace std;

```
int main()
{
    setlocale(LC_ALL, "Turkish");
    cout << "Merhaba Dünya!" << endl;
    return 0;
}
```

Programı önceki bölümlerde belirtilen araçlardan herhangi birisini kullanarak derleyip çalıştırırsanız aşağıdakine benzer bir çıktı elde edeceksiniz.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./merhaba
Merhaba Dünya!
[goksin@tardis ~/projeler/C++_Notlar]$
```

Yukarıdaki örnek basit bir mesaj yazdırmaktadır. Şimdi ise aynı programı biraz daha geliştirip bir kenar uzunluğu verilen bir karenin alanını ve çevresini hesaplayan bir programa bakalım.

```
/*
 * kare.hxx
 *
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 *
 *
 * Kaynak kodun yeniden dağıtım; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 *
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
```

- \* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİK VE ÖZEL BİR AMACA
- \* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
- \* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
- \* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
- \* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
- \* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
- \* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
- \* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
- \* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
- \* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
- \* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
- \* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
- \* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
using namespace std;
int kenar = 5;
int çevre = 4 * kenar;
int alan = kenar * kenar;
int main()
{
    setlocale(LC_ALL, "Turkish");
    cout << "Karenin bir kenarı " << kenar << " birimdir." << endl;
    cout << "Karenin çevresi " << çevre << " birimdir." << endl;
    cout << "Karenin alanı " << alan << " birim karedir." << endl;
    return 0;
}
```

Programı önceki bölümlerde belirtilen araçlardan herhangi birisini kullanarak derleyip çalıştırırsanız aşağıdakine benzer bir çıktı elde edeceksiniz.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./kare
```

Karenin bir kenarı 5 birimdir.

Karenin çevresi 20 birimdir.

Karenin alanı 25 birim karedir.

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

Programlar yazılırken, anadilimizde olduğu gibi programlama dillerinin de kendisine özgü yazım kullanıcıları ve düzeni vardır. Bu kurallara ve yazım düzenine uyulması hem kullanıcıların hem de diğer programcının, yazılan program/yazılım ile çalışmasını kolaylaştırmakla birlikte aynı zamanda ortak çalışan projelerde programcılar arasındaki iletişim ve işbirliğinin kurulmasını, geliştirilmesini ve yürütülmesini sağlar. Her yazılan program içerisinde yorum satırlarında geliştirici ekip üyeleri, iletişim bilgileri, lisans ile programın yazıldığı tarih de yer alır. Burada ders notları kapsamında programın yazıldığı tarih bilgisine kural olarak yer verilmemiştir.

### 1.1.1.Lisans

Her bir bilgisayar programının geliştiricisi tarafından yazılmış veya önceden kabul görmüş bir lisans metini bulunur. Bu metin, programcı veya geliştirici ekip ile kullanıcılar ve diğer programcılar arasında sunulan yazılımın hangi koşullar ile dağıtıldığı, programcı veya geliştirme ekibi ile kullanıcılar ve diğer programcılara, programın/yazılımın fikri mülkiyetinden başlamak üzere hangi koşullar ile dağıtilabileceği ve kullanılabileceği yanında karşılıklı olarak hakları ve sorumlulukları düzenleyen bir hukuksal belgedir. Seçilen lisansın ve bu lisansın açık bildirim metin olarak kaynak kodu ile birlikte veya programın derlenmiş, çalıştırılabilir kopyaları ile birlikte verilmesi, yasal sorumluluklar ile hukuksal süreçlerin işlemesi açısından karşılıklı olarak hakların ve sorumlulukların nasıl düzenlenliğinin yasal bir belgesi olarak sunulması hukuksal olarak gereklidir. Yazılan bir programın veya yazılımın diğer yazılım projelerinde kullanılabilmesi için programcının gereklili yasal koşulları öğrenmek ve bunların uygunluğunu kontrol etmek gibi bir süreci her yazılım için yineleyerek zaman ve emek kayıplarının önüne geçilmesi için standartlaştırılması yoluna gidilmiştir. Bu konuda günümüzde yaygın olarak kabul gören **Açık Kaynak/Özgür Yazılım – FLOSS/Free Software** modellerine uygun olarak **FSF<sup>5</sup>** ve **OSI<sup>6</sup>** lisansları arasından uygun bulunan bir tanesinin tercih edilmesi yoluna gidilebilir. Ancak programcının herhangi bir lisansı tercih etmeyeip kendi lisanslarını yazabilecekleri de dikkate alınmalıdır. Ancak temelden yazılan bir lisans metinin hukuksal olarak gereken koşulları sağlaması gereği unutulmamalıdır.

**Lisans metinleri konusunda burada tüm kaynak kodlar BSD lisansı ile sunulmaktadır. Ders notları ise Creative- Commons Atıf-Gayri Ticari-Aynı Lisansla Paylaş 4.0 Uluslararası Lisans’ı ile yayınlanmaktadır.**

### 1.1.2.Yorumlar

Programın ilk satırında bir yorum satırı bulunmaktadır. Bir C/C++ programlama dili ile yazılan bir programda, programcı tarafından kaynak kodu inceleyen diğer programcılar ve kişilere

5 <https://www.fsf.org/licensing>

6 <https://opensource.org/licenses>

yönelik olarak programın lisansı, program içerisinde bazı ifadelerin veya komutların yazılmasına ilişkin açıklamalar, yorum satırı ile belirtilir. Bir yorum satırı tek bir satır veya çok sayıdaki satıdan oluşabilir. Eğer tek satırda ifade edilebiliyorsa aşağıdaki gibi yazılır.

```
// Bu tek satırlık bir yorumdur.
```

Eğer yapılacak olan açıklama birden çok sayıdaki satıra yayılıyorsa bu durumda aşağıdaki gösterimlerden bir tanesi tercih edilir.

```
/*  
 * Bu tek satırlık bir yorum değildir.  
 * Yorum birden çok satıra yayılmıştır.  
 */
```

Yukarıdaki yorum satırı benzer şekilde aşağıda görüldüğü gibi de yazılabilir.

```
//  
Bu tek satırlık bir yorum değildir.  
Yorum birden çok satıra yayılmıştır.  
//
```

Yorum satırlarında ilk olarak yer verilen tek satırlık yazım biçimini kullanılabılır. Ancak bunun olumsuz yanı komut satırından çalışılırken programın kaynak kodunda yer alan ifadeler ile yorum satırlarının bir bakışta birbirinden ayırt edilmesini zorlaştırmıştır. Okuma kolaylığı açısından bu kitapta birden çok satıra yayılan yorum satırları ile tek satırlık yorum satırları farklı şekillerde yazılmıştır.

Derleyiciler yorum satırlarını program derlenirken dikkate almazlar. Ancak lisans metinleri kullanılan kütüphanelere bağlı olarak programın derlenmiş olan çalıştırılabilir kodun içerisinde yer alır. Bir hata ayıklayııcı ile tercihen **Gnu Debugger – GDB** ile derlenmiş olan kod incelenirse kullanılan işletim sistemindeki derleyici ve/veya kütüphanelere ait **lisans metinlerinin** GDB çıktısında<sup>7</sup> yer aldığı görülür.

### 1.1.3.Özel Karakterler

Tüm programlama dilleri için değişmez olan tanımlardan birisi bir programlama dilinde yazılabilen en küçük anlamlı birim **token – varlık** olarak tanımlanır. C++ programlama Dili’nde ise varlıklar **özel karakterler – special symbols**, **özel sözcükler – reserved words** ve **tanımlayıcılar – identifiers** olarak üç gruba ayrılır. Özel semboller adından da anlaşılacağı üzere programlama dili tarafından farklı anlamlar ile yorumlanan semboller temsil eder. Bunlar arasında aritmetik işlemler, büyüktür, küçüktür, büyük eşit veya küçük eşit ile ters bölümü gibi semboller bulunur. Aşağıdaki ilk satırda aritmetik işlemler olan sırası toplama, çıkarma, çarpma ve bölme işlemlerini belirten semboller yer almaktadır. İkinci satırda ise noktalama işaretlerinden bazıları

<sup>7</sup> GDB, derlenmiş ve makine diline çevrilmiş olan bir programın kaynak kodunu yazıldığı programlama dilinde vermez. Programın kaynak kodu olarak programın Assembly dilindeki karşılığı olarak yer alırken, lisans metinleri ise asıl şeklinde yazdırılır.

görülmektedir. Bunlar program içerisinde kullanıldıklarında çeşitli işlemlerin gerçekleştirilmesini veya özel bazı işlevleri temsil ederler. Sırası ile ? İşareti belirli bir koşulun gerçekleşip gerçekleşmediğini kontrol ederken, noktalı virgül önceki örneklerde de görüldüğü gibi bir ifadenin sona erdiğini belirtirken, virgül aynı tipteki bir den çok verinin tek bir satır içerisinde tanımlandığında birebirinden ayrılması sağlar. Üçüncü satırındaki karakterler veriler arasında gerçekleştirilen küçük eşit, eşit değil, eşittir ve büyük eşit işlemlerini belirtmektedir. Burada yazılmamış olmakla birlikte “boşluk” da özel bir karakterdir.

+	-	*	/
?	;	,	.
<=	!=	==	>=

Yukarıdaki özel semboller incelemiş olduğumda ilk iki satırda yer alan varlıkların tek karakterden oluşturuları ancak son satırın ise birden fazla karakterden oluşmaktadır. Özel anlamaya sahip olan varlıkların programlama dili tarafından tanımlanmış olan özel işlevleri olduğundan derleyiciler bu sembollerin bütün olarak yani ayırmadan işlerler. Programlar yazılırken bu bileşik olan ama tek sembol gibi işlenen sembollerin gerektiği şekilde makine diline dönüştürür. Bu bileşik semboller yazılırken karakterler arasında boşluk bırakılmamalıdır. Aksi halde derleyici hata mesajı döndürür.

#### 1.1.4. Özel Sözcükler

Özel sözcükler aynı zamanda **anahtar sözcükler – keywords** olarak da adlandırılır. Kural olarak bir özel sözcük daima küçük harfler kullanılarak yazılır. Anahtar sözcükler veya özel sözcükler birden çok sayıda karakterden oluşsalar da bunlar da bütün olarak anlaşılmırlılar. Aşağıda C++ Programlama Dili’nine ait ISO standartlarına göre özel anlamlı olan sözcüklerin güncel ve tam listesi verilmiştir.

alignas (C++11)	alignof (C++11)	and
and_eq	asm	atomic_cancel
atomic_commit	atomic_noexcept	auto (1)
bitand	bitor	bool
break	case	catch
char	char8_t (C++20)	char16_t (C++11)
char32_t (C++11)	class (1)	compl
concept (C++20)	const	consteval (C++20)
constexpr (C++11)	constinit (C++20)	const_cast
continue	co_await (C++20)	co_return (C++20)
co_yield (C++20)	decltype (C++11)	default (1)
delete (1)	do	double

dynamic_cast	else	enum
explicit	export (1) (3)	extern (1)
false	float	for
friend	goto	if
inline (1)	int	long
mutable (1)	namespace	new
noexcept (C++11)	not	not_eq
nullptr (C++11)	operator	or
or_eq	private	protected
public	reflexpr	register (2)
reinterpret_cast	requires (C++20)	return
short	signed	sizeof (1)
static	static_assert (C++11)	static_cast
struct (1)	switch	synchronized
template	this (4)	thread_local (C++11)
throw	true	try
typedef	typeid	typename
union	unsigned	using (1)
virtual	void	volatile
wchar_t	while	xor
xor_eq		

(1) — C++ 11 standardında eklenmiştir.

(2) — C++ 17 standardında eklenmiştir.

(3) — C++20 standardında eklenmiştir.

(4) — C++ 23 standardında yeni anlama sahip olmuştur.

## 1.1.5.Tanımlayıcılar

Tanımlayıcılar bir C++ Programlama Dili ile yazılan bir programda değişkenlerin, fonksiyonların ve sabitlerin tanımlanarak kullanılmasını sağlayan varlıklardır. Tanımlayıcıların yazılmasına uyulması gereken özel kurallar vardır. Bunlar sırası ile şu şekildedir:

1. Bir tanımlayıcı yazılrken harf veya “\_” ile başlamalıdır.
2. Bir tanımlayıcı harfler, rakamlar ve “\_” ile yazılabılır.

3. Özel sözcükler ve karakterler tanımlayıcı olarak kullanılamaz.
4. Tanımlayıcıların karakter olarak uzunluklarına dair bir kısıtlama söz konusu değildir.

Bunlar dışında dikkat edilmesi gereken diğer özellik de C++ Programlama Dili'nin büyük küçük harf duyarlı oluşudur. Dolayısı ise bir tanımayıcı yazılırken örneğin X ile x farklı tanımlayıcılar olarak kabul edilerek işlenirler.

### **1.1.6.Boşluk**

Yazılan bir programda sık sık boşluk kullanılır. Bu programcılar tarafından yazılan bir programın diğer programcılar tarafından da kolaylıkla okunmasını sağlamaktadır. Ayrıca boşlukları günlük yazışmalarda sıkılıkla kullanmakta olduğumuz için üzerinde durulmaz. Programların yazılımasında boşluklar sadece programcılar işini kolaylaştırmakla kalmadığı gibi aynı zamanda veri girişleri sırasında da kullanılır. Ayrıca bir programlama dilinin varlıkları arasında bırakılan boşluklar aynı zamanda yazılan programların derleyici tarafından doğru şekilde ayırtılara makine diline çevrilmesini sağlar.

## **1.2.Veri Tipleri**

Yazılan tüm programlar veri kümesi üzerinde işlemler gerçekleştirir. Örneğin bir üretim planlama yazılımı sayısal veri üzerinde aritmetik ve mantıksal işlemler gerçekleştirip bunun sonuçlarını sunarken, bir okuldaki öğrencilerin ada ve soyadına göre alfabetik olarak sıralanmasını sağlayan bir program sözel veri üzerinde işlemler yapacaktır. Bunlar gibi farklı işlemler farklı veri kümeleri üzerinde gerçekleştirir. Dolayısı ise öğrencilerin alfabetik olarak sıralanmasında aritmetik işlemler yapılmayacağı gibi üretim planlaması faaliyetlerinde de alfabetik olarak sıralama gibi bir işlem gerçekleştirilmeyecektir.

Tüm programlama dilleri üzerinde işlem yapılacak olan veriyi benzer şekilde grupperdir. Böylece belirli bir gruptaki ver ile gerçekleştirilebilecek olan işlemler önceden belirlenmiş olmaktadır. C++ Programlama Dili çeşitli veri grupları üzerinde yapılacak olan işlemler için özel kontrol mekanizmalarına sahiptir. Böylelikle doğru işlemler doğru veri üzerinde gerçekleştirilir, hataların önüne geçirilir.

Burada **veri tipi – data type** kavramını tanımlayabiliriz: Ortak özellikleri ve üzerinde gerçekleştirilebilecek olan işlemlerin belirli olduğu değerlerdir.

C++ Programla Dili'nde veri tipleri üç gruba ayrılır:

- 1. Basit veri tipleri – simple data types**
- 2. Yapılandırılmış veri tipleri – structured data types**
- 3. İşaretçiler – pointers**

### **1.2.1. Basit Veri Tipleri**

Basit veri tipleri C++ programlama Dili'nin yapısında yer alan temel veri tipleridir. Bu temel veri tipleri kullanılarak daha karmaşık veri tipleri geliştirilebilmektedir. Basit veri tipleri de kendi içerisinde üç gruba ayrılır.

- Tam sayılar – integral:** Ondalık basamaklara sahip olmayan tam sayı olarak ifade edilebilen veri tipidir.
- Kayar noktalı sayılar – floating points:** Ondalık basamaklara sahip olan ve matematikte gerçek sayılar olarak ifade edilen sayıları ifade edebilen veri tipidir.
- Sıralama – enumeration:** kullanıcı tarafından tanımlanan ve sıralı olarak ifade edilen ve sırası belirli olan verilerin ifade edilmesi için kullanılan veri tipidir.

Tam sayı veri tipi ise kendi içerisinde tekrar alt gruplara ayrılmaktadır. Bu ayrıca göre yam sayı veri tipi şu alt türlere ayrılır: **bool**, **char**, **short**, **int**, **long**, **unsigned char**, **unsigned short**, **unsigned int**, **unsigned long**, **long long** ve **unsigned long long**.

### **1.2.1.1.int Veri Tipi**

Veri tipleri arasında bu kadar çok ayırım yapılmasıının nedeni her birisinin belirttiği verinin sayısal büyülüğünün farklı olmasıdır. Örneğin mantıksal işlemler için kullanılan **bool** veri tipi bir byte büyülüktedir. Alabileceğini değer 1 veya 0 olmaktadır. **char** veri tipi ise -128 ile 127 arasında değer alır. Tam sayıları ifade etmek için kullanılan **int** veri tipi ise -2147483648 ile 2147483647 arasındaki değerleri temsil eder. Aynı şekilde **short** veri tipi ise -32768 ile 32767 arasındaki değerleri alabilir. Bu farklı veri tiplerinden hangisinin kullanılması gerekiği ise programcının çözmek durumunda olduğu probleme ve programın işleyeceği verinin özelliklerine bağlıdır. İlk bilgisayarlar için bellek pahalı bir bileşen olduğu için programcılar hem yazdıkları programı hem de kullandıkları bellek miktarını dikkatle kontrol etmeleri gerekmektedir. Bugün için bellek geçmişe göre daha ucuz olduğundan programcının bellek tüketimi konusunda dikkat etmesi gerekmemektedir. Belleği en iyi şekilde kullanmak için programcının öncelikle çözmek durumunda olduğu problemin sınır koşulları iyice anlaması gereklidir. Böylelikle hangi veri tipini kullanacağını kolaylıkla belirleyebilir. Bazı teknolojiler ise bellek kullanımını konusunda dikkatli olunmasını gerektirebilir.

Aritmetik işlemlerdeki tam sayılar bu veri tipine örnek verilebilir. Bu veri tipinin kullanımında iki önemli kural bulunmaktadır: Pozitif sayıların ifade edilmesi durumunda önlerinde "+" işaretи bulunmaz. İkincisi de sayıların bölgelerinin gösterilmesinde nokta kullanılmaz. Nokta sadece gerçek sayıların ifade edilmesinde kullanılır.

### **1.2.1.1.1.bool Veri Tipi**

Bu veri tipi için iki ayrik değer söz konusudur: Bir ve sıfır. Diğer bir deyişle doğru ve yanlış. Bu veri tipi ayrıca boolean veri tipi olarak da bilinir. Bu veri tipi sadece mantıksal ifadeler için kullanılır. C++ programlama dilinde bool, true ve false özel sözcüklerdir.

### **1.2.1.1.2.char Veri Tipi**

Tek bir karakter ile yapılan işlemler için kullanılan veri tipidir. Tek karakter dediğimiz ise harfler, rakamlar ve özel sembollerdir. Bu veri tipi klavyede görülen tüm karakterleri temsil edebilir. Karakter veri tipi olarak da bazı kaynaklarda isimlendirilir. Bu veri tipi ile bir değişken tanımlandığında kural olarak tek tırnak içerisinde gösterilmesi esastır. Özellikle de boşluk yazılrken tek karakter büyülüğünde bir boşluk olarak tek tırnak içerisinde yazılması gereklidir.

Karakterlerin gösterimi için yaygın olarak kullanılan ASCII olduğu düşünülse de başkaları da bulunmaktadır. Bunlar arasında en yaygın olarak kullanılan bir diğeri de **Extended Binary**

**Coded Decimal Interchange Code (EBCDIC)**'dir. ASCII ile temsil edilen karakter kümesinde 128 adet karakter bulunurken, EBCDIC ise 256 adet karakterden oluşur. Hem ASCII hem de EBCDIC yapısında temsil edilen karakterler belirli bir sıraya göre ikilik tabandaki ifade edildikleri sıra ile kodlanmıştır. Bu sıralamaya **birleşik sıralama – collative sequence** adı verilir. Böylece hangi karakterin diğerine göre önce geldiği belirlenebilir.

ASCII karakter tablosunun incelenmesinden dikkat edileceği üzere tablonun ilk 14 karakteri yazdırılamayan karakterlerdir. Ancak tablonun ilk ögesinin kodu 0 olduğundan yeni satır karakteri olan '\n' ise 13 ile gösterilir. Her ne kadar yeni satır karakteri iki karakterden oluşuyor olsa da tek karakter olarak kabul edilir. Aynı şekilde verinin sonunu temsil eden **null terminator** yani '\0' ile sekme karakteri olan '\t' aynı şekilde tek karakterdir.

### **1.2.1.2. Kayar Noktalı Sayılar Veri Tipi (Floating-Point Data Types)**

Gerçel sayılar üzerinde yapılacak olan işlemler için C++ Programlama Dili kayar noktalı sayı veri tipini kullanır. Gerçel sayılar doğrudan ondalık basamaklara sahip olan sayılar şeklinde ifade edilebileceği gibi aynı zaman da bilimsel gösterim olarak da bilinin  $10^x$  kuvvetleri şeklinde de ifade edilebilirler. Gerçel sayılar ile yapılan işlemler için tam sayılar da olduğu gibi kendi içerisinde faktörlü türlere ayrılır: **float, double ve long double**.

#### **1.2.1.2.1. float Veri Tipi**

Bu veri tipi 4 byte büyüklüğündedir. 1.17549e-38 ile 3.40282e+38 arasındaki değerler için kullanılır.

#### **1.2.1.2.2. double Veri Tipi**

Bu veri tipi 8 byte büyüklüğündedir. 2.22507e-308 ile 1.79769e+308 arasındaki değerler için kullanılır.

#### **1.2.1.2.3. long double Veri Tipi**

Bu veri tipi 16 byte büyüklüğündedir. 3.3621e-4932 ile 1.18973e+4932 arasındaki değerler için kullanılır.

Yukarıda verilen değerler derleyiciye, işletim sisteme ve işlemciye bağlı olarak değişmektedir. 32 bit işlemcili bir sistemde elde edilecek olan sonuçlar ile 64bit işlemcili bir sistemde elde edilecek olan sonuçlar farklı olacaktır. Bu farklar için kullanılan derleyicinin belgelerine bakmak gereklidir. Ayrıca kayar noktalı sayıların gösterimi için IEEE754 standartı kullanılır. Bu standarda göre noktadan sonra gösterilebilecek olan ondalık basamak sayısı sayının ifade edilmesinde kullanılan önemli bitlerin sayısına göre değişkenlik gösterir. Float için bu digit sayısı altı ile yedi arasında olabilirken double için ise bu sayı on beş adettir.

Ondalık basamakların sayının belirlenmesinde kullanılan digit sayısı **duyarlılık – precision** olarak tanımlanır. Eğer işlemler için **float** veri tipi kullanılırsa bu **tek duyarlılık – single precision** olarak isimlendirilirken, **double** veri tipi kullanılırsa bu **çift duyarlılık – double precision** olarak tanımlanır. Bazı derleyiciler için C++ Programlama Dili ile yazılan programlarda kayar noktalı sayılar için varsayılan veri tipi double olarak belirlenmiştir. Eğer veri tipi olarak float belirtilirse bu durumda derleyici hata mesajı döndürebilir. Bu derleyici kaynaklı bir hatadır. Programdan kaynaklı bir hata değildir. Aşağıdaki programda C++ Programlama Dili'yi tarafından desteklenen veri

tiplerinin bellekte kapladığı alanı ve bu alan kullanılarak hesaplanabilecek olan sayısal verilerin büyülü olarak alt ve üst değerleri döndürülmektedir.

```
/*
 * datatypes.hxx
 *
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımlı; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
 * SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
 * VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
```

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <climits>
#include <cfloat>
#include <locale>
using namespace std;

int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");

    //bool için okuma yazma
    cout << "bool veri tipi büyüklüğü = " << sizeof(bool) << " byte" << endl;

    //char için okuma ve yazma
    cout << "char veri tipi büyüklüğü = " << sizeof(char) << " byte" << endl;
    cout << "char değer aralık: " << CHAR_MIN << " ... " << CHAR_MAX << endl;

    //short için okuma ve yazma
    cout << "short veri tipi büyüklüğü = " << sizeof(short) << " byte" << endl;
    cout << "short değer aralık: " << SHRT_MIN << " ... " << SHRT_MAX << endl;

    //int için okuma ve yazma
    cout << "int veri tipi büyüklüğü = " << sizeof(int) << " byte" << endl;
    cout << "int değer aralık: " << INT_MIN << " ... " << INT_MAX << endl;

    //long için okuma ve yazma
    cout << "long veri tipi büyüklüğü = " << sizeof(long) << " byte" << endl;
    cout << "long değer aralık: " << LONG_MIN << " ... " << LONG_MAX << endl;

    //unsigned char için okuma ve yazma
    cout << "unsigned char veri tipi büyüklüğü = " << sizeof(char) << " byte" << endl;
    cout << "unsigned char değer aralık: " << 0 << " ... " << UCHAR_MAX << endl;

    //unsigned short için okuma ve yazma
```

```

cout << "unsigned short veri tipi büyüklüğü = " << sizeof(unsigned short) << " byte" << endl;
cout << "unsigned short için değer aralık: " << 0 << " ... " << USHRT_MAX << endl;
//unsigned int için okuma ve yazma
cout << "unsigned int veri tipi büyüklüğü = " << sizeof(unsigned int) << " byte" << endl;
cout << "unsigned int değer aralık: " << 0 << " ... " << UINT_MAX << endl;
//unsigned long için okuma ve yazma
cout << "unsigned long veri tipi büyüklüğü = " << sizeof(unsigned long) << " byte" << endl;
cout << "unsigned long değer aralık: " << "0" << " ... " << ULONG_MAX << endl;
//long long için okuma ve yazma
cout << "long long veri tipi büyüklüğü = " << sizeof(long long) << " byte" << endl;
cout << "long long değer aralık: " << LLONG_MIN << " ... " << LLONG_MAX << endl;
//unsigned long long için okuma ve yazma
cout << "unsigned long long veri tipi büyüklüğü = " << sizeof(unsigned long long) << " byte" << endl;
cout << "unsigned long long değer aralık: " << "0" << " ... " << ULLONG_MAX << endl;
//float için okuma ve yazma
cout << "float veri tipi büyüklüğü = " << sizeof(float) << " byte" << endl;
cout << "float değer aralık: " << FLT_MIN << " ... " << FLT_MAX << endl;
//double için okuma ve yazma
cout << "double veri tipi büyüklüğü = " << sizeof(double) << " byte" << endl;
cout << "double değer aralık: " << DBL_MIN << " ... " << DBL_MAX << endl;
//long double için okuma ve yazma
cout << "long double veri tipi büyüklüğü = " << sizeof( long double) << " byte" << endl;
cout << "long double değer aralık: " << LDBL_MIN << " ... " << LDBL_MAX << endl;
return 0;
}

```

Program delenip çalıştırıldığında aşağıdakine benzer bir sonuç elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./datatype
bool veri tipi büyüklüğü = 1 byte
char veri tipi büyüklüğü = 1 byte
```

```
char değer aralık: -128 ... 127
short veri tipi büyüklüğü = 2 byte
short değer aralık: -32768 ... 32767
int veri tipi büyüklüğü = 4 byte
int değer aralık: -2147483648 ... 2147483647
long veri tipi büyüklüğü = 8 byte
long değer aralık: -9223372036854775808 ... 9223372036854775807
unsigned char veri tipi büyüklüğü = 1 byte
unsigned char değer aralık: 0 ... 255
unsigned short veri tipi büyüklüğü = 2 byte
unsigned short için değer aralık: 0 ... 65535
unsigned int veri tipi büyüklüğü = 4 byte
unsigned int değer aralık: 0 ... 4294967295
unsigned long veri tipi büyüklüğü = 8 byte
unsigned long değer aralık: 0 ... 18446744073709551615
long long veri tipi büyüklüğü = 8 byte
long long değer aralık: -9223372036854775808 ... 9223372036854775807
unsigned long long veri tipi büyüklüğü = 8 byte
unsigned long long değer aralık: 0 ... 18446744073709551615
float veri tipi büyüklüğü = 4 byte
float değer aralık: 1.17549e-38 ... 3.40282e+38
double veri tipi büyüklüğü = 8 byte
double değer aralık: 2.22507e-308 ... 1.79769e+308
long double veri tipi büyüklüğü = 16 byte
long double değer aralık: 3.3621e-4932 ... 1.18973e+4932
[goksin@tardis ~/projeler/C++_Notlar]$
```

### 1.3.Veri Tipleri, Değişkenler ve Atama İfadeleri

Bilgisayar programlarının temel işlevi veri işlemek olduğundan bir verinin tanımlanabilmesi ve üzerinde işlem yapılabilmesi için bir değişken ile tanımlanması ve bu değişkenin de veri tipinin belirtilmesi gereklidir. Bu tanımlayıcının işlevidir. Bir tanımlayıcı ile bir değişken tanımlanabilmesi

İçin değişkene bir isim verilmesi ve önceden belirttiğimiz gibi veri tipinin de belirtilmesi gereklidir. Bir değişkenin tanımlanması aşağıdaki gibi yapılır.

```
veri_tipi değişken_adi;
```

Aşağıda örnekler verilmiştir:

```
int sayac;  
float kdv_oran;  
char ilk_harf;
```

Bu örneklerde ilk değişken tanımında "sayac" adı verilen bir değişken tanımlanmış ve bunun int veri tipinde olup 4 byte büyülüğünde bellek tüketmekte olduğu belirtilmektedir. İkinci değişken kdv\_oran adını taşımaktadır. Kayar noktalı veri tipi olan float veri tipinde olduğu belirtilmiş ve bunun için de dört byte bellekte alan ayrılmıştır. Son örnek ise char veri tipinde olup sadece bir byte bellekte alan tüketmektedir. Bu değişkene ise ilk\_harf adı verilmiştir. Bu değişkenler için gereken tüm tanımlamalar yapılmıştır. Ancak şu an için değerleri belirli değildir.

```
int sayac = 10;  
float kdv_oran = 0.18;  
char ilk_harf = 'A';
```

Bir önceki tanımlamadan farklı olarak yukarıdaki tanımlamalarda değişkenin adı, değişkenin hangi veri tipinde olduğu ve bu değişkenin alacağı değer de tanımlanmıştır. Bu yazım şekline atama ifadesi adı verilir. Bir atama ifadesi bir değişkenin veri tipini, adını ve alacağı değeri tanımlamayı sağlar. Yukarıda ki ifadelerde değişkenler tanımlanmış ve bu değişkenlerin barındıracığı değerler atanmıştır.

## 1.4. Aritmetik İşlemler, İşlem Öncelikleri ve İfadeler

Bilgisayarların veri işleme özelliklerinin en fazla kullanıldığı alan aritmetik işlemlerdir. Standart aritmetik işlemler bağılayıcı ile ek kütüphanelere gerek duyulmadan doğrudan tam sayı ve kayar noktalı veri tipleri ile birlikte kullanılarak program içerisinde kullanılabilirler. Aritmetik işlemler ve operatörleri sırası aşağıdaki gibidir:

- Toplama +
- Çıkarma (veya negatif çevirme) -
- Çarpma \*
- Bölme /
- Kalan (veya modüler aritmetik yada kalan) %

Bu aritmetik işlemler operatörler ile gerçekleşir. Operatörlerin kullanımı sırasında şu noktala dikkat edilmesi gereklidir:

- **Toplama, çıkarma, çarpma ve bölme** operatörleri **tam sayı** ve **kayar noktalı** veri tipleri ile kullanılabilir.

- Bölme operatörü ile yapılan işlemlerde tam sayı ve kayar noktalı veri tipleri kullanıldığından daima işlem sonucunda **bölüm** döndürülür, kalan döndürülmez.
- Modüler aritmetik operatörü ile yapılan işlemler da sadece tam sayı veri tipi kullanılabilir. İşlem sonucunda döndürüle alışık bölüm işleminin sonucunda elde edilen **kalan**'dır.

Aşağıdaki örnekte gr cinsinden verilen bir ağırlık program tarafından kg ve gr olarak ayrılmaktadır.

```
/*
 * gr_kg_cevir.cxx
 *
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
```

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    int miktar = 2123;
    int kg, gr;
    //kg olan kısım hesaplanır.
    kg = miktar / 1000;
    cout << "2123gr = "<< kg << " kg ";
    //gr olan kısım hesaplanır.
    gr = miktar % 1000;
    cout << gr << " gr'dır." << endl;
    return 0;
}
```

Programın derlenip çalıştırıldığında aşağıdaki çıktı elde edilir:

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./gr_kg_cevir
2123gr = 2 kg 123 gr'dır.
[goksin@tardis ~/projeler/C++_Notlar]$
```

Bugüne kadar matematik derslerinden bildiğimiz aritmetik işlemler hem tam sayılar hem de gerçek sayılar ile gerçekleştirılmıştır. Cebir konusunda eşitlikler ve bilinmeyenler ile çeşitli işlemler

yapılmıştır. Programlama dilleri söz konusu olduğunda alışa geldiğimiz aritmetik işlemler bildiği-mizden farklı olmamakla birlikte anlamlandırılma açısından değişir. Örneğin  $4 + 5$ ,  $7 - 2$  ve  $x + 2 * 5 + 6 / y$  olarak gösterilen aritmetik işlemler programlama dillerinde “**aritmetik ifadeler – arithmetic expressions**” olarak tanımlanır. Bu ifadelerin hepsinde yer alan sayılar ile  $x$  ve  $y$  ise “**işle-nenler – operands**” olarak tanımlanır. Benzer olarak da örneğin  $-5$  bizim için negatif bir sayı olarak kabul edilirken programcılık açısından pozitif bir sayının negatif dönüştürülmesi işlemini temsil eder. Burada negatif işaretin bilgisayarın pozitif sayı olan beşi üzerinde işlem yaparak negatif beşe çevrilmesi işlemini temsil eder. Bu işlemde ise tek bir işlenen vardır. Bu ve benzeri ifadelere ise “**tek işlenenli operatörler – unary operator**” adı verilir. Aynı şekilde  $7 - 2$  gibi ifadeler ise “**iki işlenenli operatör – binary operator**” olarak tanımlanır. Toplama ve çıkartma operatörleri her ne kadar iki işlenenli operatör gibi algılansa da bir sayının pozitif sayı olarak tanımlandığı  $+ 2$  gibi bir ifade ise tek işlenenli operatör olarak işlev görür. Ancak bölme, kalan ve çarpma ifadeleri ise her zaman için iki işlenenli operatör olarak işlev görürler.

#### **1.4.1.İşlem Önceliği – İşlem Hiyerarşisi**

C++ programlama dili bir ifade içerisinde birden çok sayıda aritmetik işlem operatörünün kullanılması durumunda bu işlemlerin hangi sıra ile gerçekleştirileceğini belirlemek için işlem öncelikleri esas alınır. Bun göre aritmetik operatörler arasında işlem öncelikleri aşağıdaki gibidir:

- $*$  / ve  $\%$
- $+$  ve  $-$

Yukarıdaki listeden görüleceği üzere  $*$ ,  $/$  ve  $\%$  öncelik derecesi aynıdır. Benzer şekilde  $+$  ve  $-$  içinde aynı durum söz konusudur. Eğer aynı öncelik derecesine sahip olan operatörler tek bir ifade içerisinde birlikte kullanılırsa, bu durumda önceliklerin belirlenmesi için yukarıdaki kurallar aynı öncelik derecesine sahip aritmetik operatörler arasında işlemlerin soldan sağa doğru yürütülerek gerçekleştirilmesi ile çözümlenir. Aşağıdaki örnekte bu durum görülebilir.

$$3 * 7 - 6 + 2 * 5 / 4 + 6$$

$$(((3 * 7) - 6) + ((2 * 5) / 4)) + 6$$

$$= ((21 - 6) + (10 / 4)) + 6$$

Önce  $*$  gerçekleştir.

$$= ((21 - 6) + 2) + 6$$

$/$  gerçekleştir. Bu tam sayılı bölme işlemidir.

$$= (15 + 2) + 6$$

$-$  gerçekleştir.

$$= 17 + 6$$

$+$  gerçekleştir.

$$= 23$$

Sonuç.

Yukarıdaki örnekte kullanılan parantezler sadece işlem sırasının daha kolay görülebilmesi amacıyla kullanılmıştır. Ancak parantezler işlemlerin gerçekleştirilmesinde işlem önceliklerinin dikkate alınmadan da gerçekleştirilmesi için kullanılabilir. Aritmetik işlemlerin önceliklerinin

soldan sağa doğru esas alınarak yürütülmesinden dolayı aritmetik operatörlerin “**birleşme – associativity**” özelliliğinin soldan sağa doğru olduğu söylenebilir.

Yukarıdaki örneklerde tam sayı tipi kullanılarak aritmetik işlemler yapıldı. Benzer şekilde karakter veri tipi de kullanılarak aritmetik işlemler yapılabilir. Örneğin karakter veri tipi olarak tanımlanan ‘8’ için ASCII karşılığı tam sayı veri tipinde 56 olurken, tam sayı olarak tanımlanmış olan 8 için ise değeri yine aynıdır.

Karakter veri tipinde tanımlanan ‘8’ + ‘7’ işleminin sonucu 56 ve 55 toplamına karşılık olan 111’dir. Ama karakter değil de tam sayı olarak tanımlanan  $8 + 7$  işlem ise yine tam sayı olarak 15 karşılık gelir. Eğer aynı işlemi ‘8’ \* ‘7’ olarak yaparsak sonuç  $56 + 55 = 3080$  olur ama bu değerin ASCII karşılığı bulunmamaktadır.

Eğer karakter veri tipinde tanımlanmış veriler ile işlem yapılacak ise elde edilecek olan sonuçların büyüklükleri konusunda programcının çok dikkatli olması gerektiği unutulmamalıdır.

Aşağıdaki örnekte C++ Programlama Dili’nde tam ve kayar noktalı sayılar ile yapılan işlemlerin sonuçları görülemektedir.

```
/*
* aritmetik_islemler.hxx
*
*
*
* Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
* takdirde serbesttir:
*
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
*
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
*
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
```

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <cfloat>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    cout << "2 + 5 = " << 2 + 5 << endl;
    cout << "30 + 25 = " << 30 + 25 << endl;
    cout << "40 - 22 = " << 40 - 25 << endl;
    cout << "4 * 8 = " << 4 * 8 << endl;
    cout << "5 / 2 = " << 5 / 2 << endl;
    cout << "21 / 3 = " << 21 / 3 << endl;
    cout << "48 % 7 = " << 48 % 7 << endl;
    cout << "2 % 3 = " << 2 % 3 << endl;
    cout << "-34 % 5 = " << -34 % 5 << endl;
```

```

cout << "34 % -5 = " << 34 % -5 << endl;
cout << "5.0 + 3.78 = " << 5.0 + 3.78 << endl;
cout << "6.2 + 4.7 = " << 6.2 + 4.7 << endl;
cout << "-22.87 - 18.45 = " << -22.87 -18.45 << endl;
cout << "4.6 * 3.9 = " << 4.6 * 3.9 << endl;
cout << "5.0 / 2.0 = " << 5.0 / 2.0 << endl;
cout << "34.5 / 6.0 = " << 34.5 / 6.0 << endl;
cout << "34.5 / 6.5 = " << 34.5 / 6.5 << endl;
return 0;
}

```

Program derlenip çalıştırıldıktan sonra aşağıdaki çıktı elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./airtmeitk_islemeler
2 + 5 = 7
30 + 25 = 55
40 - 22 = 15
4 * 8 = 32
5 / 2 = 2
21 / 3 = 7
48 % 7 = 6
2 % 3 = 2
-34 % 5 = -4
34 % -5 = 4
5.0 + 3.78 = 8.78
6.2 + 4.7 = 10.9
-22.87 - 18.45-41.32
4.6 * 3.9 = 17.94
5.0 / 2.0 = 2.5
34.5 / 6.0 = 5.75
34.5 / 6.5 = 5.30769
[goksin@tardis ~/projeler/C++_Notlar]$
```

Programın çıktısında herhangi bir hata bulunmamaktadır. Bazı İşlemlerin sonuçları veri tipi tanımlamaları doğrudan sayısal olarak yapıldığı için bu veri tipinde gerçekleştirilen işlemlerin sonuçlarının da aynı veri tipinde olacağı unutulmamalıdır. Bir diğer önemli nokta ise % işlemlerinde sonuç sizin beklediğinizden farklıdır. Tam sayılı aritmetik işlemlerin son iki satırındaki işlemde sonuçlar doğrudur. -34'ün 5'e bölümünden kalan -4'tür. Aynı şekilde 34'ün -5 ile bölümünden kalan da 4'tür.

### 1.4.1.İfadeler

C++ programlama dili için aritmetik işlemler söz konusu olduğunda üç çeşit ifadeden söz edilir. Bunlar:

- **Tam sayı ifadeleri – integral expressions:** Bu ifade içerisinde yer alan tüm işlenenler tam sayılardır. İşlem sonuçları tam sayı olur.
- **Kayar noktalı sayı ifadeleri – floating point (decimal) expressions:** Bu ifade içerisinde yer alan tüm işlenenler kayar noktalı yani gerçek sayılardır. İşlem sonuçları da kayar noktalı olur.
- **Karma ifadeler – mixed expressions:** bu ifadelerde yer alan işlenenler hem tam sayı hem de gerçek sayılardır. İşlem sonuçlarının nasıl döndürüleceği ise programcı tarafından belirlenir.

Aşağıdaki örneklerde yukarı sözü edilen ifadelere ait örnek verilmiştir. Karma ifadeler ise sonraki bölümde ele alınmıştır.

```
int x, y;  
...  
3 + y +x  
3 + x - y / 8  
x + 2 + ( y - 2 ) / 10
```

Yukarıdaki ifadelerin tamamı tam sayılı ifadelerdir.

```
double x, y;  
...  
8 + y +x  
9 + x - y / 6.5  
x + 23.2 + ( y - 22.89 ) / 10
```

Yukarıdaki ifadelerin tamamı kayar noktalı sayı ifadeleridir.

### 1.4.2. Karma İfadeler

Karma ifadelerde hem tam sayılı hem de kayar noktalı aritmetik işlemler birlikte yer alır. Aşağıdaki örnekte karma ifadeler görülmektedir.

$$3.5 + 6$$

$$6 / 5 + 1.765$$

$$5.4 * 2 - 22.7 + 24.5 + 9 / 2$$

Yukarıdaki örneklerde ilk ifadenin iki adet işleneni vardır. Bir tanesi tam sayı ve diğer kayar noktalıdır. İkinci ifadede bölme işleminin iki adet tam sayılı işleneni bulur. İşlem sonucu elde edilen tam sayı ile toplama işleminde yer alan kayar noktalı sayı ile toplama işleminin iki adet işleneni olmaktadır. Aritmetik işlemlerde öncelikler söz konusu iken karma ifadelerde C++ Programlama Dili nasıl işlemektedir?

Karma ifadeler söz konusu olduğunda şu iki kural esas alınmaktadır:

- 1 Bir karma ifadedeki operatörler şu sırada işlenirler:

1.1 Eğer operatörün üzerinde işlem yapacağı işlenenlerin her ikisinde veri tipi aynı ise işlenenlerin çeşidine göre işlemler yapılır. Tam sayı veri tipi işlenenler tam sayı veri tipinde sonuç döndürür kayar noktalı veri tipindeki işlenenler üzerinde gerçekleştirilen sonuçlar kayar noktalı veri tipinde sonuç döndürür.

1.2 Eğer operatör her iki çeşit işlenen üzerinde işlem yapacak ise, işlem sırasında tam sayı veri tipindeki işlenen kayar noktalı veri tipine çevrilir. Ondalık basamak kısmı ise sıfır olur. Bu işlemin ardından operatör işlenir ve sonuç kayar noktalı olarak döndürülür.

- 2 Tüm işlemler, öncelik sırasına göre işlenir: Çarpma, bölme ve kalan önce işlenir, ardından toplama ve çıkarma işlemleri yapılır. Aynı öncelik değerine sahip olan operatörler soldan sağa doğru işleme alınırlar. Parantezlerin kullanılması ise işlemlerin daha kolay izlenebilmesi ve olası mantık hatalarının önüne geçilmek için kullanılabilir.

Aşağıdaki örneklerde karma ifadelerin nasıl işlendiği görülmektedir.

Karma İfade	İşlemler	Uygulanan Kural
$3 / 2 + 4.5$	$= 1 + 4.5 = 5.5$	İlk olarak bölme işlemi yapılır. Sonuç tam sayıdır. (1.1) Ardından tam sayı kayar noktalı sayıya dönüştürülür ve sonuç bulunur (1.2)
$12.6 / 3 + 6$	$= 4.2 + 6 = 10.2$	Öncelikle bölme işlemi yapılabilmesi için tam sayı kayar noktalı sayıya dönüştürülür. (1.2) Ardından da toplama işlemi yapılmadan önce tam sayı tekrar kayar noktalı sayıya dönüştürülür. (1.2)
$4 + 5 / 2.0$	$= 4 + 2.5 = 6.5$	Once bölme işlemi yapılacaktır. Tam sayı kayar noktalı sayıya dönüştürülür. (1.2) Ardından bulunan sonuç ile toplama işlemi yapılır. Her iki veri tipide aynı olduğu için dönüşüm gereklidir (1.1)
$3 * 4 + 7 / 5 - 22.5$	$= 12 + 7 / 5 - 22.5$ $= 12 + 1 - 22.5$	İlk olarak çarpma işlemi yapılır. (1.1) ikinci işlem basamağı olarak toplama işlemi yapılır (1.1)

$$\begin{aligned} &= 13 - 22.5 \\ &= -9.5 \end{aligned}$$

üçüncü işlem basamağı ise çıkarma işlemidir. Ancak öncelikle tam sayı veri tipi kayar noktalı veri tipine dönüştürülür (1.2) ve ardından işlem gerçekleşir.

Aşağıda verilen program yukarıda verilen ifadelerin sonuçlarını döndürmektedir.

```
/*
* karma_ifadeler.hxx
*
*
* Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
* takdirde serbesttir:
*
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
*
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
*
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİK VE ÖZEL BİR AMACA
* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
```

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <cfloat>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    cout << "3 / 2 + 4.5 = " << 3 / 2 + 4.5 << endl;
    cout << "12.6 / 3 + 6 = " << 12.6 / 3 + 6 << endl;
    cout << "4 + 5 / 2.0 = " << 4 + 5 / 2.0 << endl;
    cout << "3 * 4 + 7 / 5 - 22.5 = " << 3 * 4 + 7 / 5 - 22.54 << endl;
    return 0;
}
```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./airtmeitk_işlemler
3 / 2 + 4.5 = 5.5
12.6 / 3 + 6 = 10.2
4 + 5 / 2.0 = 6.5
3 * 4 + 7 / 5 - 22.5 = -9.5
[goksin@tardis ~/projeler/C++_Notlar]$
```

### 1.4.3.Veri Tipleri Arasındaki Dönüşümler

Karma ifadelerde hem tam sayılı hem de kayar noktalı aritmetik işlemler birlikte yer alırken aritmetik işlemlerin gerçekleşmesi için gerekli olan tam sayı, kayar noktalı sayı dönüşümlerinin

kural gereği yapıldığını önceki bölümde gördük. Veri tipi üzerinde yapılan bu dönüşümüne “**örtük tip dönüşüm – implicit type coercion**” bir diğer deyişle “**otomatik veri tipi dönüşümü**” adı verilir. Bu dönüşüm işleminde eğer programcı veri tipleri konusunda dikkatli değilse, programın işleyişinde önemli hatalar ortaya çıkacaktır.

Bu otomatik dönüşümün neden olabileceği hataların ortadan kaldırılması için C++ Programlama Dili bu işlemin tersi olan “**açık tip dönüşüm – explicit type coercion**” yani “**manuel veri tipi dönüşümü**” yapılmasını sağlayacak araca sahiptir. Bu araca “**tip operatörü – cast operator**” adı verilir. Aynı zamanda yabancı kaynaklarda “**type conversion**” veya “**type casting**” adı da verilir. Programlar içerisinde aşağıdaki gibi yazılarak veri tipleri arasındaki dönüşüm gerçekleştirilebilir.

`static_cast <veri tipi>(ifade)`

Bu program satırında önce ifade işlenir ve sonuç elde edilir. Ardından da veri tipi dönüşümü gerçekleştirilecektir. Örneğin işlem sonucunda elde edilen sonuç kayar noktalı veri tipinde ise, bunu tam sayı veri tipine dönüştürülmesi olanaklıdır. Bu dönüşüm gerçekleştirken ondalık basamaklar ihmal edilir ve tam sayı olan kısım döndürülür. Aşağıdaki örnekte bu dönüşüm işlemleri gösterilmiştir.

İfade	İşlem sonucu
<code>static_cast&lt;int&gt;(7.8)</code>	7
<code>static_cast &lt;int&gt;(3.7)</code>	3
<code>static_cast&lt;double&gt;(22)</code>	22.0
<code>static_cast&lt;double&gt;(3 + 4)</code>	<code>static_cast&lt;double&gt;(7) = 7.0</code>
<code>static_cast&lt;double&gt;(15) / 2</code>	<code>static_cast&lt;double&gt;(15) = 15.0 15.0 / 2 = 7.5</code>
<code>static_cast&lt;int&gt;(7.8) + static_cast&lt;double&gt;(15 / 2)</code>	<code>static_cast&lt;int&gt;(7.8) = 7 static_cast&lt;double&gt;(15/2) = 7.0 7 + 7.0 = 14.0</code>
<code>static_cast&lt;int&gt;(7.8 + static_cast&lt;double&gt;(15 / 2))</code>	<code>static_cast&lt;int&gt;(7.8 + 7.5) static_cast&lt;int&gt;(15.3) = 15</code>

Aşağıda verilen program yukarıda verilen ifadelerin sonuçlarını döndürmektedir.

```

/*
 * veri_tipi_donustur.cxx
 *
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>

```

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

#include <iostream>

#include <locale>

```

#include <cfloat>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    cout << "static_cast<int>(7.8) = " << static_cast<int>(7.8) << endl;
    cout << "static_cast<int>(3.7) = " << static_cast<int>(3.7) << endl;
    cout << "static_cast<double>(22) = " << static_cast<double>(22) << endl;
    cout << "static_cast<double>(3 + 4) = " << static_cast<double>(3 + 4) << endl;
    cout << "static_cast<double>(15) / 2 = " << static_cast<double>(15) / 2 << endl;
    cout << "static_cast<int>(7.8 + static_cast<double>(15) / 2) = " << static_cast<int>(7.8 +
static_cast<double>(15) / 2) << endl;
    cout << "static_cast<int>(7.8 + static_cast<double>(15 / 2)) = " << static_cast<int>(7.8 +
static_cast<double>(15 / 2)) << endl;
    return 0;
}

```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```

[goksin@tardis ~/projeler/C++_Notlar]$ ./veri_tipi_donustur
static_cast<int>(7.8) = 7
static_cast<int>(3.7) = 3
static_cast<double>(22) = 22
static_cast<double>(3 + 4) = 7
static_cast<double>(15) / 2 = 11
static_cast<int>(7.8 + static_cast<double>(15) / 2) = 15
static_cast<int>(7.8 + static_cast<double>(15 / 2)) = 14
[goksin@tardis ~/projeler/C++_Notlar]$

```

Yukarıdaki örnekler kısmında yazan sonuçlar ile programın çıktısı arasında farklı olduğu görülmektedir. Bu farklar programın hatalı işlemsinden olmayıp işlem sonucunun nasıl biçimlendiğinin belirtilmemiş olmasından kaynaklanmaktadır. Çıktının biçimlendirilmesi daha sonraki bölümlerde ele alınacaktır.

Yukarıda gösterilen veri tipi dönüşüm örneklerinde kullanılan static\_cast ayrılmış sözcüğü yerine C Programlama Dili’ndeki yapıldığı gibi (veritipi) dönüşüm yapılması da olanaklıdır. Bu dönüşüm şeklinde “**C Tipi Dönüşüm – C Type Casting**” adı verilir. Aşağıdaki örnek program

kodunda iki tam sayının ortalaması alınıp daha sonra C Tipi Dönüşüm uygulanarak kayar noktalı sayıya dönüştürülmektedir.

```
#include <iostream>
#include <cfloat>
using namespace std
int main()
{
    int n1 = 1, n2 = 2;
    cout<<"ortalama = "<< (double) (n1 + n2)/2 << endl;
    return 0
}
```

C++ Programla Dili’nde sadece tam ve kayar noktalı sayılar arasında veri tipi dönüşümü yapılmamaktadır. Aynı zamanda karakter veri tipi ile tam sayı veri tipine manuel olarak çevrilmesi olanaklıdır. Örneğin ASCII karakter setindeki ‘A’ tam sayı olarak dönüştürülmesi için **static\_cast<int>('A')** yazılır ve sonuç **65** olarak elde edilir. Benzer şekilde **static\_cast <int>('8')** işleminin sonucu da **56** olacaktır. Bu işlemler tersine olarak da yapılabilir. **static\_cast<char>(65)** işleminin sonucu ‘A’ olurken, **static\_cast <char>(56)** ise ‘8’ olur.

## 1.5.Karakter Dizisi Veri Tipi – String Type

Karakter dizisi aynı zamanda karakter katarı olarak adlandırılır. Adından anlaşılacağı üzere birden çok sayıdaki karakterin ardışık olarak yer alarak oluşturulmuş olan bir veri tipidir. Karakter dizisi veri tipi programcı tarafından kayar noktalı veri tipinde olduğu gibi tanımlanmak durumundadır. Tanımlama işlemi yapıldıktan sonra bu veri tipi program içerisinde kullanılabilir. Karakter dizisi veri tipi ISO C++ Standardı öncesinde var olmayan birbir veri tipidir. Standart öncesinde tüm derleyici ve geliştirme araçlarını üreten işletmeler karakter dizisi işlemleri için kendi geliştirdikleri kütüphaneleri ve araçları geliştirdikleri yazılım paketleri ile birlikte sunmuşlardır. Bu da farklı yazılım paketleri arasında birbiri ile uyuşmayan ve hatta çelişen uygulamalara neden olmuştur.

Karakter dizisi veri tipi bir C++ programında çift tırnak içerisinde belirtilir. Eğer çift tırnak arasında herhangi bir veri bulunmuyor ise buna “**boş karakter dizisi – null, empty**” adı verilir. Aşağıda çeşitli karakter dizisi örnekleri gösterilmiştir.

```
“Ali Veli”
“dondurma”
“”
```

Bir karakter dizisini oluşturan karakterlerin yeri, karakter dizisinin başlangıcına göre belirlenir. İlk karakterin konumu 0, ikinci karakterin konumu 1 ve üçüncü karakterin konumu da 2 olarak belirtilir. Yukarıdaki örnekleri buna göre inceleyeceğiz olursak aşağıdaki durumu gözlemleriz.

Karakter Dizisi	Karakterin Dizi İçerisindeki Konumu	Karakter dizisinin uzunluğu
“Ali Veli”	‘A’ konumu 0 ‘l’ konumu 1 ‘i’ konumu 2 ‘ ’ (boşluk) konumu 3 ‘V’ konumu 4 ‘e’ konumu 5 ‘l’ konumu 6 ‘i’ konumu 7	8
“dondurma”	‘d’ konumu 0 ‘o’ konumu 1 ‘n’ konumu 2 ‘d’ konumu 3 ‘u’ konumu 4 ‘r’ konumu 5 ‘m’ konumu 6 ‘a’ konumu 7	8

Örneğin aşağıda verilen tümcenin uzunluğu 23 karakterdir.

“Bu gün yağmur yağıyor.”

Karakter dizisi veri tipi diğer veri tiplerine göre daha karmaşık olmakla birlikte diğer veri tiplerinde olmayan özelliklere sahiptir. Bir diğer deyişle karakter veri tipi üzerinde diğerlerinden farklı olarak daha çeşitli işlemler yapılabilir. Örneğin bir karakter veri tipinin uzunluğunu bulmak, karakter veri tipindeki karakter dizisinin bir bölümünü çıkarmak, farklı karakter dizileri arasında karşılaştırmalar yapmak gibi.

## 1.6.Değişkenler, Atama ve Girdi İfadeleri

Daha önce belirttiğimiz gibi bilgisayar programlarının amacı veri üzerinde işlemler yapılarak sonuçlar elde edilmesidir. Ancak veriler üzerinde yapılan işlemler eğer kayıt altına alınmaz ise program sona erdiğinde kaybolur. Diğer istenmeyen durum da yapılan çeşitli işlemler sonucunda elde edilen verinin program içerisinde tekrar kullanılması gerektiğinde aynı işlemlerin tekrar yazılmasının gerekmektedir. Bu tekrarlayan yazımalar sırasında olası yazım hataları nedeni ile sonuçlar yanlış elde edilebilir veya program derlenmeyebilir. Ayrıca programın da gereken daha uzun sürede sonuçlanması da söz konusu olabilir. Bu olumsuzlukların ortadan kaldırılması için program içerisinde yapılan işlemlerin kayıt altına alınarak program içerisinde daha sonra kullanılması ve hatta sonuçların da farklı biçimlerde sunulabilmesi de olanaklıdır. Bu bölümde bilgisayar belleğine bir verinin nasıl aktarılacağını ve barındırıldığını öğreneceksiniz.

## 1.6.1. Bellekte Sabitler ve Değişkenler ile Alan Ayrılması

Bir programda tanımlanan bir değişken ve/veya sabit ile bilgisayar belleğinde belirtilen veri tipini barındıracak olan, büyülüğu belli olan ve bu alanlara erişimi kolaylaştırmak için değişkenler ve sabitlere isim verilir. Böylece önceden tanımlanan sabitler ve değişkenler tekrar program içerisinde aynı isim ile çağrılarak kullanılabilir.

### 1.6.1.1. Sabitlerin Isimlendirilmesi

Bazı problemlerin çözümünde bir verinin programın çalıştığı süre boyunca değeri değişmeden bellekte varlığını koruması gerekebilir. Örneğin ağırlık birimleri olan kg ile gr arasında dönüşümler için 1kg karşılığının 1000gr olması gibi. Bunun gibi verilerin bellek üzerinde barındırılması ve değerinin değişmemesi için korunarak barındırılması gereklidir. C++ Programlama Dili’nde bu tür değerinin değişmemesi ve korunması gereken veriler için “**isimlendirilmiş sabitler – named constants**” kullanılır. Bu bellek alanları programın çalışması süresinde barındırdıkları veriyi değiştirmeden barındırır ve bu alanlar üzerinde herhangi bir değişiklik yapılmaz.

Bellek üzerinde bir sabit için alan ayrılması için aşağıdaki gibi tanımlama yapılması gereklidir.

```
const veri_tipi tanımlayıcı = değer;
```

C++ programlama Dili’nde const ayrılmış sözcüktür. Programın başında sabitlerin tanımlanması ve değerlerinin atanması gereklidir. Çünkü derleyici sabit olarak tanımlanmış bir veri alanını programın ilk çalıştırıldığı anda oluşturulmasını sağlayacak makine dili komutlarını çalıştıracak şekilde çalışır. Eğer sabit tanımlanır ama değeri atanmaz ise bu bellek alanındaki sabitin değeri daha sonra program akışı içerisinde değiştirilemez.

Aşağıda verilen örneklerde farklı veri tipinde tanımlanmış olan çeşitli sabitler görülmektedir.

```
const double GRAD = 400;  
const int SINIFTAKI_SIRA_SAYISI = 22;  
const char SICAKLIK_BIRIM = 'C';
```

İlkörnekte derleyici 8 byte büyülüğünde ve adı GRAD olan bir bellek alanı ayırip bu alana 400 değeri atamaktadır. İkinciörnekte ise 4 byte büyülüğünde ve adı SINIFTAKI\_SIRA\_SAYISI olan bir bellek alanına 22 sayısını atamıştır. Üçüncüörnekte ise 1 byte büyülüğünde bir bellek alanına SICAKLIK\_BIRIM adı verilmiş ve bu alana ‘C’ atanmıştır.

Örneklerde dikkatini çekeni üzere sabitlerin tamamı **büyük harf** ile yazılmıştır. C++ programlama Dili’nde veya başkabir kaynakta sabitlerin isimlerinin nasıl yazılacağına ilişkin herhangi bir kural bulunmamakla birlikte programcılar arasında üzerinde anlaşılmış bir kural olarak **sabitlerin isimlerinin büyük harf ile yazılması** tercih edilir. Sabitlerin isimlendirmesinin dikkat çekeni bir diğer noktada tanımlamaların **birden çok sözcük olarak değil tek sözcük olarak** işlenmekte olduğunu söylemektedir.

Sabitlerin programcıya sundukları en önemli yarar ve kolaylık, program içerisinde değişikliklere gidilmesi durumunda yer bir satırda ilgili değerin değiştirilmesi yerine bunun programın sadece ilgili sabitin tanımlandığı satır üzerinde değiştirilmesi yeterli olacaktır. Ayrıca aynı değerin

program içerisinde çeşitli yerlerde tekrar yazılmasının önüne geçileceği gibi, oluşabilecek yazım hatası kaynaklı çeşitli hataların de önüne geçirilmiş olunacaktır.

### 1.6.1.2.Değişkenler

Bazı programlarda üzerinde işlem yapılan veri belirli koşullarda veya durumlarda değişkenlik gösterebilir. Bu değişimin dikkate alarak programın işlemleri yapması gerekektir. Örneğin bir çalışanın aldığı ücretin belirli bir yüzdesi olarak hesaplanan vergi miktarının hesaplanmasıında yapılan ücret zamına bağlı olarak ödenecek olan vergi miktarı değişecektir. Burada belirleyici olan ücretin miktarıdır. Aynı şekilde tüketicilerin ödemek zorunda oldukları katma değer vergisi de benzer şekilde ürün fiyatının bir yüzdesi olarak hesaplanır. Vergi oranının değişmesi durumunda da ödenecek olan vergi miktarı da değişir. Bunun gibi belirli bir parametrenin değişebileceği durumlarda gerçekleşecek olan işlemler için bellek üzerinde değeri değiştirilebilen bellek alanlarına gereksim duyulur. Bu bellek alanları sabitlerde olduğu gibi tanımlanır ancak değeri program akışı boyunca değişime uğrar. Değişkenler sabitlerin aksine olarak değeri değiştirebilen bellek alanlarıdır.

Bir değişken aşağıda gösterildiği gibi tanımlanır.

veri\_tipi tanımlayıcı, tanımlayıcı, ...

Aşağıda örnek değişken tanımlamaları görülmektedir.

```
int sayac, iterasyon;  
string harf_notu;  
char ad_ilkharf, soyad_ilkharf;
```

Yukarıdaki ilk örnekte sayac ve iterasyon adlı değişkenler tanımlanmıştır. BU değişkenler için veri tipi tam sayı veri tipidir. İkinci örnekte ise Harf\_Notu adlı bir değişken karakter dizisi olarak tanımlanmıştır. Karakter dizileri için bellekte sabit büyülükte alan ayrılmaz. Son örnekte ise karakter veri tipinde iki adet değişken tanımlaması yapılmıştır. Birinci değişken ad\_ilkharf ve ikinci değişken de soyad\_ilkharf olarak isimlendirilmiştir. Burada dikkat çeken nokta sabitlerin tamamı büyük harf olarak yazılrken değişkenler için ise küçük harflerin tercih edilmekte olduğunu söylemektedir. Burada tekrar belirtmek gerekmektedir ki isimlendirmede üzerinde uzlaşılmış olan bir kural olmadığıdır. Farklı programcılar değişken tanımlamaları konusunda çeşitli yazım şekillerini tercih etmekte olduklarıdır. Burada ders notları içerisinde değişkenler tamamı küçük harf ve sabitler de tamamı büyük harf olarak yazılmaktadır.

Buraya kadar çeşitli veri tipleri ile değişkenler ve sabitler açıklandı. Burada bu açıklamalara dayanarak bir tanımlama yapılması yerinde olacaktır. Eğer **bir değişken veya sabit sadece tek bir veri tipinde tek bir değer alabiliyorsa bunlara basit veri tipleri** adı verilir. Örneğin yukarıdaki değişken örnekleri basit veri tipleridir. Çünkü sadece tek bir veri tipinde tek bir değer barındırmaktadırlar.

## 1.6.2. Değişkenlere Değer Atanması

Bir programda sabitlerin tanımlanması durumunda değerleri de tanımlanmaktadır. Ancak değişkenler için durum farklıdır. Bir değişkenin barındıracağı verinin tanımlanması ise iki şekilde yapılabilir:

1. C++ Programlama Dili'nin atama ifadeleri kullanılarak
2. C++ Programlama Dili'nin girdi ifadeleri kullanılarak

## 1.6.3. Atama Bildirimleri

Atama bildirimleri adından anlaşılacağı üzere değişkenlere değer atamalarının yapılması için kullanılır. Aşağıda bir atama ifadesinin yapısı gösterilmiştir.

değişken = ifade;

Bir atama bildiriminde, sağ tarafta yer alan ifadenin veri tipi ile değişkenin veri tipi aynı olmalıdır. Bildirimde sağ tarafta yer alan ifadenin sonucu işlenip hesaplanır ve değeri de değişkene atanır. Atam işlemi sonucun bellekteki ilgili alana yazılmalıdır.

Bir değişkenin tanımlanmasının ardından barındıracağı değerin ilk defa atanması durumuna **“verinin etkinleştirilmesi – variable initialization”** adı verilir. C++ programlama Dili'nde “=” atama sembolüdür.

Aşağıdaki örneklerde atama bildirimleri gösterilmiştir.

```
int sayı1, sayı2;  
double ciro;  
char ilk_harf;  
string mesaj;
```

Yukarıdaki örneklerde sayı1 ve sayı2 adlı değişkenler tam sayı veri tipinde tanımlanmış ve bellekte 4 byte yer ayrılmıştır. İkinci örnekte ciro adlı değişken kayar noktalı veri tipinde ciro isimi ile tanımlanmış ve bellekte 8 byte yer ayrılmıştır. Karakter veri tipinde tanımlanan üçüncü örnek ise 1 byte büyülüklükte bellek alanına sahiptir, ve ilk\_harf adı verilmiştir. Son örnekte ise karakter dizisi olarak tanımlanmış olan değişken mesaj adı ile oluşturulmuştur.

Aşağıda değişkenlere yapılan atamalar görülmektedir.

```
sayı1 = 4;  
sayı2 = 5*4 - 11  
ciro = 135.65 * 14567;  
ilk_harf = 'A';  
mesaj = "Bu gün yağmur yağıyor";
```

Yukarıdaki atama ifadelerinin tümünde bilgisayar atama sembolünün sağ tarafındaki ifadeyi değerlendirdip, sonucunu sol tarafta belirtilen değişkene yani değişkenin adının belirttiği bellek ala-

nına yazacaktır. İlk değişken olan sayı1 4 değerini, ikinci değişken olan sayı2 ise 5 \*4 – 11 işleminin sonucunu, ciro adlı değişkende benzer şekilde 135.65 \* 14567 işleminin sonucunu ilgili alana yazacaktır. Karakter veri tipinde tanımlanmış olan ilk\_harf adlı değişkenin temsil ettiği bellek alanında ise ‘A’ yer alacaktır. Son olarak da karakter dizisi olarak tanımlanmış olan mesaj adlı bellek alanına “Bu gün yağmur yağıyor” adlı karakter dizisi yazılacaktır.

Bu örnekteki değişkenlerin bir programın içerisinde kullanılmış olduğunu düşünürsek, değişkenler henüz etkinleştirilmemiştir. Bunlara atama yapılması durumunda etkinleştirilmiş olacaklardır. Aşağıdaki programda bu durum gösterilmiştir.

```
/*
* degisken_atama.cxx
*
*
* Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
* takdirde serbesttir:
*
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
*
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
*
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
```

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <cfloat>
#include <locale>
#include <string>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    int sayi1, sayi2;
    double ciro;
    char ilk_harf;
    string mesaj;
    sayi1 = 4;
    cout << "sayi1 = " << sayi1 << endl;
    sayi2 = 5 * 4 - 11;
    cout << "sayi2 = " << sayi2 << endl;
    ciro = 135.65 * 14567;
    cout << "ciro = " << ciro << endl;
    ilk_harf = 'A';
    cout << "ilk_harf = " << ilk_harf << endl;
    mesaj = "Bu gün yağmur yağıyor";
```

```
cout << "mesaj = " << mesaj << endl;  
return 0;  
}
```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./degisken_atama  
sayi1 = 4  
sayi2 = 9  
ciro = 1.97601e+06  
ilk_harf = A  
mesaj = Bu gün yağmur yağıyor  
[goksin@tardis ~/projeler/C++_Notlar]$
```

Program incelemiştirinde ilk kısımlar kolaylıkla anlaşılmaktadır. Çıktının yazıldığı kısmında ise daha önce veri tipleri konusunda görülen ama üzerinde durulmamış olan bir yazım şekli tekrar karşımıza gelmektedir.

```
cout << "sayi1 = " << sayi1 << endl;
```

Yazdırma işlemi için “cout <<” olarak başlayan kısım, değişkenin adının yer aldığı ve çift tırnak içerisinde yazılan kısım ile devam etmektedir. “cout << "sayi1 = " <<” yazılan kısmından sonra gelen değişken adı ise yazdırma operatörüne değişkenin temsil ettiği bellek alanındaki veriyi döndürmektedir.

Yukarıdaki örneklerden ve açıklamalardan hareketle, bir değişkenin değerinin doğrudan tanımlanabileceği gibi aynı zamanda bir ifadenin sonucu olarak da belirlenebileceği görülmektedir. Aşağıda verilen atama ifadesinde say adlı değişkenin değeri ile 22 sayısı toplanıp sonuç yeniden say adlı değişkene atanmaktadır.

```
say = say + 22;
```

Yukarıdaki atama ifadesinde belirtilen say adlı değişkenin ilk değerinin 2 olduğunu kabul edeceğiz, ifadenin değerlendirilmesinin ardından değeri 24 olacak ve değişkenin yeni değeri 24 olarak atanacaktır. Eğer programın ilerleyen bölümlerde aşağıda görülen satırın bulunduğu kabul edeceğiz, bu satırda sonra değişkenin değeri ne olur?

```
say = 10;
```

Bu satırda sonra değişkenin değeri bir önceki işlemde gelen değeri olan 24 değil, artık 10 olacaktır. Eğer önceki satırda örnekte say adlı değişkene herhangi bir atama yapılmamış ise yani değişken etkinleştirilmemiş ise bu durumda bir önceki satırda atama ifadesi program derlenirken derleyici tarafından hata mesajı döndürülerek programcı uyarılacaktır. Bazı derleyicilerin bu tür hataları nasıl ele alacağı derleme seçeneklerinde tanımlanabildiğinden etkinleştirilmemiş olan değişkenlerin kullanılmasından kaynaklı hatalar sadece uyarı mesajı ile geçistirilebilir. Bu nedenle

etkinleştirilmemiş olan değişkenlerin olması durumunda derleyicinin programı derlenmesinin önüne geçilecek şekilde derleme seçeneklerinin tanımlanması gereklidir.

Ayrıca program kodundaki atama ifadelerinin programcılar tarafından okunmasında yukarıdaki satırı esas alırsak şu şekilde okunması doğru olacaktır:

- ✓ say değişkenin değeri 10 olmuştur.
- ✓ say değişeni 10'dur.
- ✓ say değişkenine 10 atanmıştır.

Bu ifade ediliş biçimleri yanında bazı programcılar aynı atama ifadesi aşağıdaki gibi hatalı okumaktadır:

- ✗ say değişkeni 10'a eşittir.

Bu okuma şekli tamamen yanlıştır. Çünkü C++ Programlama Dili'nde eşittir ifadesi “==” şeklinde yazılır. “=” ise atama sembolüdür, eşitlik değil.

Aşağıdaki örnekte say1, say2 ve say3 adlı değişkenlerin tam sayı veri tipinde tanımlanmış olduklarını kabul edelim

say1 = 20;	(1)
say1 = say1 + 27;	(2)
say2 = say1;	(3)
say3 = say2 / 5;	(4)
say3 = say3 / 3;	(5)

Yukarıdaki işlemlerin sonuçlarının izlenebilmesi için aşağıda görülen tablo hazırlanmıştır. Tabloda değişkenlerin ilk değerleri bilinmiyor ise ? Değişkenin ilk atanan değeri siyah ve kalın olarak, değişkenin değerinin değişmesi durumu da kalın ve kırmızı renkli olarak gösterilmiştir.

	Değişkenlerin/İfadelerin Değerleri	Açıklama
(1) öncesi	?	
(1) sonrası	<b>20</b>	
(2) sonrası	<b>47</b>	say1 = 20 + 27 = 47
(3) sonrası	<b>47</b>	say2 = say1
(4) sonrası	<b>47</b>	say3 = say2 / 5
(5) sonrası	<b>47</b>	say3 = say3 / 3

Yukarıda tablo olarak gösterilen süreç programcılıkta “**üzerinden geçme – walk through**” olarak adlandırılır. Özellikler programlardaki bazı hataların bulunması için işlem basamaklarının adım adım okunarak gerçekleşen işlemlerin yorumlanarak programdaki hatanın mantıksal mı yoksa başka nedenlerden mi olduğunu tespit edilmesi açısından oldukça yararlıdır.

#### 1.6.4.Bir İfadenin Sonucunun Saklanması ve Çağrılması

Buraya kadar değişkenlerin ve sabitlerin nasıl tanımlanacağını ve bunlara nasıl değer atanabileceğini gördük. Bu bölümde ise programcılıkta sık kullanılan ve bir ifadenin işlenmesi sonucunda elde edilen sonucun program içerisinde daha sonra kullanılmak üzere nasıl saklanıp ve program içerisinde nasıl tekrar çağrılacağını göreceğiz. Bu sözünü ettigimiz işlemlerin gerçekleştirilebilmesi için değişkenler kullanılır. Bir değişkenin bir ifadenin sonucunu saklaması için şu işlemler yapılmalıdır.

1. İfadenin döndüreceği sonucun veri tipi belirlenir. Bu sonucu saklamak için aynı veri tipinde bir değişken tanımlanır.
2. Atama ifadesi kullanılarak ifadenin sonucu, değişkene atanır. Böylece değişken işlem sonucunu barındırır hale gelir.
3. İfadenin sonuca gereksinim duyulan tüm program bölümlerinde ilk adında tanımlanmış olan değişken ilgili bildirim veya ifade içerisinde yazarak barındırdığı değer kullanılabilir.

Örneğin yazılan bir programın ikinci derecen bir denklemin köklerinin hesaplayan bir program yazacağımızı düşünelim. İkinci dereceden denklemin köklerinin bulunması için aşağıdaki formülü kullanılabaktır.

$$ax^2+bx+c \quad x \in \mathbb{R} \quad x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Denklemin çözümünde katsayıların tamamı tam sayı veri tipinde olacaktır. Benzer şekilde diskriminatın değeri de tam sayı veri tipinde olacaktır. Formülde diskriminant adlı kısım çözüm için birden çok defa kullanılmak durumundadır. Diskriminatın sonucunun bir değişkende saklanması ve varsa ayrık veya çakışık köklerin hesaplanması tekrar kullanılması gerekecektir. Bu nedenle programın değişkenleri aşağıdaki gibi tanımlanabilir.

```
int a, b, c, d, x1, x2;
```

Yukarıdaki bildirimlerde denkleme ait olan katsayılar, denklemin köklerini temsil eden x1 ve x2 ile diskriminanti temsil eden d değişkenleri tam sayı veri tipinde belirtilmiştir. Bu durumda diskriminantın hesaplanmasıının ardından köklerin bulunması için aşağıdaki işlemlerin yapılması gereklidir.

```
d = b *b - 4 * a *c;  
x1 = -b - d;  
x2 = -b + d;
```

Yukarıdaki ifadelerin program akışında bir hataya neden olmadan işlenebilmesi için söz konusu değişkenlere ait değerlerin ilgili işlem öncesinde tanımlanmış olması gereklidir. Değişkenlerin hem tanımlanması hem de etkinleştirilmesi aynı anda yapılabileceği gibi aynı zamanda da klavyeden veri girilerek de yapılabilir.

## 1.6.5.Değişkenlerin Tanımlanması ve Etkinleştirilmesi

Bir değişken tanımlandığında C++ Programlama Dili'ne bu değişkene eğer bir değer atanmamış ise bellekte ayrılan alana bir veri yazılmasız. Diğer bir deyişle C++ Programlama Dili bu değişkenleri etkinleştirmez. Bazı programlama dillerinde ise bunun tersi söz konusudur. Bir değişkenin tanımlanmış olması durumunda değeri örneğin tam sayı ve kayar noktalı veri tipleri için 0 olarak otomatik olarak atanır.

İşletim sistemleri bilgisayarın belleğini yönetmekle yükümlüdür. Dolayısı ile bir programın belleğe yüklenip çalıştırılması durumunda hem program hem de üzerinde işlem yapacağı veri ve bu veriler üzerinde yapılacak olan işlemlere ait olan değişkenler için bellekte alan ayrıılır. Ancak bellekte ayrılan alanının boş veya dolu olup olmadığı işletim sistemi tarafından bildirilmez. Dolayısı ile de bir bellek alınının bir önceki işleminden kalan verileri barındırmakta olması olasıdır. Eğer bu alan programdaki bir değişken tarafından kullanılmak üzere ayrılsa ve bu alanın ayrıldığı değişkenleri etkinleştirilmemiş ise program akışında önemli hatalar ortaya çıkacaktır. Örneğin ayrılan bellek alanında bir başka programa ait olan veriler aynen kalmış ve değişken etkinleştirilmeden doğrudan bildirim içerisinde kullanılmış ise bu durumda bellekteki var olan veri doğrudan okunup işlemlerde kullanılabilir.

C++ Programlama Dili'bu konuda programcının işini kolaylaştırmak üzere değişkenlerin hem tanımlanması hem de etkinleştirilmesini sağlayan özelliğe sahiptir. Bir değişkenin veri tipi belirtilip tanımlanması ile birlikte atama ifadesinin kullanılması durumunda yukarıda sözü edilen hatalar ortadan kaldırılmış olmaktadır. Aşağıdaki örneklerde değişkenler tanılanırken değerleri de atanmaktadır. Böylece değişken etkinleştirilmiş olmaktadır.

```
int ilk, ikinci;  
char karakter;  
double kar;  
ilk = 10, ikinci = 11;  
karakter = ' '  
kar = 230608.98;
```

Yukarıdaki yazım şekli yerine aşağıdaki gibi değişkenler tanımlanıp, değer ataması yapılp etkinleştirilebilir.

```
int ilk = 10, ikinci = 11;  
char karakter = ' ';  
double kar = 230608.98;
```

Bir değişkenin programın başlangıcında etkinleştirilip etkinleştirilmeyeceği tamamen programcının tercihine kalmış bir konudur. Bazı programcılar tüm değişkenleri programın başlangıç kısmında tanımlayıp etkinleştirmeyi tercih ederken bazıları bunu programda gerektiğinde etkinleştirilecek şekilde yazmaktadır. Bir program hangi şekilde yazılırsa yazılışın, bir değişken etkinleştirilmeden bir değere sahip olamayacağı kuralı her koşulda geçerlidir.

## 1.6.6. Girdi (Okuma) Bildirimleri

Bir programın yazılması sırasında bir değişken atama ifadeleri kullanılarak bir değer barındırmaktadır. Diğer bir deyişle değişken etkinleştirilmiş olduğunda bir değer barındırmaktadır. Ancak bu aşamaya kadar programların içerisinde belirtilen değerler değişkenlere atanmıştır. Eğer bir programın sadece tek bir değer ile çalıştırılması istenmiyorsa ve programın belirli bir problem grubuna özgü farklı değerler ile işlem yapması bekleniyorsa bu durumda C++ Programlama Dili'nin “**girdi / okuma bildirimi – input/ read statement**” kullanılabilir.

Bilgisayar bilimlerinde kullanıcı klavyeyi kullanarak veri girişi yapıyorsa bu “**etkileşimli – interactive**” kullanım olarak isimlendirilir.

C++ Programlama Dili’nde klavyeden girilen verinin bir program içerisinde okunması ve verinin değişkene aktarılması “**cin**” ve “**>>**” operatörünün birlikte kullanılması ile söz konusudur. Aşağıda görüldüğü gibi bir programın içerisinde girdi bildirimi yazılabilir.

```
cin >> değişken >> değişken >> ... ;
```

Yukarıdaki yazım şekli “**girdi bildirimi – input statement**” olarak tanımlanır. Ayrıca “**>>**” operatörü “**akış ayırtıcısı – stream extractor**” olarak tanımlanır.<sup>8</sup>

Aşağıdaki örnekte km adlı değişken double veri tipinde tanımlanmış ve değeri klavyeden kullanıcı tarafından girileceğini kabul edelim. Değişkenin değeri de 1259.098 olsun.

```
cin >> km;
```

Program bu bildirimin yer aldığı satıra geldiğinde klavyeden verinin girilmesini bekleyecektir. Veri girildikten sonra enter tuşuna basıldığında değişken 1259.098 değerini barındıracaktır.

Her bir değişken için yukarıda görüldüğü gibi aynı bildirimin tekrarlanarak yazılmasına gerek yoktur. C++ Programlama Dili’nde akım operatörleri bir satırдан girilen ve aralarında boşluk bırakılan girdileri işleyerek her birisini sırası ile bir değişkene atayabilir.

Yukarıdaki örnekten devam edecek olursak, tam sayı veri tipinde tanımlanmış olan km ve hm gibi değişkenlere atanacak olan verilerin tek bir satırda girilerek atanması aşağıdaki gibi yapılabilir.

```
int km, hm;
```

```
cin >> km >> hm;
```

Bu bildirime klavyeden yapılan girdi ise sırası ile 18 ve 8 ise girdilerin atanması sonucunda değişkenlerin değerleri sırası ile 18 ve 8 olacaktır. Akım operatörleri için girdiler arasında tek bir boşluk olması zorunlu değildir. Birden çok sayıda boşluk olabileceği gibi aynı zamanda sekme de kullanılabilir.

Aşağıdaki örnek programda klavyeden girilen km ve hm değerleri m çevrilerek sonuç ekrana yazılmaktadır.

```
/*
```

<sup>8</sup> Bilgisayar teknolojisinin ilk gelişmekte olduğu günlerde verinin bellekte bir ve sıfır oluşan bir veri topluluğu olarak tutmakta olduğu ve işlemcinin de bu veri topluluğunu, bir veri akımı şeklinde alıp istediği şeklindeki tanımlama nedeni ile akım operatörü olarak isimlendirilir.

```
* km_hm_m.cxx  
*  
*  
* Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>  
* *  
* Her hakkı saklıdır.  
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak  
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği  
* takdirde serbesttir:  
*  
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,  
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.  
*  
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,  
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen  
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.  
*  
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA  
* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
* SORUMLU DEĞİLLERDİR.
```

```

/*
 */

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

int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");

    int km, hm, m;

    cout << "Uzunluğu tam sayı olarak, aralarında boşluk bırakarak 'km hm' olarak giriniz." <<
endl;

    cin >> km >> hm;

    cout << endl;

    m = km *1000 + hm * 100;

    cout << "Toplam uzunluk = " << m << " m'dir." << endl;

    return 0;
}

```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./km_nm_m
Uzunluğu tam sayı olarak, aralarında boşluk bırakarak 'km hm' olarak giriniz.
23 9
```

Toplam uzunluk = 23900 m'dir.

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

Aşağıdaki program örneği ise tam sayı, kayar noktalı ve karakter dizisi veri tiplerinin birlikte kullanılmasını göstermektedir.

```
/*
 * kisisel_bilgiler.cxx
 *
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
```

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

#include <iostream>

#include <locale>

```

#include <string>
#include <cfloat>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    string ad, soyad;
    int yas;
    double kilo;
    cout << "Adınızı, soyadınızı, yaşıınızı ve kilonu aralarında boşluk bırakarak. giriniz." << endl;
    cout << "Adınız ve soyadınız sadece harflerden oluşabilir. Yaşınız tam sayı olarak girilmelidir. Kilonuzu ondalık sayı olarak girebilirsiniz." << endl;
    cin >> ad >> soyad >> yas >> kilo;
    cout << endl;
    cout << "Adınız: " << ad << endl;
    cout << "Soyadınız: " << soyad << endl;
    cout << "Yaşınız: " << yas << endl;
    cout << "Kilonuz: " << kilo << endl;
    return 0;
}

```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./kisisel_bilgiler
```

Adınızı, soyadınızı, yaşıınızı ve kilonu aralarında boşluk bırakarak. Giriniz.

Adınız ve soyadınız sadece harflerden oluşabilir. Yaşınız tam sayı olarak girilmelidir. Kilonuzu ondalık sayı olarak girebilirsiniz.

Baturalp Dinçdarı 33 65

Adınız: Baturalp

Soyadınız: Dinçdarı

Yaşınız: 33

### 1.6.7. Değişkenlerin Etkinleştirilmesi

Değişkenlerin etkinleştirilmesi iki şekilde olabilir. Birincisi atama ifadesi kullanılması ile olurken diğer de girdi bildirimleri aracılığı yapılabılır. Aşağıda bir program kodu içerisinde yapılan iki değişken tanımlaması görülmektedir.

```
int kg, gr;
```

Bu değişkenlerin aşağıda gösterilen program kodları içerisinde iki farklı şekilde kullanımı görülmektedir. İlk kullanım şekli aşağıdaki gibi olsun.

```
kg = 345;
```

```
gr = 890;
```

```
cout << "Toplam ağırlığın gram olarak karşılığı: " << kg * 1000 + gr;
```

Aynı değişkenlerin kullanıldığı ikinci program şu şekildedir:

```
cout << "kg cinsinden ağırlık: "
```

```
cin >> kg;
```

```
cout << "gr cinsinden ağırlık: "
```

```
cin >> gr;
```

```
cout << endl;
```

```
cout << "Toplam ağırlığın gram olarak karşılığı: " << kg * 1000 + gr;
```

Birinci örnek kodda değişkenler programın başında değerleri tanımlanarak etkinleştirilmiştir. İkinciörnekte ise program değişkenlerin alacağı değerleri kullanıcıdan girmesini istemektedir. Kullanıcı giriş yapıldığında okunana değer değişkene atanıp, değişken etkinleştirilmektedir. Bu iki program da aynı işlev sahip olmakla birlikte ikisi arasında tasarım olarak farklılık bulunmaktadır. Eğer bir programda kullanılacak olan değişkenin değeri, sadece program içerisinde yapılacak bazı hesaplamalarda kullanılacak ise ilkörnekte olduğu değişkenin değeri programın başlangıç bölümünde tanımlanarak etkinleştirilmesi uygun olur. Ancak programda tanımlanan değişkenin değeri kullanıcı tarafından girilmesi beklenen ve programın asıl işlevinin bu verilerin işlenmesi olduğu durumlarda ise değişkenlerin okunarak etkinleştirilmesi yolu tercih edilmelidir.

Unutulmaması gereken C++ Programlama Dili'nin değişkenleri otomatik olarak etkinles- tirmemekte olduğudur. Bu nedenle değişkenlerin etkinleştirilmesi gereken durumlarda bu işlemin nasıl yapılacağına programcının karar ermesi ve programı buna göre kurgulayarak yazması gereklidir. Aşağıdaki program değişkenlerin atama ve okuma işlemleri nasıl etkinleştirildiğini ve program içerisinde nasıl kullanıldığını göstermektedir.

```
/*
```

```
* degisken_etkinlestir.cxx
```

\*

\*

\* Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımıyla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

```

*/
#include <iostream>
#include <locale>
#include <cfloat>
#include <string>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    int sayac, gecici;
    double uzunluk, genislik, alan;
    char karakter;
    string isim;
    // İşlemler bölümü
    sayac = 1;                                // (1)
    sayac = sayac + 1;                          // (2)
    cout << "Uzunluk ve genişliği aralarında boşluk bırakarak giriniz: ";
    cin >> uzunluk >> genislik;                // (3)
    cout << endl;
    alan = uzunluk * genislik;                  // (4)
    cout << "Bir isim yazınız: ";
    cin >> isim;                               // (5)
    cout << endl;
    uzunluk = uzunluk + 2;                      // (6)
    genislik = uzunluk * 2 - 5 * genislik;      // (7)
    alan = uzunluk * genislik;                  // (8)
    cout << "Bir karakter yazınız: ";
    cin >> karakter;                           // (9)
    cout << endl;
    gecici = sayac + static_cast<int>(karakter); // (10)
    return 0;
}

```

}

Yukarıdaki programı derleyip çalıştırıldığınızda sadece kullanıcının veri giriş yapmasını isteyecektir. Program herhangi bir çıktı üretmeyecektir. Program çalışması sırasında değişkenlerin veri tipleri tanımlanmış ve bazlarının değerleri önceden atanmıştır. Programa girilen verilerin sırası ile 23 5 Ahmet A olduğunu kabul edelim. Program çalıştırıldığında değişkenlerin durumu her bir işlem basamağında aşağıdaki gibi olacaktır. Eğer değişken etkinleştirilmemiş ise ilgili alan “?” ile, değişkenin değeri atanmamış veya değişmiş ise “**kırmızı**” Değişkenin değeri değişimemiş ise “**siyah**” olarak gösterilmiştir

**Değişkenler**

	<b>sayac</b>	<b>gecici</b>	<b>uzunluk</b>	<b>genişlik</b>	<b>alan</b>	<b>karakter</b>	<b>isim</b>
(1)	<b>1</b>	?	?	?	?	?	?
(2)	<b>2</b>	?	?	?	?	?	?
(3)	<b>2</b>	?	<b>23</b>	<b>5</b>	?	?	?
(4)	2	?	<b>23</b>	5	<b>115</b>	?	?
(5)	2	?	<b>23</b>	5	<b>115</b>	?	<b>Ahmet</b>
(6)	2	?	<b>25</b>	5	<b>115</b>	?	<b>Ahmet</b>
(7)	2	?	<b>25</b>	<b>25</b>	<b>115</b>	?	<b>Ahmet</b>
(8)	2	?	<b>25</b>	<b>25</b>	<b>625</b>	?	<b>Ahmet</b>
(9)	2	?	<b>25</b>	<b>25</b>	<b>625</b>	<b>A</b>	<b>Ahmet</b>
(10)	2	<b>67</b>	25	25	625	<b>A</b>	<b>Ahmet</b>

## 1.7.Arıtırma ve Azaltma Operatörleri

Önceki bölümlerde bir değişkenin tanımlanması ve değişkene değer atanmasını incelemiştik. Bu bölümde ise artırma ve azaltma operatörlerini inceleyeceğiz. Artırma ve azaltma operatörleri bir çok programlama dilinde bulunur ve sıkılıkla programcılar tarafından kullanılır.

Bir programın içerisinde “sayac” adlı değişkenin değerinin artırılması gerekiği bir durum olduğunu düşünelim. Bu işlem için aşağıdakine benzer bir bildirimin yazılması gereklidir.

sayac = sayac + 1;

Bu satırın işlemi için öncelikle sağ taraftaki ifade işlenecektir. Değişkenin değeri bir artırılacak ve elde edilen değer sol tarafta yer alan sayac adlı değişkene atanacaktır. Bu kullanım şekli bir çok programda bir işlemin gerçekleşme sayısının izlenmesi ve buna göre programın akışının değerlendirilmesi için kullanılır. Bu işlemler için C++ Programlama Dili özel operatörler sahiptir. Bunlara **arttırırum** **++** ve **azaltırırı** **--** operatörleridir. Sırası değişkenin değeri bir artırılır veya bir azaltılır. Bu iki operatörün programlar içerisinde kullanımı aşağıda görüldüğü gibidir.

Artırma operatörü iki şekilde kullanılabilir.

++değişken;      \ Değişken bir artar. Pre-increment

`değişken++;`      \ Değişken bir artar. Post-increment

Azaltma operatörü iki şekilde kullanılabilir

`--değişken;`      \ Değişken bir azaltılır. Pre-decrement

`değişken--;`      \ Değişken bir azaltılır. Post-decrement

Arttırma ve azaltma operatörleri için atama ifadesi yazılmasına gerek yoktur. Yukarı her iki kullanım şeklinde artırma ve azaltma işlemleri yapıldığına göre neden iki farklı kullanım şekli bulunmaktadır? Bu sorunun yanıtı program içerisinde nasıl kullanıldığı gözlemlenince anlaşılır.

Örneğin iki değişkenin olduğu bir program içerisinde yer alan aşağıdaki iki bildirimi inceleyelim. Öncelikle pre-increment işlemini inceleyelim.

`a = 5;`      \ a değişkenin değeri 5.

`b = ++a`      \ a değişkenin değeri 6 oldu. b değişkenine a'nın değeri atandı ve 6 oldu.

Yukarıdaki kodu eğer uzun yoldan yazılırsa aşağıdaki gibi olacaktır

`a = 5;`      \ a değişkenin değeri 5.

`a = a + 1`      \ a değişkenin değeri 1 arttı ve 6 oldu

`b =`      \ a değişkenin değeri b değişkenine atandı ve b değişkeni 6 oldu.

Şimdi de post-increment işlemini inceleyelim.

`a = 5;`      \ a değişkenin değeri 5.

`b = a++`      \ a değişkenin değeri b değişkenine atandı. B değişkenin değeri 5 oldu.

                        \ a'nın değeri 1 arttırıldı ve 6 oldu.

Yukarıdaki kodu eğer uzun yoldan yazılırsa aşağıdaki gibi olacaktır

`a = 5;`      \ a değişkenin değeri 5.

`b = a`      \ a değişkenin değeri b değişkenine atandı ve b değişkeni 5 oldu.

`a = a+1`      \ a değişkenin değeri 1 arttırıldı ve 6 oldu.

Arttırma operatörünün pre-increment veya post-increment olarak artırılması arasındaki fark artırma eyleminin işlem öncesi veya sonrasında gerçekleşmesidir.

## 1.8. Çıktı İşlemleri

Önceki bölümlerde yer alan programlarda komut satırına yani standart çıktıya hem bazı mesajların hemde işlem sonuçlarının yazılmasını görmüştük. Bu kısımda ise standart çıktıya yazdırma işlemlerinin nasıl gerçekleştiğini inceleyeceğiz.

C++ Programlama Dili’nde standart çıktıya – yani ekrandaki komut satırına – yazdırma işlemi “cout” ve “<<” operatörü ile yapılır. İkisinin birlikte kullanım şekli ise aşağıdaki gibidir:

`cout << ifade veya biçimlendirici << ifade veya biçimlendirici << ... ;`

Yukarıda görülen ifade “**çıktı ifadesi – output statement**” olarak tanımlanır. C++ Programlama Dili’nde ise “**<<** operatörü “**çıktı akışına ekleme – stream insertion operator**” olarak isimlendirilir. Çıktı işlemleri ile ilgili şu iki kural uygulanır:

1. İfade işlenir, sonucu da komut satırında imlecin bulunduğu yere yazılır.
2. Biçimlendirici, yazdırma işleminin biçimsel olarak düzenlenmesini sağlar. En basit biçimlendirici bir sonraki satırın başına geçilmesini sağlayan endl’dir.

Aşağıdaki örnek programda iki değişken tanımlanmış bve bu değişkenler ile birlikte bazı işlemler ve mesajlar yazdırılmıştır.

```
/*
 * yazdirma.cxx
 *
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
```

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.  
\*  
\*/

```
#include <iostream>
#include <locale>
#include <cfloat>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    int a = 78, b= 59;
    cout << 37 / 8 << endl;
    cout << 3.5 / 2 << endl;
    cout <<"Bu metin alt satıra geçer\n ve bu satırda sona erer."<< endl;
    cout << "3 + 5 = ";
    cout << 3 + 5;
    cout << " * * * * * " << endl;
    cout << a << endl;
    cout << b << endl;
    return 0;
}
```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./yazdirma
```

1.75

Bu metin alt satıra geçer

ve bu satırda sona erer.

3 + 5 = 8 \* \* \* \* \*

78

59

[goksin@tardis ~/projeler/C++\_Notlar]\$

Programın ürettiği ilk iki satırdaki çıktıda önceki programlardakine benzer bir çıktı üreten iki bildirim bulunmaktadır. İleyen bildirimde ise bir metin yazdırılmaktadır. Metin içerisinde ise “\n” yer almaktadır. Burada görülen ters bölümü işareteti “**kaçış karakteri – escape character**” oalrak kendiden sonra gelen karakter ile birlikte tek bir karakter olarak işleneceğini belirtir. Çıktan da görüleceği üzere “\n” bir sonraki satırın başına geçilerek metinin oradan devam etmesini sağlamaktadır. Bunların ardından gelen satırda bir toplama işlemi yazılmış ve sonucu da bir sonraki satırda hesaplanmasına rağmen aynı satırın sonunda yazılmış ve yine sonraki bildirimde yer alana yıldız karakteri yazılmıştır.

İki satıra yayılan mesaj aslında tek bir çıktı bildiriminden oluşmaktadır.

```
cout <<"Bu metin alt satıra geçer\n ve bu satırda sona erer."<< endl;
```

Bu bildirim iki ayrı bildirim olarak da yazılabildirdi.

```
cout <<"Bu metin alt satıra geçer" << endl;
```

```
cout <<" ve bu satırda sona erer." << endl;
```

İki ayrı bildirim olarak bir metinin yazdırılması ilk bakışta daha kolay gibi görünebilir. Ancak bazı durumlarda aynı kod üzerinde çalışan birden çok programcının olduğu projelerde birbiri ile ilişkili olan ve birden çok çıktı bildirimleri ile yazılan metinler arasında yeni kodların yazılması ile hatalı çıktılar üretilebilmektedir. Bu durum kullanıcıların mesajı yanlış yorumlamasına neden olabileceği de dikkate alınarak birden çok satıra yayılmış olarak yazılabilecek olan mesajların tek bir satır olarak yapılması tercih edilir.

Mesajın okunabilirliğini artırmak için ise mesaj yukarıdaki örnekte olduğu gibi uygun yerlerden “\n – yeni satır – new line” ile diğer satırlara ayırlabilir. Aşağıda bir programın içerisinde alınan bir açıklama metinin programcı tarafından birden çok satıra yayılarak yazılmış halini görüyoruz:

```
cout <<"Bu program kullanıcı tarafından belirtilen aralıktaki\n pozitif tam sayılar arasında bulunan asal sayıları bulmaktadır.\nAralık 2 ile " << ULLONG_MAX << " arasında olmalıdır.\n"
```

Yukarıdaki satırda programcı metinin okunabilirliğini artırmak için mesajın uygun yerlerinden enter tuşuna basarak bir sonraki satıra geçerek devam etmek isteyebilir. Ancak C++ Programlama Dili için bir karakter katarında enter kullanarak sonraki satıra geçerek karakter katarina devam etmek söz konusu değildir. Bu şekilde yazılan metinler derleyici hata mesajı döndürür.

Eğer yukarıdaki mesajı yeni satır karakteri kullanmadan yazılması tercih edilirse bu durumda C++ Programlama Dili’nde bildirimin sona erdiğini belirten “;” kullanılmadan aşağıdaki gibi yazılabılır.

```
cout <<"Bu program kullanıcı tarafından belirtlen aralıktaki"  
<< "pozitif tam sayılar arasında bulunan asal sayıları bulmaktadır."  
<<"Aralık 2 ile " << ULONG_MAX << " arasında olmalıdır.\n";
```

Satırın sona erdiğini belirten “;” kullanılmaz ise derleyici yazılan program kodundaki izleyen satırın önceki satırın devamı olarak işleyecektir. Önceki örnekte bu durum gösterilmiştir.

Cıktıyı biçimlendirmek için kullanılabilen diğer “**kaçış dizileri – escape sequence**” şunlardır:

Kaçış Dizisi	Tanımı	
\n	Yeni satır	İmleç bir sonraki satırın başına geçer.
\t	Sekme	İmleç bir sekme uzunluğu kadar sağa ilerler.
\b	Geri git	İmleç sola doğru bir karakter geri gider.
\r	Başa git	İmleç bulunduğu satırın sol başına gider
\\\	Ters bölü	Ters bölü işaretini yazar.
\'	Tek tırnak	Tek tırnak yazar.
\”	Çift tırnak	Çift tırnak yazar.

Aşağıdaki örnek programda kaçış dizilerinin kullanımı gösterilmektedir.

```
/*  
* kacis_dizileri.cxx  
*  
*  
* Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>  
* *  
* Her hakkı saklıdır.  
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak  
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği  
* takdirde serbesttir:  
*  
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,  
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
```

\*

- \* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
- \* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
- \* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

- \* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
- \* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
- \* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
- \* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
- \* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
- \* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
- \* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
- \* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
- \* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
- \* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
- \* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
- \* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
- \* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
- \* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
using namespace std;
int main()
{
    cout << "Bir sonraki satırda geçmek için \\n kullanılır." << endl;
    cout << "Bu metin birinci satırdadır.\nBu metin ikinci satırdadır."<< endl;
    cout << endl;
    cout << "Bir sekme genişliği boşluk bırakmak için \\t kullanılır." << endl;
    cout << "\\tBu satır bir sekme kadar boşluk bırakılarak yazılmıştır." << endl;
```

```
cout << endl;  
cout << "Satırın başına dönmek için \\r kullanılır." << endl;  
cout << "İmleç satırın başına döndüğünde bu metinin üzerine yazılır.\r";  
cout << "Önceki satırın üzerine yazıldı!" << endl;  
cout << endl;  
cout << "Ters bölümü işaret etmek için \" << '\\' << " kullanılır." << endl;  
cout << "\\\" << endl;  
cout << endl;  
cout << "Karakter veri tipi girdi bildirimlerinde \"\"\" << " " << \"\"\" << \" kullanılır." << endl;  
cout << "Örnek: \'A\\\'" << endl;  
cout << endl;  
cout << "Karakter dizileri \"\"\" << " " << \"\"\" << \" arasında yazılır." << endl;  
cout << "Örneğin:" << "\"Karakter dizisi\\\" " << endl;  
return 0;  
}
```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./yazdirma
```

Bir sonraki satırda geçmek için \\n kullanılır.

Bu metin birinci satırdadır.

Bu metin ikinci satırdadır.

Bir sekme genişliği boşluk bırakmak için \\t kullanılır.

Bu satır bir sekme kadar boşluk bırakılarak yazılmıştır.

Satırın başına dönmek için \\r kullanılır.

İmleç satırın başına döndüğünde bu metinin üzerine yazılır. Önceki satırın üzerine yazıldı!

Ters bölümü işaret etmek için \\ kullanılır.

\\

Karakter veri tipi girdi bildirimlerinde ' ' kullanılır.

Örnek: 'A'

Karakter dizileri " " arasında yazılır.

Örneğin: "Karakter dizisi"

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

C++ Programlama Dili'ndeki girdi ve çıktı işlemler için özel bir “**başlık dosyası – header file**” dosyası kullanılır. Sonraki bölümde başlık dosyalarının ne olduğu, nasıl eklendiği ve neden gereksinim duyulduğu açıklanmaktadır.

## 1.9.Önişlemci Yönergeleri

C++ Programlama Dili içerisinde aritmetik işlemler ve atama operatörler gibi bazı işlemler hazır olarak gelir. Programlar yazılırken kullanılacak olan çok sayıdaki işlev ve semboller kütüphaneler sunulur ve bunlar çağrılarak kullanılır. Her bir kütüphane kendisini tanımlayan bir başlık dosyası ile çağrılır. Örneğin **girdi/çıktı işlemleri** için kullanılacak olan başlık dosyası **iostream** adını taşımaktadır. Benzer şekilde sıkılıkla kullanılan matematik fonksiyonlar olan mutlak değer, karekök üs vb işlemler için ise **cmath** başlık dosyasının kullanılması gereklidir. Bu kütüphanelerin disk üzerinde bulundukları yerlerinin belirtilmesi programın çalıştırılabilir dosyalarının üretilebilmesi için gereklidir. Bu işlemin programcı tarafından yapılmasına gerek yoktur. Bu işlem “**önişlemci yönergeleri – preprocessor directives**” tarafından yapılır. Önişlemci yönergeleri önişlemci adlı bir program tarafından işlenerek yazılan program kaynak kodunu barındıran metin dosyası üzerinde düzenleme yapar ve gerekli kütüphaneleri eklenmesinin ardından derleme işlemi gerçekleştir. Tüm önişlemci yönergeleri “#” işaretleri ile başlar ve sonunda “;” yer almaz. Çünkü önişlemci yönergeleri bir C++ bildirimi değildir. Bir ön işlemci yönergesinin programda kullanılabilmesi için aşağıdaki gibi programın kaynak kodunun lisans metninin ardından yazılması gereklidir.

```
#include <önişlemci_yönergesinin_adı>
```

Yukarıdaki örneklerde sıkılıkla iki ön işlemci yönergesinin kullanılmakta olduğunu gördünüz. Birincisi girdi ve çıktı işlemleri için kullanılan **<iostream>** ve Türkçe karakterlerin kullanılmasını sağlayan **<locale>** idi . Ayrıca veri tiplerinin alt ve üst sınırlarını belirten **<climits>** ve karakter dizileri üzerindeki çeşitli işlemlerin yapılmasını sağlayan **<string>** ile kayar noktalı sayılar ile işlem yapılmasını sağlayan **<cfloat>** da kullanıldı.

Bir programın derlenmeden önce gerekli olan önişlemci komutlarının kaynak kod içerisinde yer verilmesi derleme süresinin kısalmasını sağlayacağı gibi programın da doğru bir şekilde çalışmasını sağlayacaktır.

### 1.9.1.Programlarda namespace, cin ve cout Kullanımı

ANSI/ISO Standart C++ göre cin ve cout dilin yapısında bulunur ve hazır olarak sunulmaktadır. Bu operatörler iostream başlık dosyasında hazır olarak bulunmakla birlikte kullanılabilmeleri için “**namespace – isim uzayı**” tanımının yapılması gereklidir. Bu tanımlama başlık dosyalarının

ardından yapılır. Program içerisinde bir başlık dosyası içerisinde yer alan herhangi bir işlevin veya operatörün kullanılabilmesi için isim uzayının tanımlanması gereklidir.

İsim uzayı farklı şekillerde tanımlanabilir. Bir bildirim içerisinde örneğin cin veya cout kullanılacak ise aşağıdaki gibi yapılabilir:

```
#include <iostream>

std::cout << "...";

...
std::cin >> ...;
```

Bu tanımlama şeklinde her bildirim içerisinde standart isim uzayının tanımlanması için “**std:: ...**” şeklinde yazılması gereklidir. Bunu tekrar tekrar yazmak yerine önceki program örnekle-rinde olduğu gibi yazılarak yapılması daha uygundur.

```
#include <iostream>

using namespace std;

...
```

ANSI/ISO Standart C++ göre her başlık dosyasının tanımlandığı izim uzayı standart isim uzayıdır. Dolayısı ile bu ders notları içerisinde kullanılan başlık dosyaları her zaman için standart isim uzayını kullanmakta olduğu unutulmamalıdır. Sonraki bölümlerde isim uzayı konusu ayrıntılı olarak incelenecaktır.

### 1.9.2. Programlarda Karakter Katarı Veri Tipinin Kullanımı

Karakter katarı veri tipi bir C++ Programlama Dili ile yazılan bir programda kullanılacak ise gerekli olan önişlemci yönergusonin belirtilmesi gereklidir. Çünkü karakter katarı veri tipi temel bileşen olarak yer almaz. Karakter katarları üzerinde gerçekleştirilecek çok sayıda farklı işlem olduğundan ve bu işlemlerin dilin gelişimi sırasında değişimler geçirdiği düşünülürse bunun temel bir veri tipi olarak yer almaması kolaylıkla anlaşılabılır. Karakter dizileri üzerinde yapılan işlemlerin program içerisinde kullanılabilmesi için **<string>** ön işlemci yönergesi kullanılır:

## 1.10. C++ Programlama Dili Program Yazım Kuralları

C++ programlarında her bir satır bir **bildirim**<sup>9</sup> olarak tanımlanır. Bildirim içerisinde çeşitli işlemleri belirten **ifadeleri** barındırmaktadır. İfade ise **sözel** veya **sayısal** veri üzerinde gerçekleştirilecek olan eylemleri tanımlar.

Bir C++ programında kullanılan ön işlemci komutları aslında bir dizi işlemin yapılmasını sağlayan ve programın çalıştırılması sırasında istenen işlevlerin yerine getirilmesini sağlayan **fonksiyonlardır**. C++ programlama dilinde programcılar gerek duyduklarında kendi fonksiyonlarını yazabilir. Ancak bu aşamada sadece gerekli olan fonksiyonların hazır yazılmış olanları kullanıla-

<sup>9</sup> İngilizce kaynaklarda programın kaynak kodundaki komutların yer aldığı her bir satır “statement” olarak tanımlanmaktadır. Bir aritmetik veya sözel işleme ait olan işlemler ise “expression” olarak tanımlanmaktadır. Bu kitapta bu terimler yerine sırası ile bildirim ve aritmetik/mantıksal ifade kullanılacaktır.

caktır. Programda kullanılan her bir fonksiyon matematikteki fonksiyonlarda olduğu gibi fonksiyona özgü olan bir isime sahiptir. Bunlar dışında kalan ve **programın asıl işlevini** oluşturan fonksiyon daima **ana fonksiyon** olduğunu belirtmek için “**main**” olarak isimlendirilir. Ana fonksiyon önünde yer alan “**int**” ise fonksiyonun tam sayı değer döndürecekğini belirtir. Bir fonksiyonun ne tür değer döndüreceği ve bunun C++ programlarında nasıl kullanılacağı ile sonraki bölümlerde incelenecektir. Ana fonksiyonun herhangi bir parametre almayacağını belirtmek için “( )” arasında herhangi bir tanımlama yapılmamıştır. Teknik olarak her bir fonksiyon bir veya daha çok sayıda parametre alabilir ve bunları işleyebilir. Ana fonksiyon henüz herhangi bir parametre almayıcağı için ilgili alan boş bırakılmıştır. Alt satırda yer alan “{” fonksiyonun başladığı kısmı ifade eder ve aynı şekilde an fonksiyon içerisinde yer alan ifadelerin bitisi de aynı şekilde “}” ile gösterilir. Bu parantezler arasında kalan program bölümü **blok** olarak da isimlendirilir.

```
int main ()  
{  
....  
return 0;  
}
```

Yukarıdaki gösterim şekli yaygın olarak kullanılmakla birlikte aşağıdaki gibi de yazılabilir. Her ikisi de doğrudur ve derleyiciler tarafından işlenirler.

```
int main () {  
....  
return 0;  
}
```

Ana fonksiyonun son satırında yer alan parantezden önceki “**return 0**” ifadesi, programın sona satır kadar hatasız olarak işlediğini ve aynı şekilde sona erdiğini belirten sinyali göndermek için kullanılır. Sinyalin işlevi ve önemsi ise ilerleyen bölümlerde ele alınacaktır.

Yukarıda yer verilen açıklamalardan da görüldüğü gibi, nasıl ki günlük yaşamda kullandığımız dilin kendisine özgü kuralları bulunuyorsa, programlama dillerinin de aynı şekilde kendisine özgü dil bilgisi ve yazım kuralları vardır. Bu kurallara uymadan yapılacak bir program derleyici tarafından derlenmeyecektir. Derleyicinin belirttiği hataların giderilmesinden sonra program derlenebilir ve çalıştırılabilir. C++ programlama dili kullanılarak program yazarken şu kurallara dikkate edilmesi gereklidir.

- Program yazımı belirli bir kalıp içerisinde, bloklar halinde yazılır.
- Bloklar ise { } parantezler ile oluşturulur.
- Bildirimler ve ifadeler bu bloklar içerisinde aynı satırda veya alt alta olacak şekilde yazılabılır.
- Tüm bildirimlerin sonuna “;” yazılır.

- Eğer bir bildirim bir blok başlatılıyorsa bu durumda ilk satırın sonunda “;” kullanılmaz.
- Programda kullanılan tüm değişkenlerin veri tipleri tanımlanmalı yani bildirilmelidir.
- Programda kullanılacak komutların yer aldığı kütüphaneler, program başında tanımlanarak belirtilir. Bu şekilde program çalıştırıldığında, adı geçen kütüphaneler çağrılarak belleğe yüklenir ve aktif kılınarak program tarafından kullanılabilir.
- İfadeler içerisinde değişkenler, fonksiyonlar ve programlar arasında gerçekleştirilen parametre aktarımıları işlemlerinde bir değişken rakam ile başlayamaz, ama “\_” veya bir harf ile başlayabilir ve içerisinde rakamlar barındırabilir. Uzunluk için ir sınır bulunmamaktadır.
- İfadeler içerisinde yapılan tanımlamalarda büyük ve küçük harf ayrimına dikkat edilmesi gereklidir. Örneğin bir değişken “X” ile tanımlanmış ise program içerisinde aynı değişken başka bir blok içerisinde “x” olarak kullanılmaz. “X” ile “x” farklıdır. C++ Programlama dili büyük harf, küçük harf duyarlı bir dildir.
- Program içerisinde okunabilirliği artırmak için girintiler, sekmeler ve boşluklar kullanılmıştır.

Bu kurallar C++ Programlama Dili ile yazılan bir programın derleyicinin **anlamsal – semantic** ve **yazım kuralları – syntax** açısından yaptığı kontrollerin sorunsuz bir şekilde gerçekleşmesini sağlar. Ancak farklı programcıların ifadeler içerisinde yaptıkları tanımlamaların sık sık kişisel tercihlere dayalı olduğu unutulmamalıdır. Bu nedenle program içerisinde yorum satırları ile gerekli olan açıklamaların yapılması programın anlaşılmasını kolaylaşacaktır.

## 2. Girdi ve Çıktı İşlemleri

Önceki bölümde klavyeden girilen verilerin “cin” ve “>>” kullanılarak programlara aktarıldığını ve işlemlerin sonucunun da “cout” ve “<<” operatörleri kullanılarak standart çıktı yani komut satırına yazdırıldığını görmüştük. Bu bölümde ise “**girdi/çıktı – input/output – I/O**” işlemlerinin yapılmasında kullanılan ifadelerini, çıktıının nasıl biçimlendirilebileceğini, standart gidi ve standart çıktı ifadelerinin sınırlarından dolayı farklı giriş ve çıkış aygıtlarının kullanılarak girdi/çıktı işleminin nasıl çeşitlendirileceğini göreceğiz.

### 2.1. G/Ç Akımları ve Standart G/Ç Aygıtları

Bir bilgisayar programı temel olarak veriyi okur, veriyi işler ve sonuçları sunar. Tüm programlama dillerinde bu işlemler yapılır. Bir bilgisayar programının bu işlemleri yapabilmesi için bilgisayarda verinin giriş yapıldığı aygıtlar ile çalışabilmesi, bu aygıtlardan girilen verileri okuyabilmesi gereklidir. Önceki bölümde sayısal veriler üzerinde yapılan işlemlere ait örnekler verilmiştir. Bu bölümde ise sözel veriler üzerinde yağılan işlemler üzerinde duracağız. Sözel veriler sayısal verilere göre daha karmaşık işlemler ile işlenirler. Bu nedenle de C++ Programlama Dili’nde sözel veriler üzerinde yapılan işlemler için ayrı ön işlemci yönergeleri geliştirilmiştir. Bunlar aynı zamanda çeşitli girdi/çıktı işlemleri için kullanılan aygıtlara ait olan işlevler ile birleşerek veriler üzerinde gerçekleştirilen işlemleri çeşitlendirip programlara önemli bir esneklik kazandırır.

C++ programlama Dili’nde veriler bir byte dizisi olarak ele alınır. Bu yaklaşım hem sayısal veriler hem de sözel veriler için geçerlidir. Byte dizisi ise bilgisayarda bir aygıtın bir diğer aygıtına doğru veri yolları üzerinden iletilir. Bu iletim şekli de “**akım – stream**” olarak adlandırılır. Bir akım içerisinde farklı biçimlerdeki verileri taşıyabilir. Akım yönü dikkate alınarak iki gruba ayrılır:

- **Girdi Akımı – Input Stream:** Bir girdi aygıtından bilgisayar doğru iletlenen verilerdir.
- **Çıktı Akımı – Output Stream:** Bilgisayardan bir çıktı aygıtına doğru iletlenen verilerdir.

Aksi belirtildiği sürece bu ders notlarında ve pratikte her zaman için standart girdi olarak klavye ve standart çıktı da ekran/komut satırı olarak düşünülecektir. Klavyeden girilen verinin okunabilmesi için tüm C++ programları **iostream** başlık dosyasına gereksinim duyar. Bu dosya bir ön işlemci yönergesi olarak programlarda yer almaktadır. Bu başlık dosyası aynı zamanda iki özel veri tipi için gerekli olan tanımlamaları da içermektedir. Bunlar girdiler için **istream** ve çıktılar için de **ostream** yanımlarıdır. Bunlara ek olarak başlık dosyasında iki değişken tanımlaması da bulunur. Bunlar “**common input – cin**” ve “**common output – cout**” değişkenleridir. Bu değişkenler program içerisinde yazılrken aşağıdaki gibi yazılması gereklidir:

```
istream cin;
```

```
ostream cout;
```

Bu şekilde uzun olarak yazılmamasına gerek olmaması için aşağıdaki ön işlemci yönergesinin programın tanımlamalar kısmında belirtilmesi gereklidir.

```
#include <iostream>
```

Başlık dosyasının tanımladığı **istream** değişkeni **input stream variable** yani girdi akımı değişkeni olarak tanımlanır. Basitçe girdi aygıtlarından bilgisayara doğru gerçekleşen veri akımı içerisinde veri okumayı, çıkarmayı ve yerine geri koymak gibi işlemlerin yapılması için kullanılır. Diğer değişken olan **ostream** ise **output stream variable** yani çıktı akımı değişkeni olarak tanımlanır. Bilgisayardan çıktı aygıtlarına doğru gerçekleşen veri akımına veri eklemek için kullanılır. Programlarda cin ve cout ön işlemci yönergeleri ile tanımlanmakta olduğundan programcı tarafından tekrar tanımlanmasına gerek yoktur.

Önceki bölümde “cin” ve “>>” nasıl kullanıldığını görmüşük. Şimdi ise bu değişken ile birlikte kullanılabilen diğer fonksiyonlardan get, peek ve putback ile veriler üzerinde yapılabilecek olan işlemleri göreceğiz.

### 2.1.1.cin ve >> Operatörü

Standart girdi aygıtını klavyeden girilen verilerin cin ve >> ile değişkenlere atanabilmektedir. Aşağıdaki örnekte klavyeden girilen veri float veri tipinde tanımlanmış olan yarı\_cap adlı değişkene atanmaktadır.

```
cin >> yarı_cap;
```

Bilgisayar programı işlerken, klavyeden girilen değer okunup ilgili değişkene atanmaktadır. Değişkenlerin özelliği veri barındırmak ve bu verinin üzerinde işlem yapılması durumunda da güncellenen veriyi barındırmaktır. Değişken aynı zamanda verinin bellek üzerindeki adres bilgisini de tutmaktadır. Dolayısı ile de “>>” operatörü iki işlenen almaktadır. Sol taraftaki işlenen değişken olarak bellek adresi ile veriyi barındırırken sağ taraftaki işlenen ise değişkenin değeridir. Değişken ve operatör aşağıdaki gibi de kullanılabilir.

```
cin >> değişken0 >> değişken1 >> değişken2 >> ... ;
```

Yukarıdaki gösterimden de anlaşılacağı üzere okunan veri ayırtılarak bir kerede birden çok değişkene farklı veriler atanabilir. Örneğin bir işçinin günlük çalışma süresine bağlı olarak o iş günü sonunda alacağı normal ücreti ve fazla mesai ücretini hesaplayan bir program yazıldığını düşünelim. Klavyeden sırası ile girilmesi gereken veriler günlük çalışma süresi, normal mesai ücreti ile fazla mesai ücreti olsun. Bu duruma her üç veri de tek bir seferde aşağıdaki gibi girilebilir.

```
cout << "Çalışan süreyi, normal saat ücretini ve fazla mesai ücretini aralarında"  
<< boşluk bırakarak giriniz:";  
  
cin >> calisma_suresi >> mesai_saat_ucret >> fazla_mesai_saat_ucret;
```

Klavyeden de kullanıcı tarafından aşağıdaki gibi verilerin girilmiş olsun:

```
[puantor@muhasebe.borg.net ~]$ Çalışan süreyi, normal saat ücretini ve fazla mesai ücretini aralarında boşluk bırakarak giriniz:12 80 95.75
```

Yukarıdaki örnekte görüldüğü gibi veriler aralarında boşluk bırakılarak giriş yapıldığında herhangi bir veri tipi belirtmeden doğrudan girilmekte ve ardından da enter tuşuna basılarak giriş sonlandırılmaktadır. Giriş yapılan verinin sayısal veya sözel veri tipinde olup olmadığıının ayrimının yapılması ise “>>” operatörü tarafından yapılmaktadır. Sağ taraftaki işlenenin veri tipi nasıl tanımlanmış ise bu durumda girilen veri de buna göre değişkene atanmaktadır.

Örneğin klavyeden 22 girilmiş olsun. Bu girdi aşağıdaki değişkene atanmaktadır.

cin >> B;

B adlı değişkenin veri tipi eğer karakter ise değişken tarafından barındırılan veri 2 olacaktır. 5 işleme alınmaz ama saklanır. Ancak veri tipi tam sayı ise bu durumda 25 olduğu gibi değişken tarafından barındırılır. Eğer veri tipi double olarak tanımlanmış ise bu durumda veri 25.0 olarak barındırılacaktır. Aşağıdaki tablo da bu durum gösterilmektedir.

Değişkenin Veri Tipi	Geçerli olan girdi
Karakter	2
Tam sayı	25
Kayar Noktalı (double)	25.0

“>>” operatörü klavyeden girilen veriyi okurken veri tipine bağlı olarak ilk “**beyaz boşluk – white space**” gelene kadar okuma yapar. Eğer karakter veri tipinde tanımlanana bir değişkene yapılacak olan atama sadece tek karakterdir. İzleyen karakterler diğer değişkenlere atanırlar. Aşağıdaki örnekte bu durum gösterilmiştir.

Bir programda aşağıda görülen değişken tanımlamaları yapılmış olsun.

```
int a,b;  
double c;  
char d;
```

Bu program çalıştırıldığında aşağıda görülen girişler yapılmış olsun.

İfade	Girdi	Bellekte barındırılan değer
(1) cin >> d;	A	d = 'A'
(2) cin >> d;	AB	d = 'A'
(3) cin >> a;	22	a = 22
(4) cin >> a;	28.59	a = 28
(5) cin >> c;	100.21	c = 100.21
(6) cin >> a, b, c	12 45 202.09	a = 12, b = 45, c = 202.09
(7) cin >> a, b, c, d;	1 3 22.409 A	a = 1, b = 3, c = 22.409, d = 'A'
(8) cin >> c, b;	2.10 3	c = 2.10, b = 3
(9) cin >> a, b, c, d;	1 2 3.156 d	a = 1, b = 2, c = 3.156, d = 'd'
(10) cin >> a, d, c, b;	1A32.23	a = 1, d = 'A', c = 32, b = ?

Yukarıdaki örnek incelendiğinde (7) ve (10) dikkat çeken girişler bulunmaktadır. (7) girilen veriler ilk ikisi tek satırda girilmiş ve ardından sonraki satırda geçilerek veri girişine devam edilmiştir. “cin” ve “>>” operatörü veri girişleri için veri tipine bağlı olarak klavyeden gelen tüm veri akımını işleyerek sadece ilgili değişkene atanacak olan veriyi ayırtmaktadır. Dolayısı ise

veriler doğru şekilde ilgili değişkenlere atanmıştır. Son satırda (10) girilen veriler arasında boşluk bırakılmamıştır. Bu durumda “cin” ve “>>” operatörü girilen verileri tanımlanana veri tiplerine ve sırasına göre ayırtırma yoluna giderecektir. Görüldüğü gibi c değişkenine atanacak olan değer 3 veya 32 olarak ayırtılınmak durumunda olacağına göre girdinin bekendiği gibi okunmayacağı ortaya çıkmaktadır. Söz konusu veri girişinde b değişkeni için herhangi bir veri girişi yapılmadığı için program b değişkenine atanacak olan veriyi beklemektedir.

Yukarıdaki örnekte görülen durumlar dışında bir diğer durumda klavyeden girilen verilerin program içerisinde tanımlanmış olan değişkenlerden fazla olması durumudur. Bu durumda “cin” ve “>>” operatörü sadece söz konusu değişkenlere atanacak olan verileri işler ve geri kalan ise işleme almaz. Program sona erdiğinde ise işlenmeyen ve fazla olan veriler ise bellekten silinir.

“cin” ve “>>” operatörü veri girişlerinde önceki bölümünden bildiğimi veri dönüşümü işlemlerini yapabilir. Bu veri tipi dönüşümü herhangi bir operatöre gereksinim olmadan gerçekleşir. Aşağıdaki örnekte bu durum görülmektedir.

Bir programda aşağıda görülen değişken tanımlamaları yapılmış olsun.

```
int A,B;  
double C;  
char D;
```

Bu program çalıştırıldığında aşağıda görülen girişler yapılmış olsun.

İfade	Girdi	Bellekte barındırılan değer
(1) cin >> A;	12	A = 12
(2) cin >> B;	21	B = 21
(3) cin >> C;	99	C = 99.0
(4) cin >> D;	28.59	D = ?

Girişlerde görüldüğü gibi (1) ve (2) tam sayı veri tipinde tanımlanmış olan veriler doğru şekilde atamaları gerçekleştirmiştir. (3) ise kayar noktalı veri tipindeki C değişkenine atama yapılrken veri tam sayı veri tipinde olmasına karşılık kayar noktalı veri tipine dönüştürülerek atama gerçekleştmiştir. Bu veri tipi dönüşümleri önceki bölümlerde de görüleceği gibi otomatik veri tipi dönüşümü olarak tanımlanır ve sayısal verilerde uygulanır. Son işlemde ise hata oluşmaktadır. Karakter verit tipinde tanımlanmış olan değişkene tama yapılması söz konusu değildir. Çünkü girişten okunan veri kayar noktalı veri tipindedir. Bu duruma **“girdi hatası – imput failure”** adı verilir. İlerleyen bölümde tekrar ele alınacaktır.

Girdi akımlarından ayırtırılan verilerde görüldüğü üzere “\n” ve “ ” dikkate alınmamaktadır. Bu durum sayısal veriler için olağandır. Söz verilerde ise söz konusu karakterlerin de okunarak işlenmesi gerekebilir. Örneğin bir metin dosyası satır satır okunarak işleniyorsa, bu durumda satırın nerede sona erdiğinin belirlenmesi zorunludur. Yen, i satır karakteri olmadan programın bir satırın başlangıcını ve bitişini belirleyebilmesi olanaksızdır. Sonraki bölümlerde bu tür verilerin programa aktarılması için kullanılan ve **“G/Ç Fonksiyonları – stream functions –**

**stream member functions**" olarak tanımlanırlar. Bunlara get, peek, putback örnek verilebilir. Bu fonksiyonları öğrenmeden önce fonksiyonun ne olduğunu ve nasıl işlediğini inceleyelim.

## 2.2. Ön Tanımlı Fonksiyonlar

Programcılıkta bir fonksiyon belirli bir işlevi yerine getirmek için ilgili komutların ve varsa değişkenlerin birlikte kullanılması ile oluşturulan kod bloklarıdır. Önceki bölümden bilindiği üzere bir programda sadece bir tane ana fonksiyon bulunabilir. Ancak bu ana fonksiyon yanında başka fonksiyonlarda bulunabilir. Programda yer alan an fonksiyon program çalıştırıldığında işlenirken, diğer fonksiyonlar ise ancak gerekiğinde çalıştırılır. C++ Programlama Dili'nde çok sayıda hazır fonksiyon bulunur. Bunlara "**ön tanımlı fonksiyonlar – predefined functions**" adı verilir. Programcılarda gereksinimlere göre fonksiyonlar yazabilirler. Programcılar tarafından yazılan fonksiyonlara "**kullanıcı tanımlı fonksiyonlar – user defined functions**" adı verilir.

Ön tanımlı fonksiyonlar "**başlık dosyaları – header files**" ile kullanılabilirler. Bu dosyalarda bir çok farklı problemin çözülmesinde kullanılabilecek hazır fonksiyonlar bulunmaktadır. Başlık dosyaları bu açıdan bakıldığından bir fonksiyonlar kütüphanesi olarak da tanımlanabilir. Belirli bir fonksiyonun kullanılabilmesi için öncelikle ilgili fonksiyonun hangi kütüphane tarafından sunulduğu bilinmelidir. Örneğin bir  $x^y$  işlemini yapmak için kullanılabilecek olan başlık dosyası **cmath** kütüphanesinde bulunan **pow(x,y)**'dır. Örneğin  $2^4 = 16$  sonucunu elde etmek için pow(2, 4) veya pow(2.0, 4.0) yazılabilir. Benzer olarak da  $\sqrt{16} = 16^{0.5} = 16^{1/2}$  işlemi için de pow(16.0, 0.5) yazılmalıdır. Bu ifade programcılık jargonunda "**fonksiyonun çağrılması – function call**" olarak tanımlanır. İşlemler için kullanılan değerler ise "**argüman – arguments**" veya "**parametreler – parameters**" olarak isimlendirilir.

Aşağıdaki programda ön tanımlı fonksiyonların kullanımı görülmektedir. Program bir dairesinin yarı çapı verildiğinde alanını, çevresini hesaplamakta ve ayrıca açıklama metinin karakter cinsinden uzunluğunu da sunmaktadır.

```
/*
 * daire.cpp
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 *
```

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki  
\* ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.  
\*  
\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <cmath>
#include <string>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    double r = 0;
    double alan = 0;
    double cevre = 0;
    const double pi = 3.1412567;
```

```

string mesaj = "Bu program yarı çapı verilen bir dairenin çevresini ve alanını hesaplamaktadır. Programda herhangi bir birim esas alınmamıştır.";

cout << mesaj << endl;

cout << "Direnin yarı çapını giriniz= ";

cin >> r;

cevre = 2*pi*r;

alan = pi*pow(r, 2.0);

cout << "Dairenin çevresi= " << cevre << endl;

cout << "Dairenin alanı= " << alan << endl;

cout << "Programın açıklama metinin karakter olarak uzunluğu boşluklar da dahil olmak üzere " << mesaj.length() << " karakterdir." << endl;

return 0;

}

```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./daire
```

Bu program yarı çapı verilen bir dairenin çevresini ve alanını hesaplamaktadır. Programda herhangi bir birim esas alınmamıştır.

```
Dairenin yarı çapını giriniz= Dairenin çevresi= 76.0058
```

```
Dairenin alanı= 459.759
```

Programın açıklama metinin karakter olarak uzunluğu boşluklar da dahil olmak üzere 138 karakterdir.

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

Programın yazılmamasında kullanılacak tüm değişkenlerin etkinleştirilmesi tercih edilmiştir. Programda hazır fonksiyonlar kullanılmıştır. Karakter dizisinin uzunluğunu elde etmek için de karakter dizisi fonksiyonlarından length() kullanılmış ve karakter dizisinin uzunluğu elde edilmiştir.

Tüm programlama dillerinde yukarıda sözü edilen hazır fonksiyonlar bulunmaktadır. Bunların yazılması kolay bir süreç değildir. Ayrıca bir çok programcı tarafından aynı problemi çözmek üzere yazılan fonksiyonların etkinlik ve verimlilik açısından değerlendirilmesi de zahmetlidir. Problemin bir defa da en iyi şekilde çözülmesini sağlayan bu hazır fonksiyonlar tüm programcılar tarafından kolaylıkla erişilebilir ve kullanılabilir durumdadır. Programının yapması gereken bu fonksiyonları bilmek ve doğru bir şekilde kullanmaktır. Aşağıdaki girdi

## 2.2.1.cin ve get

Bir C++ programında cin ile veri okunması hazır fonksiyonlarının kullanılması ile kolaylıkla gerçekleştirilebilir. Bu veri giriş işlemleri sırasında cin klavyeden girilen ve “**beyaz boşluk – white**

**space**" boşlukları dikkate almaz. Bu boşluklar atlanır ve gelen ilk karakter doğrudan sıradaki değişkene aktarılması durumu söz konusu ise doğrudan aktarılır. Örneğin bir programda okunacak olan ilk veri karakter tipinde tanımlanmış ve üçüncü veri de tam sayı olarak tanımlanmış olsun. Bu durumda veri tipi tanımlamasına ilişkin satırlar aşağıdaki gibi olacaktır.

```
char karakter1, karakter2
```

```
int tam_sayı1
```

Bu tanımlama teknik olarak doğrudur. Ancak kullanıcı klavyeden aşağıdaki gibi bir veri girişi yaptığında yukarıda verilen kaynak kodun işlenmesi ile faklı bir durum ortaya çıkacaktır.

G 25

Yukarıdaki girdiye göre karakter1 değişkeni "G" ve karakter2 değişkeni de "2", tam\_sayı değişkeni de "5" değerini barındıracaktır. Klavyeden girilen verilerin aslında ilk karakterin "G" ikincisinin ise " " ve üçüncü girilen verinin de 25 sayısı olduğu durumu düşündüğümüzde işlemin yanlış olduğu ve programın çalışması sırasında olabilecek hataların önüne geçilmesi gerekecektir. Bu tür girişler için cin tek başına kullanılmayıp yanında ek bir fonksiyon ile birlikte kullanılır. Bu ek fonksiyon get fonksiyonudur.

```
char karakter1, karakter2;  
int tam_sayı1;  
  
int main()  
{  
    cin.get(karakter1);  
    cin.get(karakter2);  
    cin >> tam_sayı
```

Yukardaki örenekte görülen **cin.get(karakter veri tipi)** olarak yapılan tanımlama cin ile okunan verinin sadece karakter veri tipi için kullanılabileceği görülmektedir. Klavyeden girilen sayısal veri tipleri için get() kullanılmaz. Bu fonksiyon her seferinde tek bir karakter okuyarak karakter veri tipi olarak tanımlanmış olan değişkene atama yapacaktır.

## 2.2.2.cin ve ignore Fonksiyonu

Klavyeden girilen verinin bütününe de不由得 bir kısmının okunarak işleme alınması istenen durumlarda veri girişinin karakter olarak okunması istenmemektedir. Veri girişinin sadece belirli bir sayıdaki karakter girildikten sonra yapılması isteniyorsa bu durumda belirtilen sayıdaki karakterin okunmaması gereklidir. Bunun için **cin.ignore (tam sayı, karakter ifadesi)** fonksiyonu kullanılır. Bu fonksiyon veri girişinde belirtilen karakter sayısı kadar girilen karakteri veya tanımlanmış olan karakter ile karşılaşmasından sonra gelen veriyi okuyacaktır. Bu iki koşuldan hangisi önce gerçekleşse ise fonksiyon buna göre işler. Aşağıdaki örnek kod üzerinde görüleceği üzere ilk karakter okunduktan sonra cin.ignore() fonksiyonu çalışmaktadır. Fonksiyonda belirtildiği üzere okunan 256 karakter dikkate alınmayacağı veya ilk karşılaşılan boşluk karakteri sonrasında gelen karakterler okunacaktır.

```
/*
```

```
* ad_soyad_ilk_harf.cpp  
*  
* Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>  
* *  
* Her hakkı saklıdır.  
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde  
* kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:  
*  
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki  
* ret yazısını muhafaza etmelidir.  
*  
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki  
* ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.  
*  
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA  
* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
* SORUMLU DEĞİLLERDİR.  
*  
*/  
  
#include <iostream>
```

```

#include <string>
#include <iostream>
using namespace std;
int main () {
    setlocale(LC_ALL, "tr_TR.UTF-8");
    char ad, soyad;
    cout << "Adınızı ve soyadınızı giriniz: ";
    cin.get(ad);
    cin.ignore(256, ' ');
    cin.get(soyad);
    cout << "Adınızın ve soyadınızın ilk harfleri: " << ad << " " << soyad << endl;
    return 0;
}

```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```

[goksin@tardis ~/projeler/C++_Notlar]$ ./ad_soyad_ilk_harf
Adınızı soyadınızı giriniz: Gökşin Akdeniz
Adınızın ve soyadınızın ilk harfleri: G A
[goksin@tardis ~/projeler/C++_Notlar]$

```

Eğer fonksiyon bir sayı veya karakter olmadan kullanılırsa, teknik olarak sıradaki ilk karakteri okumayacaktır. Bu kullanım şekli özellikle de enter tuşuna basılarak yapılan girişlerde yeni satır karakteri yani enter tuşunun okunmasını engeller.

Aşağıdaki örnek programda ise sayılar arasında boşluk bırakılarak giriş yapılmaktadır. İlk sayı okunup değişkene atanırken, ikinci sayının okunması ve değişkene atanması için sonraki satır geçilmesi veya ilk 20 karakterden sonra gelen ilk karakterden başlanarak okuma yapılacaktır.

```

/*
 * iki_sayı_oku.cpp
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:

```

\*

\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki

\* ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki

\* ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, Veya TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ Veya BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA Veya HİZMETLERİN TEMİNİ; KULLANIM,

\*VERİ Veya RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
```

```
#include <locale>
```

```
using namespace std;
```

```
int main () {
```

```
    setlocale(LC_ALL, "tr_TR.UTF-8");
```

```
    int sayı1, sayı2
```

```
    cout << "Sıfırdan büyük pozitif tam sayılardan istediğiniz kadarını arasında boşluk bırakarak  
veya enter tuşuna basarak da girebilirsiniz: ";
```

```
cin >> sayi1  
cin.ignore(20,'n');  
cin >> sayi2  
cout << "İlk sayı " <<sayi1 << " ve ikinci sayı " << sayi2 << endl;  
return 0;  
}
```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./iki_sayi_oku
```

Sıfırdan büyük pozitif tam sayılardan istediğiniz kadarını arasında boşluk bırakarak veya enter tuşuna basarak da girebilirsiniz: 11 24 79 67 93

```
123 678 98751
```

```
İlk sayı 11 ve ikinci sayı 123
```

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

Eğer `cin.ignore()` herhangi bir koşul tanımlanmadan kullanılırsa, bu durumda ilk karakteri ihmal edere sıradakini okuyacaktır. Bu komut sıkılıkla yeni satır karakterinin ihmal edilmesi için kullanılır.

### 2.2.3.putback ve peek Fonksiyonları

Bazı durumlarda okunacak olan veriler karakterler, sayılar veya karakter diziler gibi karışık olarak sunulabilir. Bu durumda verilerin hangisinin karakter, hangisinin sayısal veya hangisinin karakter dizisi olduğunu belirlenmesi son derece güçtür. Bu tür durumlar için C++ programlama Dili iki adet hazır fonksiyon sunmaktadır. İlk fonksiyon olan **peek** veri akımındaki okuma işleminde sıradaki karakterin ne olduğunu kontrol etmek için kullanılır. İkinci fonksiyon olan **putback** ise veri akımın içerisinde okunup çıkarılan karakterin veri akımına geri konulmasını sağlar. Bu fonksiyonlar kullanılarak veri akımını oluşturan verilerin karakter, sayı veya karakter dizisi olup olmadığı kontrol edilerek program çalıştırılabilir. Böylece okunan verilerin dönüştürülmesi gibi işlemlere gerek kalmaz.

Aşağıda **peek()** fonksiyonun kullanımı görülmektedir.

```
ch = istreamVar.peek()
```

Bu atama ifadesinde **istreamVar** girdi akımı değişkenidir. **get()** aksine olarak **peek()** bağımsız olarak kullanılmaz ve bir atama ifadesi içerisinde kullanılmalıdır. Girdi içerisinde sıradaki karakteri okumaktadır. Okunan karakteri veri akımın olmadığı için veri halen veri akımı içerisinde yer alırken, okunan veri de bellek üzerinde ilgili değişkene atanarak bellekte tutulmaktadır.

Aşağıda **putback()** fonksiyonun kullanımı görülmektedir.

```
istreamVar.putback(ch);
```

Bu atama ifadesinde **istreamVar** girdi akımı değişkenidir. **peek()** aksine olarak **putback()** bağımsız olarak kullanılır. İlgili değişkende yer alan veri burada tekrar girdi akımına katılır. Örnek kullanımda ch olarak tanımlanan değişkenin değeri veri akımına geri konulmuştur. Aşağıdaki örnekte bir veri akımında aralarında boşluk bırakılmadan girilen veriler tek karakter olarak okunmaktadır. **peek()** ve **putback()** fonksiyonları kullanılmıştır.

```
/*
 * oku_gerikoy.cpp
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
 * SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
 * VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
 * BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
```

```
* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
* SORUMLU DEĞİLLERDİR.

*
*/
```

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

int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");

    char karakter, p;

    cout << "Aralarında boşluk bırakmadan dört adet harf girip enter tuşuna basınız: ";

    // İlk karakter okunur.

    cin.get(karakter);

    cout << "İlk okunan karakter: " << karakter << endl;

    //Sıradan devam edilir ve ikinci karakter okunur.

    cin.get(karakter);

    cout << "İkinci okunan karakter: " << karakter << endl;

    // İkinci okunan karakteri geri koyuyoruz.

    cin.putback(karakter);

    // Sıradaki karaktere göz atıyoruz.

    p = cin.peek();

    cout << "Sıradaki karakter: " << p << endl;

    //Üçüncü karakter okunur.

    cin.get(karakter);

    cout << "Üçüncü okunan karakter: " << karakter << endl;

    //Dördüncü karakter okunur.

    cin.get(karakter);

    cout << "Dördüncü okunan karakter: " << karakter << endl;

    return 0;
```

}

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./oku_gerikoy
```

Aralarında boşluk bırakmadan dört adet harf girip enter tuşuna basınız: İlk okunan karakter: a

İkinci okunan karakter: b

Sıradaki karakter: b

Üçüncü okunan karakter: b

Dördüncü okunan karakter: c

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

Yukarıdaki programın çalıştırılması durumunda girilen karakterler sırası ile okunacaktır. İlk okunan karakter ‘a’ olduğundan sonuç olarak “a” yazılacaktır. İkinci okunan aynı şekilde ‘b’ olacaktır. İkinci okumadan sonra okunan karakter, yani ‘b’ tekrar geri konulacak ve sıradan okumaya devam edilecektir. Üçüncü okunan karakter aslında ‘b’ geri konulmamış olsa, ‘c’ olacak iken yerin koyma işleminden dolayı ‘b’ tekrar okunmuştur. Programda son olarak dördüncü okuma ile ‘d’ okunması gereklilikten sıradaki karakter olan ‘c’ okunmuş ve program sona ermiştir.

## 2.2.4. G/Ç Akım ve G/Ç Fonksiyonları Arasındaki Fark

Önceki bölümlerde yer alan konularda G/Ç işlemlerinde kullanılan fonksiyonlar ile akım fonksiyonları arasında “.” yer almaktadır. Eğer araya nokta konulmaz ise derleyici bu durumda yazım hatası yapıldığını belirten bir hata mesajı döndürecektir. Burada yazım hatası ile anlatılmak istenen cin.get() yerine yazılan cinget() ile tanımlanmış bir fonksiyonunun çağrılmakta olmasıdır. Program içerisinde bu tür bir fonksiyon olmadığı için derleyici tanimsız bir fonksiyona atıf yapıldığını belirtecektir.

Bölümün başında da açıklandığı üzere **istream** değişkeni ilişkili olan ve belirli bir işlevi yerine getiren çok sayıda fonksiyon bulunmaktadır. Bu fonksiyonlar istream veri tipi ile ilişki olan üye fonksiyonlarıdır. Veri tipi ile fonksiyon arasında nokta konularak yapılan gösterime “**nokta notasyonu – dot notation**” adı verilir. Bu şekilde yazılarak **istream değişkeni** ile üye fonksiyon ayrıştırılır. Aslında C++ Programlama Dili’nde nokta “**üyelik erişim fonksiyonu – member access function**” olarak tanımlanır.

Esasen istream ve ostream veri tipleri nesne yönelimli programlama paradigmına göre iki ayrı sınıfıtır. Buna göre cin vde cout da nesneler olarak tanımlanır. Dolayısı ilse cin, istream nesnesi ve cout da ostream nesnesidir. Akım değişkenleri de akım nesneleri olarak tanımlanmak durumundadır.

## 2.3. Girdi Hataları

Bir programın işleyişi sırasında bir çok farklı nedenle programın işleyişi donabilir veya program hata mesajı döndürmeden çökebilir. Bu durumlarda karşı donanım düzeyinde alınmış olan önlemler programların çökmesini ve buna bağlı olarak da işletim sisteminin donması veya çökmesi

durumunun önüne geçilebilmektedir. Programların çalışırken askıda kalmasının çeşitli nedenleri olabilir. Bunlardan bir tanesi verit tipi tanımlamaları ile girilen veri arasındaki uyumsuzluktan kaynaklı hatalar neden ile programın donmasıdır. Bu durumlarda programın donmasının önüne geçilmesi için programcının alabileceği önlemler de bulunmaktadır. Eğer kullanıcı tarafından programa veri girişi yaparken değişkene atanın hatalı veri tipindeki girdiler nedeni oluşan hata durumunda program donmaktadır. Bu duruma “**girdi hatası – input failure**” adı verilir. Girdi hatasının ortaya çıkması durumunda ise girdi akımın okunması söz konusu olmayacağı için program devam edemecektir. Bu “**hata durumu – fail state**” olarak ifade edilir.

Aşağıdaki örnekte kisisle\_veriler.cxx adlı programı kullanacağız. Program önceden hazır olduğu için doğrudan çalıştırıp aynı değişkenleri kullanacağız. Ancak giriş yapılrken hatalı veri girişi yapılmaktadır.

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./kisisel_bilgiler
```

Adınızı, soyadınızı, yaşınızı ve kilonu aralarında boşluk bırakarak. giriniz.

Adınız ve soyadını sadece harflerden oluşabilir. Yaşınız tam sayı olarak girilmelidir. Kilonuzu ondalık sayı olarak girebilirsınız.

Baturalp Dinçdarı w33 65

Adınız: Baturalp

Soyadınız: Dinçdarı

Yaşınız: 0

kilonuz: 4.94066e-324

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

Programın çıktısında gözlenen hata yaşın “33” yerine “w33” olarak girilmesinden kaynaklanmaktadır. Yaş verisinin barındırıldığı veri tipi tam sayı iken karakter veri tipinde veri girişi olmuştur. Bu durumda cin hata durumunda kalmıştır.

### 2.3.1.clear Fonksiyonu

Girdi akımındaki hatalı girişlerin neden olduğu hata durumunda ya hatalı veri okunmasından kaynaklı program hata üretebilir. Bu hatalar programın hatalı işlemler gerçekleştirip hatalı sonuç üretmesi olabileceği gibi programın donması da olabilir. Bu tür durumların önüne geçilmesi için **istreamVar** değişkeni ile birlikte **clear()** fonksiyonu kullanılarak oluşan hata durumunun ortadan kaldırılıp programın kaldığı yerden devam etmesi sağlanabilir. Aşağıdaki programda hatalı veri girişinin neden olduğu hatanın ortadan kaldırılıp programın devam etmekte olduğu düzletmenin yapıldığı program görülmektedir.

```
/*
 * kisisel_bilgiler_duzeltme.cxx
```

\*

\* Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde

\* kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki

\* ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki

\* ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* **İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN**

\* **"OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA**

\* **UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA**

\* **KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT**

\* **REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA**

\* **BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ**

\* **SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ**

\* **YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA**

\* **ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP**

\* **SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,**

\* **VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE**

\* **BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ**

\* **HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI**

\* **SORUMLU DEĞİLLERDİR.**

\*

\*/

#include <iostream>

#include <locale>

```

#include <string>
#include <cfloat>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    string ad, soyad;
    int yas = 0;
    double kilo = 0;
    cout << "Adınızı, soyadınızı, yaşıınızı ve kilonu aralarında boşluk bırakarak. giriniz." << endl;
    cout << "Adınız ve soyadınız sadece harflerden oluşabilir. Yaşınız tam sayı olarak girilmeli-  
lidir. Kilonuzu ondalık sayı olarak girebilirsiniz." << endl;
    cin >> ad >> soyad >> yas >> kilo;
    cout << endl;
    cout << "Adınız: " << ad << endl;
    cout << "Soyadınız: " << soyad << endl;
    cout << "Yaşınız: " << yas << endl;
    cout << "kilonuz: " << kilo << endl;
    // Girdi hata durumu ortadan kaldırılır.
    cin.clear();
    // Ara bellek temizlenir.
    cin.ignore(200, '\n');
    cout << "Adınızı, soyadınızı, yaşıınızı ve kilonu aralarında boşluk bırakarak. giriniz." << endl;
    cout << "Adınız ve soyadınız sadece harflerden oluşabilir. Yaşınız tam sayı olarak girilmeli-  
lidir. Kilonuzu ondalık sayı olarak girebilirsiniz." << endl;
    cin >> ad >> soyad >> yas >> kilo;
    cout << endl;
    cout << "Adınız: " << ad << endl;
    cout << "Soyadınız: " << soyad << endl;
    cout << "Yaşınız: " << yas << endl;

```

```
cout << "kilonuz: " << kilo << endl;  
return 0;  
}
```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./kisisel_bilgiler_duzeltme
```

Adınızı, soyadınızı, yaşıınızı ve kilonu aralarında boşluk bırakarak. Giriniz.

Adınız ve soyadınız sadece harflerden oluşabilir. Yaşıınız tam sayı olarak girilmelidir. Kilonuzu ondalık sayı olarak girebilirsiniz.

Baturalp Dinçdari e33 65

Adınız: Baturalp

Soyadınız: Dinçdari

Yaşınız: 0

Kilonuz 0

Adınızı, soyadınızı, yaşıınızı ve kilonu aralarında boşluk bırakarak. Giriniz.

Adınız ve soyadınız sadece harflerden oluşabilir. Yaşıınız tam sayı olarak girilmelidir. Kilonuzu ondalık sayı olarak girebilirsiniz.

Baturalp Dinçdari 33 65

Adınız: Baturalp

Soyadınız: Dinçdari

Yaşınız: 33

Kilonuz 65

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

İlk programdakinden farklı olarak değişkenler içerisinde tam sayı ve kayar noktalı değişkenler programın başlangıcında etkinleştirilmiştir. Böylece bellek üzerinde diğer programlardan kalan verilerin okunarak hatalı işlemlerin yapılmasının önüne geçilmiş olmaktadır. Ayrıca okuma sırasında gerçekleşen hata durumunun de ortadan kaldırılması için ara bellek temizlenmiştir. Klavyeden yapılan girdinin tamamı ihmäl edilmek üzere **cin.ignore(200, '\n')** ile temizlik yapılmıştır. Ardından programın tekrar giriş yapılması istenen kısmına geçilerek program devam etmektedir.

## 2.4. Çıktı ve Çıktının Biçimlendirilmesi

Programcılar öncelikli görevleri yazdıkları programların verimli ve etkin olmasıdır. Ancak bir programın sadece bunları sağlama değil aynı zamanda işlediği veriyi de okunabilir bir şekilde sunması da istenir. Örneğin bilimsel çalışmalar için kullanılan bir programda çıktıının belirli bir şekilde sunulması beklenir. Ayrıca bir raporlama yapan programda verinin işlenmesi ile elde edilen sonuçların belirli bir şekilde yazdırılması ve hatta bazen de çıktıının çeşitli semboller kullanılarak grafikler ile desteklenmesi de gerekebilir. Bu işlemler için “**cout**” ve “**<<**” ile birlikte “**biçimlendiricilerin – manipulators**” kullanılması gereklidir.

```
cout << ifade veya biçimlendirici << ifade veya biçimlendirici << ... ;
```

Yukarıdaki gösterimde ifade işlenecek ve ardından da biçimlendirici ile belirtilen biçimde yazdırılacaktır. Bu bölüme kadar kullanılan en basit biçimlendirici yani satır geçilmesini sağlayan “**endl**”dır. Yaygın olarak kullanılan bazı biçimlendiriciler şunlardır: **setprecision**, **fixed**, **showpoint** ve **setw**.

### 2.4.1. **setprecision** Biçimlendiricisi

Kayar noktalı sayılar ile yapılan işlemlerde elde edilen sonuçların ondalık basamak sayısının sınırlanması için kullanılır. Örneğin çalışanlara ay sonunda ödenecek olan toplam ücretin hesaplanması ve bordroların yazdırılmasında kuruşları belirtmek için iki basamak tercih edilir. Benzer olarak bazı bilimsel amaçlı programlarda ise bu ondalık basamak sayısı altı ile sınırlılmaktadır. C++ Programlama Dili’nde ise ondalık basamaklarının sınırlanılması aşağıdaki gibi yapılır.

```
setprecision(n)
```

Bu ifadede yer alan n değeri ondalık basamak sayısını belirtir. Örneğin programda elde edilen sonucun barındırılılığı sonuc adlı değişkenin iki ondalık basamak ile yazılması için aşağıdaki şekilde kullanılması gereklidir.

```
cout << setprecision(2) << sonuc;
```

Çıktının biçimlendirilmesi için yukarıda belirtilen biçimlendirme operatörlerinin kullanılabilmesi için **iomanip** kütüphanesinin kullanılması gereklidir. Bunun içinde aşağıdaki önişlemci yönergesinin kullanılması gereklidir.

```
#include <iomanip>
```

### 2.4.2. **fixed** ve **scientific** Biçimlendiricisi

Kayar noktalı sayıların gösteriminde bu ifade kullanıldıktan sonra tüm kayar noktalı sayıların çıktısı tüm kaya noktalı sayılar için sabitlenir. Kullanım şekli aşağıda görüldüğü gibidir.

```
cout << fixed;
```

Eğer yapılan sabit basamak işleminin sona erdirilmesi gerekiyor ise bunu aşağıdaki bildirimin kullanılması ile yapılabilir.

```
cout.unsetf(ios::fixed);
```

Bu bildirimin ardından yapılacak tüm kayar noktalı işlemlerin sonuçları çıktıda varsayılan biçimleri ile görüntülenecektir.

Kayar noktalı sayılar ile yapılan işlemlerin sonuçları sabit basamak sayısı yerine bilimsel gösterim biçiminde yazdırılması istenirse, bu durumda **scientific** biçimlendiricisi kullanılır. Bilimsel gösterim biçiminde sayılar 10 üssü şeklinde gösterilir. Çıktıda ise bu **e+** ve **e-** olarak yazılır. Aşağıdaki örnek programda işlemlerin çıktısı önce bilimsel ardından da sabit basamak sayısı ile yazılmıştır

```
/*
 * sabit_bilimsel.cxx
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
 * SONUCUNDAN OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
```

\*VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <cfloat>
#include <string>
#include <iomanip>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    const long double avagadro = 6.02214199e+12;
    long double atom_sayi = 0;
    double mol = 0;
    string mesaj = "Atom sayısı= ";
    cout << "Bu program verilen mol için toplam atom sayısını hesaplar." << endl;
    cout << "Mol sayısını tam sayı veya ondalık basamaklı olarak giriniz:";
    cin >> mol;
    // Hesaplama
    atom_sayi = avagadro * mol;
    // Standart çıktıya yazdır.
    cout << mesaj << scientific << atom_sayi << endl;
    cout << mesaj << fixed << atom_sayi << endl;
    return 0;
}
```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./sabit_bilimsel
```

Bu program verilen mol için toplam atom sayısını hesaplar.

Mol sayısını tam sayı veya ondalık basamaklı olarak giriniz: 12.301

Atom sayısı= 7.407837e+13

Atom sayısı= 74078368618990.000938

[goksin@tardis ~/projeler/C++\_Notlar]\$

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

### 2.4.3.showpoint Biçimlendiricisi

C++ programlarında eğer kayar noktalı sayıların ondalık basamaklarının değeri sıfır ise bu basamaklar çıktıda yer almaz ve sayı tam sayı biçiminde yazdırılır. Ondalık basamakların gösterilmesi istenirse bunun için **showpoint** operatörü kullanılır. Ondalık basamakların değerleri sıfır olsa da çıktıda kayar noktalı sayı biçiminde gösterilir. Ondalık basamak sayısının ne kadar olması gerektiği ise **setprecision()** ile tanımlanmalıdır veya **fixed** kullanılmalıdır.

```
cout << showpoint;  
cout << fixed << showpoint;  
cout << setprecision(2) << showpoint;
```

Aşağıda verilen programda yarı çapı ve yüksekliği cm olarak verilen bir silindirin hacminin kaç litre olduğu hesaplanmaktadır. Programın çıktısında ondalık basamaklar üzerinde bir düzenleme yapılmadan, **fixed** ve **setprecision** kullanılarak çıktı biçimlendirilmiştir.

```
/*  
 * silindir_hacim.cxx  
 *  
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>  
 * *  
 * Her hakkı saklıdır.  
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde  
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:  
 *  
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki  
 * ret yazısını muhafaza etmelidir.  
 *  
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki  
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
```

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <cfloat>
#include <string>
#include <cmath>
#include <iomanip>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    const double dm3 = 1000;
    const double pi = 3.14159265;
    double cap = 0, yukseklik = 0, hacim = 0, litre = 0;
    string mesaj1 = "Önce silindirin çapını ve sonrada yüksekliğini cm olarak giriniz: ";
```

```

string mesaj2 = "Silindirin cm3 cinsinden hacmi: ";
string mesaj3 = "Silindirin litre cinsinden hacmi: ";
// Veri girişi
cout << mesaj1;
cin >> cap >> yukseklik;
// Hesaplamalar
hacim = pow((cap * 0.5), 2.0)* pi * yukseklik;
litre = hacim / dm3;
// Standart çıktıya biçimlendirmeden yazdır.
cout << mesaj2 << hacim << endl;
cout << mesaj3 << litre << endl;
cout << fixed << showpoint;
// Standart çıktıya ondalık basamak sayısı sabit olarak yazılır.
cout << mesaj2 << hacim << endl;
cout << mesaj3 << litre << endl;
// Standart çıktıya ondalık basamak sayısını üç ile sınırlandırır.
cout << setprecision(3);
cout << mesaj2 << hacim << endl;
cout << mesaj3 << litre << endl;
return 0;
}

```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./silindir_hacim
Önce silindirin çapını ve sonradan yüksekliğini cm olarak giriniz: 10 100
Silindirin cm3 cinsinden hacmi: 7853.98
Silindirin litre cinsinden hacmi: 7.85398
Silindirin cm3 cinsinden hacmi: 7853.982
Silindirin litre cinsinden hacmi: 7.854
Silindirin cm3 cinsinden hacmi: 7853.982
Silindirin litre cinsinden hacmi: 7.854
[goksin@tardis ~/projeler/C++_Notlar]$
```

#### 2.4.4.setw Biçimlendiricisi

Programın çıktılarının belirli bir genişlikteki sütunlar şeklinde yazılması için **setw()** kullanılır.

```
cout << setw(n);
```

Aşağıda verilen programda yarı çapı ve yüksekliği cm olarak verilen bir silindirin hacminin kaç litre olduğu hesaplanmaktadır. Programın çıktısında ondalık basamaklar üzerinde bir düzenleme yapılmadan, **fixed** ve **setprecision** kullanılarak çıktı biçimlendirilmiştir.

```
/*
 * silindir_hacim.cxx
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
 * SONUCUNDAN OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
```

\*VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <cfloat>
#include <string>
#include <cmath>
#include <iomanip>
using namespace std;

int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");

    const double dm3 = 1000;

    const double pi = 3.14159265;

    double cap = 0, yukseklik = 0, hacim = 0, litre = 0;

    string mesaj1 = "Önce silindirin çapını ve sonra yüksekliğini cm olarak giriniz: ";
    string mesaj2 = "Silindirin cm3 cinsinden hacmi: ";
    string mesaj3 = "Silindirin litre cinsinden hacmi: ";

    // Veri girişi
    cout << mesaj1;
    cin >> cap >> yukseklik;

    // Hesaplama
    hacim = pow((cap * 0.5), 2.0) * pi * yukseklik;
    litre = hacim / dm3;

    // Standart çıktıya biçimlendirmeden yazdır.
    cout << mesaj2 << hacim << endl;
    cout << mesaj3 << litre << endl;
```

```

cout << fixed << showpoint;

// Standart çıktıya ondalık basamak sayısı sabit olarak yazılır.

cout << mesaj2 << hacim << endl;

cout << mesaj3 << litre << endl;

// Standart çıktıya ondalık basamak sayısını üç ile sınırlandırır.

cout << setprecision(3);

cout << mesaj2 << hacim << endl;

cout << mesaj3 << litre << endl;

return 0;

}

```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./silindir_hacim
```

Once silindirin çapını ve sonrada yüksekliğini cm olarak giriniz: 10 100

Silindirin cm3 cinsinden hacmi: 7853.98

Silindirin litre cinsinden hacmi: 7.85398

Silindirin cm3 cinsinden hacmi: 7853.982

Silindirin litre cinsinden hacmi: 7.854

Silindirin cm3 cinsinden hacmi: 7853.982

Silindirin litre cinsinden hacmi: 7.854

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

## 2.4.5.setfill Biçimlendiricisi

Programın çıktılarındaki kayar noktalı işlemlerin sonuçları için **fixed** ve **setprecision** kullanmak yeterli olmaktadır. Bunun yanında tüm çıktıının sütunlar olarak biçimlendirilmesi için **setw** kullanılması biçimlendirme işlemleri için genellikle uygun olur. Bazı durumlarda ise bundan fazlasını yapmak gerekebilir. Çünkü **setw** ile yapılan biçimlendirme eğer çıktı sayısal veri tipinde ise sağa dayalı olarak ve sol tarafta boşluklar bırakarak yazdırılmaktadır. Karakter dizisi değişkenlerin kullanılması durumunda ise yerel dil ayarları operatörün davranışını etkilemektedir. Türkçe karakter kullanılmadığında sağa dayalı olarak yazılan çıktı Türkçe karakterler kullanıldığından ortalananarak yazıdırılmaktadır. Ayrıca seçilen sütun genişliği de eğer sonuç belirlenen genişlikten büyük ise bu durumda buna büyülükle göre otomatik olarak ayarlanmaktadır. Eğer sol tarafta kalan boşluklar yerine başka karakterlerin yazılması tercih edilecek ise **setfill(karakter)** kullanılmalıdır. Kullanım şekli çıktı akış değişkeninin “<<” operatörü ile birlikte **setfill(karakter)** olarak kullanılması şeklindedir.

```
ostreamVar << setfill(karakter);
```

Yukarıdaki şablonda **ostreamVar** ile gösterilen değişken **cout** veya diğerlerinden birisi olabilir. Bu kısımda ise cout ile kullanacağız. Aşağıda boşluklar yerine “#” karakterinin kullanımı görülmektedir.

```
cout << setfill('#');
```

Program içerisinde **setfill** kullanılabilmesi için **iomanip** ön işlemci yönertesine gereksinim vardır. Aşağıdaki programda kullanımı gösterilmiştir.

```
/*
 * bosluk_doldur.cxx
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 * *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 * *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 * *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
 * SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
```

\*VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <string>
#include <iomanip>
using namespace std;

int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişken tanımlamaları
    string isim = "Ahmet Selim";
    float ortalama = 2.70;
    int burs_miktari = 2000;
    // 40 sütun işaretlenir.
    cout <<"1234567890123456789012345678901234567890" << endl;
    //Çıktının normal şekilde biçimlendirilmesi
    cout << setw(20) << isim << setw(10) << ortalama << setw(10) << burs_miktari << endl;
    //Çıktının # karakteri ile biçimlendirilmesi
    cout << setfill('#');
    cout << setw(20) << isim << setw(10) << ortalama << setw(10) << burs_miktari << endl;
    //Çıktının _ karakteri ile biçimlendirilmesi
    cout << setfill('_');
    cout << setw(20) << isim << setw(10) << ortalama << setw(10) << burs_miktari << endl;
    //Çıktının @ karakteri ile biçimlendirilmesi
    cout << setfill('@');
```

```

cout << setw(20) << isim << setw(10) << ortalama << setw(10) << burs_miktari << endl;
return 0;
}

```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir. Aşağıda komut satırı çıktısının doğrudan alınmış ham hali olduğundan fontlar ve büyülükleri nedeni ile olduğundan farklı görülmektedir.

```

[goksin@tardis ~/projeler/C++_Notlar]$ ./bosluk_doldur
1234567890123456789012345678901234567890
    Ahmet Selim    2.7    2000
#####Ahmet Selim#####2.7#####2000
    Ahmet Selim_____2.7_____2000
@@@@@@@Ahmet Selim@@@@@@@2.7@@@@@@@2000
[goksin@tardis ~/projeler/C++_Notlar]$

```

## 2.4.6.left ve right Biçimlendiricisi

**setw()** ve **setfill()** kullanarak çıktıının biçimlendirilmesi için ön tanımlı ayarların değiştirilmesi gerekiyor ise bu durumda çıktıının belirtilen sütun genişliği içerisinde sola veya sağa dayalı olarak yazılması sağlanabilir. Sağa dayalı yazılması için **setw()** teknik olarak yeterli olsa da Türkçe karakterlerin kullanılması gerekiğinde ise **left** ve **right** biçimlendiricilerinin kullanılması daha okunaklı çıktılar elde edilmesini sağlar. Türkçe karakterlerin kullanıldığı durumlarda sola çıktıının dayalı yazdırılması daha uygundur. Windows kullanıyorsanız Türkçe karakterlerde bu sorun olmayacaktır. Kullanılan karakter kümesi tanımlaması Microsoft'un tüm ürünlerinin standartlara uyamamakta olması nedeni ile çıktıı daha farklı görüntülenebilir. Sola dayalı yazdırma için:

```
çktıakımıdeğişkeni << left;
```

Sola dayalı yazdırma sürecinin sonlandırılması için de:

```
çktıakımıdeğişkeni.unsetf(ios:: left);
```

kullanılır. Bu şekilde ön tanımlı olan sağa dayalı yazdırma biçimine geri dönülmüş olur. Eğer tüm çıktı sola dayalı olarak yazdırılırken sadece belirli bir bölümü sağa dayalı yazdırılacak ise aşağıdaki gibi kullanılabilir.

```
çktıakımıdeğişkeni << right;
```

Aşağıdaki örnek program bir önceki programın çıktıısının sola ve sağa dayalı olara çıktıı biçimlendirmesi şeklinde yeniden yazılmıştır. İsimler sola dayalı yazılırken, sayısal veriler sağa dayalı olarak yazılmaktadır.

```
/*
* bosluk_doldur_hizala.cxx
```

\*

\* Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde

\* kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki

\* ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki

\* ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* **İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN**

\* **"OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA**

\* **UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA**

\* **KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT**

\* **REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA**

\* **BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ**

\* **SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ**

\* **YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA**

\* **ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP**

\* **SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,**

\* **VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE**

\* **BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ**

\* **HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI**

\* **SORUMLU DEĞİLLERDİR.**

\*

\*/

#include <iostream>

#include <locale>

```

#include <string>
#include <iomanip>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişken tanımlamaları
    string isim1 = "Ahmet Selim";
    string isim2 = "Baturalp Kalkan";
    float ortalama1 = 2.70;
    float ortalama2 = 2.90;
    int burs_miktari = 2000;
    // 40 sütun işaretlenir.
    cout <<"1234567890123456789012345678901234567890" << endl;
    //Çıktının sola dayalı olarak yazdırılması
    cout << left << fixed << setprecision(2);
    cout << setw(20) << isim1;
    cout.unsetf(ios::left);
    cout << setw(10) << ortalama1 << setw(10) << burs_miktari << endl;
    cout << left;
    cout << setw(20) << isim2;
    cout.unsetf(ios::left);
    cout << setw(10) << ortalama2 << setw(10) << burs_miktari << endl;
    return 0;
}

```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir. Aşağıda komut satırı çıkışının doğrudan alınmış ham hali olduğundan fontlar ve büyüklükleri nedeni ile olduğundan farklı görünümktedir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./bosluk_doldur_hizala
1234567890123456789012345678901234567890
```

```
Ahmet Selim      2.70  2000
Baturalp Kalkan  2.90  2000
[goksin@tardis ~/projeler/C++_Notlar]$
```

## 2.5.Girdi/Çıktı İşlemlerinde Karakter Dizileri

Herhangi bir kaynaktan okunan verinin içi ile alınması durumunda boşluk karakterine kadar olan karakter dizisi okunur ve boşluk karakteri dikkate alınmaz ve okuma sona erer. Programın örneğin ad ve soyad arasında boşluk bırakarak girilmesi beklenen programlarda yetersiz kalacaktır. Bu kısıtlamanın aşılması için karakter dizisi olarak tanımlanan değişkenin kullanılması gereki gibi aynı zamanda boşluk karakterinin de okunması da istenir. Bunun için **getline** fonksiyonu kullanılır.

```
getline(girdiakımdeğişkeni, değişken);
```

Program kodu içerisinde getline kullanımı durumunda girdinin okunduğu kaynağı belirten değişken okuma sırasında boşluklara gelince durmayacak ve karakter dizisinin sona kadar okumaya devam edecektir. Aşağıdaki örnek programda kullanıcıdan adı ve soyadı arada boşluk bırakılarak girmesi istenmektedir.

```
/*
* bosluk_oku_selamlar.cxx
*
* Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
*
* Her hakkı saklıdır.
*
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
* kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
*
*
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
* ret yazısını muhafaza etmelidir.
*
*
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
* ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
*
*
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
```

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\*VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <string>
#include <iomanip>
using namespace std;

int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");

    // Değişken tanımlamaları
    string isim;

    // isim sor
    cout <<"Adınızı ve soyadınızı aralarında boşluk bırakarak yazıp enter tuşuna basınız." <<
endl;

    // isim öğren
    getline(cin, isim);

    //selamlama
```

```
cout << "Merhaba " << isim << " tanılığımıza memnun oldum." << endl;  
return 0;  
}
```

Programın derlenip çalıştırılması ile aşağıdaki sonuçlar elde edilir. Aşağıda komut satırı çıktısının doğrudan alınmış ham hali olduğundan fontlar ve büyülükleri nedeni ile olduğundan farklı görülmektedir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./bosluk_doldur_hizala  
Adınızı ve soyadınızı aralarında boşluk bırakarak yazıp enter tuşuna basınız.  
Baturlap Dinçdarı  
Merhaba Baturlap Dinçdarı tanılığımıza memnun oldum.  
[goksin@tardis ~/projeler/C++_Notlar]$
```

## 2.6.Dosyaları Kullanarak Girdi/Çıktı İşlemleri

Klavye kullanılarak verinin bir programa girilmesi veri miktarının az olduğu durumlarda herhangi bir soruna neden olmayacağıdır. Aynı şekilde işlenen veriye aştıktan sonra da ekrana sıhhatli bir şekilde az olduğu durumlarda sonuçların ekranda görüntülenmesinin de aynı şekilde bir sakıncası yoktur. Ama işlenecek olan verinin ve bu veriden elde edilecek olan sonuçların büyülüklüğü arttıkça klavyeden giriş yapmak ve ekranda sonuç görüntülemek ve okumak tercih edilmeyecektir. Ayrıca klavyeden veri girişinin de bazı sakıncalar yaratabileceği dikkate alınmalıdır. Klavyeden girilen veride yazım hataları olabilir ve bunlar bazı istenmeyen sonuçlara neden olabilir. Bunlardan başka üzerinde işlem yapılacak verinin işlem öncesi düzenlenmesi gerekebilir. Ve bu düzenlenmeden sonra işlenebilir durumda gelebilir.

Bu ve benzeri nedenlerden dolayı giriş ve çıkış işlemlerinin standart araçlar dışındaki kaynaklar ile gerçekleştirilebilir. Bunun için kullanılan yaygın yöntem verinin dosyadan okunup sonuçlarında bir dosyaya yazılmasıdır. Ön işlemci komutlarından **iostream** sadece standart girdi ve standart çıktı için gerekli olan dosyalara sahiptir. Dosyadan okuma ve dosyaya yazma işlemleri için ise **fstream** ön işlemci komutu kullanılır. Bu kütüphane dosyadan okuma ve dosyaya yazma işlemleri için gerekli olan iki değişkeni **ifstream** ve **ofstream** barındırmaktadır. Diğer ön işlemci komutu olan **iostream** olduğu gibi doğrudan **cout**, **cin**, **<<**, **>>**, **setfill**, **setprecision** ve **left**, **right**, **peek**, **putback** vb dosyalar üzerinde kullanılamaz. Bu nedenle **fstream** kütüphanesi diğer girdi/çıktı işlemleri ile birlikte kullanılmak için bu iki değişkeni kullanır. Değişkenlerin kullanılması için de öncelikle bu değişkenin tanımlanıp birer dosya ile ilişkilendirilmesi gereklidir. Girdi ve çıktıının farklı dosya kaynakları olması gerekliliği, girdinin okunurken işlemin sonucunun bir başka dosyaya yazılması gereğindenidir. Aynı dosyadan verinin okunup aynı dosyaya tekrar yazılması sırasında veri kayıplarının yaşanması ve programın hata sonucunda işlemez duruma gelmesi olasıdır.

Dosyaların kullanılarak girdi/çıktı işlemleri yapılabilmesi süreci şu adımlardan oluşur:

1. Ön işlemci yönergesi olan **#include <fstream>** programın tanımlamlar kısmına eklenmelidir.

2. Dosya okuma ve yazma değişkenlerinin tanımlanmalıdır.
3. Dosya okuma yazma işlemleri için tanımlanan değişkenlerin girdi çıktı işlem kaynakları ile ilişkilendirilmelidir.
4. Akım değişkenleri << ve >> operatörleri ile ya da diğer G/C fonksiyonları birlikte kullanılmalıdır.
5. İşlem sona erdiğinde dosyalar kapatılmalıdır.

Aşağıdaki program yukarıda gösterilen değişkenler, operatörler ve fonksiyonları kullanarak bir dosyadan veri okumaktadır. Okunan veriler işlenip daha sonra bir başka dosyaya yazılmaktadır.

**Programlama Problemi:** Bir dersi alan öğrenciler, iki ara sınav, bir dönem sonu sınavı ve iki ödev ile değerlendirilmektedir. Her bir değerlendirme etkinliği yüz puan üzerinden notlandırılmaktadır. Öğrencinin tüm değerlendirme etkinliklerine katılması zorunlu olup, katılmadığı etkinlikler sıfır olarak notlandırılmaktadır. Tüm değerlendirme etkinliklerinin sonuçları öğrencinin dönem sonundaki başarı değerlendirmesi hesaplanırken dikkate alınmaktadır. Tüm değerlendirme etkinliklerden alınan notların başarı notuna yaptığı katkı hesaplanıp, bu katkılar toplanarak dönem sonu başarı değerlendirme hesaplanmaktadır. Öğrencilerin değerlendirme etkinlerinden aldıkları sonuçlar notlar.txt adlı dosyadan okunmaktadır. Hesaplanan başarı notları karne.txt adlı dosyaya yazılmaktadır. Bu programı yazınız.

**Cözüm:** Program taslak olduğu için tek bir veri üzerinde işlem yapacaktır. Program başlatılınca öğrencinin adı ve soyadı, değerlendirme etkinliklerinin sonuçlarını dosyadan okuyacaktır. Okunan veriler her birisi ilgili katsayı ile çarpılarak sonuçlar toplanarak öğrencinin dönem sonu başarısı belirlenecektir. Elde edilen sonuçlar bir başka dosyaya yazılacaktır. Program sona erecektir.

```
/*
* not_hesapla.cxx
*
* Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
*
* Her hakkı saklıdır.
*
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
*kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
*
*
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
* ret yazısını muhafaza etmelidir.
*
*
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
* ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
```

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <fstream>
#include <string>
#include <cfloat>
#include <iomanip>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişken tanımlamaları
    ifstream dosyadan_oku;      // Dosya okuma işlemleri için değişken
    ofstream dosyaya_yaz;        // Dosya yazma işlemleri için değişken
```

```

string ad, soyad;           // Öğrencinin adı ve soyadı
float sinav1, sinav2, odev1, odev2, donemsonu, ortalama;
float k1=0.15, k2=0.15, k3=0.10, k4=0.10, k5=0.50;
// Dosya okuma ve yazma işlemleri için kaynakların tanımlanması
dosyadan_oku.open("notlar.txt");
dosyaya_yaz.open("karne.txt");
// Çıktının biçimlendirilmesi
dosyaya_yaz << fixed << showpoint << setprecision(2);
cout << "Veriler dosyadan okunuyor ve işleniyor..." << endl;
dosyadan_oku >> ad >> soyad;
dosyaya_yaz << "Öğrencinin adı, soyadı: " << ad << " " << soyad << endl;
dosyadan_oku >> sinav1 >> sinav2 >> odev1 >> odev2 >> donemsonu;
dosyaya_yaz << "Dönem içi değerlendirme sonuçları: " << setw(6) << sinav1 << setw(6)
<< sinav2 << setw(6) << odev1 << setw(6) << odev2 << setw(6) << donemsonu << endl;
ortalama = (k1 * sinav1) + (k2 * sinav2) + (k3 * odev1) + (k4 * odev2) + (k5 * donemsonu);
dosyaya_yaz << "Dönem sonu başarı notu: " << setw(6) << ortalama << endl;
dosyadan_oku.close();
dosyaya_yaz.close();
return 0;
}

```

Programın derlenip çalıştırılması ekran sadece program içerisinde mesaj satırı yazdırılacaktır. Veriler “notlar.txt” adlı dosyadan okunacak ve çıktıda “karne.txt” adlı dosyada oluşacaktır. Çıktının olusacağı dosya hazır olarak bulunmuyor ise bu dosya işletim sistemi tarafından ve içeri de program tarafından oluşturulur.

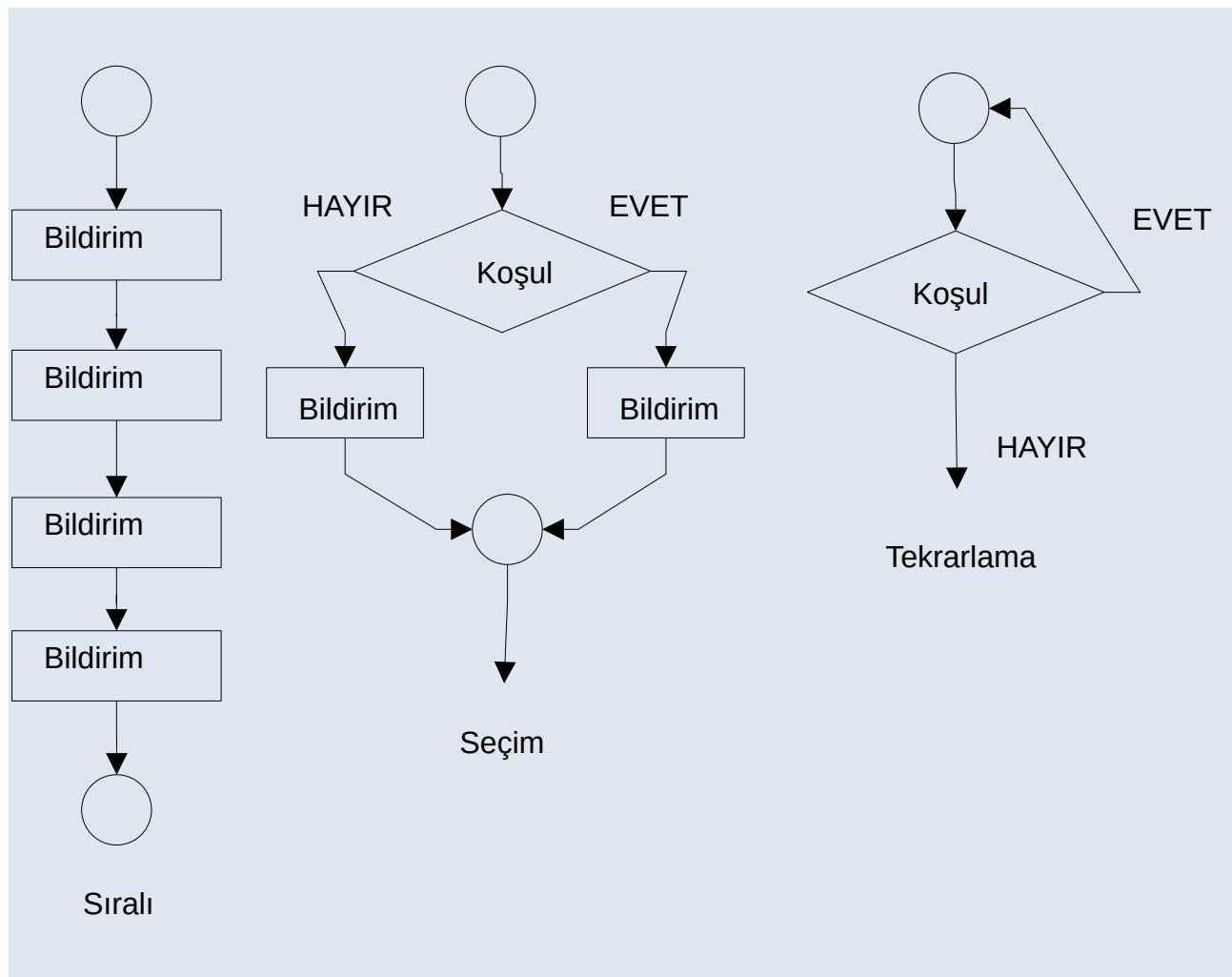
```
[goksin@tardis ~/projeler/C++_Notlar]$ ./not_hesapla
Veriler dosyadan okunuyor ve işleniyor...
[goksin@tardis ~/projeler/C++_Notlar]$
[goksin@tardis ~/projeler/C++_Notlar]$ ls
notlar.txt      karne.txt
[goksin@tardis ~/projeler/C++_Notlar]$ cat karne.txt
Öğrencinin adı, soyadı: Ahmet Selim
Dönem içi değerlendirme sonuçları: 65.00 70.00 88.00 92.00 78.00
```

Döne sonu başarı notu: 77.25

[goksin@tardis ~/projeler/C++\_Notlar]\$

### 3.Kontrol Yapıları: Seçimler

Önceki bölümlerde programlar ardışık olarak işlenen bildirimlerden oluşan yapılar olarak karşımıza çıkmıştı. Ancak gülük yaşamda çözüdüğümü bir çok problem, bir dizi yönergenin ardışık olarak işlenmesi ile çözülemez. Ardışık olarak tanımlanmış adımlar ile çözülemeyen problemlerde işlem basamakları ye önceden belirlenmiş bir koşulun gerçekleşmesi ile işlem sırasında değişiklik yapılarak sonuçlanır veya belirli bir koşulun gerçekleşmesi durumunda işlem basamakları işlenir veya işlemler belirli bir noktada sonlandırılır. Bu bölümde bilgisayarın istediği programın belirli durumlarda program akışını ardışık olarak bildirimleri işleyerek sürdürmek yerine farklı noktalara dallanarak gerçekleştirmesini, belirli koşulların gerçekleşmesi durumunda işlemlerin nasıl sonlanacağını göreceğiz.



Şekil 2: Bilgisayar Programlarının İşleyişi: Sıralı, Seçim ve Tekrarlama

Bir bilgisayar programı yukarıda belirtildiği gibi yönergeleri ardışık olarak işleyebilir, bazı yönergeleri atlayıp sadece bazlarını işleyebilir, bir dizi yönergeyi birden çok kere tekrarlayarak işleyebilir veya bir fonksiyonu çağrıp fonksiyondaki yönergeleri işleyerek sona erebilir. Sözü edilen program akışının nasıl gerçekleşeceği ise programcı tarafından kontrol yapıları kullanılarak tasarlanır. En yaygın kullanıla iki kontrol yapısı seçim ve yinelemedir. Seçim söz konusu olduğunda belirli bir koşuluk gerçekleşme veya gerçekleşmeme durumuna göre program akışı devam eder.

Yinelemede ise program akışı belirli bir koşulun gerçekleşme veya gerçekleşmemeye durumuna göre bazı yönergeleri yineleyerek işler. (**Şekil 2**)

### 3.1. “if” Ve “if ... else” İle Seçim Yapılması

Şekil 2’de görüldüğü gibi bir programın işleyişi sırasında belirli bir koşulun gerçekleşme durumunda program akışı değişmektedir .Bu nedenle öncelikle koşulların nasıl kurgulanacağı ve değerlendirileceğinin anlaşılması öncelikle gereklidir. Koşulların kurgulanmasında ve değerlendirilmesinde mantıksal ifadeler kullanılır. Bir mantıksal ifade mantıksal veri tipinde sonuç döndüren ifadelerdir. Sadece doğru ve yanlış olarak iki sonucu olabilir.

Örneğin “ $c > 3$ ” ifadesi mantıksal olarak doğru sonucunu döndürecektil. ASCII tablosuna göre  $c$ ’nin değeri 99 iken 3 değeri 51’dir. Bu ifadenin yazılmasında kullanılan büyükür işareti C++ Programlama Dili’nde büyükür operatörü olarak isimlendirilir. Aşağıdaki tabloda C++ Programlama Dili’nde mantıksal ifadelerde kullanılan operatörler aşağıda verilmiştir.

İlişkisel Operatör	Tanımı
$= =$	Eşittir
$! =$	Eşit değildir
$<$	Küçüktür
$>$	Büyükür
$< =$	Küçük veya eşittir
$> =$	Büyük veya eşittir

C++ Programlama Dili’nde atama ifadesi tek “ $=$ ” simbolü ile ifade edilir ve eşitlik ifadesi “ $= =$ ” ile ifade edilir. Yukarıda görülen “**ilişkisel operatörler – relational operators**” tekil operatörlerdir, sadece tek bir değişken ile işlem yaparlar. Yapılan karşılaştırmanın sonucu daize doğru veya yanlış olacağından bu operatörlerin doğru ve yanlış dışında bir sonuç döndürmezler.

#### 3.1.1. Basit Veri Tipleri ve İlişkisel Operatörler

Basit veri tipleri olan tam sayılar, karakter ve kayar noktalı sayılar ile ilişkisel operatörler kullanılabilir. Aşağıdaki örnekte ilişkisel operatörler ile gerçekleştirilen işlemler ve sonuçları gösterilmektedir.

İfade	Açıklama	Sonuç
$1 < 10$	1 10’dan küçüktür.	Doğru
$2 != 2$	2, 2’ eşit değildir.	Yanlış
$2.5 >= 1$	2.5,1’den büyükür	Doğru
$8.0 <= 11$	8, 11’den küçüktür	Doğru
$11.3 < 9.98$	11.3, 9.98’den küçüktür	Yanlış

### 3.1.2.Karakterlerin Karşılaştırılması

Karakterlerin karşılaştırılmasında ASCII karakter tablosu (Bkz: EK A) esas alınır. ASCII karakter tablosu bir karakterin ikilik tabandaki sayısal karşılığını belirtir. Dolayısı ile karakterlerin tekil olarak birbirine göre karşılaştırılması yapılabilir.

İfade	Açıklama	Sonuç
'A' < 'a'	65 < 97	Doğru
'3' > '#'	51 > 35	Doğru
'a' > 'c'	97 > 99	Yanlış
'Q' <= 'R'	81 <= 82	Doğru
12 < 'Z'	12 < 90	Doğru

Yukarıdaki örneklerde karakterler, ASCII tablosu esas alınarak birbiri ile karşılaştırılmaktadır. Bu işlem karakter dizilerinin alfabetik olarak sıralanmasında kullanılır. Son satırda ise bir tam sayı, bir karakter ile karşılaştırılmaktadır. Karakter ASCII tablosundaki sayısal değeri ile işlem alınacağından karşılaştırıldığı tam sayının değerine göre karşılaştırma yapılacaktır. Bundan dolayı da işlemin sonucu doğru olacaktır.

### 3.1.3.Tek Seçenekli Bildirimler

Bir işlemin gerçekleştirilmesinden önce, daha önceden belirlenmiş olan bir koşulun gerçekleşme durumunun kontrol edildiği ve sonucun doğru olması durumunda bir veya birden çok sayıdaki işlemin gerçekleştirildiği ifadelerdir. Eğer koşulun gerçekleşmediği tespit edilirse yani önceki işlemler sonucunda karşılanması gereken işlemler gerçekleşmedi ise, bu durumda söz konusu işlemler gerçekleşmez.

Örneğin sağlık sigortalarında müşterinin ödeyeceği prim miktarı belirlenirken sigorta yapacak olan kişinin sağlığı ve alışkanlıkları ile ilgili bilgilere bakılır. Eğer kişi sigara içme alışkanlığına sahipse, bu alışkanlığa sahip olmayan birisine göre daha ödeyeceği primin parasal miktarı fazladır. Benzer bir şekilde, bir banka müşterisinin vadesiz hesabına tanımlanan kredili mevduat hesabı, müşteri tarafından kullanılması durumunda hesap kesim tarihinde müşteri bankaya borçlu olacak ve borcunu fizi ile birlikte ödemek durmunda kalacaktır. Eğer hesap kullanılmamış ise müşteri bankaya herhangi bir ödeme yapmak durumunda olmayacağındır.

C++ Programlama Dili’nde mantıksal ifadelerin kullanıldığı ve sadece belirli bir koşuluk gerçekleşme durmuna göre program içerisinde belirli kod bloklarının işlenip işlenmeyeceğinin belirlendiği ifadelere “**tek seçenekli ifadeler – one way selection**” adı verilir. Aşağıdaki gibi yazılarılar. Bildirimin tek satır veya birden çok sayıdaki satırдан oluşması fark etmeksızın bildirimlerin “{ }” arasında yazılması derleyicinin hangi uyarıları döndürecekine ve bunlara hangi eylemleri gerçekleştireceğine göre değişir. Bu nedenle alışkanlık olarak “{ }” arasında yazılması uygundur.

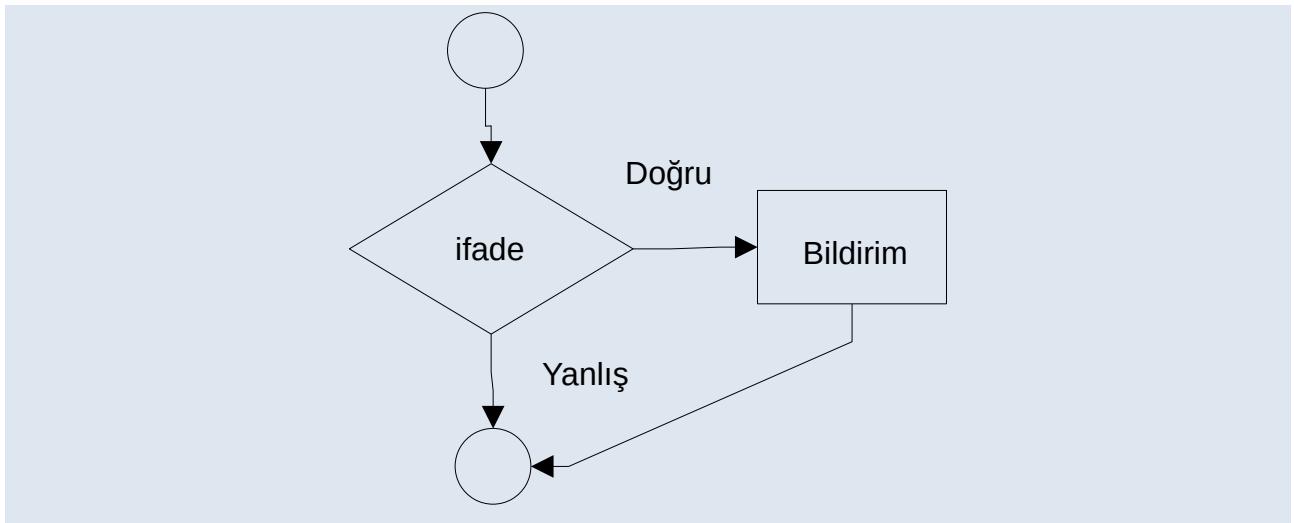
```
if (ifade)
```

```
{
```

```
    bildirim
```

}

Bu ifadeler içerisinde yer alan ve parantez içerisinde yazılan ifadeye “**karar verici – decision maker**” adı verilir. Karar verici ifade yaygın olarak mantıksal bir ifadedir. Mantıksal ifadenin işlenmesi sonucunda doğru veya yanlış sonucuna göre bildirim işlenir. Bu bildirim aynı zamanda “**eylem bildirimi – action statement**” olarak isimlendirilir. (**Şekil 3**)



**Şekil 3: Tek seçenekli ifadeler**

Aşağıdaki örnekte bir hayali bankaya ait kredi kartı borcunun son ödeme tarihinden önce borcun tamamı ödenmemesi durumunda bir sonraki hesap kesimine eklenecek olan faiz miktarı hesaplanmaktadır

```
/*
 * faiz_hesapla.cxx
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 */
```

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <cfloat>
#include <iomanip>
using namespace std;

int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // sabitlerin tanımlanması
    const float FAIZ_ORAN = 0.35;
    // Değişken tanımlamaları
    float donem_borcu = 0.0, odeme = 0.0, kalan_borc = 0.0, odenecek_faiz=0.0,
toplam_borc=0.0;
```

```

// Çıktının biçimlendirilmesi
cout << fixed << showpoint << setprecision(2);

// Verilerin girilmesi
cout << "Dönem borcunu giriniz: ";
cin >> donem_borcu;

cout << endl << "Yapılan ödeme miktarını giriniz: ";
cin >> odeme;

cout << endl;

// Hesaplama
kalan_borc = donem_borcu - odeme;
if (kalan_borc > 0.0)
{
    odenecek_faiz = FAIZ_ORAN * kalan_borc;
    toplam_borc = kalan_borc + odenecek_faiz;
}

// İşlem sonuçlarının yazdırılması
cout << "Gelecek hesap kesimine devreden borç: " << kalan_borc << endl;
cout << "Gelecek hesap kesiminde ödenecek borç faizi: " << odenecek_faiz << endl;
cout << "Gelecek hesap kesimine devreden toplam borç: " << toplam_borc << endl;
return 0;
}

```

Programın derlenip çalıştırılması ile aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./faiz_hesapla
```

```
Dönem borcunu giriniz: 23456
```

```
Yapılan ödeme miktarını giriniz: 1234
```

```
Gelecek hesap kesimine devreden borç: 22222.0
```

```
Gelecek hesap kesiminde ödenecek borç faizi: 7777.70
```

```
Gelecek hesap kesimine devreden toplam borç: 29999.70
```

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

Aşağıdaki program örnekleri bir tek seçenekli ifade yazılırken yapılabilecek olan bazı hataları göstermektedir.

Aşağıdaki program blokunda yazım hatası söz konusudur. Çünkü mantıksal ifade parantez içerisinde yazılmamıştır.

```
if basari_notu <= 33  
    harf_ntou = "FF";
```

Aşağıdaki kod blokunda parantezler kullanılmış ancak mantıksal bir hata söz konusudur. Mantıksal ifade parantez içerisinde yazılmış ancak parantez sonunda ";" ile satır sonlandırıldığı için bir bir "if ... null" yapısı olarak değerlendirilir. Diğer bir deyişle işlem sonucunda "{}" arasındaki kısım üstteki bölüm ile ilişkilendirilmemiş olmaktadır. "{}" arasındaki bildirim yukarıdaki mantıksal ifadenin sonucundan bağımsız olarak işleyecektir.

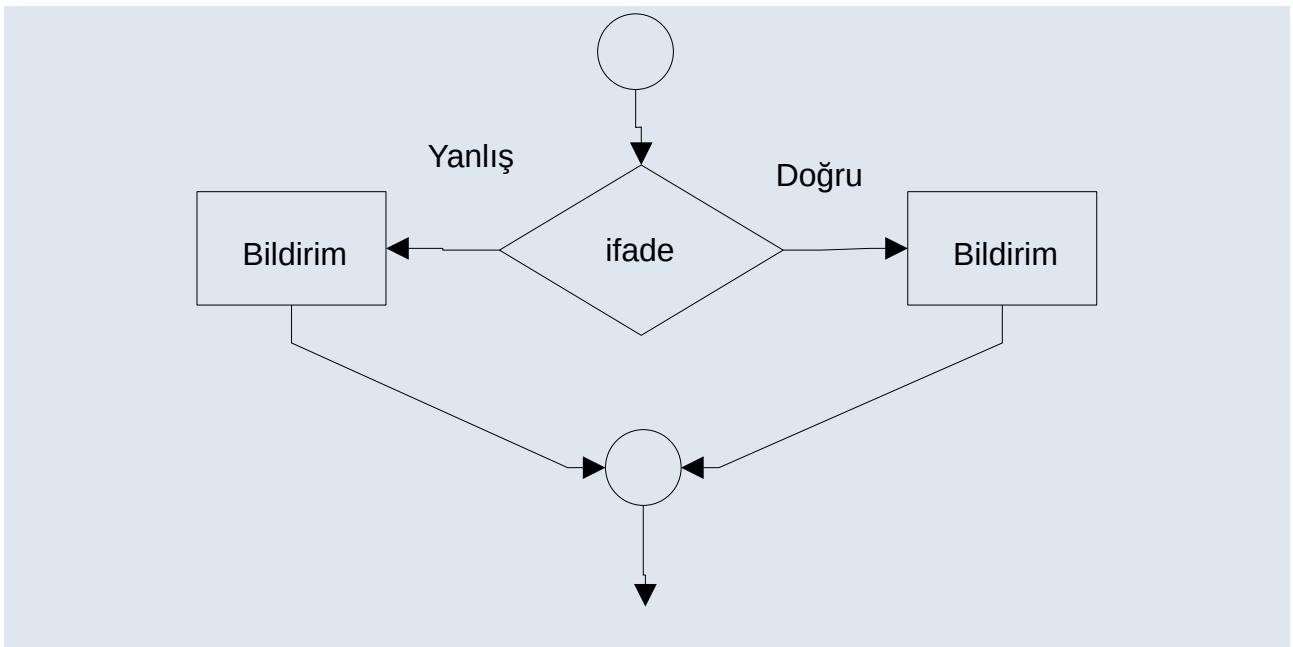
```
if (basari_notu <= 33);  
{  
    harf_ntou = "FF";  
}
```

### 3.1.4.Çift Seçenekli Bildirimler

Bir çok durumda karşı karşıya kalınan ve aralarından seçim yapılması gereken iki farklı seçenek olabilir. C++ Programlama Dili ile yazılan programlarda bu işlemler için iki ayrı mantıksal ifadenin kullanıldığı ve sadece belirli bir koşuluk gerçekleşme durmuna göre iki seçenekten birisinin seçilerek işlendiği kod bloklarına sahip olan ifadelere **"çift seçenekli ifadeler – two way selection"** adı verilir. Aşağıdaki gibi yazılrıllar. Bildirimin tek satır veya birden çok sayıdaki satırdan oluşması fark etmeksızın bildirimlerin "{}" arasında yazılması derleyicinin hangi uyarıları döndüreceğine ve bunlara hangi eylemleri gerçekleştireceğine göre değişir. Bu nedenle alışkanlık olarak "{}" arasında yazılması uygundur.

```
if (ifade)  
{  
    bildirim  
}  
else  
{  
    bildirim  
}
```

Bu ifadelerin yapısında iki adet **"karar verici – decision maker"** bulunur. Mantıksal ifadelerin sonucuna göre bildirimlerden bir tanesi işlenir. (**Şekil 4**)



**Şekil 4: Çift seçenekli ifadeler**

Aşağıdaki örnekte bir iş yerinde çalışmakta olan işçiler için çalışma süresine göre ay sonunda ödenecek olan ücreti hesaplayan program görülmektedir. Aylık normal çalışma süresi 45 saat olarak belirlenmiş ve buna göre ücret hesapları yapılmaktadır.

```

/*
 * ucret_hesapla.cxx
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

```

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <cfloat>
#include <iomanip>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // sabitlerin tanımlanması
    const float FAZLA_MESAI_UCRET = 1.5;
    // Değişken tanımlamaları
    float mesai_ucret = 125.0, fazla_mesai_ucret = 0.0, toplam_ucret = 0.0, sure=0.0,
ucret=5625.0 ;
    // Çıktının biçimlendirilmesi
    cout << fixed << showpoint << setprecision(2);
```

```

// Verilerin girilmesi

cout << "Aylık toplam çalışma süresini saat olarak giriniz: ";
cin >> sure;

// Hesaplamalar

if (sure > 45.0)

{
    sure = sure - 45.0;

    fazla_mesai_ucret = FAZLA_MESAI_UCRET * sure * mesai_ucret;

    toplam_ucret = ucret + fazla_mesai_ucret;
}

else

{
    toplam_ucret = mesai_ucret * sure;
}

// İşlem sonuçlarının yazdırılması

cout << "Bu ay sonunda ödenecek ücret: " << toplam_ucret << endl;

return 0;
}

```

Programın derlenip çalıştırılması ile aşağıdakine benzer bir çıktı elde edilecektir.

```

[goksin@tardis ~/projeler/C++_Notlar]$ ./ücret_hesapla
Aylık toplam çalışma süresini saat olarak giriniz: 35
Bu ay sonunda ödenecek ücret: 4375.00
[goksin@tardis ~/projeler/C++_Notlar]$ ./ücret_hesapla
Aylık toplam çalışma süresini saat olarak giriniz: 55
Bu ay sonunda ödenecek ücret: 7500.00
[goksin@tardis ~/projeler/C++_Notlar]$ ./ücret_hesapla
Aylık toplam çalışma süresini saat olarak giriniz: 45
Bu ay sonunda ödenecek ücret: 5625.00
[goksin@tardis ~/projeler/C++_Notlar]$

```

### 3.1.5.Tam Sayı Veri Tipi ve Mantıksal İfadeler

C++ Programlam Dili'nin ilk sürümlerinin duyurulduğu günlerde dilin yapısında “**mantıksal – boolean**” veri tipleri bulunmuyordu. Mantıksal ifadelerin sonucunun **0** veya **1** olabileceğinden hareketle mantıksal ifadelerin sonuçlarının tam sayı veri tipinde barındırılması tercih edilmişti. Bundan dolayı bugünde yazılan programlarda mantıksal ifadelerin sonuçları tam sayı veri tipi ile işlenebilmektedir.

Aşağıdaki programördeklarımda klavyeden girilen yaşı değerine göre bireyin reşit olup olmadığı belirlenmektedir.

```
/*
 * int_bool.hxx
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
```

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\*VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişken tanımlamaları
    int yas, yasal_yas;
    // Verilerin girilmesi
    cout << "Yaşını tam sayı olarak giriniz: ";
    cin >> yas;
    // Hesaplamalar
    yasal_yas = (yas >= 18);
    if ( yasal_yas == 1 )
    {
        cout << "Reşittir." << endl;
    }
    else
    {
        cout << "Reşit değildir." << endl;
    }
    return 0;
}
```

Programın derlenip çalıştırılması ile aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./int_bool
```

Yaşını tam sayı olarak giriniz: 22

Reşittir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./int_bool
```

Yaşını tam sayı olarak giriniz: 18

Reşittir.

Yaşını tam sayı olarak giriniz: 17

Reşit değildir.

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

Yukarıdaki programda yer alan aşağıdaki bildirim, bir mantıksal sınama işlemidir.

```
yasal_yas = (yas >= 18);
```

Klavyeden girilen yas değişkenin değeri ons sekizden büyük veya eşit ise ifadenin sağ tarafının değeri 1 olmaktadır. Aksi durumda ise sıfır olmaktadır. Dolayısı ile de kontrol yapısı içerisinde ilgili olan kod bloku çalıştırılmaktadır.

Mantıksal sınamalarda değil işlemi yani “!( )” parantez içerisinde yer alan ifadenin sonucunu sıfır ise bir, bir ise sıfır yapacaktır. Eğer ifade !(true) olarak yazılsa sonucu “**true – doğru**” yani bir olarak döndürülür. İfadeden içindeki mantıksal işlemin sonucu “**false – yanlış**” olduğundan bir daha değil işleminden geçirilmesi sonucunda değer “**true – doğru**” olmaktadır. Eğer aynı mantıksal ifade bir sayısal değer üzerinde kullanılırsa örneğin !(2) ifadesinin sonucu bir yani “**true – doğru**” olmaktadır. Çünkü !(2) ifadesinin sonucu sıfır olurken !0 ise 1 olmaktadır.

### 3.1.6.Mantıksal Veri Tipi ve Mantıksal İfadeler

C++ Programlam Dili’nin güncel sürümlerinde mantıksal veri tipleri bulunduğuundan işlem sonuçlarının mantıksal veri tipleri ile değerlendirmek olanaklıdır. Yukarıda yazılan program mantıksal veri tipi kullanılarak yeniden düzenlenmiştir. Dikkat edilmesi gereken nokta C++ Programlama Dili’nde **bool**, **true** ve **false** sözcüklerinin ayrılmış sözcükler olduğunu. Programlarınızda bunu dikkate alarak yazmanız gerekmektedir.

Aşağıdaki program örneğinde klavyeden girilen yaş değerine göre bireyin reşit olup olmadığı belirlenmektedir.

```
/*
 * bool_bool.cxx
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 * *
```

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM, VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
using namespace std;
int main()
{
```

```

// Dil ayarları
setlocale(LC_ALL, "tr_TR.UTF-8");

// Değişken tanımlamaları
int yas;
bool yasal_yas;

// Verilerin girilmesi
cout << "Yaşınızı tam sayı olarak giriniz: ";
cin >> yas;

// Hesaplamalar
yasal_yas = (yas >= 18);

if ( yasal_yas == true )
{
    cout << "Reşittir." << endl;
}
else
{
    cout << "Reşit değildir." << endl;
}

return 0;
}

```

Programın derlenip çalıştırılması ile aşağıdakine benzer bir çıktı elde edilecektir.

```

[goksin@tardis ~/projeler/C++_Notlar]$ ./bool_bool
Yaşınızı tam sayı olarak giriniz: 24
Reşittir.

[goksin@tardis ~/projeler/C++_Notlar]$ ./bool_bool
Yaşınızı tam sayı olarak giriniz: 18
Reşittir.

[goksin@tardis ~/projeler/C++_Notlar]$ ./bool_bool
Yaşınızı tam sayı olarak giriniz: 16
Reşit değildir.

[goksin@tardis ~/projeler/C++_Notlar]$

```

### 3.1.7. Mantıksal Operatörler ve Mantıksal İfadeler

Yukarıdaki örneklerde program akışını kontrol eden tek veya çift seçimli bildirimler içerisinde tek bir işlenene sahip mantıksal operatörler kullanılmıştır. Bir çok problemin bilgisayar ile çözümünde ise tek işlenenli mantıksal operatörler problemin çözümü için yeterli olmayabilir. Örneğin aşağıdaki ifadede iki farklı mantıksal koşul bir mantıksal ifade içerisinde birlikte yer almaktadır.

```
cinsiyet == 'E' and boy > 170 or boy < 190 and kilo > 95 or kilo < 65
```

Bu mantıksal ifadede yer alan koşullar “and” ve “or” mantıksal operatörleri ile birlikte **“bileşik ifade – compound expression”** olarak kurgulanmıştır. Bu tür ifadelerin C++ Programlama Dili ile işlenebilmektedir. Bileşik ifadede kullanılan “**ve – and**”, “**veya – or**” gibi operatörlere **“mantıksal operatörler – boolean/logical operators”** adı verilir. Aşağıda C++ programlama Dilinde kullanılan mantıksal operatörlerden en yaygın kullanıma sahip olanlara yer verilmiştir.

Mantıksal Operatör	Tanımı
!	Değil
&&	Ve
	Veya

Yukarıda gösterilen mantıksal operatörler içerisinde sadece “**değil – not**” tek işlenene sahiptir. “**ve – and**” ve “**veya – or**” operatörleri ise iki işlenene sahiptir. Ve ile veya operatörlerini oluşturan semboller arasında boşluk bulunmamaktadır. Aşağıda yukarıda belirtilen mantıksal operatörlerin işlem sonuçları gösterilmiştir.

NOT - Değil	Sonuç
Doğru - True	Yanlış – False - 0
Yanlış - False	Doğru – True -1

Aşağıda “**değil – not**” operatörü ile gerçekleştirilen mantıksal işlemlere örnekler verilmiştir.

İfade	Açıklama	Sonuç
!(‘A’ < ‘a’)	65 < 97, sonuç 0 olur ama değil ile sonuç 1 olur	Doğru
!(‘3’ > ‘#’)	51 > 35, sonuç 1 olur ama değil ile sonuç 0 olur	Yanlış
!(‘a’ > ‘c’)	97 > 99, sonuç 0 olur ama değil ile 1 olur	Doğru
!(‘Q’ <= ‘R’)	81 <= 82, sonuç 1 olur ama değil ile 0 olur	Yanlış
!(12 < ‘Z’)	12 < 90, sonuç 1 olur ama değil ie 0 olur	Yanlış

Aşağıda gösterilen tabloda “**ve – and**” işleminin sonuçları görülmektedir. “**Ve – and**” işlemi iki işlenen ile işlem yapmaktadır. Her iki ifadenin sonuçları ile “**ve – and**” işlem yaparak sonuç döndürmektedir. “**ve – and**” mantıksal operatörü ancak her iki ifadenin sonucunun 1 olması durumunda 1 sonucunu döndürür, diğer durumlarda sonuç daima sıfırdır.

İfade	İfade	İfade && ifade
True – Doğru (1)	True – Doğru (1)	True – Doğru (1)
True – Doğru (1)	False – Yanlış (0)	False – Yanlış (0)
False – Yanlış (0)	True – Doğru (1)	False – Yanlış (0)
False – Yanlış (0)	False – Yanlış (0)	False – Yanlış (0)

Aşağıda “**ve – and**” operatörü ile gerçekleştirilen mantıksal işlemlere örnekler verilmiştir.

İfade	Sonuç	Açıklama
(‘A’ < ‘B’) && (‘a’ < ‘c’)	Doğru, 1	(‘A’ < ‘B’) ifadesinin sonucu doğru yani 1’dır. Aynı şekilde (‘a’ < ‘c’) ifadesinin sonucu da birdir. Dolayısı ile 1 VE 1 işleminin sonucu 1 olur.
(!(‘3’ > ‘#’)) && (‘2’ < ‘3’)	Yanlış, 0	(!(‘3’ > ‘#’)) ifadesinin sonucu yanlış yani 0’dır. (‘2’ < ‘3’) ifadesinin sonucu doğru yani 1’dır. 0 VE 1 işleminin sonucu 0 olur

Aşağıdaki tabloda “**veya – or**” işleminin sonuçları gösterilmektedir. “**Veya – or**” işlemi iki işlenen ile işlem yapmaktadır. Her iki ifadenin sonuçları ile “**veya – or**” işlem yaparak sonuç döndürmektedir. “**veya – or**” mantıksal operatörü her iki ifadenin veya ifadelerden en az birini sonucunun 1 olması durumunda 1 sonucunu döndürür, her iki ifadenin sonucu 0 olduğu durumlarda sonuç daima sıfırdır.

İfade	İfade	İfade    ifade
True – Doğru (1)	True – Doğru (1)	True – Doğru (1)
True – Doğru (1)	False – Yanlış (0)	False – Yanlış (1)
False – Yanlış (0)	True – Doğru (1)	False – Yanlış (1)
False – Yanlış (0)	False – Yanlış (0)	False – Yanlış (0)

Aşağıda “**veya – or**” operatörü ile gerçekleştirilen mantıksal işlemlere örnekler verilmiştir.

İfade	Sonuç	Açıklama
(‘A’ < ‘B’)    (‘a’ < ‘c’)	Doğru, 1	(‘A’ < ‘B’) ifadesinin sonucu doğru yani 1’dır. Aynı şekilde (‘a’ < ‘c’) ifadesinin sonucu da birdir. Dolayısı ile 1 VEYA 1 işleminin sonucu 1 olur.
(!(‘3’ > ‘#’))    (‘2’ < ‘3’)	Doğru, 1	(!(‘3’ > ‘#’)) ifadesinin sonucu yanlış yani 0’dır. (‘2’ < ‘3’) ifadesinin sonucu doğru yani 1’dır. 0 VEYA 1 işleminin sonucu 0 olur
(‘A’ > ‘B’)    (‘a’ > ‘c’)	Yanlış, 0	(‘A’ > ‘B’) ifadesinin sonucu yanlış yani 0’dır. Aynı şekilde (‘a’ > ‘c’) ifadesinin sonucu da 0’dır. Dolayısı ile 0 VEYA 0 işleminin sonucu 0 olur.

### 3.1.8. Mantıksal Operatörlerde İşlem Öncelikleri

Mantıksal operatörlerin de aritmetik operatörlerde olduğu gibi işlem öncelikleri bulunur. Operatörler arasındaki öncelik belirlemek işlemlerin doğru sıra ile yapılarak doğru sonucun lede edilmesi içindir. Mantıksal operatörlerde de önceki bölümün başında yer verilen örnek mantıksal bildirimde olduğu gibi birde çok sayıda operatörün birlikte kullanılması ile oluşturulan bileşik bildirimler veya bileşik ifadelerin sonuçları beklentiği gibi olmayabilir. Burada bir hata durumu söz konusu değildir. Çünkü mantıksal operatörler arasındaki işlem önceliklerine göre işlemler doğru sırada işlenerek doğru sonuç bulunmuştur. Aşağıdaki tabloda görülen çeşitli operatör işlem öncelikleri karmaşık ifadeler ve karmaşık bildirimlerde işlemlerin doğru sırada okunmasını ve sonuçların bulunmasını kolaylaştıracaktır.

Operatör	Öncelik Sırası
!, + (pozitife çevir) - (negatif çevir)	Birinci
*, /, %	İkinci
+, -	Üçüncü
<, <=, >=, >	Dördüncü
= =, ! =	Beşinci
&&	Altıncı
	Yedinci
= (atama operatörü)	Sekizinci

Burada gösterilen mantıksal ve ilişkisel operatörlerin işlem öncelikleri aynı bileşik ifadede kullanılmaları durumunda soldan sağa doğru işlenmesi kuralına uymaktadır.

Aşağıdaki örneklerde mantıksal ve ilişkisel operatörlerin değişkenler ile birlikte basitten başlayarak karmaşağa doğru giderek mantıksal ve ilişkisel operatörlerin sonuçları gösterilemektedir.

İfade	Sonuç ve açıklama
!girdi	Sıfır – yanlış. Girdi adlı değişkenin değeri ne olursa olsun mantıksal olarak doğru – 1 sonucunu verir. Değil ile işlendiinde ise sonuç yanlış – 0 olur.
sure > 45.00	Değişkenin değeri 45.00'dan büyük olduğu sürece sonuç doğru – 1 olur. Aksi durumda yanlış – 0 olur.
!yas	Değişkenin değeri sıfırdan farklı herhangi bir veri ise mantıksal olarak değeri doğru – 1 olur. Ancak Değil operatörü bunu yanlış – 0 olarak değiştirir.
!girdi && (yas >= 18)	Sol taraftaki ifadenin değeri yanlış – 0 olduğundan sağ taraftaki ifadenin değeri her ne olursa olsun sonuç yanlış – 0 olacaktır.
mesai + fazla_mesai <= 75.00	Sol taraftaki ifadede yer alan mesai değişkenin değeri ile fazla-mesai_değeri değişkenin değerinin toplamının belirtilen ilişkisel operatördeki değer olan 75.00'dan küçük

olduğu sürece doğru – 1 olur.

(sayac >=0) && (sayac <=100)

Her iki ifade için ortak değişken olan sayac adlı değişkenin değeri belirtilen 0 ile 100 kapalı aralığında olduğu sürece ifadenin sonucu doğru – 1 olarak döndürülür.

('A' <= karakter && karakter <= 'z')

İşlem önceliğine göre öncelikle ilişkisel operatörlerin işlenmesi gereklidir. Karakter adlı değişken sadece büyük veya küçük harflerden herhangi birisi olursa bu durumda her iki ifadenin sonucu doğru -1 olacaktır. Bu durumda doğru – 1 ve doğru – 1 ifadelerinin sonucu doğru – 1 olur.

Mantıksal ve ilişkisel operatörlerde işlem öncelikleri belirtildiği sırada yapılır. Ancak programın daha kolay olunması ve gerektiğinde bu işlemlerin önceliğinin değiştirilmesi gerekiyor ise programcı tarafından parantezler kullanılarak işlem süreci yeniden düzenlenebilir. Aşağıdaki ifade de bu durum gösterilmiştir.

5 > 3 || 2 < 8 && 8 >= 12

Yukarıdaki ifade nin sonucu doğru – 1 olacaktır. Ancak bu ifadeyi daha kolay okunabilir ve öncelikleri değiştirmek için aşağıdaki gibi düzenlenenebilir.

(5 > 3) || ( 2 < 8 && 8 >= 12 )

Yukarıdaki ifadenin de sonucu doğru – 1 olur.

(5 > 3 || 2 < 8) && 8 >= 12

Yukarıdaki ifadenin de sonucu doğru – 1 olur.

C++ Programlama Dili'nin yapısında bulunan bazı algoritmalar mantıksal işlemlerin yürütülmesini hızlandırır. Örneğin aşağıda verilen ifadelerin sonuçlarının elde edilmesi için tüm ifadenin işlenmesine gerek olmayacağıdır.

(x >= 3) && (x < y)

Yukarıdaki ifadenin sonucunun bulunması için “**ve – and**” operatörünün her iki tarafında bulunan iki işlenin de işlenerek sonuçlarının alınmasına gerek yoktur. Çünkü bildiğimiz üzere “**ve – and**” operatörünün işlem sonucunun “**doğru – true – 1**” olması için her iki işlenin de sonucunun “**doğru – true – 1**” gereklidir. Mantıksal operatörler soldan sağa doğru işlendikler için sol taraftaki ifadenin sonucunun “**yanlış – false – 0**” durumunda sağ taraftaki ifade işlenmez. Sonu olarak doğrudan “**yanlış – false – 0**” döndürülür.

(x >= 3) || (x < y)

Yukarıdaki ifadenin sonucunun bulunması için “**veya – or**” operatörünün her iki tarafında bulunan iki işlenin de işlenerek sonuçlarının alınmasına gerek yoktur. Çünkü bildiğimiz üzere “**veya – or**” operatörünün işlem sonucunun “**doğru – true – 1**” olması için her iki işlenden en az birinin sonucunun “**doğru – true – 1**” gereklidir. Mantıksal operatörler soldan sağa doğru işlendikler için sol taraftaki ifadenin sonucunun “**doğru – true – 1**” durumunda sağ taraftaki ifade işlenmez. Sonu olarak doğrudan “**true – doğru – 1**” döndürülür.

Aşağıdaki örnek program da mantıksal ve ilişkisel operatörler kara yapıları içerisinde kullanılmıştır. Bir kargo taşımacılığı yapan işletmenin kabul ettiği kargo paketlerinin boyut ve ağırlık açısından incelenerek taşıma ücretinin belirlenmesi gösterilmiştir.

**Problem:** Kargo işletmesi şubelerinden kabul edilecek olan kargolar için fiyat üç aşamalı olarak hesaplanmaktadır. Birinci aşama belirtilen taşıma uzaklığına (kısa, orta ve uzak) göre taban fiyat hesaplanmaktadır. İkinci aşamada ise hacim veya ağırlık hesabı yapılarak hangisi daha büyük ise ek fiyat buna göre hesaplanmaktadır. Her ikisi de birbirine eşit ise kg esas alınmalıdır. Üçüncü aşama ise ikinci aşamada hesapta esas alınan büyülüüğün 100'ü geçmesi durumunda 500TL daha ek ücret alınmaktadır. En son hesaplana fiyata ek olarak da %10 KDV eklenmektedir.

**Çözüm:** İlk olarak taşıma için seçilecek olan uzaklık kategorisi belirlenip bu kategoriden gelen taban ücret hesaplanacaktır. İkinci aşamada kargo ağırlığı girilecektir. Ayrıca kargo paketinin genişlik, derinlik ve yükseltik girilip aşağıda belirtilen formüle göre desı değeri hesaplanacaktır. Bir desı/kg için taban fiyat 20TL olup her ilave desı/kg için 2 TL ücret daha eklenecektir. Üçüncü aşamada eğer desı/kg 100'den büyük ise ek olarak 500TL daha fiyata eklenir. Son olarak toplam fiyata %10 KDV eklenecektir.

**Algoritma:** aşağıdaki gibi olacaktır.

- 1 Başla
- 2 Uzaklık girin.
  - 2.1 Kısa ise mesafe fiyatı 50TL olur
  - 2.2 Orta ise mesafe fiyatı 100TL olur
  - 2.3 Uzun ise mesafe fiyatı 200 TL olur.
- 3 Kargo paketinin desı veya kg göre hesaplanacağını belirleyin
  - 3.1 kargonun ağırlığını girin (kg)
  - 3.2 kargonun yükseklik, genişlik ve derinliği girin (cm)
  - 3.3 
$$ax^2+bx+c=0 \quad X \Rightarrow X \in \mathbb{Z} \quad x_{1,2} = \frac{-b \mp \sqrt{b^2 - 4ac}}{4a}$$
- 4 kargonun desı veya kg hangisi büyük olduğunu belirleyin.
  - 4.1 Kg göre hesaplanacak ise
  - 4.2 desı göre hesaplanacak ise
  - 4.3 desı ve kg birbirine eşit ise kg esas alınır.
  - 4.4 desı veya kg 100 fazla ise taba fiyata 500 TL eklenir
- 5 Toplam fiyat mesafe ile taban fiyatın toplamıdır.
- 6 Tüketicije olan fiyat, taban fiyata %10 KDV eklenerek bildirilir.
- 7 Son

Yukarıdaki algoritmanın kaynak kod olarak uygulanmış hali aşağıdadır:

```
/*
 * kargo_hesapla.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
 * SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
 * VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
 * BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
 * HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
```

```
* SORUMLU DEĞİLLERDİR.  
*  
*/  
  
#include <iostream>  
  
#include <locale>  
  
#include <cfloat>  
  
#include <iomanip>  
  
using namespace std;  
  
int main()  
{  
    // Dil ayarları  
    setlocale(LC_ALL, "tr_TR.UTF-8");  
    // Değişken tanımlamaları  
    int derinlik =0, genislik=0, yukseklik=0, mesafe=0;  
    float desi=0.0, kg=0.0, fiyat=0.0, uzak_fiyat=0.0;  
    // Verilerin girilmesi  
    cout << "Taşıma mesafesinin türünü giriniz." << endl;  
    cout << "Kısa mesafe için 1" << endl;  
    cout << "Orta mesafe için 2" << endl;  
    cout << "Uzun mesafe için 3" << endl;  
    cout << "Seçiminiz: ";  
    cin >> mesafe;  
    // Ondalık basamak sayısı 2 olsun.  
    cout << fixed << setprecision(2);  
    // Hesaplamlar  
    // Uzaklık sınıfına göre taban fiyat hesabı  
    if ( mesafe == 1 )  
    {  
        uzak_fiyat = 50.0;  
    }  
    if ( mesafe == 2 )
```

```

{
    uzak_fiyat = 100.0;;
}

if ( mesafe == 3 )
{
    uzak_fiyat = 200.0;
}

// Kargo ağırlığının girilmesi

cout << "Kargonun ağırlığını (kg) olarak giriniz: ";
cin >> kg;

// Desi hesabı

cout << "Kargonun boyutlarını genişlik(cm), derinlik(cm) ve yükseltik(cm) olarak giriniz: ";
cin >> derinlik >> genislik >> yukseklik;

desi = (yukseklik * genislik * derinlik) / 3000;

// Karşılaştırma ve seçim

if (kg >= desi)
{
    fiyat = 20 + (kg * 2);
}

else
{
    fiyat = 20 + (desi *2);
}

if ((kg > 100) or (desi > 100))
{
    fiyat = 500 + fiyat;
}

fiyat = (fiyat + uzak_fiyat) * 1.1;

// Ücretlendirme

cout << "Toplam ücret: " << fiyat << " TL" << endl; ;

return 0;

```

}

Programın derlenip çalıştırılması ile aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./kargo_hesapla
```

Taşıma mesafesinin türünü giriniz.

Kısa mesafe için 1

Orta mesafe için 2

Uzun mesafe için 3

Seçiminiz: 1

Kargonun ağırlığını (kg) olarak giriniz: 1

Kargonun boyutlarını genişlik(cm), derinlik(cm) ve yükseltik(cm) olarak giriniz: 10 10 30

Toplam ücret: 79.20 TL

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

### 3.2.İlişkisel Operatörler ve Karakter Dizileri

İlişkisel operatörler sayısal veriler ile çalışırken sonuçları kolaylıkla ön görülebilirdir. Matematik eğitimi sayısal veriler ile çalışmayı kolaylaştırmaktadır. Aynı durum sözel verilerin ifade edildiği karakter veri tipleri ile de söz konusu olmakla birlikte, karakter dizilerinde görece olarak zor görünmektedir. Karakter dizileri üzerinde ilişkisel operatörler ile gerçekleştirilen işlemlerde ise her bir tekil karakter üzerinde gerçekleştirilir. İşlemler sol baştan başlayarak sağa doğru devam eder. Karşılaştırmalarda ASCII karakter tablosu esas alınır. Karşılaştırma işleminde belirleyici olan karakter sayısı az olan yani kısa karakter dizileridir. Böylece karşılaştırma işlemi kısa uzunluktaki karakter dizisinin sonuna gelindiğinde sona erecektir. Eğer her iki karakter dizisinin uzunluğu eşit ise bu durumda karşılaştırma ilk eşleşmeyen karaktere kadar devam eder ve bu eşleşmeyen karakterlerin karşılaşılmasından sonra sona erer. Eğer her iki karakter dizisinin elaman sayısı eşit ve her bir dizideki karakterler aynı ise işlem son karaktere kadar devam eder.

Aşağıda verilen örnek karakter dizileri üzerinde ilişkisel operatörler ile işlemler yapılmaktadır.

```
string karakterdizisi1 = "Merhaba";  
string karakterdizisi2 = "Merhamet";  
string karakterdizisi3 = "Merhem";  
string karakterdizisi4 = "Kabak";  
string karakterdizisi5 = "Küçük";
```

#### İfade

#### Sonuç ve açıklama

karakterdizisi1 < karakterdizisi2

İşlemin sonucu **true – doğru – 1** olur. Çünkü karakterdizisi1 değişkenindeki dizinin uzunluğu karakterdizisi2 değişkenin-

	deki dizinin uzunluğundan kısadır.
karakterdizisi1 > "Meram"	İşlemin sonucu <b>yansı - false - 0</b> olur. Çünkü her iki dizideki ilk üç karakter aynı iken, dördüncü karakter 'a' ve 'h' olduğundan 'a' < 'h'dir ve sonuç 0 döndürülür.
karakterdizisi2 < "Mehmet"	İşlemin sonucu <b>doğru - true - 1</b> olur. Çünkü her iki dizideki ilk iki karakter aynı iken, üçüncü karakter 'h' ve 'r' olduğundan 'h' < 'r'dir ve sonuç 1 döndürülür.
karakterdizisi2 > karakterdizisi3	İşlemin sonucu <b>yansı - false - 0</b> olur. Çünkü her iki dizideki ilk dört karakter aynı iken, beşinci karakter 'a' ve 'e' olduğundan 'a' < 'e'dir ve sonuç 0 döndürülür.
karakterdizisi4 < "Kelek"	İşlemin sonucu <b>doğru - true - 1</b> olur. Çünkü her iki dizideki ilk karakter aynı iken, ikinci karakter 'a' ve 'e' olduğundan 'a' < 'e'dir ve sonuç 1 döndürülür.
karakterdizisi5 < "Küçüktür"	İşlemin sonucu <b>doğru - true - 1</b> olur. Çünkü ilk değişkenin uzunluğu diğerinden kısadır. Sonuç 1 döndürülür.
karakterdizisi3 <= karakterdizisi4	İşlemin sonucu <b>doğru - true - 1</b> olur. Her iki değişkendeki karakter dizisinin ilk dört karakteri eşittir. Beşinci karakterler ise 'e' ve 'm' dir. 'e' <= 'm' koşulunu sağlar ve sonuç 1 döndürülür.

### 3.2.1. Bileşik Bildirimler ve İfadeler

Önceki bölümde yer verdiğimiz "if ..." ve "if ... then" ifadeleri sadece tek bir mantıksal veya ilişkisel koşulu kontrol ederek program akışını düzenler. Eğer ilişkisel veya mantıksal ifadenin sonucuna göre tek bir ifade işlenecek ise "if ..." ve "if ... else" ifadelerinin altında yer alana kod editör tarafından girintilerek yazılması sağlanır. Aşağıdaki örnek programda seçimler öncesi oy kullanabilecek olan vatandaşların belirlenmesi için kullanılan bir programın bir bölümü yer almaktadır.

```
if (yas >= 18)
    cout << "Oy kullanabilir." << endl;
else
    cout << "Oy kullanamaz." << endl;
```

Yukarıdaki örnek kod sadece tek bir çıktı döndürmektedir. Eğer sadece seçme hakkı değil de hem seçme hem de seçilme hakkı açısından kontrol eden bir program yazılmış olsa ide bu durumda aşağıdaki gibi düzenlenmesi gerekecektir.

```
if (yas >= 18)
{
    cout << "Oy kullanabilir." << endl;
    cout << "Aday olabilir." << endl;
```

```

    }

else

{

    cout << "Oy kullanamaz." << endl;

    cout << "Aday olamaz." << endl;

}

```

Yukarıdaki kod blokundan görüldüğü üzere birden çok sayıda ifadenin işlenmesi gerekiyor ise bu durumda “{ ... }” arasında kalacak şekilde ifadelerin yazılması gereklidir. Bu yazım şekli ile kaynak kodun okunabilirliği iyileşirken, koşulun gerçekleşme durumuna bağlı olarak işlenecek olan bildirimler de gruplanabilir.

### 3.2.2.Çoklu Seçimler

Önceki bölümde bir program akışının düzenlenmesinde belirli bir koşulun gerçekleşme durumuna göre bir seçenekin işleneceği veya işlenmeyeceği program ile benzer şekilde iki seçenekten birisinin seçilerek devam edileceği program örneklerine yer verilmiştir. Bazı durumlarda ise birden bir koşulun gerçekleşme durumunda göre program akışının farklı seçimler üzerinden ilerlemesi gerekebilir. (Kargo ücretinin hesaplandığı örnek programda olduğu gibi) Bu durumda ayrık “**if ...**” ifadelerin kullanılması en basit yaklaşım olmakla birlikte her bir seçenek için ilgili koşulun yinelerek kontrol edilmesi gerekecektir. Programın akışında aynı mantıksal ifadenin bir çok kez işlenmesi gereksiz yere programın uzamasına ve bazen de hataların yapılmasına neden olabilir. Bu olumsuzlukların önüne geçilmesi için iç içe geçen “**if ... else**” ifadeleri kullanılır. Böylelikle bir koşulun gerçekleşmesi durumunda sadece ilgili kod bloku işlenir ve diğer seçenekler için koşulun gerçekleşme kontrolü tekrarlanmaz.

Aşağıda bir bankada açılacak olan vadeli mevduat için vade sonunda elde edilecek olan brüt getiri ile vade sonunda elde edilecek olan net getiri hesaplayan bir programda kullanılmak üzere farklı ana para ve vade miktarları için faiz ve vergi kesintisi hesaplayan bölümler yazılmıştır.

```

if (anapara > 500000)

    faiz_oran = 0.20;

else

    if (anapara > 250000)

        faiz_oran = 0.15;

    else

        if (anapara > 100000)

            faiz_oran = 0.10;

        else

            if (anapara > 50000)

```

```
faiz_oran = 0.05;
```

```
else
```

```
faiz_oran= 0.02;
```

Bir “if ... else” yapısında dikkat edilmesi gereken en önemli konu bir “if” ile hangi “else” ilişkilidir. Çünkü C++ Programlama Dili’nde “else” bağımsız değildir ve mutlaka bir “if” ile ilişkili olmak durumundadır. Yukarıdaki kod blokundan da görüleceği üzere “else” bir üst satırda bulunan ilk “if” ile eşleştirilir. Yukarıdaki örnek programda bu eşleştirmenin daha kolay görülebilmesi için girintileme kullanılmıştır. Ancak girintilemenin bu şekilde kullanılması karmaşık ifadelerde zorluklar yaratmaktadır. Bunun yerine aynı kod bloku aşağıdaki gibi yazılabilir.

```
if (anapara > 500000)
    faiz_oran = 0.20;
else if (anapara > 250000)
    faiz_oran = 0.15;
else if (anapara > 100000)
    faiz_oran = 0.10;
else if (anapara > 50000)
    faiz_oran = 0.05;
else
    faiz_oran= 0.02;
```

Yukarıda gösterilen iki kod bloğu da aynı işlemi yapmaktadır. Aralarında hız açısından herhangi bir fark bulunmamaktadır. Aşağıdaki kod örnekleri aynı işlemi yerine getirmektedir.

```
if (hafta_gun = 1 )
    cout << "Pazartesi" << endl;
if (hafta_gun = 2 )
    cout << "Sali" << endl;
if (hafta_gun = 3 )
    cout << "Çarşamba" << endl;
if (hafta_gun = 4 )
    cout << "Perşembe" << endl;
if (hafta_gun = 5 )
    cout << "Cuma" << endl;
if (hafta_gun = 6 )
```

```
cout << "Cumartesi" << endl;  
if (hafta_gun = 7 )  
    cout << "Pazar" << endl;
```

Yukarıdaki kod aşağıdaki tekrar yazılabilir.

```
if (hafta_gun = 1 )  
    cout << "Pazartesi" << endl;  
else if (hafta_gun = 2 )  
    cout << "Salı" << endl;  
else if (hafta_gun = 3 )  
    cout << "Çarşamba" << endl;  
else if (hafta_gun = 4 )  
    cout << "Perşembe" << endl;  
else if (hafta_gun = 5 )  
    cout << "Cuma" << endl;  
else if (hafta_gun = 6 )  
    cout << "Cumartesi" << endl;  
else if (hafta_gun = 7 )  
    cout << "Pazar" << endl;
```

Yukarıda yer verilen ikinci kod ise aynı program içerisinde sadece ilgili bölümlerin değiştirilerek kullanılması durumunda ilkine göre daha hızlı çalışmaktadır.

### 3.2.3.Kayar Noktalı Sayılarda Eşitlik Kontrolü

Kayar noktalı sayıların bilgisayarda gösterimi ve üzerinde yapılacak işlemlerin nasıl gerçekleştirileceği IEEE 754 standardında belirtilmektedir. Bu standarda göre bir kayar noktalı sayı üzerinde işlemler yapılabilir ve sonuçları görüntülenebilir. Ancak işlemcinin işlem kapasitesinin sınırlılığı nedeni ile matematikçilerin günlük yaşamda karşı karşıya kaldıkları ondalık basamakların hesaplanması ve yazılması sorunu programının da karşısına gelecektir. Bu tür durumlarda yazılabilecek olan ondalık basamak sayısının sınırlı olmasından dolayı işlemlerde hata (**data corruption**) oluşacaktır. Oluşan hatanın belirli sınırlar içerisinde kalması durumunda işlemin geçerli olarak kabul edilebilmesinden dolayı da yapılan işlemlerde kabul edilen hata sınırının yani toleransın başından belirlenmesi gereklidir. Aşağıda görülen programda kayar noktalı sayılar ile yapılan aritmetik işlemlerin sonucunda ondalık basamak sayısında gerçekleşen hatanın sınırı önceden belirlendiği için sonuçlar buna göre değerlendirilmektedir.

```
/*  
* tolerans.cxx
```

\*

\*

\* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

```

*/
#include <iostream>
#include <locale>
#include <iomanip>
#include <cfloat>
#include <cmath>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişken tanımlamaları
    double x = 1.0;
    double y = 3.0 / 9.0 + 2.0 / 9.0 + 4.0 / 9.0;
    cout << fixed << showpoint << setprecision(17);
    // Hesaplama ve sonucun yazdırılması
    cout << "4.0 / 9.0 + 3.0 / 9.0 + 2.0 / 9.0 = " << 4.0 / 9.0 + 3.0 / 9.0 + 2.0 / 9.0 << endl;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    // Karşılaştırma
    if (x == y)
        cout << "x ve y eşittir." << endl;
    else
        cout << "x ve y eşit değildir." << endl;
    if (fabs(x - y) < 0.000001)
        cout << "x ve y 0.000001 tolerans ile eşittir." << endl;
    else
        cout << "x ve y arasındaki fark 0.000001 toleranstan büyüktür." << endl;
    return 0;
}

```

Programın derlenip çalıştırılması ile aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./tolerans
4.0 / 9.0 + 3.0 / 9.0 + 2.0 / 9.0 = 0.9999999999999989
x = 1.0000000000000000
y = 1.0000000000000000
x ve y eşittir.
x ve y 0.000001 tolerans ile eşittir.
[goksin@tardis ~/projeler/C++_Notlar]$
```

Programın sonucundan görüleceği üzere kayar noktalı sayılar ile yapılan işlemler de sayılar arasındaki fark önceden belirlenmiş olan hata sınırı içerisinde kaldığı sürece işlemler doğru kabul edilir. Programda kayar noktalı sayılar ile gerçekleştirilen bazı işlemler için geliştirilmiş olan **cmath** kütüphanesi kullanılmış ve böylece iki kayar noktalı sayının arasındaki farkın mutlak değeri hesaplanabilmiştir. Ayrıca derleyici seçeneklerinden de görüleceği üzere kayar noktalı sayılar üzerinde gerçekleştirilecek olan aritmetik eşitlik işlemleri için de derleyici uyarı mesajı döndürmekte ve programcının dikkati çekilmektedir. Program uyarı mesajına karşılık derlenebilmektedir.

### 3.2.4. Girdi Hatalarının if ile Tespit edilmesi

Önceki bölümde girdi hatalarının programın işleyişinde girdi işlemlerinde hatalı veri girişlerinin girdi akışını çalışmaduruma getirdiğini görmüştük. Bu durumda programın akışının kesintiye uğramamış olmasına karşılık devam ederken hatalı veriler üzerinde işlem yaparak yanlış sonuçlar döndürebileceğini görmüştük. Bu olumsuz durumdan kaçınmak için veri girişinin tekrar başlatılması yanında hatanın tespit edilip bununla ilgili hata mesajının döndürülmesi, program akışının sonlandırılması gibi seçeneklerde söz konusudur. Hatalı veri girişlerinin tespit edilerek hata mesajı döndürülmesi ve programın sonlandırılması kolaylıkla uygulanabilecek olan bir seçenekdir. Bunun için “if ... ” kara yapısı ile girdi operatörü olan “cin” birlikte kullanılabilir. Benzer şekilde dosyadan okuma yapılrken kaynak dosyanın bulunaması durumunda programın hata mesajı döndürüp sonlanması sağlanabilir.

Bu işlemlerin sonucunda programın sonlanması için “**return 0**” kullanılır. Bu yönerge, programın hatasız sonlandığının ve işletim sistemine dönüldüğünü belirtir. Girişlerde hata olması durumunda ise programın sonlanması için gereken yerde yine “**return**” kullanılabilir. İşletim sisteminin desteklemesi durumunda ise “**return**” ile programın neden sonlandığının bilgisi iletilebilir. Programcının önceden belirlediği hata durumları için hazırladığı hata kodları, N sayı olmak üzere “**return N**” şeklinde işletim sistemine iletilebilir. \*BSD işletim sistemlerinde ve Linux dağıtımlarında kullanıcının kullandığı etkileşimli ortamda “\$?” ile programın işletim sistemine döndürdüğü hata kodu bulunuyor ise bu öğrenilebilir. Eğer program hata kodu döndürmeden sadece sonlanması istenirse bu durumda önceki örneklerde olduğu gibi “**return 0**” ile program sonlandırılabilir. İşletim sistemi açısından bu durum programın başarı ile sonlandığı anlamına gelir.

Eğer program Microsoft'un ticari işletim sistemleri üzerinde çalışıyor ise bu durumda programın sonlanması durumunu bir hata mesajı ile kullanıcıya bildirmek ve hata kodu döndürmeden de doğrudan programın sonlandırılması sağlanabilir.

Aşağıdaki program önceki bölümde incelenen öğrenci başarı notunu belirlemek için kullanılan programın iki olası hata durumunu (okunacak olan dosyanın bulunamaması durumu ve dosyada geçersiz verilerin bulunması durumunu dikkate almaktadır. Belirtilen hata durumları ile karşılaşmasında uyarı mesajını yazdırıp ilgili hata kodunu işletim sistemine de döndürmektedir.

```
/*
 * not_hesapla2.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
```

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
#include <locale>
#include <fstream>
#include <string>
#include <cfloat>
#include <iomanip>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Dosya okuma ve yazma işlemleri için değişkenlerin tanımlanması
    ifstream dosyadan_oku;
    ofstream dosyaya_yaz;
    // Öğrencinin adı ve soyadı
    string ad, soyad;
    // Değişken tanımlamaları: Öğrencinin notları ve sayısal değişkenler etkinleştirilsin.
    float sinav1=0.0, sinav2=0.0, odev1=0.0, odev2=0.0, donemsonu=0.0, ortalama=0.0;
    float k1=0.15, k2=0.15, k3=0.10, k4=0.10, k5=0.50;
    // Sayısal veri hata kontrol değişkeni
    bool hataliveri = false;
    // Okunacak olan dosyanın kontrolü
    dosyadan_oku.open("notlar1.txt");
```

```

if (!dosyadan_oku)
{
    cout << "Kaynak dosya olan \"notlar1.txt\" bulanamadi." << endl;
    cout << "Program sonlanıyor. Hata kodu: 1" << endl;
    return 1;
}

dosyaya_yaz.open("karne1.txt");
// Çıktının biçimlendirilmesi
dosyaya_yaz << fixed << showpoint << setprecision(2);
dosyadan_oku >> ad >> soyad;
dosyaya_yaz << "Öğrencinin adı, soyadı: " << ad << " " << soyad << endl;
dosyadan_oku >> sinav1 >> sinav2 >> odev1 >> odev2 >> donemsonu;
// Dosyadaki notların geçerliliği kontrol ediliyor.
if (hataliveri == ((sinav1 < 0.0 or sinav1 >100.0) && (sinav2 < 0.0 or sinav2 >100.0) && (odev1 < 0.0 or odev1 >100.0) && (odev2 < 0.0 or odev2 >100.0) && (donemsonu < 0.0 or donemsonu >100.0)))
{
    cout << "Kaynak dosyada hatalı veri girişi bulundu." << endl;
    cout << "Program sonlanıyor. Hata kodu: 2" << endl;
    return 2;
}
else
{
    dosyaya_yaz << "Dönem içi değerlendirme sonuçları: " << setw(6) << sinav1 << setw(6) << sinav2 << setw(6) << odev1 << setw(6) << odev2 << setw(6) << donemsonu << endl;
    ortalama = (k1 * sinav1) + (k2 * sinav2) + (k3 * odev1) + (k4 * odev2) + (k5 * donemsonu);
    dosyaya_yaz << "Dönem sonu başarı notu: " << setw(6) << ortalama << endl;
    cout << "Veriler dosyadan okunuyor ve işleniyor..." << endl;
    dosyadan_oku.close();
    dosyaya_yaz.close();
}

```

```
    return 0;  
}
```

Programın derlenip ilk çalıştırıldığında dosyanın bulunamaması nedeni programın sonlanması test edilmiştir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./not_hesapla1
```

Kaynak dosya olan "notlar1.txt" bulanamadı.

Program sonlanıyor. Hata kodu: 1

```
[goksin@tardis ~/projeler/C++_Notlar]$ $? 
```

bash: 1: komut yok

```
[goksin@tardis ~/projeler/C++_Notlar]$ $? 
```

Programın ikinci çalıştırılmasında öğrenin notlarından en az bir tanesi 100'den büyük olarak yazılabilir veya notlardan bir tanesi negatif olarak yazılabilir. Programın test edilmesinde kullanılan notlar1.txt dosyasının içeriği aşağıdaki gibidir. İlk örnek negatif sayıları barındırmaktadır.

```
Ahmet Selim 100.01 76.50 65.0 86.50 77.75
```

Aşağıdaki ikinci örnek dosya içeriğinde yüzden büyük pozitif sayı yer almaktadır.

```
Ahmet Selim 100.0 76.50 65.0 86.50 -77.75
```

Son örnekte karakter veya karakter dizisi olan verilerin kontrolü için kullanılmaktadır.

```
Ahmet Selim 100.0 76.50 65.0 86.50 ab}\`
```

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./not_hesapla1
```

Kaynak dosyada hatalı veri girişi bulundu.

Program sonlanıyor. Hata kodu: 2

```
[goksin@tardis ~/projeler/C++_Notlar]$ $? 
```

bash: 2: komut yok

```
[goksin@tardis ~/projeler/C++_Notlar]$ $? 
```

Yukarıda verilen program dosyadan okuma yaparak değişkenlere atayarak işlem yapmaktadır. Önceki bölümlerde incelediğimiz üzere program herhangi bir kaynaktan girdiyi okurken geçersiz veriler programın çalışmasını durdurmayıp sadece hatalı verilerden kaynaklı olarak programın yanlış sonuçlar döndürmekte olduğunu görmüştük. Burada programın olası yanlış verileri işleyerek hatlı sonuç döndürmesinin önüne geçilmesi için yukarıdaki programda tüm veri giriş yapıldıktan sonra işlem öncesi hata kontrolü yapılmakta idi. Program hatalı veriyi bulduğunda tek bir hata mesajı döndürüp sona eriyor. Bu durum çok fazla veri üzerinde işlem yapılması durumunda hatalı veri kaynağının bulunması için uygun olmayacağındır. Bu durumun iyileştirilmesi olanağlıdır. Öncelikle C++ Programlama Dili'nin veri girişi ve çıkış ile ilgili işleyişini dikkat alınmalıdır. Veri girişi sırasında bir hata olması durumunda okuma işlemi devam etmektedir. Programın hatalı veri ile işlem yaparak sonuç döndürmemesi için programın okuma işlemi sırasında

ortaya çıkan hataları yakalaması ve buna göre işlemesi daha uygun olacaktır. Aşağıdaki program girdinin okunması sırasında ilk hata oluştuğu zaman işlemi sonlandırmaktadır.

Bir önceki bölümde yer verilen başarı notu hesaplayan programda iyileştirmeler yapılmıştır. Programın gerçekleştirmesi istenen işlemler şunlardır:

- a) Program dosyadan öğrencinin adı, soyadı, birinci ve ikinci ara sınav notları ile birinci ve ikinci ödev notlarını okuyacaktır.
- b) Program okuduğu notları kat sayılar ile çarparak dönem sonu başarı notunu hesaplayacaktır.
- c) Hesaplanan başarı notunun karşılık geldiği harf not bulunacaktır.
- d) Öğrencini adı ve soyadı ile dönem içi notları ve dönem sonu başarı notu ile harf notu dosyaya yazdırılacaktır.

Programın algoritması aşağıda verilmiştir.

1. Başla
2. Bilgilerin okunacağı dosyanın dizinde bulunup bulunmadığını kontrol et. Dosya dizinde bulunuyorsa “4.” adıma geç, dosya dizinde bulunmuyorsa dosyanın bulunmadığını belirten hata mesajını yaz ve programı sonlandır.
3. Dosyanın geçerli verilere sahip olduğunu kontrol et. Veriler geçersiz ise hatalı veriyi belirtip programı sonlandır.
4. Başarı notunu şu formüle göre hesapla: “ortalama = (k1 \* sinav1) + (k2 \* sinav2) + (k3 \* odev1) + (k4 \* odev2) + (k5 \* donemsonu)”
5. ortalama 84 ve üzeri ise harf notu AA olarak belirle.
6. ortalama 77 ve üzeri ise harf not AB olarak belirle.
7. ortalama 71 ve üzeri ise harf not BA olarak belirle.
8. ortalama 66 ve üzeri ise harf not BB olarak belirle.
9. ortalama 61 ve üzeri ise harf not BC olarak belirle.
10. ortalama 56 ve üzeri ise harf not CB olarak belirle.
11. ortalama 50 ve üzeri ise harf not CC olarak belirle.
12. ortalama 46 ve üzeri ise harf not CD olarak belirle.
13. ortalama 40 ve üzeri ise harf not DC olarak belirle.
14. ortalama 33 ve üzeri ise harf not DD olarak belirle.
15. ortalama 33'ten az ise FF olarak belirle.
16. Sonuçları dosyaya yaz.
17. Son

Yukarıda verilen algoritmanın kaynak kod olarak karşılığı aşağıdadır.

```
/*
 * harf_notu_hesapla.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
 * SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
 * VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
 * BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
 * HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
```

```
* SORUMLU DEĞİLLERDİR.  
*  
*/  
/*  
  
#include <iostream>  
  
#include <locale>  
  
#include <fstream>  
  
#include <string>  
  
#include <cfloat>  
  
#include <iomanip>  
  
using namespace std;  
  
int main()  
{  
    // Dil ayarları  
    setlocale(LC_ALL, "tr_TR.UTF-8");  
    // Dosya okuma ve yazma işlemleri için değişkenlerin tanımlanması  
    ifstream oku;  
    ofstream yaz;  
    // Öğrencinin adı ve soyadı  
    string ad, soyad, harf_not;  
    // Değişken tanımlamaları: Öğrencinin notları ve sınav oranları tanımlanıp etinleştirilsin.  
    float sinav1=0.0, sinav2=0.0, odev1=0.0, odev2=0.0, donemsonu=0.0, ortalama=0.0;  
    float k1=0.15, k2=0.15, k3=0.10, k4=0.10, k5=0.50;  
    // Okunacak olan dosyanın kontrolü yapılsın.  
    oku.open("notlar.txt");  
    if (!oku)  
    {  
        cout << "Kaynak dosya olan \"notlar1.txt\" bulanamadı." << endl;  
        cout << "Program sonlanıyor. Hata kodu: 1" << endl;  
        return 1;  
    }  
}
```

```
// Dosyadan veriler okunuyor.

cout << "Veriler dosyadan okunuyor." << endl;
oku >> ad >> soyad;
if (!oku)
{
    cout << "Öğrenci adı ve/veya soyadı hatalı girilmiştir. İşlem sonlandırıldı." << endl;
    cout << "Hata kodu 2" << endl;
    oku.close();
    return 2;
}
oku >> sinav1;
if ((!oku) || ((sinav1 < 0.0) || (sinav1 > 100.0)))
{
    cout << "Öğrencinin birinci sınav notu hatalı girilmiştir. İşlem sonlandırıldı." <<
endl;
    cout << "Hata kodu 3" << endl;
    oku.close();
    return 3;
}
oku >> sinav2;
if ((!oku) || ((sinav2 < 0.0) || (sinav2 > 100.0)))
{
    cout << "Öğrencinin ikinci sınav notu hatalı girilmiştir. İşlem sonlandırıldı." << endl;
    cout << "Hata kodu 3" << endl;
    oku.close();
    return 3;
}
oku >> odev1;
if ((!oku) || ((odev1 < 0.0) || (odev1 > 100.0)))
{
```

```

cout << "Öğrencinin birinci ödev notu hatalı girilmiştir. İşlem sonlandırıldı." <<
endl;

cout << "Hata kodu 3" << endl;
oku.close();
return 3;
}

oku >> odev2;
if ((!oku) || ((odev2 < 0.0) || (odev2 > 100.0)))
{
    cout << "Öğrencinin ikinci sınav notu hatalı girilmiştir. İşlem sonlandırıldı." << endl;
    cout << "Hata kodu 3" << endl;
    oku.close();
    return 3;
}

oku >> donemsonu;
if ((!oku) || ((donemsonu < 0.0) || (donemsonu > 100.0)))
{
    cout << "Öğrencinin dönem sonu sınav notu hatalı girilmiştir. İşlem sonlandırıldı."
<< endl;
    cout << "Hata kodu 3" << endl;
    oku.close();
    return 3;
}

// Veriler okundu ve okuma işlemi sona erdi.

oku.close();
cout << "Veriler üzerinde işlem yapılıyor." << endl;
ortalama = (k1 * sinav1) + (k2 * sinav2) + (k3 * odev1) + (k4 * odev2) + (k5 * donemsonu);
// Harf notunun hesaplanması
if (ortalama>=84.0)
    harf_not = "AA";
else if(ortalama>=77.0)

```

```

        harf_not = "AB";
    else if(ortalama>=71.0)
        harf_not = "BA";
    else if(ortalama>=66.0)
        harf_not = "BB";
    else if(ortalama>=61.0)
        harf_not = "BC";
    else if(ortalama>=56.0)
        harf_not = "CB";
    else if(ortalama>=50.0)
        harf_not = "CC";
    else if(ortalama>=46.0)
        harf_not = "CD";
    else if(ortalama>=40.0)
        harf_not = "DC";
    else if (ortalama>=33.0)
        harf_not = "DD";
    else harf_not = "FF";
// Yazılacak olan dosya açılır, yoksa işletim sistemi oluşturur.
yaz.open("karne.txt");
// Çıktının biçimlendirilmesi
yaz << fixed << showpoint << setprecision(2);
yaz << "Öğrencinin adı, soyadı: " << ad << " " << soyad << endl;
// Değerlendirme dosyaya yazılır.
yaz << "Dönem içi değerlendirme sonuçları: " << setw(6) << sinav1 << setw(6) << sinav2
<< setw(6) << odev1 << setw(6) << odev2 << setw(6) << donemsonu << endl;
yaz << "Dönem sonu başarı notu: " << setw(6) << ortalama << endl;
yaz << "Dönem sonu harf notu: " << setw(4) << harf_not << endl;
// Dosya yazma sonlanıyor
yaz.close();
return 0;

```

}

Programın derlenip çalıştırıldığında çalıştırıldığı dizinde gerekli olan dosyayı kontrol etmektedir. Dosya bulunuyorsa işleme devam etmekte bulunmuyorsa hata mesajı döndürüp işlemi sonlandırmaktadır. Dosyadan okunan veriler sırası ile geçerli olup olmadıkları kontrol edilmekte ve ardından geçerlik durumuna göre program akışı devam etmektedir. Veriler geçerli değilse program hata mesajı döndürüp sonlanmaktadır. Geçerli verilerin olması durumunda ise başarı notu hesaplanıp, ardından harf not karşılığı bulunup dosyaya yazılmaktadır. Dosyaya yazma işleminin ardından program sonlanmaktadır.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./harf_not_hesapla1
```

Veriler dosyadan okunuyor.

Veriler üzerinde işlem yapılıyor.

```
[goksin@tardis ~/projeler/C++_Notlar]$ $?
```

bash: 0: komut yok

```
[goksin@tardis ~/projeler/C++_Notlar]$ cat karne.txt
```

Öğrencinin adı, soyadı: Ahmet Selim

Dönem içi değerlendirme sonuçları: 77.00 98.00 76.00 56.00 98.00

Dönem sonu başarı notu: 88.45

Dönem sonu harf notu: AA

### 3.3.Switch ... Case ... Yapısı

C++ Programlama Dili’nde program akışını kontrol etmek için kullanılan yapılar bilgisayar programlama terminolojisinde “**dallanma – branching**” olarak adlandırılır. Bu yapılar “if ... ” ve “if ... else ... ” şeklinde kullanılarak ilişkisel ve mantıksal operatörler ile program akışının kontrol edilmesi yanında değişkenlere atamalar yapılması için de kullanılabilir. Son kullanım şekli için söz konusu olabilecek çeşitli kullanım şekilleri arasında bir değişkenin belirli koşulların gerçekleşme durumuna alabileceği değerlerin veya programın belirli koşulların gerçekleşmesi durumunda işleyişinin buna göre sürdürülmesi içinde kullanılabilir. Bu işlemler için “if ... else ... ” yapısının kullanılması yerine “**switch ... case ...** ” yapısı tercih edilir. “**switch ... case ...** ” yapısının biçimini aşağıdaki gibidir.

```
switch (ifade)
{
    case değer:
        ifade;
        break;
    case değer:
```

```
        ifade;  
        break;  
  
    case değer:  
        ifade;  
        break;  
  
    case değer:  
        ifade;  
        break;  
  
    ...  
  
    case değer:  
        ifade;  
        break;  
  
default:  
    ifade;
```

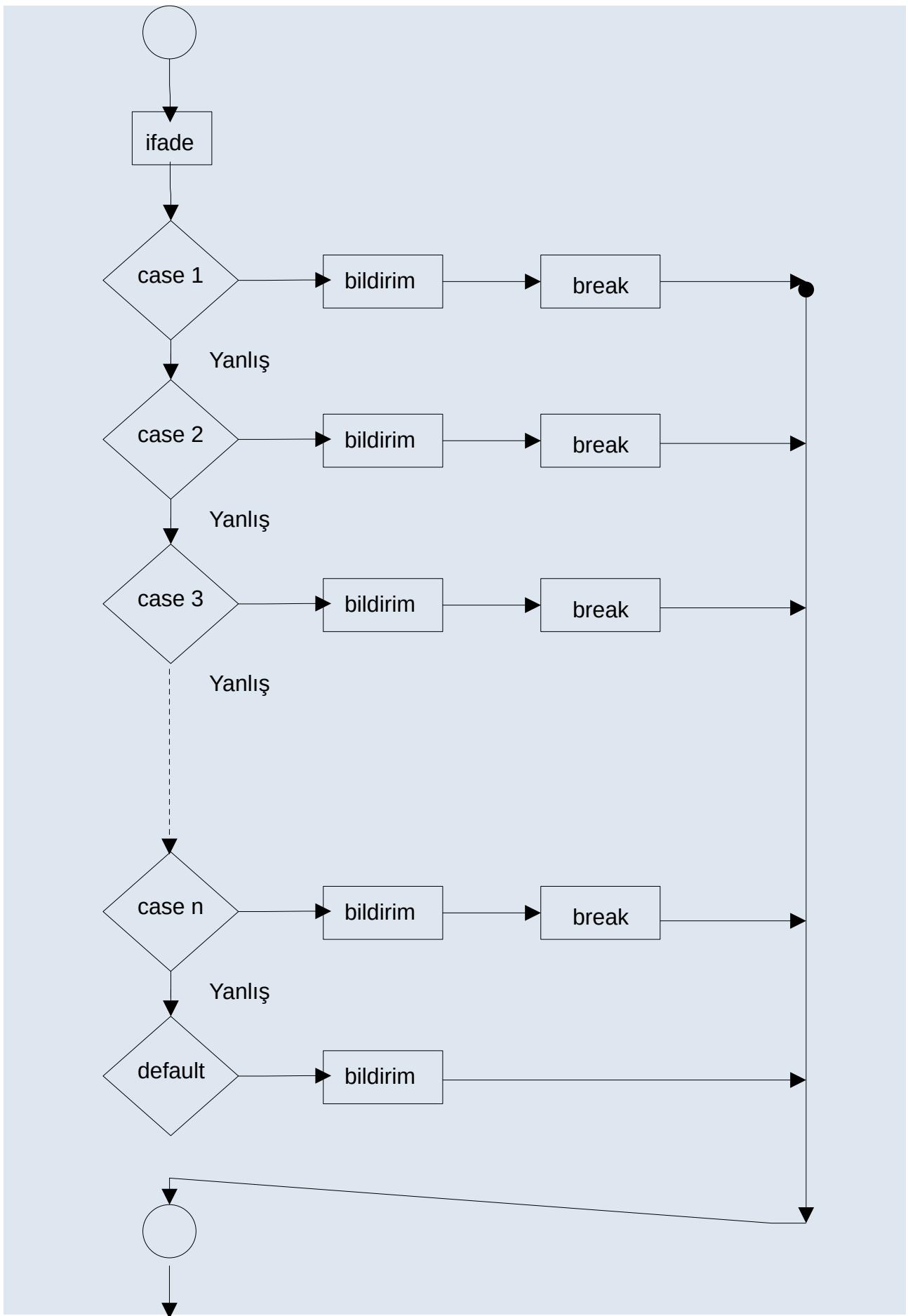
Bu yapının işleyişinde yapının başlangıcında bulunan switch (ifade) kodunun işlem sonucunun belirlenmesi gereklidir. Bu kodun değeri daima **tam sayıdır**. Bu değerin belirlenmesin ardından ilgili “**case**” alanına program geçiş yapar ve buradan işlemeye devam eder. Başlangıçtaki ifadenin değeri tam sayı olduğundan case ile kullanılan değerler C++ Programala Dili’nde bulunan temel veri tiplerinden tam sayı veya karakter olmak durumundadır. İlgili değer karşılık gelen case blokunda ifadeler işlenir. İşlem bittiğinde ise bloğun sonunda yer alan “**break**” yönergesi programın bu noktadan sonra “**switch ... case ...**” yapısını terk ederek programın geri kalan bölümüne geçilerek işleme devam edilmesini sağlar.

“switch – case” yapısının klasik algoritma ve programlama kitaplarında “if ... else ...” yapısı şeklinde gösterilmektedir. Bazı kaynaklarda ise yapıda yer alan break kullanılarak bu belirsizlik giderilmeye çalışmaktadır.

“switch ... case” yapısı aşağıdaki kurallara göre işler:

1. Yapının başında belirtilen ifadenin sonucu belirlendikten sonra eşleşen “case” kod blokuna geçilir. Buradaki kodlar “break” yönergesine gelinene kadar işlenir. “break” yönergesine gelindiğinde ise yapı terk edilir.
2. Yapının başında belirtilen ifadenin sonucu belirlendikten sonra eşleşen case kod blokuna bulunmuyor ise “default” blokuna geçilir. Burada yukarıdaki işlem yapılır. “break” yönergesine gelinene kadar işlenir. “break” yönergesine gelindiğinde ise yapı terk edilir.

Aşağıdaki örnek programda “switch ... case” yapısı kullanılmaktadır. Program 1 ile 5 arasındaki tam sayıları girdi olarak kabul etmektedir. Belirtilen kapalı aralık dışındaki değerler geçersiz olarak değerlendirilmektedir.



```
/*
 * test_say.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
 * SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
 * VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
 * BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
 * HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
```

```
* SORUMLU DEĞİLLERDİR.  
*  
*/  
/*  
  
#include <iostream>  
  
#include <locale>  
  
using namespace std;  
  
int main()  
{  
    // Dil ayarları  
    setlocale(LC_ALL, "tr_TR.UTF-8");  
  
    int girdi;  
  
    cout << "1 ile 5 arasında bir tam sayı giriniz: ";  
  
    cin >> girdi;  
  
    // switch case bölümü  
    switch (girdi)  
    {  
        case 1:  
            cout << "1 yazdınız." << endl;  
            break;  
        case 2:  
            cout << "2 yazdınız." << endl;  
            break;  
        case 3:  
            cout << "3 yazdınız." << endl;  
            break;  
        case 4:  
            cout << "4 yazdınız." << endl;  
            break;  
        case 5:  
            cout << "5 yazdınız." << endl;
```

```
        break;  
  
    default:  
  
        cout << "Geçersiz girdi." << endl;  
    }  
  
    return 0;  
}
```

Programın derlenip çalıştırıldığında klavyeden girilen sayılarla karşılık gelen kod blokundaki mesaj görüntülenmektedir. Belirtilen aralık dışında kalan değerler için “default” bölümünde programın sonlanmasıından önce kullanıcıyı uyaran mesaj yazılmıştır.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./test_say
```

```
1 ile 5 arasında bir sayı giriniz: 1
```

```
1 yazdınız.
```

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./test_say
```

```
1 ile 5 arasında bir sayı giriniz: 2
```

```
2 yazdınız.
```

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./test_say
```

```
1 ile 5 arasında bir sayı giriniz: 5
```

```
5 yazdınız.
```

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./test_say
```

```
1 ile 5 arasında bir sayı giriniz: A
```

```
Geçersiz girdi.
```

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

```
[goksin@tardis ~/projeler/C++_Notlar/2022-2023/Güz]$
```

### 3.4.Assert ile Programda Hata Yakalama

Bir programcı bir bilgisayar programın yazarken programın işleyişi sırasında olabilecek bir çok hataya karşı önceden önlem almak durumundadır. Bunu yapabilmenin ilk koşulu programın ne yapacağı, ne tür bir veri ile çalışacağını, verinin büyüklüğü ile programın üreteceği çıktıının hangi veri tipinde olacağını, çıktıının büyülüğünün ne kadar olacağını bilmektir. Eğer bu bilgiler olmadan sadece bir programın çözümü için önerilen çözüm yönteminin uygulanması sadece bir algoritma ve kaynak kod yazma çalışmasından başka bir anlama gelmeyecektir.

Program yazılırken her ne kadar probleme ilişkin tüm bilgiler ile verilerin derlenmesi için özenle ve dikkatle çalışılsa da eksiklikler olabilir. Bu eksikliklerin neden olabileceği çeşitli hataların da programın işleyişinde ortaya çıkması kaçınılmaz olacaktır. Olası hataların belirlenmesi için tüm

programlama dillerinde çeşitli araçlar bulunmaktadır. Buraya kadar C++ programlama dilinde mantıksal operatörler ile if ... else yapıları kullanılarak olası hataların program akışını aksatmaması için programın nasıl sonlandırılacağı incelendi. Bir diğer araç olan **assert** ise kullanıldığı yerdeki hata olduğunu belirten bir mesaj üretecek programı sonlandırmaktadır. Aşağıdaki örnekte ikinci dereceden bir polinomun reel köklerinin bulunmasını sağlayan bir program bulunmaktadır. Programın işleyişinin çözümlenmesinde diskriminantın negatif olması durumunda polinomun reel kökleri bulunmayacaktır. Hatanın olabileceği yer ise negatif sayının karekökünün hesaplanması olacaktır.

$$\sqrt{16} = 16^{0.5} = 16^{1/2}$$

Bir programda **assert** fonksiyonu kullanılacak ise önişlemci yönergelerinden “**#include <cassert>**” eklenmesi gereklidir. Ayrıca assert fonksiyonu kullanılırken fonksiyon içerisinde mantıksal ifade kullanılır. Mantıksal ifadenin değerlendirilmesi sonucunda eğer sonuç “doğru – true” ise programa devam edilir. Aksi durumda program hata mesajı döndürülerek sonlandırılır. Aşağıdaki program kodunda farklı noktalarda girdiden kaynaklı geçersiz işlemlerin olabileceği düşünülerek assert fonksiyonu diskriminatın hesabında “a” katasyısının ve diskriminat değerinin negatif olması durumlarını kontrol etmektedir.

```
/*
* ikinci_dereceden_polinom_kok_bul.cxx
*
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
*
*
* Her hakkı saklıdır.
*
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
* takdirde serbesttir:
*
*
* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
*
*
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
*
```

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
```

```
#include <locale>
```

```
#include <cmath>
```

```
#include <cassert>
```

```
#include <cfloat>
```

```
#include <iomanip>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    // Dil ayarları
```

```
    setlocale(LC_ALL, "tr_TR.UTF-8");
```

```
    // Değişken tanımlamaları: Öğrencinin notları ve sayısal değişkenler etkinleştirilsin.
```

```
    float kok1 = 0.0, kok2 = 0.0, a = 0.0, b = 0.0, c = 0.0, d = 0.0;
```

```
    bool kontrol = true;
```

```

cout << "Bu program ikinci dereceden bir polinomun reel köklerini bulmaktadır." << endl;
cout << "Polinom ax2 + bx +c biçimindedir. Sırası ile a, b ve c giriniz. ";
cin >> a >> b >> c;
if (a == 0)
{
    kontrol = false;
    assert (kontrol);
}

// Çıktının biçimlendirilmesi
cout << fixed << showpoint << setprecision(2);

// Diskriminant hesabı: 1. Adım
d = (pow(b,2.0) - (4* a *c));
if (d < 0.0)
{
    kontrol = false;
    assert(kontrol);
}
if (d > 0.0)
{
    d= pow(d,0.5);
    kok1 = (-b+d)/(4*a);
    kok2 = (-b-d)/(4*a);
    cout << "Birinci kök: " << kok1 << endl;
    cout << "İkinci kök: " << kok2 << endl;
    return 0;
}
if (d == 0.0)
{
    kok1 = (-b/(4*a));
    cout << "Çakışık kök vardır. Birinci kök ikinci köke eşittir. " << kok1 << endl;
}

```

```

    return 0;

}

if (d < 0.0)

    cout << "Buraya kadar gelmemeliydi! 0_o" << endl;

return 0;

}

```

Programın derlenip ilk çalıştırıldığında klavyeden “0 1 2” girilmiştir. Program hata döndürüldüğü için ilgili bellek verisi programın çalıştığı dizine yazılmıştır. Program ikinci defa çalıştırıldığında “1 2 3” girilmiştir. Yine program akışı kesilmiş ve ilgili bellek verisi programın çalıştığı dizine yazılmıştır. Bu dosyalar daha sonrası için programın neden hata üretip sonlandığının bulunması için gerekli incelemelerde kullanılır. Hata mesajının incelenmesinden programın hangi satırında hatanın olduğu da görülmektedir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./ikinci_dereceden_polinom_kok_bul
```

Bu program ikinci dereceden bir polinomun reel köklerini bulmaktadır.

Polinom  $ax^2 + bx + c$  biçimindedir. Sırası ile a, b ve c giriniz. 0 1 2

Assertion failed: (kontrol), function main, file ikinci\_dereceden\_polinom\_kok\_bul.cxx, line 66.

Abort trap (çekirdek döküldü)

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./ikinci_dereceden_polinom_kok_bul
```

Bu program ikinci dereceden bir polinomun reel köklerini bulmaktadır.

Polinom  $ax^2 + bx + c$  biçimindedir. Sırası ile a, b ve c giriniz. 1 2 3

Assertion failed: (kontrol), function main, file ikinci\_dereceden\_polinom\_kok\_bul.cxx, line 78.

Abort trap (çekirdek döküldü)

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

Hata yakalam fonksiyonu olan assert ile program akışındaki çeşitli durumların neden olabileceği hataların yakalanması için kullanılmasının sonlandırılması için program kodu içerisindeki assert fonksiyonlarının bulunduğu kod blokları ile ön işlemci yönergesinin kaynak koddan çıkarılması yeterlidir. Ancak program kodu içerisinde halen fonksiyonun bulunması ama derlenmiş dosyada yer alması ancak işlenmemesi için “**#include <cassert>**” ön işlemci yönergesinden önce **#define NDEBUG** yazılmalıdır.

## 4.Kontrol Yapıları: Döngüler

Bir program ardışık olarak çalışan yönergelerden oluşmak durumunda değildir. Önceki bölümde görüldüğü gibi bazı durumlarda program akışında yapılacak olan seçimler ile bazı kod blokları atlanıp diğer kod blokları üzerinden program işlemeye devam edebilir. Bazı problemlerin çözümünde seçimlere bağlı olarak program akışı düzenlenebilir olmakla birlikte bazı durumlarda ise program akışının seçimlere bağlı olmadan aynı kod blokunun bir çok defa tekrarlanması gereklidir. Bu tekrar işlemi ise “**döngü**” olarak adlandırılır. Döngüler aynı kod blokunun belirli sayıda veya tanımlanmış olan bir koşulun gerçekleşmesi durumuna bağlı olarak yine belirli bir sayıda tekrarlanması ile kurgulanır. Bu bölümde döngülerin tasarıımı ve analizi üzerinde durulacaktır.

### 4.1.Döngüler neden kullanılır?

Bir fabrikada üretimin takip edilmesi için yazılacak olan bir program belirli parçaların günlük üretim sayılarının girilmesi ile haftalık üretim miktarının hesaplayacak bir program yazılmaktır. Bu programın yapacağı işlem için haftanın her günü yapılan üretim için bir değişken tanımlaması yapılabilir. Bu değişken tanımlaması ile her bir gün için yapılan üretim miktarları hafta sonunda girilerek haftalık üretim miktarı belirlenebilir. Bu programda üretim miktarı tam sayı veri tipinde olarak tanımlanabilir. Bir haftalık üretim için yedi adet girdiyi okuyarak ve ilgili değişkene atayacak girdi okuma satırı yazılması gerekecektir. Bu işlemin ardından haftalık üretim miktarının saklandığı bir tam sayı veri tipinde değişkene ve bu değişkenin de haftalık üretim miktarlarının toplamını barındırması gerekecektir.

```
/*
 * uretim_miktar.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtım; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
```

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen  
\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.  
\*  
\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
```

```
#include <locale>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    // Dil ayarları
```

```
    setlocale(LC_ALL, "tr_TR.UTF-8");
```

```
    // Değişken tanımlamaları ve etkinleştirilmeleri
```

```
    int gun1 = 0, gun2 = 0, gun3 = 0, gun4 = 0, gun5 = 0, gun6 = 0, gun7 = 0, toplam = 0;
```

```
    cout << "Bu program haftalık toplam üretim miktarını belirler." << endl;
```

```
    cout << "Pazartesi gününe ait üretim miktarını giriniz: ";
```

```

cin << gun1;
cout << "Salı gününe ait üretim miktarını giriniz: ";
cin >> gun2;
cout << "Çarşamba gününe ait üretim miktarını giriniz: ";
cin >> gun3;
cout << "Perşembe gününe ait üretim miktarını giriniz: ";
cin >> gun4;
cout << "Cuma gününe ait üretim miktarını giriniz: ";
cin >> gun5;
cout << "Cumartesi gününe ait üretim miktarını giriniz: ";
cin >> gun6;
cout << "Pazar gününe ait üretim miktarını giriniz: ";
cin >> gun7;
toplam = gun1 + gun2 + gun3 + gun4 + gun5 + gun6 + gun7;
cout << "Yapılan toplam üretim miktarı " << toplam << " adettir." << endl;
return 0;
}

```

Programın derlenip çalıştırıldığında aşağıdaki çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./uretim_miktar
```

Bu program haftalık toplam üretim miktarını belirler.

Pazartesi gününe ait üretim miktarını giriniz: 123

Salı gününe ait üretim miktarını giriniz: 234

Çarşamba gününe ait üretim miktarını giriniz: 245

Perşembe gününe ait üretim miktarını giriniz: 276

Cuma gününe ait üretim miktarını giriniz: 232

Cumartesi gününe ait üretim miktarını giriniz: 222

Pazar gününe ait üretim miktarını giriniz: 229

Yapılan toplam üretim miktarı 1561 adettir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Program şimdilik haftalık üretim miktarını bulmaktadır. Programın haftalık değil de aylık üretim miktarını bulmasını istersek, yukarıdaki programın işlem yapan kod kısmını dört defa daha

tekrar yazmamız gerekecektir. Ayrıca haftalık üretim miktarları için dört ayrı haftalık üretim miktarını barındıracak değişkene ve aylık toplam üretim miktarını hesaplayacak aritmetik ifadeye ve bu işlemin sonucunu barındıracak olan aylık üretim miktarını barındıran bir tam sayı veri tipinde bir değişkene gereksinim olacaktır.

Programın kapsamının genişletilmesi beraberinde programın tamamının yeniden yazılmasını gerektirecektir. Programın yeniden yazılması sırasında işlemleri basitleştirmek için aylık üretimi temsil eden 30 adet günlük üretim miktarı verisi barındıran değişken ve bir tane de aylık üretiminin toplam miktarını barındıracak olan aylık toplam üretim miktarı değişkeni tanımlanabilir. Böylece aylık üretim miktarı için gerekli olan haftalık üretim miktarı değişkeni oradan kaldırılabilir. Ancak bu bağıntıların tekrarlanarak yazılmaması yazım hatalarını ve hatta gözden kaçan bazı değişkenler dolayısı ile de ortaya çıkabilecek olan mantık hatalarını de beraberinde getirebilir.

Bu tekrarlayan kod bloklarının yeniden yazilarak ortaya çıkan uzun programlardan kaçınmak için döngüler tercih edilir. C++ Programlama Dili yapısında üç çeşit döngü barındırmaktadır. Bu döngü yapılarından probleme uygun olanı seçilerek kullanılabilir.

## 4.2.while Döngü Yapısı

C++ programlama Dili’nde yer alan döngü yapılarında olan “**while()**” döngüsü belirlenmiş bir mantıksal koşulun doğru olduğu sürece işleyen bir yapıdır. Bir while döngüsünün yapısı aşağıdaki gibidir.

```
while(ifade)
```

```
    bildirim
```

Döngüde belirtilen bildirim basit bir ifade olabileceği gibi aynı zamanda karmaşık bir ifade veya birden çok ifadenin bir arada olduğu kod bloku da olabilir. Döngünün koşulun geçerli olduğunu kontrol edilmesi ile başlayan ve işlenen bildirimin veya kod bloğunun sona ermesine kadar yapılan tüm işlemler bir çevrim olarak tanımlanır. Bir çevrim sona erdiğinde döngü yeniden başa döner. Koşul olarak belirtilen ifadenin geçerliliği sınanır. Eğer ifadenin mantıksal olarak sonucu **doğru – true** ilse döngü çevrimi yinelenir. Tekrar ifadenin mantıksal sonucu kontrol edilir. Eğer sonuç **yanlış – false** ise döngü çevrimi yinelenmez ve program akışı döngüden sonra gelen bildirim ile devam eder.

Buradan da anlaşılacağı üzere **while()** döngüsünde döngünün gerçekleşmesini sağlayan koşulun döngü içerisinde yani döngü gövdesinde düzenlenmesi gerekecektir. **Döngü gövdesi**, “{ ... }” arasında kalan kod blokudur. Yapılacak olan düzenleme döngünün yapısında bulunan kod bloku tarafından işlenecek olan verinin durumuna, sayısına veya belirli bir koşulun gerçekleşme durumuna bağlı olarak düzenlenebilir. Eğer bu düzenleme yapılmaz ise **sonsuz döngü – infinite loop** durumu ortaya çıkar. Sonsuz döngü, adından anlaşılacağı üzere döngü yapısının kesintisiz olarak çalışması durumudur. Bu istenmeyen bir durumdur ve genellikle programcı hatasından kaynaklanır. Yukarıdaki üretim miktarı hesaplayan programın haftalık üretim miktarını esas alan ama veriyi döngü kullanarak okuyan ve işleyen şekli aşağıdadır.

```
/*
```

```
* uretim_miktar_while.cxx
```

\*

\*

\* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımıyla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

```

*/
/*
#include <iostream>
#include <locale>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişken tanımlamaları ve etkinleştirilmeleri
    int gun = 1, toplam = 0, miktar =0;
    cout << "Bu program haftalık toplam üretim miktarını belirler." << endl;
    while (gun <=7)
    {
        cout << gun << ". güne ait üretim miktarını giriniz: ";
        cin >> miktar;
        toplam = toplam + miktar;
        gun++;
    }
    cout << "Yapılan toplam üretim miktarı " << toplam << " adettir." << endl;
    return 0;
}

```

Programın derlenip çalıştırıldığında aşağıdaki çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./uretim_miktar_while
```

Bu program haftalık toplam üretim miktarını belirler.

1. güne ait üretim miktarını giriniz: 123
2. güne ait üretim miktarını giriniz: 234
3. güne ait üretim miktarını giriniz: 245
4. güne ait üretim miktarını giriniz: 276
5. güne ait üretim miktarını giriniz: 232
6. güne ait üretim miktarını giriniz: 222

7. güne ait üretim miktarını giriniz: 229

Yapılan toplam üretim miktarı 1561 adettir.

[goksin@tardis ~/projeler/C++\_Notlar/]\$

Programdaki döngünün doğru şekilde işlemesini sağlayan ifade aşağıda görülmektedir.

gun = gun + 1;

Bu ifade döngünün çevrim sayısını kontrol eden değişken olan “gun” değişkenin değerini her döngü sonuna gelindiğinde bir artırmaktadır. Döngünün işlenmesini sağlayan koşul da döngünün başında yer alan ifadedir.

while(gun <= 7)

Bu ifade döndüğün işlenmesi için gerekli olan koşulu barındırmaktadır. Eğer kontrolü sağlayan bir “gun = gun + 1” ifadesi döngüden çıkarılacak olursa program sonuz döngüye girecek ve sona ermeden çalışmaya devam edecektir.

Aşağıda görülen program ve çıktısını inceleyelim.

```
/*
 * dongu_kontrol.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
```

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA  
 \* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
 \* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
 \* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
 \* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
 \* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
 \* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
 \* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
 \* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
 \* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
 \* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
 \* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
 \* SORUMLU DEĞİLLERDİR.  
 \*  
 \*/  
 /\*  

```

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

int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");

    // Değişken tanımlamaları ve etkinleştirilmeleri
    int i = 1;
    cout << "i=" << i << endl;
    while (i <=20)
    {
        cout << i << " ";
        i = i + 4;
    }
}

```

```

cout << endl << "i=" << i << endl;
return 0;
}

```

Programın derlenip çalıştırıldığında aşağıdaki çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./dongu_kontrol
i=1
1 5 9 13 17
i= 21
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Programın ilk çıktısı döngüyü kontrol eden değişkenin belirlenmiş olan başlangıç değeridir. Program ilk çalıştırıldığında döngü işlenmeye başlanmada önce bu değeri yazdırmaktadır. Döngü işlenmeye başlandığında *i* değişkenin değeri her çevrimde dört arttırılmaktadır. Çıktıdan anlaşıla-cağı üzere döngünün çevrimi bittiğinde *i* değişkenin değeri kontrol edilmektedir. Koşul gereği *i* değişkenin değeri 20'den küçük ise ifadenin sonucu mantıksal olarak **doğru – true** olduğundan döngü yeniden çevrime girip işlenmektedir. Döngü çıktıdan görüleceği üzere beş defa tekrar etmiştir. Son çevrimde *i* değişkenin değeri 21 olmuştur. Döngü çevrimine yeniden başlanacak iken öncekilerde olduğu gibi değişkenin değeri kontrol edilmiştir. Değişkenin değeri 21 olmuş ve koşul gereği çevrim gerçekleşmemiştir. Kontrol amaçlı kullanılan değişken olan *i* değeri se 21 olmuştur. Aşağıdaki tabloda **while** döngüsünün her bir çevriminde *i* değişkenin değeri ve programın döngü içerisinde ürettiği çıktı görülmektedir.

<b>Cevrim sayısı</b>	<b>Değişken <i>i</i> değeri</b>	<b>Kontrol ifadesi</b>	<b>Artum ifadesi</b>	<b>Program çıktısı</b>
1	1	<i>i</i> <= 20 / Doğru – True	<i>i</i> = 1+ 4 = 5	1
2	5	<i>i</i> <= 20 / Doğru – True	<i>i</i> = 5+ 4 = 9	5
3	9	<i>i</i> <= 20 / Doğru – True	<i>i</i> = 9+ 4 = 13	9
4	13	<i>i</i> <= 20 / Doğru – True	<i>i</i> = 13+ 4 = 17	13
5	17	<i>i</i> <= 20 / Yanlış – False	<i>i</i> = 17+ 4 = 21	17

Döngülerin kurgulanmasında yapılan bir hata, döngünün başlangıç ifadesinein sonua “;” konulmasıdır.

```

while (i <=20);
{
    cout << i << " ";
    i = i + 4;
}

```

Yukarıdaki kod blokunda da görüleceği üzere döngünün başlangıcını belirten bildirimin sonunda ";" yer aldığı için bu döngünün "{ ... }" arasında yer alan bir gövdesi bulunmamaktadır.

### 4.2.1.while Döngüsünün Tasarımı

Önceki bölümlerde yer alan örneklerde görüldüğü gibi bir while() döngüsü, döngüyü kontrol eden değişkenin durumuna göre işlenir. Önceki ekteki örnekte i değişkeni döngünün işleyişini kontrol eden değişken idi. Tüm while() döngülerinde döngüyü kontrol eden değişkenin durumuna bağlı olarak belirlenen bir kontrol ifadesi bulunur. Bu kontrol ifadesinin kurgulanma biçimini ise döngünün tasarımını ve işleyişini belirler.

#### 4.2.1.1. Sayaç ile Kontrol Edilen while() Döngülerı

Bu döngülerin tasarımında önceki ekteki gibi döngünün çevrim sayısının önceden belirlenmiş olan bir sayıdan fazla olmaması esastır. Bu sayı döngünün kaç defa işleneceğini kontrol eden bir sayaç görevi görmektedir. Aşağıdaki örnek programda bir işletmenin mağazalarında bir ayda gerçekleştirilen mikroişlemci satışlarına ait sayılar girilerek elde edilen gelir hesaplanmaktadır.

```
/*
 * dongu_sayac.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
```

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
#include <locale>
#include <cfloat>
#include <string>
#include <iomanip>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişken tanımlamaları ve etkinleştirimieleri
    const float FIYAT = 5480.75;
    int magaza_say = 0, satis = 0, i = 1;
    float gelir = 0.0;
    string magaza;
```

```

cout << "Mağaza sayısını girdikten sonra mağaza adını ve satış sayısını girip enter basınız."
<< endl;

cout << "Mağaza sayısını giriniz: ";
cin >> magaza_say;
while (i <= magaza_say)
{
    cout << "Mağaza adını ve satış sayısını giriniz. ";
    cin >> magaza >> satis;
    gelir = gelir + (satis * FIYAT);
    i++;
}
cout << fixed << setprecision(2);
cout << "Elde edilen gelir: " << gelir << " TL'dir."<< endl;
return 0;
}

```

Programın derlenip çalıştırıldığında aşağıdaki çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./dongu_sayac
```

Mağaza sayısını girdikten sonra mağaza adını ve satış sayısını girip enter basınız.

Mağaza sayısını giriniz: 6

Mağaza adını ve satış sayısını giriniz. Eskişehir 43

Mağaza adını ve satış sayısını giriniz. Bursa 22

Mağaza adını ve satış sayısını giriniz. İzmir 68

Mağaza adını ve satış sayısını giriniz. Ankara 98

Mağaza adını ve satış sayısını giriniz. İstanbul 234

Mağaza adını ve satış sayısını giriniz. İzmit 56

Elde edilen gelir: 2855470.75 TL'dir

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Programda mağaza sayısı kontrol değişkeni olarak kullanılmaktadır. Sayaç olarak ise “i” adlı değişken kullanılmıştır. Sayaç değişkeni kontrol değişkeninden küçük olduğu sürece döngü işlemektedir .Döngüde her bir mağazanın adı “magaza” adlı değişkene atanmaktadır. Satış miktarı da “satis” adlı değişkende barındırılmaktadır. Bu değerler okunduktan sonra o mağazadan elde edilen gelir hesaplanmaktadır. Hesaplana değer gelir adlı değişkende barındırılmakta ve döngünün her

tekrar edilmesi sırasında gelir hesaplanı önceden belirlenen değeri ile toplanarak toplam gelir elde edilmekte ve sonuç yine aynı değişkene atanarak saklanmaktadır. Döngünün gövdesinin sonuna gelindiğinde ise sayaç değişkeni olan i bir arttırılmaktadır. Döngü ise sayaç değişkenin mağaza sayısından büyük olduğu belirlendiğinde sona ermektedir ve elde edilen gelir miktarı da programın son bölümünde yazdırılmaktadır. Bu döngü tasarımda döngünün çevrim sayısı döngü kontrol değişkeninin büyülüğe eşittir.

#### **4.2.1.2. Bitiş Sözcüğü ile Kontrol Edilen while() Döngülerı**

Döngünün tasarımda kullanılabilen bir diğer kontrol değişkeni de bitiş simgelerinin veya sözcüklerinin kullanılmasıdır. Bitiş simgesinin veya sözcüğünün ana fikri whiile() döngünün işlemesi için gerekli olan mantıksal ifadenin sonucunun bitiş simgesi girilmediği sürece işlemeye devam etmesine dayanmaktadır. Bir diğer deyişle yapılan giriş sırasında önceden belirlenmiş olan karakter veya karakter dizisi girilmediği sürece döngü işlenir. Aşağıdaki örnekte bir önceki programın bitiş simgesi kullanılarak yazılmış olanı görülmektedir.

```
/*
 * dongu_simge.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtım; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
```

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
#include <locale>
#include <cfloat>
#include <iomanip>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Sabitlerin tanımlanması
    const float FIYAT = 5480.75;
    const string BITIS = "SON";
    // Değişken tanımlamaları ve etkinleştirimieleri
    int satis = 0;
    float gelir = 0.0;
    string magaza;
```

```

cout << "Mağaza sayısını girdikten sonra mağaza adını ve satış sayısını girip enter basınız."
<< endl;

cout << "Programı sonlandırmak için \"SON\" girip enter basınız." << endl;
cout << "Mağaza adını ve satış sayısını giriniz. " << endl;
cin >> magaza;
while (magaza != BITIS)
{
    cin >> satis;
    gelir = gelir + (satis * FIYAT);
    cin >> magaza;
}
cout << fixed << setprecision(2);
cout << "Elde edilen gelir: " << gelir << " TL'dir."<< endl;
return 0;
}

```

Programın derlenip çalıştırıldığında aşağıdaki çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./dongu_sayac
```

Mağaza sayısını girdikten sonra mağaza adını ve satış sayısını girip enter basınız.

Programı sonlandırmak için "SON" girip enter basınız.

Eskişehir 43

Bursa 22

İzmir 68

Ankara 98

İstanbul 234

İzmit 56

SON

Elde edilen gelir: 2855470.75 TL'dir

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Programda döngünün sona ermesi için önceden belirlenmiş olan karakter dizisi kullanılmaktadır. Döngüyü sonlandırmak için “SON” yazılması ve enter basılması gereklidir. Programdaki döngünün işlemesi için öncelikle mağaza isimleri değişken olarak kullanıldığından burad aynı veri tipinde olmak üzere “SON” adlı bitiş sözcüğü kullanılmıştır. Anca döngünün başlaması için önce-

likle “magaza” adlı değişkenin ilk başlangıç değerinin verilmesi zorunludur. Döngünün bitiş sözcüğü veya simgesi ile kontrol edilmesi durumunda while() döngünün başlatılabilmesi için bu girişin yapılması zorunludur. Döngü öncesinde klavyeden “magaza” adlı değişkene giriş yapılmaktadır. Döngü giriş yapılan verinin kontrol sözcüğü veya simgesi ile karşılaşmakta değerlendirem sonucunun doğru olması durumunda döngü işlemektedir .Döngü gövdesinde tekrar döngü çevrimin gerçekleşip gerçekleşmeyeceğinin belirlemek için yine “magaza” adlı değişkene giriş yapılması gereklidir. Döngü sona erdiğinde ise önceki programda olduğu gibi gerekli hesaplamalar yapılp program sona ermektedir.

#### **4.2.1.3.Durum ile Kontrol Edilen while() Döngüleri**

Döngünün tasarımda kullanılabilen durum kontrolü önceden tanımlanmış olan bir durumun gerçekleşme durumunun mantıksal operatör ile yorumlanması dayanmaktadır. Bu döngü tasarımda döngünün işlemesini sağlayan ifadenin her zaman için doğru olması gereklidir. O halde mantıksal olarak işlenecek olan ifadenin yapısında yer alana bir diğer mantıksal değerin değillenmesi bu döngünün kurgulanarak işletilmesi için yeterli ve gerekli koşuldur. Örneğin sayı tahin oyunu oyuncunun belirlenmiş olan sayıyı bulması durumunda sona ermektedir. Bu oyunu program olarak yazılması için tahminin gerçekleşme durumu kurgulanacak döngü için kontrol değişkeni görevi görecektir. Aşağıdaki programda tahmin oyunu while() döngüsü ile kurgulanmıştır. Oyuncu sayıyı tahmin ettiğinde oyun sona ermektedir. Tahmin edemez ise oyun devam etmektedir.

```
/*
 * tahmin.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtıımı; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
```

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
#include <locale>
#include <ctime>
#include <cstdlib>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Sabitlerin tanımlanması
    bool BULDUN = false;
    // Değişken tanımlamaları ve etkinleştirimieleri
    int sayac = 0, tahmin = 0, sayı = 0;
```

```

// Tahmin oyunu için ratsgele sayı oluştur.
srand(time(0));
sayi = rand () % 100;
cout << "Bu bir tahmin oyunudur. Bir ve yüz arasında olan bir sayıyı tahmin etmeniz gereklidir. Oyun sayı tahmin edildiğinde sona ermektedir. " << endl;
while (!BULDUN)
{
    cout << "Bir ile yüz arasında bir tam sayı girip enter basın. ";
    cin >> tahmin;
    sayac++;
    if (tahmin == sayi)
    {
        cout << sayac << ". denemediğinizde buldunuz, tebrikler." << endl;
        BULDUN = true;
    }
    else if(tahmin < sayi )
    {
        cout << "Bilemediniz, sayı bundan büyüktür. Tekrar deneyin. " << endl;
    }
    else
        cout << "Bilemediniz, sayı bundan küçüktür. Tekrar deneyin. " << endl;
}
return 0;
}

```

Programın derlenip çalıştırıldığında aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./tahmin
[goksin@tardis ~/projeler/C++_Notlar/2022-2023/Güz/04]$ ./tahmin
```

Bu bir tahmin oyunudur. Bir ve yüz arasında olan bir sayıyı tahmin etmeniz gereklidir. Oyun sayı tahmin edildiğinde sona ermektedir.

Bir ile yüz arasında bir tam sayı girip enter basın. 22

Bilemediniz, sayı bundan büyüktür. Tekrar deneyin.

Bir ile yüz arasında bir tam sayı girip enter basın. 45

Bilemediniz, sayı bundan küçüktür. Tekrar deneyin.

Bir ile yüz arasında bir tam sayı girip enter basın. 40

Bilemediniz, sayı bundan büyüktür. Tekrar deneyin.

Bir ile yüz arasında bir tam sayı girip enter basın. 43

4. denemede buldunuz, tebrikler.

[goksin@tardis ~/projeler/C++\_Notlar/2022-2023/Güz/04]\$

Program oyuncunun bilgisayar tarafından üretilen rastgele bir sayıyı doğru tahmin ettiğinde sona ermektedir. Tahmin sayısının kaç adet olacağı bilinmediği için bunun bir döngü ile tasarlanması uygundur. Döngü while() yapısı ile tasarlanmıştır. Döngünün sona erme koşulu ise döngünün bir çevrimi sırasında sayının tahmin edilmesidir. Bunun için de döngü yapısında kullanılacak bir **mantıksal değişken** tanımlanıp değeri “yanlış – false” olarak seçilmiştir. Döngünün işlemesi için mantıksal değerlendirme sonucunda koşulun “doğru – true” olması gereklidir. Bu nedenle döngü gövdesinde mantıksal işlem sonucunun yanlış olması için doğru olan değerin “değil – not” işleminden geçmesi gereklidir. İşlem sonucunun mantıksal değerlendirme “yanlış” olduğundan döngü sona erer.

#### **4.2.1.4.Dosya Sonu – EOF Bildirimi ile Kontrol Edilen while() Döngülerı**

Sıklıkla verinin değiştiği durumlarda döngüler kullanılarak verilerin okunması ve işlenmesi tercih edilen bir durumdur. Veri sayısının değişkenliğinden dolayı verinin sonlandığının belirlenmesi için bitiş sembollerini kullanılabılır. Ancak bu yaklaşımın olumsuz yanı, verilerin bulunduğu dosyada düzenleme yapan çok sayıda kişinin olması durumunda veri kümesinin sona erdiğini belirten bitiş sembolünün/simgesinin de yanlışlıkla silinmesi sonucunda döngünün hatalı işlemesidir. Silme işlemini yapan kullanıcının bunu iyi niyetle yaptığı bildiğimizden bu konuda bitiş sembolü/simgesi kullanmak yerinde dosyanın sonuna gelindiğini belirten ve dosyanın son satırı olan “**EOF – End Of File**” girdisi kullanılabilir. Bu yaklaşım üzerinde çok sık değişim yapılan ve çok sayıda veriyi barındıran dosyalardan okunan veriler üzerinde çalışan programlarda en iyi çözümüdür. Aşağıdaki durumlarda “**girdi akım değişkeni – input stream variable**” “yanlış – false” veya “**doğru – true**” değerini döndürür.

1. Klavyeden girilen verilerin okunması “cin” ve “>>” kullanılmaktadır. “>>” operatörü eğer girdileri dosyadan okuyor ise dosyanın sonuna gelindiğinde başka veri okumayacağı için girdi akım değişkeni “yanlış – false” değerini döndürecektr.
2. Bir program girdileri okurken tanımlanmış olan veri tipinden farklı, örneğin int, bir veri tipinde bir veriyi okuyup, örneğin char, bunu tanımlanmış veri tipindeki değişkene atanması durumunda girdi çıktı işlemlerinde hata oluşur ve sonlanır. Bu durumda da girdi akım değişkenin değeri “yanlış – false” olarak tanımlanır. Bu durumda işlemci herhangi bir hata döndürmez, çalışmaya devam eder ve program hatalı girdilerden verileri işlemeye çalışır ve çıktı da hatalı olur.

3. Yukarıda belirtilen iki durum dışında “**girdi akım değişkeni – input stream variable**” değişkeni “**doğru – true**” değerini döndürür.

Girdi akım değişkeninin doğru veya yanlış değer döndürmesi veri okuma işleminin sonuna gelindiğini belirlemek için kullanılabilir. Bunu while() döngülerindeki mantıksal değerlendirme işlemi ile birlikte kullanılması ile bir veri girişi işleminin sona erip ermediği kontrol edilerek okuma ve diğer işlemler gerçekleştirilebilir.

```
cin >> değişken;
```

```
...
```

```
while (cin)
```

```
{
```

```
...
```

```
cin >> değişken;
```

```
...
```

```
}
```

C++ programlama Dili, bir girdi akım değişkenin değerinin, örneğin cin gibi, kontrol edilerek değerinin değişmesi durumunda dosyanın sonuna gelindiğini belirlemek yanında bu iş için özel hazırlanmış bir fonksiyona da sahiptir. Bu fonksiyona **eof** fonksiyonu adı verilir. Diğer G/Ç fonksiyonları olan cin, peek ve putback gibi eof fonksiyonu da **istream** üyesidir. Kullanımı aşağıdaki gibidir:

```
girdiakımdeğişkeni.eof()
```

Yukarıdaki örnek gösterimde istream fonksiyonun değişkenleri eof() fonksiyonu ile birlikte kullanılır. Dosyadan okuma yapılacağı için kullanım şekli de aşağıda açıklandığı gibi olacaktır.

Dosyadan okunacak olan veril için ifstream değişkenin tanımlanması gereklidir. Bu değişken tanımlaması aşağıdaki gibi yapılmış olsun.

```
ifstream dosya_ac_oku;
```

Program içerisinde dosyadan okuma işleminin yapılması sırasında değişkenin okunacak olan dosyanın sonuna gelinene kadar mantıksal olarak değeri **doğru – true** olacaktır. Son satır okunduktan sonra ve dosyanın sonuna gelinip de işlem devam edildiğinde ise değişkenin mantıksal değeri **yansız – false** olacaktır. Yukarıdaki tanımlamanın program içerisindeki yeri aşağıdaki gibi olsun.

```
ifstream dosya_ac_oku;
```

```
char karakter;
```

```
dosya_ac_oku.open("kaynak");
```

Yukarıdaki kod bölümü girdi akım değişkeni olan dosya\_ac\_oku tanımlanmıştır. Karakter veri tipinde tanımlanmış olan değişken karakter adını taşımaktadır. Okunacak veri de program ile

aynı dizinde bulunan kaynak adlı dosyadır. Dosyanın ilk satırdan son satıra kadar okunmasını sağlamak için bir döngü kurulmalıdır. Bu while() döngüsü ile aşağıdaki gibi yapılabilir.

```
ifstream dosya_ac_oku;  
char karakter;  
dosya_ac_oku.open("kaynak");  
while(!dosya_ac_oku.eof())  
{  
    dosya_ac_oku.get(ch);  
    cout << karakter;  
}
```

Dosyadan veri okunurken program dosyanın sonuna gelmediği sürece dosya\_ac\_oku.eof() fonksiyonun mantıksal durumu **y yanlış – false** olarak kalacaktır. Döngü işlenecektir. Ancak son satır okunduktan sonra, başka okunacak satır kalmayınca, ise durum **true – doğru** olacaktır. Bu durumda da yukarıda görülen döngü sonlanacaktır. Döngünün işlemesi koşulu daima mantıksal olarak **doğru – true** olmasını gerektirmekte olduğu unutulmamalıdır.

Aşağıda döngü örneği bir program içerisinde kullanıldığından dosyada yer alan tüm tam sayıların toplamını bulmaktadır.

```
int toplam =0;  
int sayı;  
cin >> sayı  
while(cin)  
{  
    toplam = toplam + sayı           // Okunan sayı toplama eklenir.  
    cin >> sayı                   // sıradaki sayıyı okur.  
}  
cout << "Toplam=" << " " << toplam << endl;
```

### 4.3.do ... while Döngü Yapısı

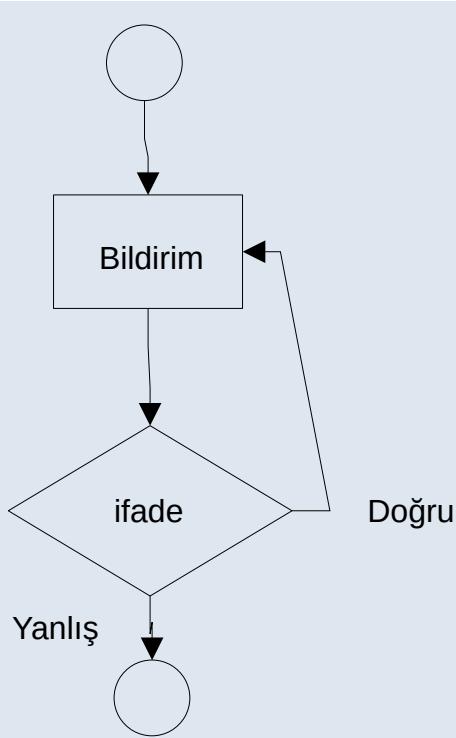
C++ programlama Dili’nde yer alan döngü yapılarından bir diğeri olan “**do ... while()**” döngüsü işleyiş olarak while() döngüsüne benzer olsa da farklıdır. Bu farklılık döngünün yapısından kaynaklanır. Döngü kontrol amaçlı kullanılan ifadenin değerinin bulunması ancak döngünün bir defa işlenmesinin ardından gerçekleşir. Bu öylece döngü en az bir defa işlenir. İlk çevrimin sonunda önceden belirlenmiş olan ifadenin mantıksal durumu kontrol edilir. İfadenin döndürdüğü değere göre döngü tekrar işlenir veya işlenmez. Bir “do ... while” döngüsünün yapısı aşağıdaki gibidir.

```

do
{
    bildirim;
    ...
}
while(ifade);

```

Bu döngüde belirtilen bildirim basit bir ifade olabileceği gibi aynı zamanda karmaşık bir ifade veya birden çok ifadenin bir arada olduğu kod bloku da olabilir. Döngünün ilk çalıştırılmasının ardından koşulun geçerli olduğunu kontrol edilir. Koşul olarak belirtilen ifadenin geçerliliği sınanır. Eğer ifadenin mantıksal olarak sonucu **doğru – true** ise döngü çevrimi yinelenir. Tekrar ifadenin mantıksal sonucu kontrol edilir. Eğer sonuç **yanyılış – false** ise döngü çevrimi yinelenmez ve program akışı döngüden sonra gelen bildirim ile devam eder. Aşağıdaki şekilde döngünün işleyişi gösterilmektedir.



Aşağıda görülen “do ... while()” döngüsü gösterilmiştir. Bu döngünün işlemesinde koşul kontrol edilmeden döngü işlemeye başlamaktadır. Değişken olan i için ilk değer sıfır olsa da ilk çevrimin yani **while(i <=20)** bildirimine gelene kadar tüm kod bloku işlenmektedir. Koşul kontrol edilip ifade mantıksal olarak doğru ise döngü tekrar işlenmektedir .Ancak son döngü çevriminde ise i = 25 olduğundan döngü sona ermektedir.

```

int i = 0;
do
{

```

```
cout << "i = " << i;  
i = i+5;  
}  
while(i <= 20)
```

Bu kod boku programa çevrilip çalıştırılınca 0 5 10 15 20 çıktısını üretecektir. Son çevrimde i değeri 25 olacağından koşul gerçekleşmeyecek ve döngüden çıkışacaktır.

while() döngüsünde, döngünün gerçekleşme koşulu, döngü çevrimi başlamadan önce kontrol edilir. Bunun tersi ise do ... while döngüsünde yer alır. Önce döngü en az bir defa ilenir ve sonra koşulun geçerliliği kontrol edilir. Aşağıda iki farklı döngü gösterilmektedir.

```
int i = 13;  
do  
{  
    cout << "i = " << i;  
    i = i+5;  
}  
while( i <= 10 );  
cout >> endl;
```

Bu döngü işlediğinde ekrana i değeri yazılır. Ancak aşağıdaki döngüde bu durum söz konusu değildir.

```
int i = 13;  
while( i <= 10 )  
{  
    cout << "i = " << i;  
    i = i+5;  
}  
cout >> endl;
```

Bu döngü işlediğinde ekrana i değeri yazılmaz.

Kullanıcıdan giriş yapılması istenen durumlarda do ... while() döngülerinin kullanılması girdinin kontrol edilmesi için uygun bir araçtır. Aşağıdaki döngü örneğinde kullanıcıdan giriş yapılması istenen sonuç belirtilen aralığın dışında girilmesi durumunda döngü tekrar işleyecek ve kullanıcının aralık içerisinde kalan değeri girmesi durumunda döngü sona erecektir.

```
int deger;  
do
```

```

{
    cout << "0 ile 100 arasında bir tam sayı giriniz.";

    cin >> deger;

    cout << endl;

}

while(deger <0 || deger >100);

```

Aşağıdaki program örneğinde do ... while () döngüsü kullanılarak verilen bi sayının 3 veya 9 tam olarak bölünüp bölünmeyeceği belirlenmektedir. Bir sayının her iki sayıya da tam olarak bölünebilmesi için gerekli olan koşul sayının basamaklarının rakam değerlerinin toplamının 3 veya 9 katı olması gereklidir.

Problem bir sayının basamaklarının rakamlarının toplamının bulunmasını gerektirmektedir. O halde öncelikle sayının basamaklarının toplanması için gerekli olan çözümün belirlenmesi gereklidir. Bölünebilirlik için girdiyi tam sayı olarak aldığımız için basamakların büyüklüğünün bulunması için sayı ona bölünebilir. Bu durumda modulo – % işlemi bize ona bölümünden kalanı verecektir. Kalan ise sayının toplanıp tekrar ona bölünmesi ile her seferinde sayı bir basamak azaltılmış olurken aynı zamanda da basamak toplamı elde edilmiş olmaktadır. Bu işlemin ona bölümünden kalan olmadığı duruma kadar tekrarlanması gereklidir. Bunun için de normal bölmeye işleminin yapılması gereklidir. Sayının basamakları toplamının ona bölmeye döngünü yeteri kadar tekrarlanması durumunda ona bölüm ile sıfır sonucuna verecektir. Bu durum sağlandığında döngü sona erecektir. Elde edilen basamaklar toplamının 9 veya 3 tam bölünmesinin kontrolü için de yine modulo – % işlemi yapılır ve elde edilen işlem sonucuna göre sayının bölünebilme durumu yazdırılır.

```

/*
 * bolunebilirlik.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *

```

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,  
 \* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen  
 \* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.  
 \*  
 \* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
 \* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA  
 \* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
 \* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
 \* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
 \* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
 \* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
 \* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
 \* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
 \* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
 \* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
 \* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
 \* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
 \* SORUMLU DEĞİLLERDİR.  
 \*  
 \*/  
 /\*  
 // Dil ayarları  
 setlocale(LC\_ALL, "tr\_TR.UTF-8");  
 int toplam=0, sayı=0, tmp=0;  
 cout << "Bir pozitif tam sayı giriniz. ";  
 cin >> sayı;  
 cout << endl;  
 tmp = sayı;  
 do  
 {  
 // son basamak elde edilir ve sayı ile toplanır.

```

toplam = toplam + (sayi % 10);

// son basamak kaldırılır.

sayi = sayi / 10;

}while(sayi>0);

cout << "Basamakların toplamı = " << toplam << endl;

if (toplam % 9 == 0)

    cout << tmp << " sayısı 3 ve 9 ile tam bölünebilir." << endl;

else if (toplam % 3 == 0)

    cout << tmp << " sayısı 3 ile tam bölünebilir." << endl;

else

    cout << tmp << " sayısı 3 veya 9 ile tam bölünemez." << endl;

return 0;
}

```

Programın derlenip çalıştırıldığında aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./bolunebilirlik
```

```
Bir pozitif tam sayı giriniz. 609321
```

```
Basmakların toplamı = 21
```

```
609321 sayısı 3 ile tam bölünebilir.
```

```
[goksin@tardis ~/projeler/C++_Notlar/2022-2023/Güz/04]$
```

**Örnek Program:** Aşağıdaki program belirtilen sayıdaki Fibonacci<sup>10</sup> sayılarını hesaplamaktadır.

```
/*
 * fibonacci_sayisi.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdakiler koşullar yerine getirildiği

```

10 [https://en.wikipedia.org/wiki/Fibonacci\\_number](https://en.wikipedia.org/wiki/Fibonacci_number)

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
```

```
#include <locale>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
// Dil ayarları
setlocale(LC_ALL, "tr_TR.UTF-8");

// Değişkenler
int son, sayac, istenen;
int birinci = 1, ikinci = 1;

cout << "Bu program belirtilen sayıdaki Fibonacci sayılarını bulur." << endl;
cout << "Hesaplanacak olan Fibonacci sayısı adedini belirtiniz ve enter tuşuna basınız: ";
cin >> istenen;
if(istenen == 1)
{
    son = birinci;
    cout << "İstenen Fibonacci sayısı: " << birinci;
}
else if (istenen == 2)
{
    son = ikinci;
    cout << "İstenen Fibonacci sayıları: " << birinci << " " << ikinci;
}
else
{
    sayac = 3;
    cout << "İstenen Fibonacci sayıları: " << endl;
    cout << birinci << " " << ikinci << " ";
    while(sayac <= istenen)
    {
        son = birinci + ikinci;
        birinci = ikinci;
        ikinci = son;
        cout << son << " ";
        sayac++;
    }
}
```

```
}

cout << endl;

return 0;

}
```

Programın derlenip çalıştırıldığında aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./fibonacci_sayıları
```

Bu program belirtilen sayıdaki Fibonacci sayılarını bulur.

Hesaplanacak olan Fibonacci sayısı adedini belirtiniz ve enter tuşuna basınız: 5

İstenen Fibonacci sayıları:

```
1 1 2 3 5
```

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

## 4.4.for Döngü Yapısı

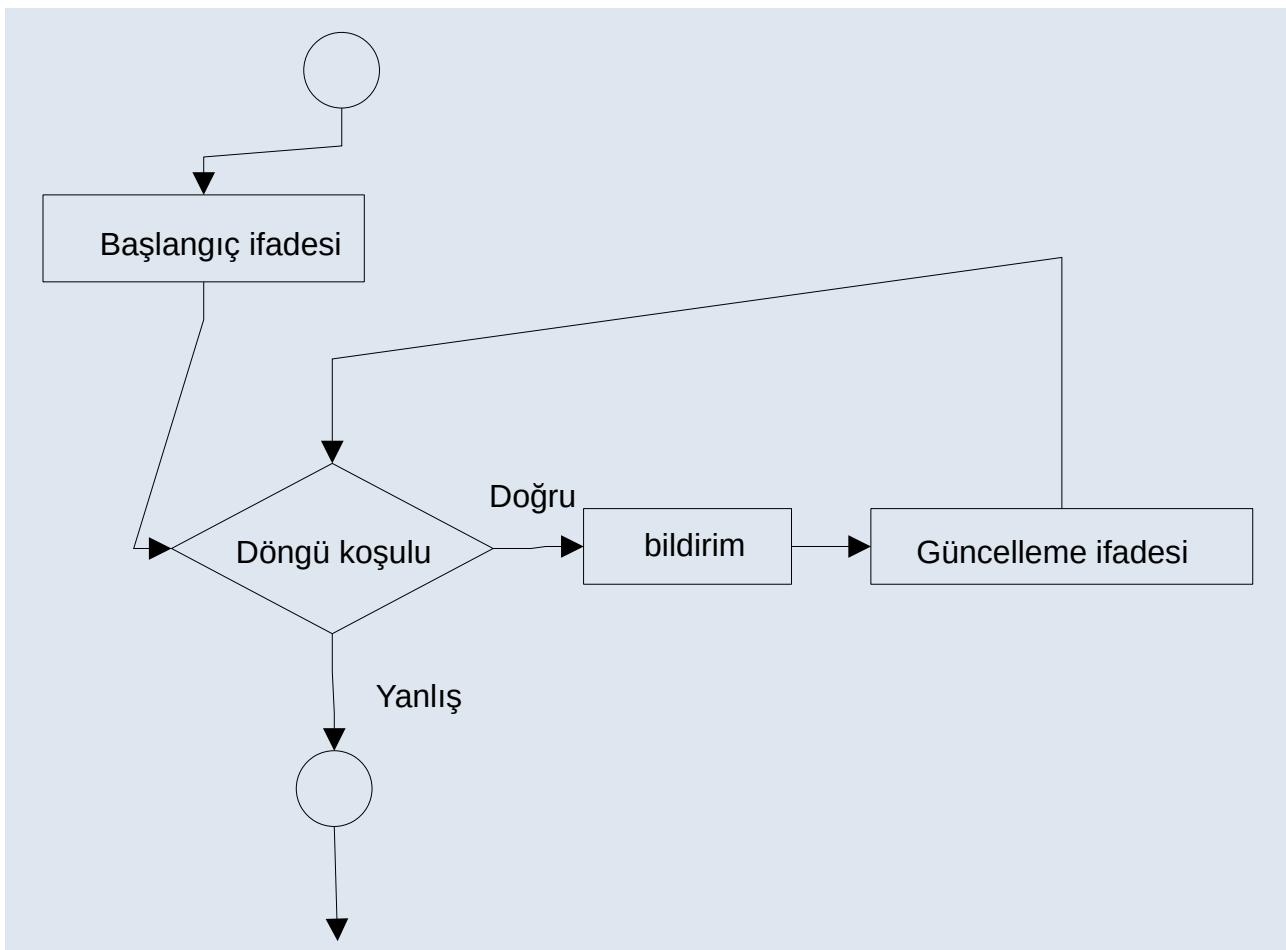
Önceki bölümde incelenen while() döngü yapısı bir çok problemin çözümünde yeteli olmaktadır. Bir döngünün çevrim sayısının esas alınarak tasarlandığı durumlarda ise daha dolaylı ve basitçe yazabilmek için for() döngü yapısı kullanılabilir. Bu döngü yapısı basitçe “**sayaç döngü yapısı – counted / indexed loop**” olarak isimlendirilebilir. Bir for döngüsün yapısı aşağıdaki gibidir.

```
for(başlangıç ifadesi; döngü koşulu; güncelleme ifadesi)
{
    ...
}
```

Bir for döngüsü şu kurallara işlenir:

- 1 Başlangıç ifadesi işlenir.
- 2 Döngü koşulu işlenir ve mantıksal sonucu belirlenir. Buna göre:
  - 2.1 Eğer mantıksal sonuç **doğru – true** ise
    - 2.1.1 Döngü çevrimi başlar.
    - 2.1.2 Döngü gövdesindeki ifadeler işlenir.
  - 2.2 Eğer mantıksal sonuç **yanlış – false** döngü işlenmez, izleyen kod blokuna geçiş yapılır.

Aşağıda bir for döngüsünün işleyişi gösterilmiştir.



Aşağıdaki örnek kod for döngüsünü kullanarak 1'den başlayarak 10'a kadar sayıları yazdırmaktadır.

```

for(i=1; i<=10; i++)
{
    cout << i << endl;
}
  
```

Döngü yapıları içerisinde for() yaygın olarak kullanılan basit bir döngü yapısıdır. Ancak bu döngü yapısı tasarlanıp program içerisinde uygulanırken şu noktalara dikkat edilmesi gereklidir.

- Eğer döngüde tanımlanmış olan başlangıç koşulunun mantıksal sonucu yanlış – **false** ise döngü yapısı işlenmez.
- Güncelleme ifadesi işlendiğinde kontrol değişkeni yani sayıç olarak kullanılan değişkenin değeri güncellenir. Bu güncelleme sonucun döngü koşulunun mantıksal değerinin **yansız – false** olmasına kadar döngü işlenir. Döngü koşulunun mantıksal sonucu **doğru – true** olduğu sürece döngü işlenir.
- C++ Programlama Dili’nde for döngülerinde kayar noktalı sayılar kullanılabilir. Ancak bu özellik kullanılmamalıdır. Çünkü farklı derleyiciler kullanılarak derlenen kayar nokta kul-

lanılarak kurgulanan değerlendirme ifadelerinde programda beklenmeyen sonuçlar döndürülmemekte ve döngü işlemekte veya işlerse de hatalı çalışmaktadır.

- Döngünün tanımlandığı for() ifadesinin sonunda ";" kullanılırsa bu döngü asla işlevsizdir. Çünkü bu anlamsal olarak hatalı bir yazım şeklidir. Derleyici için döngü gövdesi boştur ve döngü sırasında herhangi bir işlem yapılmaz.
- Bir for döngüsüne döngü koşulu yazılmış ise mantıksal sonucu her zaman için **doğru – true** kabul edilir. Sonsuz döngü yaratılmış olur.
- Aynı şekilde bir for döngüsü tanımlanırken, başlangıç ifadesi, döngü koşulu ve güncelleme ifadesi yazılmalıdır ama ";" kullanılırsa bu durumda da bir sonsuz döngü yaratılmış olur. Örneğin for (;;) { ... }

Aşağıdaki for döngü yapısı için örnekler verilmiştir.

Aşağıda verilen program kullanıcı tarafından verilen bir pozitif sayıdan başlayarak sıfıra doğru geri sayım yapmakta ve ekrana yazmaktadır.

```
/*
 * geri_sayim.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
```

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
 \* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
 \* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
 \* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
 \* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
 \* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
 \* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP  
 \* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
 \* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
 \* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
 \* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
 \* SORUMLU DEĞİLLERDİR.  
 \*  
 \*/  
 /\*  

```

#include <iostream>
#include <locale>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    int i=0,sayı=0;
    cout << "Bir pozitif tam sayı giriniz. ";
    cin >> sayı;
    cout << endl;
    for(i=sayı;i>=0;i--)
    {
        // Geri sayılm sonucunu ekrana yazdırır.
        cout << i << " ";
    }
}

```

```
    cout << endl;  
  
    return 0;  
}
```

Programın derlenip çalıştırıldığında aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./geri_sayim
```

```
Bir pozitif tam sayı giriniz. 13
```

```
13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Döngülerde kontrol değişkeninin arttırılması veya azaltılması hem pozitif hem de negatif olarak yapılabilir. Artım veya azaltım ise birer birer yapılması için herhangi bir zorunluluk bulunamamaktadır. Aşağıdaki program 1'den başlayarak verilen üst sınıra kadar olan tek ve çift sayıları yazdırmaktadır.

```
/*  
  
 * tek_cift_sayilar.cxx  
  
 *  
  
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>  
 * *  
  
 * Her hakkı saklıdır.  
  
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak  
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği  
 * takdirde serbesttir:  
  
 *  
  
 * Kaynak kodun yeniden dağıtıımı; yukarıdaki telif hakkı uyarısını,  
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.  
 *  
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,  
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen  
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.  
 *
```

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
```

```
#include <locale>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    // Dil ayarları
```

```
    setlocale(LC_ALL, "tr_TR.UTF-8");
```

```
    int i, j, sayı;
```

```
    cout << "Bu program 1'den başlayarak belirtilen üst sınıra kadar olan tek sayıları hesaplamaktadır. " << endl;
```

```
    cout << "Bir üst sınır belirtiniz ve enter tuşuna basınız: ";
```

```
    cin >> sayı;
```

```
    cout << endl;
```

```
    for(i=1; i <= sayı; i = i + 2)
```

```

{
    // Tek sayılar yazdırılır.

    cout << i << " ";

}

cout << endl;

for(j=2; j <= sayi; j = j + 2)

{
    // çift sayılar yazdırılır.

    cout << j << " ";

}

cout << endl;

return 0;
}

```

Programın derlenip çalıştırıldığında aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./tek_cift
```

Bu program 1'den başlayarak belirtilen üst sınıra kadar olan tek sayıları hesaplamaktadır.

Bir üst sınır belirtiniz ve enter tuşuna basınız: 13

Tek sayılar:

1 3 5 7 9 11 13

Çift Sayılar:

2 4 6 8 10 12

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Aşağıda verilen program örneklerinde çeşitli for döngüleri gösterilmiş ve açıklanmıştır.

```

for(i =11; i <=10; i++)

{
    cout << i << endl;
}

```

Yukarıdaki döngünün bir program içerisinde kullanılması durumunda döngünün çalışmaya-  
ceği görülmektedir. Döngünün başlangıç ifadesinde değişken 11 olarak tanımlanmış olsa da, döngü  
koşulu sağlanamayacağından bu döngü işlenmeyecektir.

```
for(i = 8; i >=10; i--)
```

```
{  
    cout << i << endl;  
}
```

Yukarıdaki döngünün bir program içerisinde kullanılması durumunda döngünün çalışmaya-  
cağı görülmektedir. Döngünün başlangıç ifadesinde değişken 8 olarak tanımlanmış ancak döngü  
koşulu mantıksal olarak “**yansı – false**” sonuç döndürecekinden bu döngü işlenmeyecektir.

```
for(i = 10; i >=10; i++)  
{  
    cout << i << endl;  
}
```

Yukarıdaki döngünün bir program içerisinde kullanılması durumunda döngü sadece bir defa  
işlenir. Döngünün başlangıç ifadesinde değişken 10 olarak tanımlanmış ve döngü koşulu mantıksal  
olarak “**doğru – true**” sonuç döndürecektir. Döngü ilk çevrimden sonra başlangıç ifadesindeki  
değişkenin değerini güncelleyecek ve 11 yapacaktır. Bu işlemden sonra döngü koşulu mantıksal  
olarak “**yansı – false**” değerini döndürecektir. Bu nedenle döngü çevrimi bir defa gerçekleşir.

```
for(i = 10; i >=10; i++);  
{  
    cout << i << endl;  
}
```

Yukarıdaki döngüde, for() döngüsün sonunda bulunan “;” nedeni ile döngünün gövdesi  
bulunmamaktadır. Döngü işlenmiş olsa da gövde bulunmadığı için bekleniği gibi sonuç döndür-  
meyecektir. Döngünün başlangıç ifadesindeki değişken olan i döngü sona erdiğinde değeri 11 ola-  
cağından sadece 11 yazdırılacaktır.

```
for(i = 10; ; i++);  
{  
    cout << i << endl;  
}
```

Yukarıdaki döngüde, for() döngüsün yapısında kontrol ifadesi yer almadığı için döngü  
program içerisinde kullanılırsa sonsuz döngü olarak çalışacaktır. Program akışı bu döngünün içinde  
kalır ve dışına geçmez.

```
for(i = 1; i < 100; i= i * 4);  
{  
    cout << i << endl;  
}
```

Yukarıdaki döngüde, for() döngüsün yapısındaki güncelleme ifadesinde, başlangıç ifade-sinde yer alan i değişkeni, dördün katları olarak güncellenmektedir. Kontrol ifadesine göre de i değişkenin büyülüğu 100'den büyük olamaz. Program içerisinde kullanıldığındá ise çıktı olarak “1, 4, 16, 64” yazdıracaktır.

**Örnek Program:** Aşağıdaki program önceki bölümde yer verilen ve while() döngüsü kullanılarak yazılmış olan Fibonacci sayılarının hesaplanmasıında for() döngüsünü kullanmıştır.

```
/*
 * fibonacci_sayisi1.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
```

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
#include <locale>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişkenler
    int son, sayac, istenen;
    int birinci = 1, ikinci = 1;
    cout << "Bu program belirtilen sayıdaki Fibonacci sayılarını bulur." << endl;
    cout << "Hesaplanacak olan Fibonacci sayısı adedini belirtiniz ve enter tuşuna basınız: ";
    cin >> istenen;
    if(istenen == 1)
    {
        son = birinci;
        cout << "İstenen Fibonacci sayısı: " << birinci;
    }
    else if (istenen == 2)
    {
        son = ikinci;
        cout << "İstenen Fibonacci sayıları: " << birinci << " " << ikinci;
```

```

    }

else

{

    cout << "İstenen Fibonacci sayıları: " << endl;

    cout << birinci << " " << ikinci << " ";

    for(sayac=3; sayac <= istenen; sayac++)

    {

        son = birinci + ikinci;

        birinci = ikinci;

        ikinci = son;

        cout << son << " ";

    }

}

cout << endl;

return 0;
}

```

Programın derlenip çalıştırıldığında aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./fibonacci_sayıları1
```

Bu program belirtilen sayıdaki Fibonacci sayılarını bulur.

Hesaplanacak olan Fibonacci sayısı adedini belirtiniz ve enter tuşuna basınız: 5

İstenen Fibonacci sayıları:

```
1 1 2 3 5
```

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

**Örnek Program:** Aşağıdaki program birden başlayarak kullanıcı tarafından belirtilen bir pozitif tam sayıya kadar olan tüm pozitif tam sayıların karesini, küpünü ve kare kökünü hesaplayarak yazdırmaktadır.

```
/*
 * tam_sayı_kare_küp_kök.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
```

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

#include <iostream>

```

#include <locale>
#include <cmath>
#include <cfloat>
#include <iomanip>
using namespace std;

int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");

    // Değişkenler
    unsigned int i, son, kare, kup;
    double kok;

    cout << "Bu program birden başlayarak kullanıcı tarafından belirtilen bir pozitif tam sayıya kadar olan tüm pozitif tam sayıların karesini, küpünü ve kare kökünü hesaplayarak yazdırmaktadır. Girilebilecek pozitif tam sayı için üst sınır 1625'tir." << endl;

    cout << "Hesaplamalar için bir pozitif tam sayı giriniz ve enter tuşuna basınız: ";
    cin >> son;

    if(son <= 1625)
    {
        cout << left << fixed << setprecision(4);
        for(i=1; i <= son; i++)
        {
            kare = pow (i,2);
            kup = pow (i,3);
            kok = pow (i, 0.5);

            cout << setw(10) << i << setw(10) << kare << setw(10) << kup << setw(10)
            << setw(10) << kok << endl;
        }
    }
    else
    {
        cout << "Hesaplanabilir aralığın dışında bir sayı girdiniz. Program sonlanıyor." << endl;
    }
}

```

```

        return 1;
    }

    cout << endl;

    return 0;
}

```

Programın derlenip çalıştırıldığında aşağıdakine benzer bir çıktı elde edilecektir. Çıktının uzunluğunun sınırlanılması için ilk değer olarak 5 kullanılmıştır. Programın kullandığı veri tipi işaretetsiz tam sayı olduğu için hesaplanabilir aralığın dışına çıkmaması için işaretetsiz tam sayı veri tipinin küçük köküne en yakın pozitif tam sayı sınır olarak belirlenmiştir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./tam_sayı_kare_kup_kok
```

Bu program birden başlayarak kullanıcı tarafından belirtilen bir pozitif tam sayıya kadar olan tüm pozitif tam sayıların karesini, küpünü ve kare kökünü hesaplayarak yazdırmaktadır. Girilebilecek pozitif tam sayı için üst sınır 1625'tir.

Hesaplamlar için bir pozitif tam sayı giriniz ve enter tuşuna basınız: 20

```

1      1      1      1.0000
2      4      8      1.4142
3      9      27     1.7321
4      16     64     2.0000
5      25     125    2.2361

```

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

## 4.5.break ve continue İfadeleri

Önceki bölümlerde incelenen “**switch()** ... **case**” yapısında **break** kullanıldığından işlem sona ermektedir ve program bu yapının dışına çıkararak devam etmeyecektir. Aynı şekilde **break** ifadesi bir döngüde kullanılacak olursa, söz konusu döngü içerisinde bu ifadeye gelindiğinde döngü sonlanacaktır ve program akışı aynı şekilde döngü yapısının dışındaki kod blokuna geçerek devam edecektir. Buradan da anlaşılabileceği üzere break ifadesi iki amaçla kullanılmaktadır:

1. Döngünün tamamlanmadan sonlandırılması gerekiyorsa
2. Bir “**switch()** ... **case**” yapısının sonuna kadar işlenmesi gerekliliğinde

Aşağıdaki kod blokunda bir program içerisinde okunan verilerin içerisinde negatif veri olması durumunda işlem yapılmadan sonraki veriye geçilerek devam edilmektedir. Kod bloku önce girdi akım değişkeninin “**doğru – true**” ve okunan değişkenin de negatif olmaması koşulu geçerli oldukça yani “**doğru – true**” olduğu sürece işlem yapmaktadır. Bu koşullardan negatif olmama koşulu sağlanmaz ise işlem yapılmamaktadır.

```
unsigned int toplam = 0;
```

```

bool girdi_negatif = false;
cin >> deger;
while(cin && !girdi_negatif)
{
    if (deger < 0 )
    {
        cout << "Negatif değer bulundu! << endl;
        girdi_negatif = true;
    }
    else
    {
        toplam = toplam + deger;
        cin >> deger;
    }
}

```

Yukarıdaki kod blokundaki döngüden girilen sayıların toplamını bulmaktadır. `girdi_negatif` adlı değişken mantıksal veri tipinde tanımlanmış ve değeri yanlış – `false` olarak tanımlanmıştır. Eğer `girdi` içerisinde negatif sayı bulunursa bunun bildirilmesi için değeri doğru – `true` olarak değiştirilmektedir. Böylece döngü bir sonraki çevrime geçtiğinde ifadenin mantıksal sonucu yanlış – `false` olmakta ve döngü işlenmemektedir. Yukarıdaki kod bloku mantıksal değişken olmadan yeniden yazılmış ve negatif sayı olması durumunda döngünü sonlanması için `break` kullanılmıştır.

```

unsigned int toplam = 0;
cin >> deger;
while(cin)
{
    if (deger < 0 )
    {
        cout << "Negatif değer bulundu! << endl;
        break;
    }
    toplam = toplam + deger;
    cin >> deger;
}

```

}

Aşağıdaki program iki kısımdan oluşmaktadır. İlk kısımda toplama işlemi birden başlayarak ona kadar olan tüm sayıların toplamını bir döngü kullanılarak hesaplamaktadır. İkinci bölüm ise kullanıcının belirlediği sayıyı toplamın dışında tutmaktadır.

```
/*
* atla.cxx
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
* takdirde serbesttir:
*
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
*
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
*
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
```

```

* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
* SORUMLU DEĞİLLERDİR.

*
*/
/*
#include <iostream>
#include <locale>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişkenler
    unsigned int i=0, sayac=10, atla=0;
    int toplam=0;
    cout << "Bu programın ilk bölümü birden ona kadar olan sayıları toplamaktadır. İkinci
bölüm ise belirtilen bir sayıyı toplama dahil etmeden aynı işlemi yapmaktadır." << endl;
    cout << "Atlanacak olan sayıyı giriniz ve enter tuşuna basınız: ";
    cin >> atla;
    if (cin.fail() or (atla < 0) or (atla > 10))
    {
        cout << "Geçersiz bir sayı girdiniz. Program sonlanıyor." << endl;
        return 1;
    }
    for(i=1; i <= sayac; i++)
    {
        toplam = toplam + i;
    }
}

```

```

cout << "Sayıların eksiksiz toplamı= " << toplam << endl;
i=0;
toplams =0;
for(i=1; i <= sayac; i++)
{
    if (i == atla)
        continue;
    else
        toplams = toplams + i;
}
cout << "Sayıların toplamı= " << toplams << endl;
return 0;
}

```

Programın derlenip çalıştırıldığında aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./atla
```

Bu programın ilk bölümünden birden ona kadar olan sayıları toplamaktadır. İkinci bölüm ise belirtilen bir sayıyı toplama dahil etmeden aynı işlemi yapmaktadır.

Atlanacak olan sayıyı giriniz ve enter tuşuna basınız: 7

Sayıların eksiksiz toplamı= 55

Sayıların toplamı= 48

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

## 4.6.İç İçे Kurgulanın Döngüler

BAzı programlarda iç içe döngülerin kullanılması gerekebilir. Yapı olarak ilk döngü gövdesinde bir diğer döngünün yer olması şeklinde düşünülebilir. Bu tür döngülerin kurgulanması çok boyutlu diziler üzerinde gerçekleştirilen işlemlerde tercih edilir. Bu döngülerin işlenmesi için önce en dıştaki döngü ile işleme başlanır. Ardından da döngü gövdesinde yer ikinci döngü işlenmeye başlanır. İçeride yer alan ikinci döngü bitinceye kadar dışında bulunan ilk döngü işlenmez. Bu yapının doğası gereği ilk döngünün her çevriminde içeride yer alan döngü tekrarlanır. Bu tür döngülerin kurgulanmasında eğer döngülerin çevrim sayıları biliniyor ise for() döngü yapısı kolaylığı nedeni ile tercih edilir.

Aşağıdaki program ekran birden ona kadar olan sayılara ait çarpım tablosunu yazdırmaktadır.

```
/*
```

```
* carpim_tablo.cxx
*
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
* takdirde serbesttir:
*
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
*
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
*
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
* SORUMLU DEĞİLLERDİR.
```

```

/*
*/
/*
#include <iostream>
#include <locale>
#include <iomanip>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişkenler
    unsigned int i=0, j=0;
    cout << "Bu program birden ona kadar olan tam sayıların çarpım tablosunu yazdırır." <<
endl;
    cout << endl;
    for(i=1; i<=10; i++)
    {
        for (j=1; j<=10; j++)
        {
            cout << i << "*" << j << "=" << i*j <<"\t";
        }
        cout << endl;
    }
    cout << endl;
    return 0;
}

```

Programın derlenip çalıştırıldığında aşağıdakine benzer bir çıktı elde edilecektir.

[goksin@tardis ~/projeler/C++\_Notlar/]\$ ./carpim\_tablo

1\*1=1 1\*2=2 1\*3=3 1\*4=4 1\*5=5 1\*6=6 1\*7=7 1\*8=8 1\*9=9 1\*10=10

```

2*1=2 2*2=4 2*3=6 2*4=8 2*5=10 2*6=12 2*7=14 2*8=16 2*9=18 2*10=20
3*1=3 3*2=6 3*3=9 3*4=12 3*5=15 3*6=18 3*7=21 3*8=24 3*9=27 3*10=30
4*1=4 4*2=8 4*3=12 4*4=16 4*5=20 4*6=24 4*7=28 4*8=32 4*9=36 4*10=40
5*1=5 5*2=10 5*3=15 5*4=20 5*5=25 5*6=30 5*7=35 5*8=40 5*9=45 5*10=50
6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36 6*7=42 6*8=48 6*9=54 6*10=60
7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49 7*8=56 7*9=63 7*10=70
8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64 8*9=72 8*10=80
9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81 9*10=90
10*1=10 10*2=20 10*3=30 10*4=40 10*5=50 10*6=60 10*7=70 10*8=80 10*9=90 10*10=100
[goksin@tardis ~/projeler/C++_Notlar/]$

```

**Örnek Program:** Aşağıdaki program kullanıcı tarafından belirtilmiş olan sayıdaki ve klavyeden girilmiş olan pozitif ve/veya negatif tam sayıların en büyüğünü, en küçüğünü, bunların sırası ile kaçinci sırada girilmiş olduğunu bulan programı yazınız.

**Çözüm:** Program çalıştırıldığında girilecek olan sayıların adedi bilinmemektedir. Kullanıcıdan gireceği tam sayı adedi istenecektir. Tam sayı adedi ile döngü kurulacaktır. Döngü ile klavyeden girilen tam sayılar karşılaştırılacak en büyük ve en küçük olanı bulunacaktır. En büyük ve en küçük sayıların girildiği sıra da belirlenecek ve yazdırılacaktır.

#### Algoritma:

- 1 Başla
- 2 Sayı adedini oku.
- 3 En büyük sayı o olsun.
- 4 En küçük sayı 0 olsun.
- 5 Sayac=0 olsun.
  - 5.1 Klavyeden sayı girilsin.
  - 5.2 Girilen sayı sayı değişkenine atansın
  - 5.3 Girilen sayı 0'dan büyük mü? Büyükse en büyük sayı, okunan sayı ile değiştirilsin ve en büyük sayı sırası olarak sayac değeri atansın.
  - 5.4 Girilen sayı 0'dan küçük mü? Küçükse en küçük sayı, okunan sayı ile değiştirilsin ve en küçük sayı sırası olarak sayac değeri atansın.
  - 5.5 Sayaç bir artırılsın.
  - 5.6 Sayac değeri sayı adedinden küçük veya eşit ise 5.1 adımına gidilsin değilse 6 adıma geçilsin.
- 6 En büyük sayı yazdırılsın.

7 En büyük sayının sırası yazdırılsın.

8 En küçük sayı yazdırılsın.

9 En küçük sayının sırası yazdırılsın.

10 Son

Programın kaynak kodu aşağıdaki gibidir:

```
/*
 * en_buyuk_en_kucuk.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
```

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
```

```
#include <locale>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    // Dil ayarları
```

```
    setlocale(LC_ALL, "tr_TR.UTF-8");
```

```
    // Değişkenler
```

```
    unsigned int i=0, sayac=2;
```

```
    int en_buyuk=0, en_kucuk=0, en_buyuk_sira=1, en_kucuk_sira=1, sayi=0, son=0;
```

cout << "Bu program klavyeden girilen belirtilen adetteki tam sayıları okuyup en büyük ve en küçük olanları ve sırası ile bunların kaçinci sırada girildiğini belirtir. Girilmesi gereken en az sayı adedi ikidir. Her giriş sonrası enter tuşuna basınız. " << endl;

cout << "İşlenecek olan tam sayı adedini pozitif tam sayı olarak giriniz ve enter tuşuna basınız: ";

```
        cin >> son;
```

```
        if (cin.fail() or (son < 0))
```

```
{
```

```
            cout << "Geçersiz bir sayı girdiniz. Program sonlanıyor." << endl;
```

```
            return 1;
```

```
}
```

```
        sayac = son;
```

```

if(sayac >= 2)
{
    for(i=1; i <= sayac; i++)
    {
        cout << i << ". sayıyı giriniz: ";
        cin >> sayi;
        if (cin.fail())
        {
            cout << "Geçersiz bir sayı girdiniz. Program sonlanıyor." << endl;
            break;
        }
        if (sayi>=en_buyuk)
        {
            en_buyuk = sayi;
            en_buyuk_sira =i;
        }
        else if (sayi <= en_kucuk)
        {
            en_kucuk = sayi;
            en_kucuk_sira = i;
        }
    }
    cout << "Girilen en büyük sayı: " << en_buyuk << endl;
    cout << "Girilen en büyük sayının sırası: " << en_buyuk_sira << endl;
    cout << "Girilen en küçük sayı: " << en_kucuk << endl;
    cout << "Girilen en küçük sayının sırası: " << en_kucuk_sira << endl;
}
else
{
    cout << "Geçersiz sayı adedi girildi. Program sonlanıyor." << endl;
    return 1;
}

```

```
    }  
  
    return 0;  
}
```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./en_buyuk_en_kucuk
```

Bu program klavyeden girilen belirtilen adetteki tam sayıları okuyup en büyük ve en küçük olanları ve sırası ile bunların kaçinci sırada girildiğini belirtir. Girilmesi gereken en az sayı adedi ikidir. Her giriş sonrası enter tuşuna basınız.

İşlenecek olan tam sayı adedini pozitif tam sayı olarak giriniz ve enter tuşuna basınız: 5

1. sayıyı giriniz: -5
2. sayıyı giriniz: 0
3. sayıyı giriniz: 1
4. sayıyı giriniz: -4
5. sayıyı giriniz: 6

Girilen en büyük sayı: 6

Girilen en büyük sayının sırası: 5

Girilen en küçük sayı: -5

Girilen en küçük sayının sırası: 1

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Programın içerisinde karşılaştırma yapılacak olan sayıların tam sayı olma koşulu vardır. Eğer gerçel sayılar girilirse, veri tipindeki uyumsuzluk nedeni ile girdi akım değişkeni hata döndürür. Bu nedenle hatalı durumun belirlenmesi ve programın sona ermesi için “**cin.fail()**” koşul içerisinde kullanılarak kontrol işlemi kurgulanmıştır. Eğer hatalı giriş olursa program sonlanmaktadır. Girilen sayıların en büyüğü ve en küçüğü için ilk değer sıfırdır. Girilen ilk sayı sıfırdan büyükse en büyük sayı, eğer sıfırdan küçük ise en küçük sayı olarak kabul edilir. Girilen her sayı en büyük ve en küçük sayı ile karşılaştırılır, eğer herhangi birisi var olan değerden büyükse veya küçükse bu değer ilgili değişkene atanır ve sıra numarası da saklanır. Döngü bittiğinde program sonuçları ekrana yazdırır.

Yukarıdaki çarpım tablosu örneğinde döngülerin çevrim sayısı bilindiği için for() döngüsü kullanılmaktadır. Ancak her problemin çözümünde for() döngüsünün kullanılması söz konusu olmayabilir. Aşağıdaki program bu for() döngü yapısı kullanılamayan bir problemde while() döngülerinin nasıl kullanılacağını göstermektedir.

**Örnek Program:** Bir üniversitedeki kullanılan başarı notu belirleme sistemi öğrencilerin iki adet dönem içi sınavı, iki adet ödev notu ile bir adet dönem sonu notunun standart oranlar ile çarpılarak toplamlarının alınması ile belirlenmektedir. Bu elde edilen not daha sonra önceden belirlenmiş harf

notu aralıklarına karşılık getirilecek öğrencinin harf notu belirlenmektedir. Harf notları ders için öğrenci özelinde belirlenmektedir. Bir dersin açılması için gereken öğrenci sayısının alt sınırı beştir. Dersleri alan öğrenci sayısı için üst sınır bulunamaktadır. Program veri olarak öğrenci adı ve soyadı, ara sınav ve ödev notları ile dönem sonu sınav notlarını okuyarak başarı notunu hesaplayıp işlem sonuçlarını bir dosyaya yazacaktır.

**Çözüm:** Programın yazılabilmesi için başarı notunun hesaplanması gerekecek olan not katsayıları gerekecektir. Bunlar ara sınav notlarının her birisi için %15, ödev notlarının her birisi için 510 ve dönem sonu sınav notu için de %50'dir. Öğrencinin sınav notu aralıkları 0 ile 100 arasındadır. Harf notlarının aralıkları da aşağıda verilmiştir.

AA	100 – 84	AB	83 – 77
BA	76 – 71	BB	70 – 66
BC	65 – 61	CB	60 – 56
CC	55 – 50	CD	49 – 46
DC	45 – 40	DD	39 – 33
FF	32 – 0		

Programın işleyişine ilişkin özel koşullar şunlardır:

- Program değerlendirme verisini notlar.txt adlı dosyadan okuyacaktır.
- Program değerlendirme verisi dosyası bulunmuyorsa, dosyadaki not verilerinden herhangi birisi hatalı ise (100 ile 0 aralığında değilse veya sayı olarak yazılmamış ise) hatayı belirtip sonlanacaktır.
- Hatalar kayıt altına alınıp “hata.txt” adlı dosyaya yazılacaktır.
- İşlenen veriler “başarı\_notu.txt” adlı dosyaya yazılacaktır.

#### **Programın algoritması:**

- 1 Başla
- 2 “notlar.txt” dosyasını okumak için aç.
- 3 Hataları kayıt atlana almak için “hata.txt” dosyasını yazmak için aç.
- 4 “notlar.txt” dosyasını kontrol et.
  - 4.1 Dosya bulunmuyorsa;
    - 4.1.1 “hata.txt” dosyasına “dosya bulunamadı mesajını yaz.”
    - 4.1.2 Standart çıktıya hata mesajını yaz
    - 4.1.3 Hata kodu 1 ile program sonlandır.
- 5 “başarı\_notu.txt” dosyasını yazmak için aç.
- 6 Öğrencinin adını oku, soyadını oku.

- 7 Eğer dosyanın sonuna gelinmiş
  - 7.1 "veri işleme tamamlandı" mesajını standart çıktıya yaz
  - 7.2 Açılan tüm dosyaları kapat.
  - 7.3 Programı başarılı olarak sonlandır.
- 8 Öğrencinin 1. ara sınav notunu oku. Eğer;
  - 8.1 Not verisi 0 ile 100 aralığında mı yoksa farklı mı kontrol et, bu koşullar sağlanmıyorsa,
    - 8.1.1 "hata.txt" dosyasına Öğrencinin "adını, soyadını ve hatalı 1. ara sınav notu" mesajını yaz.
    - 8.1.2 Hatalı veri belirlendiğini belirten mesajı yaz.
    - 8.1.3 Açılan tüm dosyaları kapat.
    - 8.1.4 Programı hata kodu 2 ile sonlandır.
- 9 Öğrencinin 2. ara sınav notunu oku. Eğer;
  - 9.1 Not verisi 0 ile 100 aralığında mı yoksa farklı mı kontrol et, bu koşullar sağlanmıyorsa,
    - 9.1.1 "hata.txt" dosyasına Öğrencinin "adını, soyadını ve hatalı 2. ara sınav notu" mesajını yaz.
    - 9.1.2 Hatalı veri belirlendiğini belirten mesajı yaz.
    - 9.1.3 Açılan tüm dosyaları kapat.
    - 9.1.4 Programı hata kodu 2 ile sonlandır.
- 10 Öğrencinin 1. ödev notunu oku. Eğer;
  - 10.1 Not verisi 0 ile 100 aralığında mı yoksa farklı mı kontrol et, bu koşullar sağlanmıyorsa,
    - 10.1.1 "hata.txt" dosyasına Öğrencinin "adını, soyadını ve hatalı 1. ödev notu" mesajını yaz.
    - 10.1.2 Hatalı veri belirlendiğini belirten mesajı yaz.
    - 10.1.3 Açılan tüm dosyaları kapat.
    - 10.1.4 Programı hata kodu 2 ile sonlandır.
- 11 Öğrencinin 2. ödev notunu oku. Eğer;
  - 11.1 Not verisi 0 ile 100 aralığında mı yoksa farklı mı kontrol et, bu koşullar sağlanmıyorsa,
    - 11.1.1 "hata.txt" dosyasına öğrencinin "adını, soyadını ve hatalı 2. ödev notu" mesajını yaz.

- 11.1.2 Hatalı veri belirlendiğini belirten mesajı yaz.
  - 11.1.3 Açılan tüm dosyaları kapat.
  - 11.1.4 Programı hata kodu 2 ile sonlandır.
- 12 Öğrencinin done sonu sınav notunu oku. Eğer;
- 12.1 Not verisi 0 ile 100 aralığında mı yoksa farklı mı kontrol et, bu koşullar sağlanmıyorsa,
    - 12.1.1 "hata.txt" dosyasına Öğrencinin "adını, soyadını ve hatalı dönem sonu sınav notu" mesajını yaz.
    - 12.1.2 Hatalı veri belirlendiğini belirten mesajı yaz.
    - 12.1.3 Açılan tüm dosyaları kapat.
    - 12.1.4 Programı hata kodu 2 ile sonlandır.
- 13 Öğrencinin başarı notunu ( $k_1AraSınavNotu + k_2AraSınavNotu + k_3ÖdevNotu + k_4ÖdevNotu + k_5DönemSonuSınavNotu = BaşarıNotu$ )
- 14 Öğrencinin başarı notuna karşılık gelen harf notu değerini belirle.
- 15 Öğrencinin adı, soyadı, dönem içi ve dönem sonu notlarını, başarı notunu ve harf notunu "başarı\_notu.txt" dosyasına yaz.
- 16 Açılan "başarı\_notu.txt" dosyasını kapat.
- 17 Açılan "kaynak.txt" dosyasını kapat.
- 18 Açılan "hata.txt" dosyasını kapar.
- 19 Programı başarı ile sonlandır.
- 20 Son

**Programın Yazılması:** Programın işleyeceği veriler öğrenci adı ve soyadı, öğrenci notlarıdır. **Öğrenci adı ve soyadı karakter dizisi – string** olarak tanımlanacaktır. **Öğrenci notları**, 0 ile 100 arasında olduğu için **tam sayı – int** veri tipinde tanımlanabilir. Bu aralık dışında olabilecek olası hatalı veriler, karakter veya negatif sayılar olabilir. Bunların kontrol edilmesinde girdi akımındaki hata durumu izlenerek yapılabilir. Verilerin olmaması durumu için **G/Ç İşlemleri** yardımı ile kontrol edilebilir. Çıktı dosyalarının önceden hazırlanmasına gerek yoktur. İşletim sistemi bunları oluşturacaktır. Bir dersi alan öğrenci sayısının alt sınırı bulunmakla birlikte üst sınır yoktur. Bu durumda programda for() döngüsü kullanılamaz. Bunun yerine **while()** döngüsü kullanılabilir. Döngünün sona ermesi için gerekli olan koşul dosyanın sonuna gelinmiş olması koşuludur. Belirtilen hata durumlarına program sona erme koşulları algoritmda tanımlanmıştır. Algoritma ayrıntılı olarak tasarlandığı için doğrudan algoritma esa alınarak program yazılacaktır. Programın verileri işlemesi sırasında program akışının izlenmesi için program içerisinde hata mesajları ve ilerlemeye durumunu belirten standart çıktıya yazdırılacaktır.

Programın kaynak kodu aşağıdaki gibidir:

\*  
\* harf\_not\_ortalama\_hata\_eof.cxx  
\*  
\*  
\* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>  
\* \*  
\* Her hakkı saklıdır.  
\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak  
\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği  
\* takdirde serbesttir:  
\*  
\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,  
\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.  
\*  
\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,  
\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen  
\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.  
\*  
\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

```
* SORUMLU DEĞİLLERDİR.  
*  
*/  
/*  
  
#include <iostream>  
  
#include <locale>  
  
#include <fstream>  
  
#include <string>  
  
#include <cfloat>  
  
#include <iomanip>  
  
using namespace std;  
  
int main()  
{  
    // Dil ayarları  
    setlocale(LC_ALL, "tr_TR.UTF-8");  
  
    // Dosya okuma ve yazma işlemleri için değişkenlerin tanımlanması  
    ifstream dosya_ac_oku;  
    ofstream dosya_ac_yaz;  
    ofstream dosya_hata_yaz;  
  
    // Öğrencinin adı ve soyadı  
    string ad, soyad, harf_not;  
  
    // Değişken tanımlamaları: Öğrencinin notları ve ortalama  
    float sinav1=0.0, sinav2=0.0, odev1=0.0, odev2=0.0, donemsonu=0.0, ortalama=0.0;  
  
    // Sabitler: Sınav yüzdeleri ve sonlanma kodu  
    const float k1=0.15, k2=0.15, k3=0.10, k4=0.10, k5=0.50;  
  
    // Okunacak olan dosya  
    dosya_ac_oku.open("notlar.txt");  
  
    // Hataların kayıt edileceği dosya  
    dosya_hata_yaz.open("hata.txt");  
  
    if (!dosya_ac_oku)  
    {
```

```

// Dosyaya hata kayıt edildi.

dosya_hata_yaz << "hata1" << endl << "Kaynak dosya olan \"notlar.txt\" bulanamadı." << endl;

// Kullanıcıya bilgi verildi.

cout << "Kaynak dosya olan \"notlar.txt\" bulanamadı." << endl;
cout << "Program sonlanıyor. Hata kodu: 1" << endl;

// Hata bildirimi yapıldı, dosya kapatıldı.

dosya_hata_yaz.close();

// Program sona erdi.

return 1;
}

// Yazılacak olan dosya açılır, yoksa işletim sistemi oluşturur.

dosya_ac_yaz.open("başarı_notu.txt");

cout << "Veriler dosyadan okunuyor ve işleniyor." << endl;
while(dosya_ac_oku)

{
    // Dosyadan veriler okunuyor.

    dosya_ac_oku >> ad >> soyad;

    // Dosyada okunacak satır bulunmuyor ise döngüyü sonlandıralım.

    if (dosya_ac_oku.eof())

        break;

    // Kullanıcıya işlem hakkında bilgi verilir.

    cout << "İşlem yapılan öğrenci: " << left << setw (10) << ad << setw(10) << soyad
<< endl;

    // 1. Ara sınav notu kontrol edilir.

    dosya_ac_oku >> sinav1;

    if ((sinav1 < 0.0) || (sinav1 > 100.0))

    {
        // Dosyaya hata kayıt edildi.

        dosya_hata_yaz << "hata2" << " " << ad << " " << soyad << endl;

        // Kullanıcıya işlem hakkında bilgi verilir.

```

```
cout << "Öğrencinin birinci sınav notu hatalı girilmiştir. İşlem sonlandırıldı."
<< endl;

cout << "Hata kodu 2" << endl;
// Hata bildirimi yapıldı, dosyalar kapatılır.

dosya_ac_yaz.close();
dosya_ac_oku.close();
dosya_hata_yaz.close();

return 2;
}

dosya_ac_oku >> sinav2;
if ((sinav2 < 0.0) || (sinav2 > 100.0))
{
    // Dosyaya hata kayıt edildi.

    dosya_hata_yaz << "hata2" << " " << ad << " " << soyad << endl;
    cout << "Öğrencinin ikinci sınav notu hatalı girilmiştir. İşlem sonlandırıldı."
<< endl;

cout << "Hata kodu 2" << endl;
// Hata bildirimi yapıldı, dosyalar kapatılır.

dosya_ac_yaz.close();
dosya_ac_oku.close();
dosya_hata_yaz.close();

return 2;
}

dosya_ac_oku >> odev1;
if ((odev1 < 0.0) || (odev1 > 100.0))
{
    // Dosyaya hata kayıt edildi.

    dosya_hata_yaz << "hata2" << " " << ad << " " << soyad << endl;
    cout << "Öğrencinin birinci ödev notu hatalı girilmiştir. İşlem sonlandırıldı."
<< endl;

cout << "Hata kodu 2" << endl;
```

```

// Hata bildirimi yapıldı, dosyalar kapatılır.

dosya_ac_yaz.close();

dosya_ac_oku.close();

dosya_hata_yaz.close();

return 2;

}

dosya_ac_oku >> odev2;

if ((odev2 < 0.0) || (odev2 > 100.0))

{

    // Dosyaya hata kayıt edildi.

    dosya_hata_yaz << "hata2" << " " << ad << " " << soyad << endl;

    cout << "Öğrencinin ikinci ödev notu hatalı girilmiştir. İşlem sonlandırıldı." << endl;

    cout << "Hata kodu 2" << endl;

    // Hata bildirimi yapıldı, dosyalar kapatılır.

    dosya_ac_yaz.close();

    dosya_ac_oku.close();

    dosya_hata_yaz.close();

    return 2;

}

dosya_ac_oku >> donemsonu;

if ((donemsonu < 0.0) || (donemsonu > 100.0))

{

    // Dosyaya hata kayıt edildi.

    dosya_hata_yaz << "hata2" << " " << ad << " " << soyad << endl;

    cout << "Öğrencinin dönem sonu sınav notu hatalı girilmiştir. İşlem sonlandı." << endl;

    cout << "Hata kodu 2" << endl;

    // Hata bildirimi yapıldı, dosyalar kapatılır.

    dosya_ac_yaz.close();

    dosya_ac_oku.close();

```

```

        dosya_hata_yaz.close();

        return 2;

    }

    ortalama = (k1 * sinav1) + (k2 * sinav2) + (k3 * odev1) + (k4 * odev2) + (k5 *
donemsonu);

    // Harf notunun hesaplanması

    if (ortalama>=84.0)

        harf_not = "AA";

    else if(ortalama>=77.0)

        harf_not = "AB";

    else if(ortalama>=71.0)

        harf_not = "BA";

    else if(ortalama>=66.0)

        harf_not = "BB";

    else if(ortalama>=61.0)

        harf_not = "BC";

    else if(ortalama>=56.0)

        harf_not = "CB";

    else if(ortalama>=50.0)

        harf_not = "CC";

    else if(ortalama>=46.0)

        harf_not = "CD";

    else if(ortalama>=40.0)

        harf_not = "DC";

    else if (ortalama>=33.0)

        harf_not = "DD";

    else harf_not = "FF";

    // Çıktının biçimlendirilmesi

    dosya_ac_yaz << left << setw (10) << ad << setw(10) << soyad << fixed <<
showpoint << setprecision(2) << setw(6) << sinav1 << setw(6) << sinav2 << setw(6) << odev1 <<
setw(6) << odev2 << setw(6) << donemsonu << setw(6) << ortalama << setw(4) << harf_not <<
endl;

```

```
}

// Veriler okundu ve okuma işlemi sona erdi.

dosya_ac_oku.close();

// Dosya yazma işlemi sonlanıyor

dosya_ac_yaz.close();

// Hata bulunmadan program sona erdi.

dosya_hata_yaz.close();

return 0;

}
```

Programın işleyeceği verilerin yer aldığı notlar dosyasının içeriği aşağıdaki gibidir.

```
Ahmet Selim 77 98 76 56 98
```

```
Ayaz Mert 34 56 65 78 88
```

```
Zeynel Kaya 54 67 87 73 54
```

```
Ali Can 45 67 56 77 80
```

```
Kemal Demir 43 56 55 67 80
```

```
Cem Balta 43 32 68 59 78
```

```
Necati Uzun 78 54 80 65 55
```

```
Asya Nur 45 65 76 89 53
```

```
Zeynep Atabek 54 63 50 60 59
```

Bu dosya “notlar.txt” adı ile kayıt edilip programın bulunduğu dizine kayıt edilecektir. Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./harf_not_ortalama_hata_eof
```

Veriler dosyadan okunuyor ve işleniyor.

İşlem yapılan öğrenci: Ahmet Selim

İşlem yapılan öğrenci: Ayaz Mert

İşlem yapılan öğrenci: Zeynel Kaya

İşlem yapılan öğrenci: Ali Can

İşlem yapılan öğrenci: Kemal Demir

İşlem yapılan öğrenci: Cem Balta

İşlem yapılan öğrenci: Necati Uzun

İşlem yapılan öğrenci: Asya Nur

İşlem yapılan öğrenci: Zeynep Atabek

[goksin@tardis ~/projeler/C++\_Notlar/]\$

Programın çalışması bitince aynı dizinde “hata.txt” ve “başarı\_notu.txt” adlı dosyalar da oluşturulmuş olacaktır. Dosya içeriklerine bakıldığından programın çıktısı görülebilir. Program herhangi bir hata mesajı üretmediği için hata.txt dosyası boş olacaktır. Öğrencilerin başarı durumlarının kayıt edildiği başarı\_notu.txt dosyası da işlem sonucunu barındıracaktır.

[goksin@tardis ~/projeler/C++\_Notlar/]\$ ls \*.txt

başarı\_notu.txt                    hata.txt                    notlar.txt

[goksin@tardis ~/projeler/C++\_Notlar/]\$ cat hata.txt

[goksin@tardis ~/projeler/C++\_Notlar/]\$

[goksin@tardis ~/projeler/C++\_Notlar/]\$ cat başarı\_notu.txt

Ahmet	Selim	77.00	98.00	76.00	56.00	98.00	88.45	AA
Ayaz	Mert	34.00	56.00	65.00	78.00	88.00	71.80	BA
Zeynel	Kaya	54.00	67.00	87.00	73.00	54.00	61.15	BC
Ali	Can	45.00	67.00	56.00	77.00	80.00	70.10	BB
Kemal	Demir	43.00	56.00	55.00	67.00	80.00	67.05	BB
Cem	Balta	43.00	32.00	68.00	59.00	78.00	62.95	BC
Necati	Uzun	78.00	54.00	80.00	65.00	55.00	61.80	BC
Asya	Nur	45.00	65.00	76.00	89.00	53.00	59.50	CB
Zeynep	Atabek	54.00	63.00	50.00	60.00	59.00	58.05	CB

[goksin@tardis ~/projeler/C++\_Notlar/]\$

**Örnek Program:** Bir üniversitede her öğretim yılının ikinci döneminde öğrenci temsilcisi seçimi yapılmaktadır. Temsilci seçim sürecinde aday olan öğrenciler her fakülte ve melsek yüksekokulunda kurulan seçim sandıklarında kullanılan oyların sayımı sonucunda kazanan öğrenci belirlenmektedir. Adayların tüm akademik birimlerde aday olmalarına karşın, tüm akademik birimlerden oy kullanan öğrencilerden oy almaları beklenmemektedir. Yazılacak olan program ile akademik birimlerden elde edilen oy sayıları toplanarak hangi adayın kaç oy aldığı belirlenecektir.

**Çözüm:** Programın yazılabilmesi için oy verilerinin bir dosyadan okunması gerekecektir. Bu dosya “oylar.txt” adlı dosya olacaktır. Dosyada her adayın adı ve soyadı ile akademik birimden aldığı oyların sayısı yer alacaktır. Her birimden adayın oy alması beklenmediği için oy alınan akademik birimlerin sayısı değişkenlik gösterecektir. Bir adayım seçim sonucu değerlendirem sürecine katılabilmesi için en az bir birimden en az bir oy alması gereklidir. Her adayın aldığı toplam oy adının yanına yazılacaktır. Sonuçlar hem standart çıktıya hemde dosyaya yazdırılacaktır. Sonuçların yazılıacağı dosya oylama\_sonuclu.txt dosyası olacaktır.

**Programın algoritması:**

- 1 Başla
- 2 “oylar.txt” dosyasını okumak için aç.
- 3 Hataları kayıt atlına almak için “hata.txt” dosyasını yazmak için aç.
- 4 “oylar.txt” dosyasını kontrol et.
  - 4.1 Dosya bulunmuyorsa;
    - 4.1.1 “hata.txt” dosyasına “dosya bulunamadı mesajını yaz.”
    - 4.1.2 Standart çıktıya hata mesajını yaz
    - 4.1.3 Hata kodu 1 ile program sonlandır.
- 5 “oylama\_sonucu.txt” dosyasını yazmak için aç.
- 6 Adayın adını oku, soyadını oku.
- 7 Eğer dosyanın sonuna gelinmiş
  - 7.1 “veri işleme tamamlandı” mesajını standart çıktıya yaz
  - 7.2 Açılan tüm dosyaları kapat.
  - 7.3 Programı başarılı olarak sonlandır.
- 8 Adayın aldığı oy sayılarını oku. Eğer;
  - 8.1 Oy sayısı sıfırdan büyük olup olmadığını kontrol et, bu koşullar sağlanmıyorsa,
    - 8.1.1 “hata.txt” dosyasına Öğrencinin “adını, soyadını ve hatalı oy sayısı bulunduğu” mesajını yaz.
    - 8.1.2 Hatalı veri belirlendiğini belirten mesajı yaz.
    - 8.1.3 Açılan tüm dosyaları kapat.
    - 8.1.4 Programı hata kodu 2 ile sonlandır.
- 9 Adayın aldığı oyları sırası ile oku ve topla.
- 10 Adayın aldığı oyların okunurken verinin sonuna gelinmesi “-1” ile temsil edilir. Eğer “-1” okunduysa toplama işlemi bitmiştir.
- 11 Adayın adını, soyadını ve aldığı oy sayısını standart çıktıya yaz.
- 12 Adayın adını, soyadını ve aldığı oyları “oylama\_sonucu.txt” dosyasına yaz.
- 13 Sıradaki adayın oylarını bulmak için 6.Adıma geçilir.
- 14 Son

**Programın Yazılması:** Programın işleyeceği veriler öğrenci adı ve soyadı, öğrenci notlarıdır. **Öğrenci adı ve soyadı karakter dizisi – string** olarak tanımlanacaktır. **Adayın aldığı oylar**, pozitif tam sayı olacağı için **tam sayı – int** veri tipinde tanımlanabilir. Bu aralık dışında olabilecek olası hatalı veriler, karakter veya negatif sayılar olabilir. Bunların kontrol edilmesinde girdi akımındaki

hata durumu izlenerek yapılabilir. Verilerin olmaması durumu için **G/Ç İşlemleri** yardımı ile kontrol edilebilir. Çıktı dosyalarının önceden hazırlanmasına gerek yoktur. İşletim sistemi bunları oluşturacaktır. Bu durumda programda for() döngüsü kullanılamaz. Bunun yerine while() döngüsü kullanılabilir. Her bir adayın adayı ve soayadı okunacak ve sonradı aldığı oy toplamı yazılacaktır. Adayın hangi akademik birimden ve kaç adet oy aldığı bilinmeyeceği için “-1” okuna kadar adayın oyları okunup toplanacaktır. Bir adayın oylarının işlenmesinin ardından diğer adaya geçilecektir. Aday konusunda sınırlama olmadığından tüm adayların oy sayımı ancak dosyanın sonuna gelinmiş olması durumunda sona erecektir. Bu programda iki adet while() veya iki adet do ... while() döngüsü kullanılmalıdır. Aday olmaması durumunda program çalıştırılmayacağı için ilk döngü while() olurken iç döngü do while() olarak kurgulanabilir. Böylece adayın aldığı oy sayısı hesaplanır. Belirtilen hata durumlarında program sona erme koşulları algoritmada tanımlanmıştır. Algoritma ayrıntılı olarak tasarlandığı için doğrudan algoritma esa alınarak program yazılacaktır. Programın verileri işlemesi sırasında program akışının izlenmesi için program içerisinde hata mesajları ve ilerleme durumunu belirten mesajlar standart çıktıya yazdırılacaktır.

Programın kaynak kodu aşağıdaki gibidir:

```
*  
* oylama.cxx  
*  
*  
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>  
* *  
* Her hakkı saklıdır.  
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak  
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği  
* takdirde serbesttir:  
*  
* Kaynak kodun yeniden dağıtım; yukarıdaki telif hakkı uyarısını,  
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.  
*  
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,  
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen  
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.  
*  
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
```

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

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

int main()
{
    // Dil ayarları.
    setlocale(LC_ALL, "tr_TR.UTF-8");

    // Dosya okuma ve yazma işlemleri için değişkenlerin tanımlanması.
    ifstream dosya_ac_oku;
    ofstream dosya_ac_yaz;
    ofstream dosya_hata_yaz;
    // Adayın seçim kimliği, adı ve soyadı.
    string aday_ad, aday_soyad;
    // satır sonu belirteci
    int SON = -1;
```

```
// Değişken tanımlamaları: Öğrencinin notları, dönem sonu ve ortalama tanımlanıp etinleştirilsin.

int oy, toplam_oy;

// Okunacak olan dosyanın kontrolü yapılsın.

dosya_ac_oku.open("oylar.txt");

if (!dosya_ac_oku)

{

    cout << "Kaynak dosya olan \"oylar.txt\" bulanamadı." << endl;

    cout << "Program sonlanıyor. Hata kodu: 1" << endl;

    // Dosyaya hata kayıt edildi.

    dosya_hata_yaz << "hata1" << " " << "Kaynak dosya olan \"oylar.txt\" bulanamadı."

<< endl;

    // Hata bildirimi yapıldı, dosya kapatıldı.

    dosya_hata_yaz.close();

    return 1;

}

// Yazılacak olan dosya açılır, yoksa işletim sistemi oluşturulur.

dosya_ac_yaz.open("oylama_sonucu.txt");

// Dosyadan veriler okunuyor.

cout << "Veriler dosyadan okunuyor ve işleniyor." << endl;

// Dosyada kaç adet aday olduğu bilinmediğinden sona gelinene kadar okuma yapılır.

while(dosya_ac_oku)

{

    // Dosyadan veriler okunur.

    dosya_ac_oku >> aday_ad >> aday_soyad;

    // Dosyada okunacak satır bulunmuyor ise döngüyü sonlandırıralım.

    if (dosya_ac_oku.eof())

        break;

    // her adayın oyları okunurken önceden değişkende barındırılan veriler temizlenir.

    oy=0;

    toplam_oy=0;
```

```

// Aday en az bir birimden oy almış ise oy sayımına katılır.

do
{
    dosya_ac_oku >> oy;

    // Satır sonu karakteri -1 olduğundan dolayı
    if (oy < -1)

    {
        // Dosyaya hata kayıt edildi.

        dosya_hata_yaz << "hata2" << " " << aday_ad << " " << aday_soyad
<< endl;

        // Kullanıcıya işlem hakkında bilgi verilir.

        cout << "Adayın oylarında hatalı veri bulunmuştur. İşlem
sonlandırıldı." << endl;

        cout << "Hata kodu 2" << endl;

        // Hata bildirimi yapıldı, dosyalar kapatılır.

        dosya_ac_yaz.close();
        dosya_ac_oku.close();
        dosya_hata_yaz.close();

        return 2;
    }

    // Adayın aldığı oylar okunur ve toplanır.

    if(oy != -1)
        toplam_oy = toplam_oy + oy;
}while(oy != SON);

// Durum kullanıcıya bildirilir.

cout << "Adayın adı soyadı: " << aday_ad << endl;
cout << "Aldiği oy sayısı: " << toplam_oy << endl;

// Durum dosyaya kayıt edilir.

dosya_ac_yaz << "Adayın adı soyadı ve aldığı toplam oy sayısı: " << aday_ad << " "
<< aday_soyad << " " << toplam_oy << endl;
}

```

```
// Veriler okundu ve okuma işlemi sona erdi.  
dosya_ac_oku.close();  
// Dosya yazma işlemi sonlanıyor  
dosya_ac_yaz.close();  
// Dosya hata yazma işlemi sonlanıyor  
dosya_hata_yaz.close();  
return 0;  
}
```

Programın işleyeceği verilerin yer aldığı oylar dosyasının içeriği aşağıdaki gibidir.

```
Ahmet Selim 99 32 87 90 54 -1
```

```
Yavuz Keser 25 92 17 54 -1
```

```
Nalan Alaca 125 92 97 48 99 123 -1
```

Bu dosya “oylar.txt” adı ile kayıt edilip programın bulunduğu dizine kayıt edilecektir. Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./oylama
```

Veriler dosyadan okunuyor ve işleniyor.

Adayın adı soyadı: Ahmet

Aldığı oy sayısı: 362

Adayın adı soyadı: Yavuz

Aldığı oy sayısı: 188

Adayın adı soyadı: Nalan

Aldığı oy sayısı: 584

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Programın çalışması bitince aynı dizinde “hata.txt” ve “oylama\_sonuc.txt” adlı dosyalar da oluşturulmuş olacaktır. Dosya içeriklerine bakıldığından programın çıktısı görülebilir. Program herhangi bir hata mesajı üretmediği için hata.txt dosyası boş olacaktır. Adayların ekran yazdırılan sonuçları aynı şekilde oylama\_sonucu.txt dosyasında yer alacaktır.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ls *.txt
```

```
başarı_notu          hata.txt          notlar.txt
```

```
[goksin@tardis ~/projeler/C++_Notlar/]$ cat hata.txt
```

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

```
[goksin@tardis ~/projeler/C++_Notlar/]$ cat oylama_sonucu.txt
```

Adayın adı soyadı ve aldığı toplam oy sayısı: Ahmet Selim 362

Adayın adı soyadı ve aldığı toplam oy sayısı: Yavuz Keser 188

Adayın adı soyadı ve aldığı toplam oy sayısı: Nalan Alaca 584

[goksin@tardis ~/projeler/C++\_Notlar/]\$

## 5.Programcı Tarafından Tanımlanan Fonksiyonlar

Bu bölümde önce yer verilen örnekler içerisinde her programda bir tane ana fonksiyon olan main() yer aldığı programlar yazıldı. Tüm bilgisayar programları için geçerli olan koşul bir programda sadece bir tane ana fonksiyon bulunabileceğidir. Bu kural, programlama eğitimlerinin hepsinde uyulan ama sözü edilmeyen bir kuraldır. Bunun nedeni programlama dillerinin ve bilgisayar teknolojisinin gelişimi sırasında verilen kararlarda yatomaktadır. Ayrıca programlamaların dillerinin de gelişimi boyunca bilgisayarla problem çözümü sırasında karşılaşılan bazı yeni problemlerin çözümlerinin de tasarım süreçlerini etkilemesidir.

Bilgisayar programlarının bir adet ana fonksiyona sahip olması ilk bakışta yeterli bir çözüm gibi görünebilir. Ancak üç önemli etken bu yaklaşımın etkinliğini ve verimliliğini değiştirebilmektedir: Problemin boyutu, işlenecek olan veri ve elde edilmesi istenen sonuçların karmaşıklığı ve çeşitliliği.

Bu etkenler klasik bir ana fonksiyon içerisinde çözüm üretmeyi zorlamaktadır. Öncelikle problemin karmaşıklaması çözümün de karmaşıklmasını beraberinde getirmektedir. İleride görüleceği üzere problem çözümünde kullanılabilecek olan çok sayıda yöntem vardır. Bunlardan birisi de problemin daha küçük ve özünde tek bir işlev ile çözülebilecek olan alt problemlere ayırmaktır. Böylece her bir problem tek olarak çözülebilir ve elde edilen çözümler birleştirilerek asıl problemin çözümü elde edilebilir. Bu yöntemde elde edilen alt problemler farklı ekiplere atanarak problem çözülebilir.

Problemin bu şekilde parçalanarak çözülmesi programlama açısından tek bir ana fonksiyon ile problemin çözülmesi yerine problemin fonksiyon özeline bölünderek çözülmesini sağlayacaktır. Fonksiyonlar bu açıdan bakıldığından problemin çözümünde şu yararları sağlamaktadır:

- Bir fonksiyon üzerinde çalışmak, problemin belirli bir bölümü üzerinde çalışmayı sağlar. Böylece probleme uygun çözümün kaynak kodunun yazılmasını, mantık, anlam ve yazım hatalarının kolaylıkla bulunmasını, yazılan kaynak kodun mükemmelleştirilmesini sağlar.
- Problem farklı fonksiyonlar ile çözüldüğünde, her bir fonksiyon bir ekip tarafından geliştirilebilir. Böylece geliştirme süreci kısalır.
- Eğer bir alt problem asıl problem içerisinde birden çok yerde kullanılıyor ise, fonksiyon olarak çözülmesi ile aynı kodun program içerisinde tekrar yazılmasına gerek kalmaz. Fonksiyon program içerisinde farklı yerlerde çağrılarak çalıştırılabilir.
- Fonksiyonlar ana fonksiyonun karmaşıklasmasının önüne geçer. Böylece hem ana fonksiyon hem de diğer fonksiyonlar kolaylıkla okunur ve anlaşılabilir.

Programlama kitaplarında bazen fonksiyonlar modül olarak da tanımlanır. Fonksiyonlar yapısal özelliği sayesinde bir çok farklı fonksiyon birlikte kullanılarak program yazılabılır. Her ne kadar bazı fonksiyonlar programlama araçları içerisinde hazır olarak sunuluyor olsa da, bunlar programı yazan programcının yazdığı kaynak koddan farklıdır. Bu fonksiyonlar hazır olarak sunulmakta olsa da özel olarak geliştirilmiş ve denendikten sonra kabul edilerek geliştirem araçlarının bir parçası olmuşlardır. Bu fonksiyonlar hazır olarak kullanılması zaman eve emekten tasarruf edilmesini sağlamak yanında sık sık karşılaşılan bazı problemler için çözümlerde sunarlar. Önce-

likle bu hazır fonksiyonlara yani ön tanımlı fonksiyonlar incelenecek daha sonra programcı tarafından tanımlanan ve yazılan fonksiyonlar üzerinde durulacaktır.

## 5.1.Ön Tanımlı – Hazır – Fonksiyonlar

Matematikte fonksiyonlar konusu hepimizin bildiği bir konudur. Bir fonksiyon bir tanım kümesinde yer alan elemanları bir görüntükümesine aktarır. Bu yaklaşım C++ Programlam Dili'ndeki fonksiyon kavramı ile benzerdir. Fonksiyonlar hazır olarak yani ön tanımlı olarak sunulduğu gibi programcı tarafından da yazılabılır. Ön tanımlı fonksiyonlar ister matematik işlemler için ister mantıksal veya metin parçaları üzerinde işlem yapmakta olsun doğrudan kaynak kod içerisinde ilgili ön işlemci komutları ile veya bazı durumlarda da programlama dilinin temel yapısında hazır olarak bulunduğu için doğrudan kullanılabilir. Aşağıda bazı yaygın kullanılan C++ Ön işlemci komutları ile çağrılarak kullanılabilen fonksiyon kütüphanelerine yer verilmiştir.

<b>cassert ön işlemci komutu (assert.h)</b>		
assert(ifade)	İfade mantıksal veya tam sayı veri tipinde olabilir.	İfadenin sonucu veya değeri sıfırdan farklı ise (doğru – true) program işlemey devam eder. Aksi durumda programın işleyışı sona erer. Standart çıktıda dosyanın adı, ifde ve bulunduğu satır yazılır.

<b>cctype ön işlemci komutu (ctype.h)</b>		
isalnum(ch)	ch karakter veri tipindedir.	Fonksiyon bir tamayı değeri döndürür: Eğer karakter olarak tanımlanan değişken 'A'-'Z', 'a'-'z', '0'-'9' aralığında ise sıfırdan farklı bir değer (doğru – true) döndürür. Aksi durumda ise sıfır (yanlış – false) döndürür.
iscntrl(ch)	ch karakter veri tipindedir.	Fonksiyon bir tamayı değeri döndürür: ASCII tablosuna göre bir kontrol karakteri ise (0 – 31 veya 127) sıfırdan farklı bir değer döndürür. (doğru – true). Aksi durumda sıfır (yanlış – false) döndürür.
isdigit(ch)	ch karakter veri tipindedir.	Fonksiyon bir tam sayı değeri

		döndürür: Karakter '0'-'9' arasında ise sıfırdan farklı (doğru – true) değeri döndürür .Aksi durumda sıfır (yanlış – false) döndürür.
islower(ch)	ch karakter veri tipindedir.	Fonksiyon bir tam sayı değeri döndürür: Karakter 'a'-'z' arasında ise sıfırdan farklı (doğru – true) değeri döndürür .Aksi durumda sıfır (yanlış – false) döndürür.
isprint(ch)	ch karakter veri tipindedir.	Fonksiyon bir tam sayı değeri döndürür: ASCII tablosuna göre karakter yazdırılabilir ise sıfırdan farklı (doğru – true) değerini döndürür. Aksi durumda ise sıfır (yanlış – false) döndürür.
ispunct(ch)	ch karakter veri tipindedir.	Fonksiyon bir tam sayı değeri döndürür: Eğer karakter bir noktalama işaretü ise sıfırdan farklı (doğru – true) değerini döndürür. Aksi durumda ise sıfır (yanlış – false) döndürür.
isspace(ch)	ch karakter veri tipindedir.	Fonksiyon bir tam sayı değeri döndürür: Eğer karakter beyaz boşluk (sekme, boşluk, yeni satır vb) ise sıfırdan farklı (doğru – true) değerini döndürür. Aksi durumda ise sıfır (yanlış – false) döndürür.
isupper(ch)	ch karakter veri tipindedir.	Fonksiyon bir tam sayı değeri döndürür: Eğer karakter büyük harf 'A'-'Z' ise sıfırdan farklı (doğru – true) değerini döndürür. Aksi durumda ise sıfır (yanlış – false) döndürür.
tolower(ch)	ch karakter veri tipindedir.	Fonksiyon bir tam sayı değeri

		döndürür: Eğer karakter büyük harf 'A'-'Z' ise küçük harf olan karşılığını 'a'-'z' döndürür. Aksi durumda ise ASCII karakter tablosundaki karşılığını döndürür.
toupper(ch)	ch karakter veri tipindedir.	Fonksiyon bir tam sayı değeri döndürür: Eğer karakter büyük harf 'a'-'z' ise küçük harf olan karşılığını 'A'-'Z' döndürür. Aksi durumda ise ASCII karakter tablosundaki karşılığını döndürür.

<b>cfloat ön işlemci komutu (float.h)</b>	
FLT_DIG	float veri tipinde kayar noktalı değerin duyarlığını belirten bit sayısıdır.
FLT_MAX	Maksimum pozitif kayar noktalı değer (float)
FLT_MIN	Minimum negatif kayar noktalı değer (float)
DBL_DIG	double veri tipinde kayar noktalı değerin duyarlığını belirten bit sayısıdır.
DBL_MAX	Maksimum pozitif kayar noktalı değer (double)
DBL_MIN	Minimum negatif kayar noktalı değer (double)
LDBL_DIG	Long double veri tipinde kayar noktalı değerin duyarlığını belirten bit sayısıdır.
LDBL_MAX	Maksimum pozitif kayar noktalı değer (long double)
LDBL_MIN	Minimum negatif kayar noktalı değer (long double)

<b>climits ön işlemci komutu (limit.h)</b>	
CHAR_BIT	Bir byte oluşturan bit sayısı
CHAR_MAX	karakter veri tipinin üst sınırı

CHAR_MIN	karakter veri tipinin alt sınırı
SHRT_MAX	short veri tipinin üst sınırı
SHRT_MIN	short veri tipinin alt sınırı
INT_MAX	int veri tipinin üst sınırı
INT_MIN	int veri tipinin alt sınırı
LONG_MAX	long veri tipi üst sınırı
LONG_MIN	long veri tipi alt sınırı
LLONG_MAX	long long veri tipi için üst sınır
LLONG_MIN	long long veri tipi için alt sınır
UCHAR_MAX	unsigned char veri tipi için üst sınır
USHRT_MAX	unsigned char veri tipi için alt sınır
UINT_MAX	unsigned int veri tipi için üst sınır
ULONG_MAX	unsigned int veri tipi için alt sınır

<b>cmath ön işlemci komutu (math.h)</b>		
acos(x)	x kayar noktalı ifadedir. $-1.0 \leq x \leq 1.0$ arasındadır.	Sonuç 0 ile $\pi$ arasındadır.
asin(x)	x kayar noktalı ifadedir. $-1.0 \leq x \leq 1.0$	Sonuç $-\pi/2$ ile $\pi/2$ arasındadır.
atan(x)	x kayar noktalı ifadedir.	Sonuç $-\pi/2$ ile $\pi/2$ arasındadır.
ceil(x)	x kayar noktalı ifadedir.	Sonuç tam sayı $\geq x$ olarak dönürülür.
cos(x)	x kayar noktalı ifadedir ve Açıının trigonometrik cos değerinden radyan cinsinden ölçülmelidir.	ridir.
cosh(x)	x kayar noktalı ifadedir.	x'in hiperbolik kosinüs değerini

		döndürür.
exp(x)	x kayar noktalı ifadedir.	$e^x$ değerini döndürür.
fabs(x)	x kayar noktalı ifadedir.	x mutlak değerini döndürür.
floor(x)	x kayar noktalı ifadedir.	Sonuç tam sayı $\leq x$ olarak dön-dürülür.
log(x)	x kayar noktalı ifadedir ve $x > 0.0$ olmalıdır.	$x$ 'in doğal logaritmasını dön-dürür.
log10(x)	x kayar noktalı ifadedir ve $x > 0.0$ olmalıdır.	$x$ 'in 10 tabanındaki logaritma-sını döndürür.
pow(x,y)	x ve y kayar noktalı sayılardır; $x^y$ değerini döndürür. eğer $x = 0.0$ , y pozitif olmalıdır. Eğer $x \leq 0.0$ ise y tam sayı olmalıdır. (örnek: $y = 6.0$ olabilir ama 4.9 olamaz)	
sin(x)	x kayar noktalı ifadedir ve Açıının trigonometrik sin dege-radyan cinsinden ölçülmelidir.	ridir.
sinh(x)	x kayar noktalı ifadedir.	$x$ 'in hiperbolik sinüs değerini döndürür.
sqrt(x)	x kayar noktalı ifadedir ve $x \geq 0.0$ olmalıdır.	$x$ karekökünü döndürür.
tan(x)	x kayar noktalı ifadedir ve Açıının trigonometrik tan dege-radyan cinsinden ölçülmelidir.	ridir.
tanh(x)	x kayar noktalı ifadedir.	$x$ 'in hiperbolik tan değerini döndürür.

#### cstdddef ön işlemci komutu (stddef.h)

NULL	İşletim sistemine bağlı olarak değişen NULL pointer değerini döndürür. (teknik olarak 0)
------	--

#### string ön işlemci komutu (string.h)

strcat(#kaynakStr, hedefStr)	kaynakStr and hedefStr aslında null terminator eklenmiş karakter dizileridir. hedefStr ise işlem sonucunu barındıracak kadar büyük olmak durumundadır.	kaynakStr başlangıç adresi döndürülür, dizinin sonunu temsil eden null-terminator de hedefStr içerisinde eklenir.
strncat(#hedefStr,kaynakStr,limit)	kaynakStr and hedefStr aslında null-terminator eklenmiş karakter dizileridir. hedefStr ise işlem sonucunu barındıracak kadar büyük olmak durumundadır. limit ise pozitif bir tam sayıdır.	strcmp fonksiyonundaki gibidir, ek olarak sınırı belirten karakterler de eklenir.
strcmp(str1, str2)	str1 ve str2 null-terminator eklenmiş karakter dizisidir.	Karşılaştırma işleminin sonucu aşağıda belirtildiği gibi döndürülür: <ul style="list-style-type: none"> <li>• Eğer str1 &lt; str2 durumunda negatif tam sayı döndürülür.</li> <li>• Eğer str1 == str2 durumunda sıfır değeri döndürülür.</li> <li>• Eğer str1 &gt; str2 ise sıfırdan büyük bir tam sayı döndürülür.</li> </ul>
strncmp(#str1, str2,limit)	str1 and str2 aslında null-terminator eklenmiş karakter dizileridir. limit ise pozitif bir tam sayıdır.	Önceki strcmp fonksiyonu gibidir, ek olarak sınırlırma karakterleri de karşılaştırılır.
strcpy(#hedefStr,kaynakStr)	kaynakStr and hedefStr aslında null-terminator eklenmiş karakter dizileridir.	kaynakStr başlangıç adresi döndürülür, dizinin tamamı hedefStr içerisinde eklenir.
strncpy(#hedefStr,kaynakStr,limit)	str1 and str2 aslında null-terminator eklenmiş karakter dizileridir. limit ise pozitif bir tam sayıdır.	kaynakStr başlangıç adresi döndürülür, dizinin tamamı hedefStr içerisinde eklenir, ek olarak da sınırlırma karakterlerinin büyük kısmı da eklenmiş olur.
strlen(str)	str null-terminator ile sonlandırılmış bir karakter dizi-	Karakter dizisinin uzunluğu null-terminator olmadan dön-

Yukarıda yer verilen **string** kütüphanesine ek olarak bir diğer karakter dizisi kütüphanesi daha bulunmaktadır. Bu kütüphanenin adı da benzer şekilde **string** olarak tanımlanmıştır. İki arasındaki fark

Aşağıda verilen örnekte ön işlemci komutları ile kullanılabilen bazı fonksiyonlar bir program içerisinde kullanılmıştır.

```
/*
 * ön_tanımlı_fonksiyonlar.cxx
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
```

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\*VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <cmath>
#include <cctype>
#include <iomanip>
using namespace std;
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    int say=0;
    double birinci_sayı=0, ikinci_sayı=0;
    char karakter ='T';
    cout << showpoint << fixed << setprecision(2) << endl;
    cout << karakter << " küçük harf midir? " << islower(karakter) << endl;
    cout << "a harfinin büyük harf olarak karşılığı, " << static_cast<char>(toupper('a')) << endl;
    cout << "4.5 sayısının 6.0 kuvveti, " << pow (4.5,6.0) << " eder." << endl;
    cout << "İki sayı giriniz. ";
    cin >> birinci_sayı >> ikinci_sayı ;
    cout << endl << "Birinci sayının ikinci sayı kuvveti, " << pow (birinci_sayı, ikinci_sayı) <<
    " eder." << endl;
    cout << "5.0'in 4 kuvveti, " << pow(5.0,4) << " eder." << endl;
    birinci_sayı = birinci_sayı + pow (3.5, 3.5);
```

```

cout << "Birinci girilen sayı üzerinde işlem yapıldıktan sonra değeri, " << birinci_sayı << "
oldu." << endl;

say = -32;

cout << "-32'inin mutlak değeri, " << abs (say) << " eder." << endl;

cout << "42'in kare kökü " << sqrt(42) << " eder." << endl;

return 0;

}

```

Programın derlenip çalıştırılması ile aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./ön_tanımlı_fonksiyonlar
```

T küçük harf midir? 0

a harfinin büyük harf olarak karşılığı, A

4.5 sayısının 6.0 kuvveti, 8303.77 eder.

İki sayı giriniz. 12 45

Birinci sayının ikinci sayı kuvveti, 3657261988008837087910294309823673092868510056448.00 eder.

5.0'in 4 kuvveti, 625.00 eder.

Birinci girilen sayı üzerinde işlem yapıldıktan sonra değeri, 92.21 oldu.

-32'inin mutlak değeri, 32 eder.

42'in kare kökü 6.48 eder.

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

Programın incelenmesi ile ön tanımlı fonksiyonların bir program içerisinde asıl kullanılacağı görülebilir. “islower()” fonksiyonu verilen bir karakterin küçük har olup olmadığını belirlemektedir. Sonuç mantıksal olarak doğru ise bir değilse sıfır sonucu döndürülür. Ardından gelen “a” karakterini büyük harf olarak karşılığı veren satırda “toupper()” fonksiyonu kullanılmıştır. Karakter olarak verilen ‘a’ fonksiyonda kullanıldığındaysa ASCII karlılığı verilir. “a” karakterinin ASCII karşılığı 65 olurken, büyük harf olarka ASCII Karlılığı 135’tir. Bu fonksiyon verilen karakterin ASCII karşılığını pozitif tam sayı olarak verir. Bu sonuç yerine elde edilen tam sayı değer “static\_cast <char>()” ile karakter veri tipine dönürtür ve karakter olarak karşılığı yazdırılır. C++ Programlam dilinde üs işlemlerinin yapılması için pow() fonksiyonu kullanılır. Bu fonksiyon iki parametre alır. Birincisi taban ve ikincisi de üs değeridir. Hem taban hem de sayı kayar noktalı olarak işlenir. Bunun yapılmasının nedeni karekök fonksiyon olan sqrt() doğrudan verilen bir kayar noktalı sayının kare kökünü bulmakta olmasıdır. Bu fonksiyon genelleştirilmiş bir çözüm olan kaya noktalı tabanın kayar noktalı kuvvetinin hesaplanması algoritmasına göre daha hızlı sonuç vermektedir. Bu nedenle de doğrudan karekök fonksiyonu kullanılması gereken işlemlerde tercih

edilmelidir. Ardından program kullanıcıdan iki sayı girmesini istemektedir. Girilen sayılar pow() fonksiyonunda işlenip sonucu yazdırılmaktadır. Bu aşamadan sonra gelen fonksiyonlar öncekilere benzer şekilde işlemler gerçekleştirmektedir.

## 5.2.Programcı Tarafından Tanımlanan Fonksiyonlar

Yukarıdaki örnekte görüldüğü gibi ön tanımlı fonksiyonların kullanılması bir programdaki bulunan ana fonksiyonun karmaşıklaşmasını azaltmaktadır. Ayrıca hazır fonksiyonlar üzerinde önceden çalışılmış ve etkinliği ve verimliliği kabul edilmiş fonksiyonlar olduğundan aynı işlevlerin yeniden yazılması, derlenmesi ve hatalarının giderilmesi gibi süreçlerin tekrarlanmasıının önüne geçer. C++ programlama Dili’nde bir çok fonksiyon hazır olarak sunuluyor olsa da bir programcının gereksinim duyacağı tüm fonksiyonlar bulunmamaktadır. Bu nedenle bir programcının kendi gereksinim duyduğu fonksiyonları nasıl yazabileceğini bilmesi gereklidir. C++ Programlama Dili’nde programcı tarafından yazılan fonksiyonlar iki grupta toplanır:

- **Değer Döndüren Fonksiyonlar:** Bu fonksiyonların işlenmesinin ardından elde edilen sonuç ana fonksiyona iletilir. Bu sonuç belirli bir veri tipinde tanımlanır ve “return” ile ana fonksiyona iletimi sağlanır. Ana fonksiyon olan main() sona erdiğinde return0 ile iletim sisteme değer döndürmektedir.
- **Değer Döndürmeyecek Fonksiyonlar:** Bu fonksiyonlar işlendiklerinde herhangi bir sonuç üretmez. Dolayısı ile de yapısında bir “return” ifadesi bulunmaz.

## 5.3.Değer Döndüren Fonksiyonlar

Önceki bölümde yer verilen programda kullanılan abs() toupper(), islower() ve pow() gibi fonksiyonlar değer döndüren fonksiyonlardandır. Bu fonksiyonların programda kullanılabilmesi için hangi ön işlemci komutu ile çağrılabileceğinin bilinmesi gereklidir. Ön tanımlı fonksiyonların program içerisinde kullanılabilmesi için şu bilgilere gereksinim vardır:

1. Fonksiyonun adı,
2. Fonksiyonun varsa aldığı parametreler,
3. Her bir parametrenin veri tipi,
4. Fonksiyonun döndüreceği sonucun veri tipi, (Bu aynı zamanda fonksiyonun tipi olarak da adlandırılır.)

Bu tür fonksiyonlar sadece tek bir sonuç döndürmekte olduğu için, bu sonucun kullanılması için şu yöntemlerden birisi tercih edilmelidir.

- Fonksiyonun döndürdüğü sonucun diğer işlemler tarafından de kullanılabilmesi için bir değişkende saklanmalıdır. Örneğin, “kare = pow(a, 2);”
- Fonksiyon doğrudan bir işlemin içerisinde kullanılabilir. “diskriminant = pow(b, 2) – (4 \*a \*c);”
- Fonksiyonun sonucu doğrudan standart çıktıya yazdırılabilir. “cout << abs(sonuc);”

Bunların incelenmesinden de görüleceği üzere değer döndüren bir fonksiyon atama ifadesinde, bir fonksiyon çağrısında parametre olarak veya bir çıktı ifadesinde kullanılabilir.

Bi değer döndüren fonksiyonlar, bir ifade içerisinde kullanılabilir. Yukarıda tanımlanan dört özelliğe ek olarak da fonksiyonun kullanılabilmesi son bir özelliğin gerekli olduğunu göstermektedir.

##### 5. Gerçekleştirilmesi istenen görevi yerine getirecek olan kaynak kod.

İlk dört özellik, fonksiyonun **başlık/fonksiyon başlığı** bölümünü, beşinci özellik ise fonksiyonun gövde kısmını oluşturur. Tüm beş özellik birden ise fonksiyonun **tanımı** olarak tanımlanır.

Bir fonksiyon yukarıdaki özellikleri ile tanımlanır ve yazılır. Buradan hareketle bir fonksiyonun **formal parametreleri** olacağını ancak çağrılığında ise fonksiyona atanın **asıl parametreler** ile işlem yaptığını belirtebiliriz.

#### 5.3.1. Değer Döndüren Fonksiyonların Yazım Kuralları

Bir değer döndüren fonksiyon yazılrken yukarıdaki açıklamalara dikk edilerek aşağıdaki biçimde yazılabılır.

```
fonksiyonTipi fonksiyonAdı (formal parametreler)
{
    ifadeler;
}
```

Bu yazım şeklinde fonksiyonTipi fonksiyonun döndürügü değerin veri tipidir. Formal parametreler fonksiyonun işleyeceği veri veya verilerin veri tipi ile birlikte tanımlanmış halidir. İfadeler bölümü ise tanımlamalar veya işlenecek olan ifadelerin yer aldığı bölümdür. Fonksiyon gövdesi daima { } arasında yazılmalıdır.

#### 5.3.2. Formal Parametrelerin Yazım Kuralları

Bir değer döndüren fonksiyonda formal parametreler yazılrken her bir parametrenin veri tipi tanımlanmalıdır.

```
fonksiyonTipi fonksiyonAdı (veriTipi parametre, veriTipi parametre)
{
    ifadeler;
}
```

Bu açıklamalardan yararlanarak aşağıda bir fatura programında kullanılan katma değer vergisi hesaplayan fonksiyon görülmektedir.

```
double kdv (int kategori, double miktar)
{
    double oran;
```

```

switch(kategori)
{
    case 1:
        oran = 0,25;
    case 2:
        oran =0,20;
    case 3:
        oran =0,18;
    case 4:
        oran =0,07;
    case 5:
        oran =0,03;
    default:
        oran = 0,0;
    kdv = oran * miktar;
}
return kdv;
}

```

### 5.3.3.Fonksiyonların Çağrılması ve Asıl Parametreler

Bir değer döndüren fonksiyonun kullanıldığı program içerisinde çağrılması için aşağıdaki şekilde çağrı işleminin ifade edilmesi gereklidir.

**fonksiyonAdı (asıl parametreler)**

Burada asıl parametreler fonksiyonun adından sonra gelen ( ) içerisinde tanımlanmalıdır. Tanımlama yapılırken fonksiyona aktarılacak olan asıl parametreler bir değer, değişken veya bir ifade olabilir. Yukarıdaki örnek fonksiyonun kullanıldığı bir programda katma değer vergisinin hesaplanması için kdv fonksiyonu program içerisinde aşağıdaki gibi çağrılacaktır.

**double kdv(4, 22.76)**

Bu ifadede, fonksiyona aktarılan asıl parametreler, ilk değişken olan kategoriye ait olan “4” ve ikinci değişken olan “miktar” ait olan “22.76” değeridir.

Eğer bir fonksiyona aktarılacak olan herhangi bir parametre bulunmuyor ise bu durumda yine fonksiyon adının sonunda ( ) yer almalıdır.

**double kdv()**

Eğer fonksiyon herhangi bir parametre almayacak ise yukarıdaki örnekte “switch () case” görüldüğü gibi bir varsayılan değer işlenir. Fonksiyon çağrılarında fonksiyona aktarılan asıl parametreler ister bir değer barındırsın ister barındırmamasın bu durum programcı tarafından dikkate alınmalıdır. Bazı durumlarda fonksiyonların ön tanımlı değerleri olabilir. Bu durumda fonksiyon ön tanımlı olan değerini işleyerek sonuç döndürecektir.

### 5.3.4.Fonksiyonların Değer Döndürmesi: return

Bir değer döndüren fonksiyona parametre aktarılmış ise, fonksiyon gövdesinde bulunan ifadelerin işlenmesi ile bir sonuç elde edilir. Elde edilen sonucun ana fonksiyona iletilmesi için “return” kullanılır. Yazım kuralı aşağıdaki gibidir.

return ifade;

Burada ifade ilk bakışta düşünüldüğü gibi hep ifade olmak durumunda değildir. Bir değişken, sabit veya bir ifade olarak tasarılanabilir. Döndürülecek olan sonucun veri tipi daima fonksiyonun veri tipi ile aynı olmalıdır. Buna göre eğer döndürülecek olan değer bir ifade ile belirleniyor ise, ifadenin değeri hesaplandığında elde edilen sonucun veri tipinin fonksiyon ile aynı olması gereğine özellikle dikkate edilmelidir.

Bir değer döndüren fonksiyonda fonksiyon gövdesinde yer alan “return” işlendiğinde fonksiyonun işlenmesi durdurulur. Döndürülecek olan sabit değer, değişken veya ifadenin sonucu belgeye aktarılır ve program akışı doğrudan ana fonksiyona geçerek devam eder. Eğer ana fonksiyon içerisinde “return” işlenecek olursa, program sonlandırılır.

Aşağıdaki örnekte verilen iki sayıyı karşılaştırın bir fonksiyon yazılmıştır. Büyük olan sayı ana fonksiyona geri döndürülmemektedir. Fonksiyon ve sayılar aynı veri tipinde olup double kullanılmıştır.

```
double buyuktur(double sayı1, double sayı2)
{
    double buyuk;
    if sayı1 >= sayı2
        buyuk = sayı1;
    else
        buyuk = sayı2;
    return buyuk;
}
```

Bu fonksiyon içerisinde iki adet formal parametre barındırmaktadır. Öte yandan kendi içerisinde kullanmak için de bir adet “**yerel değişken**” olan “**buyuk**” tanımlanmıştır. Bu yerel değişken sadece fonksiyon içerisinde işlenmektedir.

Yukarıdaki fonksiyon bir program içerisinde aşağıdaki şekillerde kullanılabilir:

- Fonksiyona asıl parametreler verilerek sonuç değişkene atanabilir.

```
sonuc = double buyuktur(12.34, 22.09)
```

- Fonksiyona asıl parametreler olarak program içerisinde tanımlanmış olan diğer değişkenler verilebilir ve sonuç yine bir değişkene atanır.

```
sonuc = double buyuktur(girdi_1, girdi_2)
```

- Fonksiyona asıl parametreler olarak program içerisinde tanımlanmış olan bir değişken ile bir sayı birlikte verilebilir ve sonuç yine bir değişkene atanır.

```
sonuc = double buyuktur(345.18, girdi_2)
```

Yuakridaki fonksiyon içerisinde tanımlanmış olan yerel değişken veya değişkenler sadece fonksiyon içerisinde işlenir. Fonksiyonlarda yerel değişkenlerin tanımlanması bir gereklilik değildir. Ancak fonksiyonun yerine getirmesi gereken işlevin özelliği gereği yerel değişkenlerin program içerisinde kullanılması gerekebilir. Yukarıdaki fonksiyon aşağıda verilen örneklerde yerel değişkenler olmadan koşul ifadeleri kullanılarak yeniden yazılmıştır.

```
double buyuktur(double sayı1, double sayı2)
{
    if sayı1 >= sayı2
        return sayı1;
    else
        return sayı2;
}
```

Yukarıdaki örnek fonksiyonda yerel değişken kullanılmadan iki durumu temsil eden bir ifade kullanılmıştır. Bunun daha dolayız olan şekli de aşağıda verilmiştir.

```
double buyuktur(double sayı1, double sayı2)
{
    if sayı1 >= sayı2
        return sayı1;
    return sayı2;
}
```

Aşağıdaki örnek programda ana fonksiyon dışında iki fonksiyon yazılmıştır. İlk fonksiyon yukarıdaki örnekte yer verilen buyuktur() fonksiyonu ve ikincisi de karsilastir() adlı fonksiyondur. Program üç adet farklı sayıyı karşılaştırıp büyük onalı döndürdüğü için sadece üç adet değişkene gereksinim vardır. Bunlar programın başlangıç bölümünde tanımlanmıştır. Önce girilen iki sayı karşılaşılacaktır. Ardında üçüncü sayı girilecek ve ilk karşılaşırma işleminde kullanılan fonksiyon üç sayının karşılaşılması için kullanılacaktır.

```
/*
 * hangisi_büyük.cxx
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
 * SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
 * VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
 * BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
 * HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
 * SORUMLU DEĞİLLERDİR.
 *
 */
```

```

#include <iostream>
#include <locale>
using namespace std;
// Değişkenler
double sayi1=0, sayi2=0, sayi3=0;
// Prototip fonksiyonlar
double buyuktur(double sayi1, double sayi2);
double karsilastir(double sayi1, double sayi2, double sayi3);
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    cout << "Bu program klavyeden girilen sayılarından büyük olanını bulur ve yazdırır." << endl;
    cout << "Lütfen aralarında boşluk bırakarak iki sayı giriniz. ";
    cin >> sayi1 >> sayi2;
    cout << endl;
    cout << "Büyük olan sayı " << buyuktur(sayi1, sayi2) << "'dir." << endl;
    cout << "Bir sayı daha girin: ";
    cin >> sayi3;
    cout << endl;
    cout << "Girilen üç sayıdan en büyüğü " << karsilastir(sayi1, sayi2, sayi3) << "'dir." << endl;
    return 0;
}
double buyuktur(double sayi1, double sayi2)
{
    double deger;
    if (sayi1>=sayi2)
        deger = sayi1;
    else
        deger = sayi2;
}

```

```
        return deger;  
    }  
  
double karsilastir(double sayı1, double sayı2, double sayı3)  
{  
    return buyuktur(sayı1, buyuktur(sayı2, sayı3));  
}
```

Programın derlenip çalıştırılması ile aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./hangisi_büyük
```

Bu program klavyeden girilen sayılardan büyük olanını bulur ve yazdırır.

```
Lütfen aralarında boşluk bırakarak iki sayı giriniz. 22.87 93.75
```

Büyük olan sayı 93.75'dir.

```
Bir sayı daha girin: 103.33
```

Girilen üç sayıdan en büyüğü 103.33'dür.

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

Programın incelenmesi ile ana fonksiyondan önce prototip fonksiyonlar adlı bir alan ve fonksiyon tanımlamaları yer almaktadır. Ana fonksiyonun ardından da adı geçen fonksiyonlar yazılmıştır. İkinci fonksiyon önceden yazılmış olan bir fonksiyon yapısında barındırmaktadır. Dolayısı ile aslında önceden yazılmış olan fonksiyon içe içe kullanıldığından önce en iç kısımdaki değerleri le çalıştırılıp elde edilen sonuç ile üçüncü sayının işlenmesi ile tekrar kullanılmaktadır. Elde edilen sonuç ise ikinci fonksiyonun sonucu olarak ana fonksiyona döndürülür. Bu tür fonksiyonlara **öz yinelemeli – recursive functions** adı verilir.

### 5.3.5.Prototip Fonksiyonlar

Buraya verilen örnekler ile yapılan açıklamlar ışığında ana fonksiyon içerisinde yer alan bir ifadenin ana fonksiyondan önce yazılması gereği bilinmektedir. Bu kural fonksiyonlar içinde geçerlidir. Ancak C++ programcıları bu kurala uymayıp fonksiyonları ana fonksiyondan sonra yazarlar. Derleyicinin çalışma prensibine göre ana fonksiyon içerisinde yer alan tüm bildirimler, önceden tanımlanmış olmalı ve hatta program akışına uygun olarak doğru yerlerde konumlanması gereklidir. Bir fonksiyonun yararı aynı kodun bir program içerisinde farklı noktalarda tekrar yazılmadan kullanılmasını sağlama olduğu dikkate alındığında bu istisnai durumun neden varlığını sürdürüğü anlaşılabilir. Bir çok yazılım projesi birden çok sayıda programın bir araya getirilmesi ile oluşmakta olduğu bilindiğinden benzer bir durumda çok sayıda tekrarlanan işleve sahip bir olan programın farklı kod bloklarının yazılması ve sonrasında birleştirilmesini gerektirdiği durumlarda prototip fonksiyonlar kod yazma sürecini kolaylaştırmaktadır. Yazılması gereken farklı fonksiyonlar

önceden tanımlanabilir ve daha sonra asıl program kodunda yerini alabilir. Eksik kalan veya çıkarılan fonksiyonlar için sadece prototip fonksiyon tanımının silinmesi yeterlidir. Ancak programın doğru şekilde yazılması istenirse an fonksiyondan önce her bir fonksiyonun programdaki sırası dikkate alınarak ana fonksiyondan önce yazılması gerekecektir. Önceki örneğin bu kural dikkate alınarak yazılmış şekli aşağıdadır.

```
/*
 * hangisi_büyük.cxx
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
 * SONUCUNDAN OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
 * VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
 * BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
```

```

* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
* SORUMLU DEĞİLLERDİR.

*
*/



#include <iostream>
#include <locale>
using namespace std;
// Değişkenler
double sayi1=0, sayi2=0, sayi3=0;
// Fonksiyonlar
double buyuktur(double sayi1, double sayi2)
{
    double deger;
    if (sayi1>=sayi2)
        deger = sayi1;
    else
        deger = sayi2;
    return deger;
}
double karsilastir(double sayi1, double sayi2, double sayi3)
{
    return buyuktur(sayi1, buyuktur(sayi2, sayi3));
}
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    cout << "Bu program klavyeden girilen sayılarından büyük olanını bulur ve yazdırır." << endl;
    cout << "Lütfen aralarında boşluk bırakarak iki sayı giriniz. ";
    cin >> sayi1 >> sayi2;
    cout << endl;
}

```

```

cout << "Büyük olan sayı " << buyuktur(sayı1, sayı2) << "\dir." << endl;
cout << "Bir sayı daha girin: ";
cin >> sayı3;
cout << endl;
cout << "Girilen üç sayıdan en büyüğü " << karsilastir(sayı1, sayı2, sayı3) << "\dir." <<
endl;
return 0;
}

```

### 5.3.6.Değer Döndüren Fonksiyonlarda Önemli Konular

Değer döndüren fonksiyonlarda dikkat edilmesi gereken önemli bir nokta fonksiyonun da adından anlaşılacağı üzere fonksiyonun çağrılması ve çalıştırılması durumunda mutlaka bir değer döndürmek durumunda olmalıdır. Aşağıdaki fonksiyon örneğini ele alalım.

```

double birfonksiyon(int x)
{
    if( x> 2)
        return pow (x,2.0)
}

```

Bu fonksiyon değer döndüren fonksiyon olarak tasarlanmıştır. Eğer fonksiyona aktarılan asıl parametrenin değeri 2'den küçük olursa fonksiyon herhangi bir işlem yapmayacaktır. Çünkü fonksiyon gövdesindeki mantıksal koşul parametrenin ikiden büyük olması durumunda yani mantıksal ifadenin sonucunun **doğru – true** olması durumunda işlem yapılmasını gerektirmektedir. Asıl parametre ikiden küçük ise mantıksal ifadenin sonucu **yanlış – false** olmaktadır. Bu durumda fonksiyon gövdesi işlenmez. Ancak fonksiyon değer döndüren bir fonksiyon olduğundan döndürülecek olan sonucun ne olacağı bilinemez. (işlemcide o an ne işleniyorsa ondan arta kalanlar döndürülür) Bu durumun neden olacağı sorunların ortadan kaldırılması için yukarıda verilen fonksiyonun aşağıdaki gibi yazılması problemi ortadan kaldıracaktır.

```

double birfonksiyon(int x)
{
    if( x> 2)
        return pow (x,2.0)
    return x;
}

```

Bazı fonksiyonlarda ise programcılar satır sayısını azaltmak için önceki örnek programdaki gibi “return ifade” yapısını tercih eder. Bu yaklaşım beraberinde ifadenin yazılış biçimine göre bazı

**hataları – bugs** beraberinde getirir. Değer döndüren fonksiyonlar sadece tek bir değer döndürmektedir .Bu nedenle ifadenin yazılış biçimini hataya neden olabilir. Derleyicinin işleyişini açısından ifadenin yazılışında herhangi bir hata olmadığı için hatası derlenmiş gibi algıansa da fonksiyonun döndürdüğü değer beklenen değer olmayacağıdır. Aşağıdaki örnek program derlenirken derleyici hata mesajı döndürmekte ama derleme işlemini bitirmektedir .

```
/*
 * bug_fonksiyon.cxx
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak ve ikili biçimlerde
 * kullanmak, aşağıdaki koşullar yerine getirildiği takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını, şartlar listesini ve aşağıdaki
 * ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı, şartlar listesi ve aşağıdaki
 * ret yazısı dağıtımla birlikte gelen belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
 * SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
 * VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
 * BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
```

```

* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
* SORUMLU DEĞİLLERDİR.

*
*/



#include <iostream>
#include <locale>
using namespace std;
// Değişkenler
int sayı=5;
// Fonksiyonlar
int bug_fonksiyon1()
{
    int x = 2;
    return 22, x;
}
int bug_fonksiyon2(int k)
{
    int a = 2, b = 3;
    return a*b, k*a;
}
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    cout << "Birinci fonksiyonun sonucu: " << bug_fonksiyon1() << endl;
    cout << "İkinci fonksiyonun sonucu: " << bug_fonksiyon2(sayı) << endl;
    return 0;
}

```

Programın derlenmesi sırasında “return” ifadelerinde kullanılmayan değerler ortaya çıkarılır. Bu uyarı mesajı görüntülenir ama program derlenir. Programın çalıştırılması sırasında görülebilir.

ceği üzere birden çok sayıda ifadenin tek bir return ile kullanılması durumunda en son ifadenin sonucu döndürülecektir. Programın çalıştırılması ile aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar]$ ./bug_fonksiyon
```

Birinci fonksiyonun sonucu: 2

İkinci fonksiyonun sonucu: 10

```
[goksin@tardis ~/projeler/C++_Notlar]$
```

**Örnek Program:** Bir çift zarın aynı anda atılması sonucunda üst yüzlerinde gelen sayıların toplamını bulan bir program yazınız.

**Çözüm:** Programda zarların atılmasını temsil etmek için rastgele sayı üretici kullanılacaktır. Bulunan değerin sıfır olması durumu dikkat alınarak hesaplanan değer bir eklenecek sonuç elde edilecektir. Bu işlem iki adet zar istediği için iki defa tekrarlanacaktır. İki zar toplamı istenen sonuca eşit olup olmadığı kontrol edilecektir. Eğer eşit değil ise zar atma işlemi tekrarlanacaktır. İşlem sonucu tekrar kontrol edilecek ve sonuç elde edilene kadar tekrarlanacaktır.

### Programın algoritması:

- 1 Başla
- 2 Elde dilmesi istenen toplamı oku.
- 3 Eğer okunan değer onikiden büyük ise hata mesajı döndürüp programı sonlandır.
- 4 Sayac değerini bir yap.
- 5 Birinci zarın değerini hesapla
- 6 İkinci zarın değerini hesapla
- 7 İki zarın değerini topla
- 8 Eğer toplam istenen sayıya eşit ise sonucun kaç adımda elde edildiğini belirten mesajı yaz ve 10. adıma git.
- 9 Eğer toplam istenen sayıya eşit değilse 5. adıma git.
- 10 Son.

**Programın Yazılması:** Program kullanıcından istenilen toplam sayısını girmesini isteyecektir. İki det zarı canlandırması için **rand()** fonksiyonu kullanılacak. Her iki zar için ayrı olarak değerler hesaplanacaktır. Her zar atma işlemi için bir **sayaç** kullanılarak atım sayısı izlenecektir. Eğer sonuç istene toplama eşit ise program sonlanacak, eşit değilse tekrar zar atma işlemi canlandırılacaktır. Döngü elde edilmek istene sonuç bulunan kadar tekrarlanacağı için **do .. while** döngüsü kullanılmalıdır.

Programın kaynak kodu aşağıdaki gibidir:

```
/*
* zar.cxx
*
```

\*

\* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
/*
#include <iostream>
#include <locale>
#include <cstdlib>
#include <ctime>
using namespace std;
// Değişken tanımlama
int toplam;
// Fonksiyon
int ZarAt(int toplam)
{
    int zar1;
    int zar2;
    int sonuc = 0;
    int zar_atim = 0;
    srand(time(0));
    do
    {
        zar1 = rand()%6 + 1;
        zar2 = rand()%6 + 1;
        sonuc = zar1 + zar2;
        zar_atim++;
    }
    while (sonuc != toplam);
    return zar_atim;
}
int main()
{
    setlocale(LC_ALL, "Turkish");
    cout << "İki zarın toplamı ne olsun? ";
    cin >> toplam;
```

```

if (toplam > 12)
{
    cout << "Girilen değer en çok 12 olabilir. Hatalı giriş yapıldığı için program sonlanıyor." << endl;
    return 0;
}

cout << "Belirttiğiniz toplam " << ZarAt(toplam) << ". atışta elde edildi" << endl;
return 0;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./zartoplasm
```

İki zarın toplamı ne olsun? 22

Girilen değer en çok 12 olabilir. Hatalı giriş yapıldığı için program sonlanıyor

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./zartoplasm
```

İki zarın toplamı ne olsun? 11

Belirttiğiniz toplam 7. atışta elde edildi.

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

**Örnek Program:** Klavyeden girilen bir sayı veya cümlenin palindrom<sup>11</sup> olup olmadığını belirleyen programı yazınız.

**Çözüm:** Palindrom olan sayılar sözcükler veya sayılar düz veya ters okunduğunda da aynı şekilde okunan sayı, sözcük ve cümlelerdir. Bu problemin çözümünde klavyeden girilen veri karakter dizisi olarak işlem alınmalıdır. Tersten ve düzden okunuşun aynı olup olmadığını kontrol etmek için bir baştan ve bir sondan alınan karakterler karşılıklı olarak kontrol edilir. Kontrol edilen tüm karakterler birbiri ile aynı ise sayı, sözcük veya cümle palindrom olarak kabul edilir. Bunun için karakter dizisinin uzunluğu elde edilmelidir. Karakter dizisinin uzunluğu belirlendikten sonra dizideki ilk karakter ile son karakter karşılaştırılır. Ardından sıradaki karakter ile sondan bir önceki karakter karşılaştırılır. Bu işlem baştan ve sondan eşleştirilen karakterler karşılaşırıldığında tamamlanıncaya kadar veya karşılaşırılan karakterlerden herhangi ikisi aynı olmadığı belirlenene kadar devam eder.

A	B	C	D	E	D	C	B	A
0	1	2	3	4	5	6	7	8

Yukarıdaki örnek karakter dizisinin uzunluğu dokuzdur. Ancak ilk karakterin indisini sıfırdan başladığı için son elamanın indisini sekiz olmaktadır. Karşılaştırma işleminde [0] ile [8], [1] ile [7], [2] ile [6], [3] ile [5] karşılaştırılır. Ortada kalan elaman için işlem yapılmaz.

<sup>11</sup> <https://tr.wikipedia.org/wiki/Palindrom>

## **Programın algoritması:**

- 1 Başla
- 2 Kontrol edilecek olan sayı, sözcük veya sayı oku.
- 3 Okunan verinin uzunluğunu bul.
- 4 Sayac 0 olsun.
- 5 Veri dizisinin [sayac] elamanını oku.
- 6 Veri dizisinin [uzunluk – 1 – sayac] elamanını oku.
- 7 Okunan elemanları karşılaştır.
  - 7.1 Eğer okunan elemanlar aynı değilse değildir döndür ve 8 adıma geç
  - 7.2 Eğer okunan elemanlar aynı ise sayacı bir arttır ve 5 adıma dön
- 8 Palindrom değildir yazdır ve 10 adıma git.
- 9 Palindromdur yazdır
- 10 Son.

**Programın Yazılması:** Program yazılırken palindrom olma durumunu kontrol etmek için bir fonksiyon kullanılacaktır. Fonksiyon okunan verinin uzunluğunu belirleyecektir. Döngü kullanılarak baştan ve sondan birer elaman okunara karşılaştırılacaktır. Bunun için bir döngü kurulup döngünün adım sayısı uzunluk değerinin yarısı olarak alınacaktır. Karakter dizisinin [0] indisli elmanı ilse [uzunluk -1 adım\_sayısi] karşılaşılacaktır. Eğer iki karakter de aynı ise işleme devam edilip sayaç bir artılır, aksi durumda döngüden çıkarılır. Ana fonksiyon fonksiyondan döndürülen değeri kontrol edecektir. Eğer döndürulen değer 2 ise palindrom, 1 ise palindrom değildir yazdırılacaktır.

Programın kaynak kodu aşağıdaki gibidir:

```
/*
 * palindrom.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
```

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
#include <locale>
#include <string>
using namespace std;
// Değişken tanımlama
string veri;
// Fonksiyon
int kontrol (string islenecek_veri)
```

```

{
    int uzunluk = islenecek_veri.length();
    for (int i=0; i < uzunluk / 2; i++)
        if (islenecek_veri[i] != islenecek_veri[uzunluk - 1 - i])
            return 1;
    return 2;
}

int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    cout << "Bir sayı, sözcük veya cümle yazıp enter tuşuna basınız: ";
    getline(cin, veri);
    kontrol(veri);
    if (kontrol(veri) == 2)
    {
        cout << "Girilen sayı, sözcük veya cümle palindromdur." << endl;
        return 0;
    }
    cout << "Girilen sayı, sözcük veya cümle polindrom değildir." << endl;
    return 0;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./palindrom
Bir sayı, sözcük veya cümle yazıp enter tuşuna basınız: 12321
Girilen sayı, sözcük veya cümle palindromdur.

[goksin@tardis ~/projeler/C++_Notlar/]$ ./palindrom
Bir sayı, sözcük veya cümle yazıp enter tuşuna basınız: AMA
Girilen sayı, sözcük veya cümle palindromdur.

[goksin@tardis ~/projeler/C++_Notlar/]$
[goksin@tardis ~/projeler/C++_Notlar/]$ ./palindrom
Bir sayı, sözcük veya cümle yazıp enter tuşuna basınız: 122
```

Girilen sayı, sözcük veya cümle palindrom değildir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./palindrom
```

Bir sayı, sözcük veya cümle yazıp enter tuşuna basınız: anastas mum satsana

Girilen sayı, sözcük veya cümle palindromdur.

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

### 5.3.7. Programın Derlenmesi ve Program İşleyışı

Önceki bölümlerde bir C++ programının birden çok sayıdaki fonksiyonun birleşiminden oluştuğunu belirtmiştik. Ayrıca fonksiyonların yazılışında belirli bir sıra izlenmesi gerekmekte olmasına rağmen bunun dışına çıkışabildiğini de görmüştük. Bu durumda prototip fonksiyonlar kullanılırsa, programın derlenmesi sırasında ilk derlenecek olan fonksiyon main() olacaktır. Bundan sonra sırası ile gelen tüm fonksiyonlar derlenir. Eğer protoip fonksiyonlar kullanılmıyor ise fonksiyonlar sırası ile ilkinden başlanarak sona doğru derlenir.

Programın derlenmesinde yazım sırası önemli olsa da, programın işleyişinde ise bu sıra önemli değildir. Program çalıştırıldığında önce belleğe yüklenir. Belleğe yüklenen program parçalar halinde işlemciye aktarılır ve ilk olarak main() çalıştırılır. Program içerisinde diğer fonksiyonlar ise ancak main() içerisinde çağrıldıklarında işlenir. Bir fonksiyon çağrılmınca program akışı çağrı yapılan noktada kesilir ve program fonksiyonun olduğu satırda geçerek oradan işlemeye devam eder. Fonksiyonun işlenmesi ve ardından değeri döndürmesi ile program akışı tekrar kesintiye uğradığı noktaya geçer ve oradan işlemeye devam eder.

## 5.4. Void – Değer Döndürmeyen Fonksiyonlar

Değer döndüren fonksiyonlar ile değer döndürmeyen fonksiyonlar arasında yapısal olarak tek fark, bu fonksiyonların bir veri tipi ile tanımlanmamalıdır. Teknik olarak değer döndüren fonksiyonlarda olduğu gibi fonksiyonun adı, formal parametreleri ile fonksiyon gövdesi bulunmak tadır. Bu fonksiyonlar prototip fonksiyon olarak yazılabilcekleri gibi aynı zamanda normal olarak da yazılabilir. Bir değer döndürmeyen fonksiyon aşağıdaki gibi yazılabılır.

```
void fonksiyonAdı (veri tipi formal parametre)  
{  
    ifade;  
}
```

Bu fonksiyonlarda da formal parametrelerin sayısında bir sınırlama bulunmamaktadır. Bir değer döndürmeyen fonksiyon program içerisinde çağrılabileceği zaman fonksiyonun adı ve bu fonksiyona aktarılacak olan asıl parametreler ile çağrılabılır.

Değer döndürmeyen fonksiyonların kullanılmasının yararı, değer döndüren fonksiyonlarda sadece tek bir değer döndürebilecek iken bu fonksiyonlarda bu kısıtlamaların olmamasıdır. Böylece birden çok sayıda değer ana fonksiyona döndürülebilir. Bunun yapılabilmesi için de değer dön-

dürmeyen fonksiyonlarda kullanılan **formal ve asıl parametrelere** ek olarak **referans ve değer parametreleri** kullanılır.

**Değer parametreleri:** Tek bir formal parametreye karşılık gelen ve üzerinde işlem yapılan asıl parametredir. Parametreye atanın değer işlem yapılması sonucunda değişir.

**Referans parametreleri:** Fonksiyona tanımlanmış olan tek bir formal parametreye karşılık gelen ve asıl parametrenin bellek üzerinde yerinini belirtir.

**Bir fonksiyonda tanımlanan formal parametreye ait olan veri tipinin sonuna getirilen & ile parametre, bir referans parametresine dönüşür.**

Aşağıda gösterilen program çalıştırıldığında ekrana “\*” karakterini kullanarak bir üçgen bölge oluşturmaktadır. Program üçgen bölge oluşturmak için önce kullanılacak boşluk sayısını ve ardından da her bir satırda bulunacak olan “\*” karakter sayısını hesaplamaktadır. Her bir satırda yıldız sayısı artarken, boşluk sayısı azalmaktadır. Bu işlemler için bir döngü kullanılmaktadır. Döngü iki parametreyi esas bir fonksiyonu kullanarak çalışmaktadır. Satır sayısı sayı ile tutulurken boşluk sayısı da her döngü sonunda azaltılmaktadır. Programın ekranda kaplayacağı yer düşünülerek satır sayısı 20 ile sınırlandırılmıştır. Kullanıcının tek bir hata yapması sonucunda programın sona ermesi yerine doğru aralıkta giriş yapabilmesi için de bir while döngüsü kullanılmıştır. Programın kaynak kodu aşağıda verilmiştir.

```
/*
* yıldızlar.hxx
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
* takdirde serbesttir:
*
* Kaynak kodun yeniden dağıtım; yukarıdaki telif hakkı uyarısını,
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
*
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
*
```

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
```

```
#include <locale>
```

```
using namespace std;
```

```
void YildizYaz (int bosluklar, int yildizlar );
```

```
int main()
```

```
{
```

```
    setlocale(LC_ALL, "tr_TR.UTF-8");
```

```
    int satirlar, sayac, boslukSayisi;
```

```
    cout << "Satır sayısını ( 1 ile 20 arası) giriniz: ";
```

```
    cin >> satirlar;
```

```
    while ( satirlar <0 || satirlar > 20)
```

```
{
```

```
    cout << "Yıldızların yer alacağı satır sayısını giriniz: ";
```

```
    cin >> satirlar;
```

```

    }

cout << endl << endl;
boslukSayisi = 30;
for (sayac = 1; sayac <= satirlar; sayac++)
{
    YildizYaz(boslukSayisi, sayac);
    boslukSayisi--;
}

cout << endl << endl;
return 0;
}

void YildizYaz (int bosluklar, int yildizlar )
{
    int adet;
    for (adet = 1; adet <= bosluklar; adet++)
        cout << ' ';
    for (adet = 1; adet <= yildizlar; adet++)
        cout << "*";
    cout << endl;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./yildizlar
```

Satır sayısını ( 1 ile 20 arası) giriniz: 5

```

*
*
* *
* * *
* * * *
* * * * *
```

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

## 5.5.Değer Parametreleri

Onceki bölümde parametrelerin çeşitleri arasında değer parametrelerinden söz edilmişti. Yukarıdaki program da, iki adet parametreyi kullanan bir örnek olarak sunulmuştur. Bir değer parametresi fonksiyonda kullanıldığında, fonksiyonda tanımlanmış olan forma parametrelerle verinin aktarılması için kullanılır. Veri program içerisinde bir değişken veya ifadenin sonucu olarak aktarılmış olsa da fonksiyon içerisinde yer alan formal parametreye aktarılmış olduğunda fonksiyon içerisinde sadece tanımlı parametre aracılığı ile işlenir. Değişkendeki değer aynen korunurken fonksiyondaki işlem sonucunda farklı bir değer elde edilir. Fonksiyona ait verileri barındırdığı bellek alanı ile programa ait bellek alanı birbirinden ayrıdır. Aşağıdaki programda bu durum gösterilmektedir .

```
/*
 * fonksiyon_parametre_değer.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
```

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
 \* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
 \* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
 \* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
 \* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
 \* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
 \* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
 \* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
 \* SORUMLU DEĞİLLERDİR.  
 \*  
 \*/  
 /\*  

```

#include <iostream>
#include <locale>
using namespace std;
void fonksiyon_parametre_değer(int sayı);
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    int sayı = 6;
    cout << "Fonksiyon çağrılmadan önce değişkenin değeri: " << sayı1 << endl;
    fonksiyon_parametre_değer(sayı);
    cout << "Fonksiyon çağrıldıktan sonra değişkenin değeri: " << sayı1 << endl;
    return 0;
}
void fonksiyon_parametre_değer(int sayı);
{
    cout << "Fonksiyon içerisinde değişkenin işlenmeden önceki değeri: " << sayı1; << endl;
    sayı = 10;
    cout << "Fonksiyon içerisinde değişkenin işlendikten sonraki değeri: " << sayı1; << endl;
}
```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./fonksiyon_parametre_değer
```

Fonksiyon çağrılmadan önce değişkenin değeri: 6

Fonksiyon içerisinde değişkenin işlenemeden önceki değeri: 6

Fonksiyon içerisinde değişkenin işlendikten sonraki değeri: 10

Fonksiyon çağrıldıktan sonraki değişkenin değeri: 6

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Programın incelenmesinden görüleceği üzere ana fonksiyon içerisinde tanımlanmış olan değişkenin değeri belirlidir. Fonksiyon çağrılmadan önceki değer de bu değerdir. Fonksiyona çağrı yapıldıktan sonra program akışı fonksiyona geçmektedir. Fonksiyona asıl değer olarak değişkenin değeri aktarılmaktadır. Fonksiyon öncelikle değer olarak aldığı veriyi yani değişkenin ilk değerini yazdırmaktadır. Fonksiyon içerisinde ana fonksiyonda tanımlanmış olan değişken ile aynı isime sahip olan bir değişken kullanılmıştır. Fonksiyon çalışırken ikinci işlem değişkenin değişkenin değiştirilmesidir. Bundan sonra değişkenin yeni değeri yazdırılmaktadır. Fonksiyonun çalışması sona erinde ana fonksiyona geri dönülmektedir. Son olarak da ana fonksiyonda değişkenin değeri tekrardan yazdırılmaktadır.

Bu tür fonksiyonlarda parametre kullanılarak değer aktarıldığında, fonksiyon kendisine aktarılan değeri işleyerek sonuç elde eder. Ancak fonksiyon değer döndürmediği için de ana fonksiyon içerisinde parametre ile aktarılan değer fonksiyon içerisinde yapılan işlemden bağımsız olarak varlığını korur. Fonksiyon işlem yaptıktan sonra elde ettiği değeri yapısında barındırır. Eğer bu sonuç ana fonksiyona herhangi bir şekilde aktarılmayacak ise fonksiyon sona erdiğinde bellekten silinir.

## 5.6.Referans (Değişkenin Bellek Adresi) Parametreleri

Bir değer aktarımı için kullanılan parametrenin asıl fonksiyondan bağımsız olarak işlemekte olduğu durumlarda fonksiyonun işlem sonucu dışarının aktarılabilmesi için bir dosyaya yazılması ve b dosya ile verinin aktarılması söz konusu olacaktır. Bu dolaylı yöntem yerine fonksiyon parametre ile değeri okuyup işlemek yerinde doğrudan ilgili verinin bulunduğu bellek adresini parametre olarak alabilir. Bu durumda parametre bellek adresini referans olarak gösterdiği için de fonksiyon yaptığı işlemler sonucunda söz konusu bellek adresindeki veri üzerinde işlem yapılır.

Referans parametreleri şu durumlarda kullanılması tercih edilmektedir:

- Parametrenin barındırdığı veri üzerinde işlem yapılması gerekiyor ve bu değişikliğinde tüm program için geçerli olması isteniyor ise,
- Bir fonksiyonun tek bir değer döndürmesi yerine birden çok sayıdaki değeri döndürmesi gerekiyor ise,
- Büyük miktardaki verinin bellek üzerinde kopyalanması, taşınması ve yer değiştirmesi gibi yüksek maliyetli işlemlerin yapılması istenmiyorsa ve doğrudan bellek üzerinde asıl veri üzerinde işlemlerin yapılması isteniyor ise.

İlk iki durum, bu ders notlarında bu bölüme kadar olan konular içerisinde yer verilmiştir. Son durum ise diziler ve sınıflar konusu ele alındığında incelenecektir.

Bir parametrenin veri tipin tanımlamasının yanına “&” işaretini konulursa bu durumda bir referans parametresi yaratılmış olur.

Aşağıda verilen program bir adet öğrenci için sınav notlarını girildikten sonra buna göre dönem sonu başarı notunu hesaplamaktadır. Programda üç adet fonksiyon kullanılmıştır. Bunlar NotOku, HarfNotu ve main fonksiyonlarıdır. Bu fonksiyonların sırası ile yaptıkları işlemler şunlardır:

## 1 NotOku

- 1.1 Kullanıcıdan notları girmesini iste
- 1.2 Girilen otları oku
- 1.3 Dönem sonu harf notunu hesapla ve yaz
- 1.4 Dönem sonu harf notunu döndür

## 2 HarfNotu

- 2.1 Dönem sonu başarı notunu oku
- 2.2 Dönem sonu harf notunu yaz

## 3 main

- 3.1 Dönem sonu başarı notunu oku
- 3.2 Dönem sonu harf notunu yaz

Programın kaynak kodu aşağıdaki gibidir

```
/*
* harf_notu_ne_olur.cxx
*
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
*
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
* takdirde serbesttir:
*
*
* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,
```

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
#include <locale>
#include <cfloat>
#include <iomanip>
using namespace std;

const float k1=0.20, k2=0.20, k3=0.60;

float basari_not=0.0;

void NotOku(int& ara_sinav1, int& ara_sinav2, int& donem_sonu);
```

```

void HarfNotu(float& basari_not);

int main()
{
    setlocale(LC_ALL, "Turkish");

    int ara_sinav1=0, ara_sinav2=0, donem_sonu=0;

    cout << "Bu program dönem içi değerlendirmelerinden alınan notları kullanulanarak dönem sonu harf notunu belirler." << endl ;

    cout << "Birinci ara sınav notunu giriniz ve enter tuşuna basınız: ";

    cin >> ara_sinav1;

    cout << "İkinci ara sınav notunu giriniz ve enter tuşuna basınız: ";

    cin >> ara_sinav2;

    cout << "Dönem sonu sınav notunu giriniz ve enter tuşuna basınız: ";

    cin >> donem_sonu;

    NotOku(ara_sinav1, ara_sinav2, donem_sonu);

    HarfNotu(basari_not);

}

void NotOku(int& ara_sinav1, int& ara_sinav2, int& donem_sonu)
{
    basari_not = (k1*ara_sinav1) + (k2*ara_sinav2) + (k3*donem_sonu);

    cout << fixed << setprecision(2);

    cout << "Dönem sonu başarı notu: " << basari_not << endl;
}

void HarfNotu(float& basari_not)
{
    if (basari_not >= 84.0)

        cout << "Harf notu: AA" << endl;

    else if(basari_not >= 77.0)

        cout << "Harf notu: AB" << endl;

    else if(basari_not>=71.0)

        cout << "Harf notu: BA" << endl;

    else if(basari_not>=66.0)
}

```

```

cout << "Harf notu: BB" << endl;
else if(basari_not>=61.0)
    cout << "Harf notu: BC" << endl;
else if(basari_not>=56.0)
    cout << "Harf notu: CB" << endl;
else if(basari_not>=50.0)
    cout << "Harf notu: CC" << endl;
else if(basari_not>=46.0)
    cout << "Harf notu: CD" << endl;
else if(basari_not>=40.0)
    cout << "Harf notu: DC" << endl;
else if (basari_not>=33.0)
    cout << "Harf notu: DD" << endl;
else cout << "Harf notu: FF" << endl;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./harf_notu_ne_olur
```

Bu program dönem içi değerlendirmelerinden alınan notları kullanarak dönem sonu harf notunu belirler.

Birinci ara sınav notunu giriniz ve enter tuşuna basınız: 62

İkinci ara sınav notunu giriniz ve enter tuşuna basınız: 78

Dönem notunu giriniz ve enter tuşuna basınız: 69

Dönem sonu başarı notu: 69.40

Harf Notu: BB

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Programın kaynak kodu incelediğinde öğrencinin başarı notunun hesaplanabilmesi için gerekli olan sınavlar ait oranlar sabit olarak tanımlanlığı görülmektedir. Ardından gelen satırda başarı notunun hesaplanması be bellekte barındırılması için gerekli olan değişkenin float veri tipinde ve basari\_notu adı ile tanımlanıldığı görülmektedir. Bu değişken tanımlaması bir genel değişken yani global değişken tanımlamasıdır. Bunun kullanılmasının nedeni ise değer döndürmeyen fonksiyon kullanılmamasındandır. Bir değer döndürmeyen fonksiyon farklı şekillerde kullanılabilir. Burada ise genel bir değişken üzerinde yapılacak olan işlemler bulunduğu için kulla-

nîması tercih edilmiştir. Programın yapısında bulunan fonksiyonlar prototip olarak tanımlanmıştır. Ardından da ana fonksiyonun ardından sırası ile yazılmışlardır.

Programın tasarımindan, başarı notunun belirlenmesinde değişimyeni çekirdek işlevlerin ana fonksiyon yapısında yapılması tercih edilmiştir. Bunlar öğrencinin dönem içi değerlendirme sonuçları ile dönem sonu değerlendirme sonucudur. Bu değişkenler program çalıştırıldığında bellekte etkinleştirilmektedir. Programın başlangıç kısmında tanımlanmış olan basari-not adlı değişken de önceden etkinleştirilmiştir.

Sabitler	<b>k1=0.20</b>	<b>k2=0.20</b>	<b>k3=0.60</b>	
<b>Genel değişken</b>				<b>basari_not=0</b>
<b>Fonksiyon</b>				<b>basari_not=0</b>
<b>main</b>	<b>ara_sinav1=0</b>	<b>ara_sinav2=0</b>	<b>donem_Sonu=0</b>	<b>basari_not=0</b>

Ana fonksiyon çalıştırıldığında öğrenciye ait sınav verisi okunup ilgili değişkenlere atanmaktadır. Ana programda sıradaki işlem olarak NotOku fonksiyonu çağrılmaktadır. Bu fonksiyonda ara\_sinva1, ara\_sinav2 ve donem\_sonu notlarını okuyup k1, k2 ve k3 sabitleri ile çarpıp sonucu basari\_not adlı değişkene atamaktadır.

Sabitler	<b>k1=0.20</b>	<b>k2=0.20</b>	<b>k3=0.60</b>	
<b>Genel değişken</b>				<b>basari_not=0</b>
<b>Fonksiyon</b>				<b>basari_not=0</b>
<b>main</b>	<b>ara_sinav1=62</b>	<b>ara_sinav2=78</b>	<b>donem_Sonu=69</b>	<b>basari_not=0</b>
<b>NotOku</b>	<b>62</b>	<b>78</b>	<b>69</b>	<b>basari_not=69.40</b>

NotOku fonksiyonun işlemi bittiğinde yukarıdaki tabloda görülen durum elde edilir. NotOku fonksiyonu için formal değişkenler aynı zamanda ana fonksiyonda tanımlanmış olan değişkenlerdir. Bu nedenle de bu fonksiyon içerisinde yeni değişkenler tanımlanmayıp doğrudan na fonksiyonda tanımlanan değişken değerleri kullanılmıştır. Son işlem basamağı olarak NotOku adlı değişken, basari\_not adlı değişkenin değerini değiştirmiştir. Bu değişken genel bir değişken olduğu için programdaki tüm fonksiyonlar ve ifadeler bu değişken üzerinde işlem yapabilir. Bu nedenle bu değişkenin değeri NotOku adlı fonksiyonun işlem yapmasının ardından değişmiştir. Bu fonksiyon işlemini bitirdiğinde ana fonksiyona geri dönüş gerçekleşir. Ana fonksiyondaki akış sırası gereği sıradaki fonksiyon olan HarfNot adlı fonksiyon çağrılr. Bu fonksiyon sadece basari\_not adlı değişkenin değeri ile işlem yapmaktadır.

Sabitler	<b>k1=0.20</b>	<b>k2=0.20</b>	<b>k3=0.60</b>	
<b>Genel değişken</b>				<b>basari_not=0</b>
<b>Fonksiyon</b>				<b>basari_not=0</b>
<b>main</b>	<b>ara_sinav1=62</b>	<b>ara_sinav2=78</b>	<b>donem_Sonu=69</b>	<b>basari_not=0</b>
<b>NotOku</b>	<b>ara_sinav1=62</b>	<b>ara_sinav2=78</b>	<b>donem_Sonu=69</b>	<b>basari_not=69.40</b>
<b>HarfNot</b>				<b>basari_not=69.40</b>

HarfNot adlı fonksiyon de NotOku adlı fonksiyonun yaptığı gibi genel değişkenin değeri ile işlem yapmaktadır. Bu fonksiyon bir dizi if ... then ... else ifadesinden oluşmaktadır. İfadenin mantıksal sonucuna göre ilgili mesaj standart çıktıya döndürülmektedir .

Yukarıdaki program örneğinde değer döndürmeyen fonksiyonların özelliği gereği bir sonuç döndürmezler. Fonksiyonun işlenmesi bittiğinde işlem sona erer. Ancak genel değişkenler ise bu tür durumlarda bir programda çeşitli değer döndürmeyen fonksiyonların tanımlanıp belirli bir bellek alanında bulunan bir değişken üzerinde işlem yapabilirler. Bu referans gösterme yolu ile yapılan işlemler sonucunda ilgili bellek alındıktaki değişkenin değeri değişir.

## 5.7.Değer ve Referans Parametreleri ile Bellek Atamaları

Bir fonksiyon üzerinde işlem yapacağı değişkenler ile fonksiyona ait formal parametreler için bilgisayar belleğinde alan ayrılır. Bir fonksiyonun yapısında tanımlanan değişkenler ile fonksiyona ait olan parametrelere yerel değişken adı verilir. Yerel değişkenler de adından da anlaşılacağı üzere sadece fonksiyon tarafından erişilir ve üzerinde işlem yapılabilir. Eğer formal parametrelər, bir referans olarak tanımlanmış ise bu durumda bellek üzerinde bulunan bir verinin barındırıldığı bellek adresinin bilgisini edinir ve bu bellek alındıktaki veri üzerinde işlem yapar. Bu durumda hem formal parametrelər hem de referans parametreləri aynıdır. Eğer program çalışırken bir fonksiyon bu referans verilen bellek adresi üzerindeki bilgi ile işlem yapabilir ve değişkenin değeri değişir. Parametrelerin aktarımı tüm programlama dilleri için önemli olduğu unutulmamalıdır. Aşağıdaki örneklerde parametrelərin aktarımı incelenmiştir.

```
/*
 * referansparamtre.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtım; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
```

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen  
\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

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

void fonksiyon_bir(int a, int& b, char v);
void fonksiyon_iki(int& x, int y, char& z);

int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    int say1, say2;
    char karakter;
    say1 = 11,
```

```

say2 = 13;
karakter = 'B';
cout << "Ana fonksiyondaki değişkenler:" << endl;
cout << "say1= " << say1 << " " << "say2= " << say2 << " " << "karakter= " << karakter <<
" " << endl;
cout << "Fonksiyon_bir geçiliyor." << endl;
fonksiyon_bir (say1, say2, karakter);
cout << "Ana fonksiyona geri dönüldü. Değişkenlerin durumu aşağıdaki gibidir:" << endl;
cout << "say1= " << say1 << " " << "say2= " << say2 << " " << "karakter= " << karakter <<
" " << endl;
fonksiyon_iki (say1, 22, karakter);
cout << "Ana fonksiyona geri dönüldü. Değişkenlerin durumu aşağıdaki gibidir:" << endl
<< endl;
cout << "say1= " << say1 << " " << "say2= " << say2 << " " << "karakter= " << karakter <<
" " << endl;
return 0;
}

void fonksiyon_bir (int a, int& b, char v)
{
    int bir;
    bir = a;
    a++;
    b = b * 2;
    v = 'C';
    cout << "fonksiyon_bir işlendi. Değişkenlerin durumu aşağıdaki gibidir:" << endl;
    cout << "a = " << a << " " << "b= " << b << " " << "v= " << v << " " << endl;
}

void fonksiyon_iki (int& x, int y, char& z)
{
    x++;
    y = y * 2;
    z = 'G';
}

```

```

cout << "fonksiyon Bir işlendi. Değişkenlerin durumu aşağıdaki gibidir:" << endl;
cout << "x = " << x << " " << "y= " << y << " " << "z= " << z << " " << endl ;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./referansparametre
```

Ana fonksiyondaki değişkenler:

say1= 11 say2= 13 karakter= B

Fonksiyon\_bir geçiliyor.

fonksiyon\_bir işlendi. Değişkenlerin durumu aşağıdaki gibidir:

a = 12 b= 26 v= C

Ana fonksiyona geri dönüldü. Değişkenlerin durumu aşağıdaki gibidir:

say1= 11 say2= 26 karakter= B

fonksiyon\_iki işlendi. Değişkenlerin durumu aşağıdaki gibidir:

x = 12 y= 44 z= G

Ana fonksiyona geri dönüldü. Değişkenlerin durumu aşağıdaki gibidir:

say1= 12 say2= 26 karakter= G

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Programın kaynak kodu incelediğinde ilk olarak main fonksiyonu çalıştırıldığından bu fonksiyonda tanımlanan değişkenler için bellek alanı ayrılmıştır. Ancak bu bellek alanlarına herhangi bir değer atanmamıştır. Ana fonksiyon içerisinde sona değişkenlere değerleri atanmıştır.

Fonksiyon	Değişkenler	Fonksiyon	Değişkenler	Açıklama
main	say1 = 11			Değişkenlere, değerleri atanmıştır.
	say2 = 13			
	karakter = 'B'			

Program akışı içerisinde fonksiyon\_bir geçiş yapılmıştır. Bu fonksiyonda üç adet formal parametre tanımlıdır. Ayrıca fonksiyon içerisinde de bir tane yerel değişken "bir" tanımlanmıştır. Bu fonksiyonda "b" adlı formal parametre aynı zamanda referans parametresidir. Böylece ana fonksiyondaki karşılık gelen değişkenin bellek adresine refere eder durumdadır. Bir diğer deyişle bu durumda değeri say2 adlı değişkenin değeri olmuştur.

Fonksiyon	Değişkenler	Fonksiyon	Değişkenler	Açıklama
main	say1 = 11	fonksiyon_bir	a = 11	
	say2 = 13		b = 13	

	karakter = 'B'		v= 'A'	
			bir = a = 11	

Bu işlemden sonra fonksiyon\_bir içerisinde a değişkenin değeri bir arttırılıp kendisine eşitlenmiştir. Böylece a değişkenin değeri 12 olmuştur. Fonksiyon\_bir içerisinde işleme devam edilmiş ve b adlı değişken iki çarpılara kelde edilen sonuç yine aynı adlı değişkene atanmıştır.

Fonksiyon	Değişkenler	Fonksiyon	Değişkenler	Açıklama
main	say1 = 11	fonksiyon_bir	a = 12	a++
	say2 = 26		<b>b = 26</b>	<b>b = b * 2</b>
	karakter = 'B'		v= 'B'	
			bir = a = 11	

Bu işlemlerin ardından fonksiyon\_bir içerisinde son işlem olarak v değişkeninin değeri C olarak değiştirilmiştir.

Fonksiyon	Değişkenler	Fonksiyon	Değişkenler	Açıklama
main	say1 = 11	fonksiyon_bir	a = 12	a++
	say2 = 26		<b>b = 26</b>	<b>b = b * 2</b>
	karakter = 'B'		v= 'C'	<b>v = 'C'</b>
			bir = a = 11	

Bu işlemlerin ardından fonksiyon\_bir işlenmesi sona ermiş ve program akışı yeniden main fonksiyonuna geçmiştir. Fonksiyon\_bir değişkenleri bellekten kaldırılmıştır. Ana fonksiyona ait olan değişkenler ve değerleri bellekte bulunmaktadır.

Fonksiyon	Değişkenler			
main	say1 = 11			
	say2 = 26			
	karakter = 'B'			

Ana fonksiyonda sırada ifade fonksiyon\_iki çağrılmaktadır. Bu fonksiyonda üç adet formal parametre bulunmaktadır. Birinci ve üçüncü parametreler aynı zamanda referans parametresidir. Böylece birinci ve üçüncü değişkenler doğrudan atanana değişkenlerin bellek adreslerini edinmişlerdir.

Fonksiyon	Değişkenler	Fonksiyon	Değişkenler	Açıklama
main	say1 = 11	fonksiyon_iki	<b>x = 11</b>	Referans pa.
	say2 = 26		y = 22	
	karakter = 'B'		<b>z= 'B'</b>	Referans pa.

Fonksiyon içerisindeki ikinci ifade y değişkenin iki ile çarpılarak sonucun yeniden y adlı değişkene atanmasıdır. Ardından gelen üçüncü ifade ise z değişkenin değerinin ‘G’ olarak değiştirilmesidir. Bu işlemler gerçekleştirildikten sonra an fonksiyon tarafından tanımlanmış olan say1 ve karakter adlı değişkenlerinin değerleri fonksiyon\_iki tarafından değiştirilmiş olmaktadır. Ana fonksiyon tarafından tanımlanmış olan saty2 adlı değişkenin değeri ise değişmemiştir.

Fonksiyon	Değişkenler	Fonksiyon	Değişkenler	Açıklama
main	say1 = 12	fonksiyon_iki	x = 12	x++
	say2 = 26		y = 44	y = y * 2
	karakter = ‘G’		z= ‘G’	z = ‘G’

İşlenmekte olan fonksiyon\_iki ile yapılan işlemler yazdırıldıktan sonra fonksiyon-iki ve ilgili olan değişkenler bellekten silinmektedir. Geriye yapılan işlemler sonucunda değişkenlerde gerçekleşen değişiklikler ana fonksiyon tarafından da kalıcıdır.

Fonksiyon	Değişkenler			
main	say1 = 12			
	say2 = 26			
	karakter = ‘G’			

Program içerisindeki ana fonksiyon sona ermeden önce işlem sonuçları yazdırılır. Bellekteki değişkenlerin barındırdıkları veri yukarıda görülmektedir.

Aşağıda verilen programda benzer şekilde fonksiyonların yapısında referans parametreleri kullanılmaktadır. Ana fonksiyonda tek bir değişken tanımlanmış ve bu değişkenin değeri de bilinmektedir. Her bir fonksiyon çağrılığında referans parametrelerin fonksiyonlarda kullanılması ile her bir fonksiyon tamamlandıktan sonra değişkenin değeri değişmektedir.

```
/*
* ne_oluyor.cxx
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
```

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
```

```
#include <locale>
```

```
using namespace std;
```

```
void toplama_yap(int& birinci, int& ikinci);
```

```
void iki_ile_carp(int bir, int iki);
```

```

void ikinci_kuvvet(int& referans, int deger);

int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");

    int say = 2;

    cout << "Ana fonksiyondaki değişkenin değeri:" << say << endl;
    toplama_yap(say, say);

    cout << "Ana fonksiyona geri döndü, değişkenin değeri: " << say << endl;
    iki_ile_carp(say, say);

    cout << "Ana fonksiyona geri döndü, değişkenin değeri: " << say << endl;
    ikinci_kuvvet(say, say);

    cout << "Ana fonksiyona geri döndü, değişkenin değeri: " << say << endl;
    return 0;
}

void toplama_yap(int& birinci, int& ikinci)
{
    cout << "toplama_yap fonksiyonunda değişkenler: " << birinci << " " << ikinci << endl;
    birinci = birinci +1;

    cout << "toplama_yap fonksiyonunda değişkenler: " << birinci << " " << ikinci << endl;
    ikinci = ikinci * 2;

    cout << "toplama_yap fonksiyonunda değişkenler: " << birinci << " " << ikinci << endl;
    cout << "Ana fonksiyona geri dönülüyor." << endl << endl;
}

void iki_ile_carp(int bir, int iki)
{
    cout << "iki_ile_carp fonksiyonunda değişkenler: " << bir << " " << iki << endl;
    bir = bir * 2;

    cout << "iki_ile_carp fonksiyonunda değişkenler: " << bir << " " << iki << endl;
    iki = iki +1;

    cout << "iki_ile_carp fonksiyonunda değişkenler: " << bir << " " << iki << endl;
    cout << "Ana fonksiyona geri dönülüyor." << endl << endl;
}

```

```
}

void ikinci_kuvvet(int& referans, int deger)
{
    cout << "ikinci_kuvvet fonksiyonunda değişkenler: " << referans << " " << deger << endl;
    referans = referans * referans;

    cout << "ikinci_kuvvet fonksiyonunda değişkenler: " << referans << " " << deger << endl;
    deger = deger +1;

    cout << "ikinci_kuvvet fonksiyonunda değişkenler: " << referans << " " << deger << endl;
    cout << "Ana fonksiyona geri dönülüyor." << endl << endl;
}
```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./ne_oluyor
```

Ana fonksiyondaki değişkenin değeri:2

toplama\_yap fonksiyonunda değişkenler: 2 2

toplama\_yap fonksiyonunda değişkenler: 3 3

toplama\_yap fonksiyonunda değişkenler: 6 6

Ana fonksiyona geri dönülüyor.

Ana fonksiyona geri dönüldü, değişkenin değeri: 6

iki\_ile\_carp fonksiyonunda değişkenler: 6 6

iki\_ile\_carp fonksiyonunda değişkenler: 12 6

iki\_ile\_carp fonksiyonunda değişkenler: 12 7

Ana fonksiyona geri dönülüyor.

Ana fonksiyona geri dönüldü, değişkenin değeri: 6

ikinci\_kuvvet fonksiyonunda değişkenler: 6 6

ikinci\_kuvvet fonksiyonunda değişkenler: 36 6

ikinci\_kuvvet fonksiyonunda değişkenler: 36 7

Ana fonksiyona geri dönülüyor.

Ana fonksiyona geri dönüldü, değişkenin değeri: 36

Programın kaynak kodu incelendiğinde ana fonksiyon içerisinde tek bir değişkenin tanımlanıldığı görüşmektedir. Bu değişkenin değeri 2 olarak tanımlanmıştır. İlk fonksiyon olan toplama\_yap iki tane formal parametreye sahiptir. Her iki formal parametre aynı değişkeni değer olarak alırken, referans parametresi oldukları için de yapılan her işlemin sonucu ana fonksiyondaki değişkenin değeri değiştirmektedir.

Fonksiyon	Değişkenler	Fonksiyon	Değişkenler	Açıklama
<b>main</b>	say = 2			Değişkenlere, değerleri atanmıştır.

İlk fonksiyon çağrıldıktan sonra program akışı buradan devam etmektedir. Fonksiyonun her iki parametresi de aynı değişken üzerinde işlem yapmaktadır. Aşağıda ilk fonksiyonda gerçekleştirilen işlemler sonucunda değişkenlerin değeri değişimini göstermektedir. Fonksiyonlar ana fonksiyonda tanımlanmış olan değişkenin bellek adresine refere ettiğinden işlem sonucu doğrudan ilgili bellek adresindeki değeri değiştirmektedir.

Fonksiyon	Değişkenler	Fonksiyon	Değişkenler	Açıklama
<b>main</b>	say = 2	<b>toplama_yap</b>	<b>birinci=2, ikinci =2</b>	Atama yapılır
	say = 3		<b>birinci=3, ikinci =3</b>	birinci=birinci+1
	say = 6		<b>birinci=6, ikinci =6</b>	ikinci=ikinci*2

İkinci fonksiyonda ilki gibi iki adet formal parametreye sahiptir. Bu formal parametrelerin değerleri ilk fonksiyondakinden farklı olarak değer parametresidir. Herhangi bir bellek adresini refere etmedikler için fonksiyon çalışırken yapılan işlemler sadece fonksiyon içerisinde değişikliklere neden olmaktadır. Ana fonksiyonun tanımadığı değişken değerini korumaktadır.

Fonksiyon	Değişkenler	Fonksiyon	Değişkenler	Açıklama
<b>main</b>	say = 6	<b>iki_ile_carp</b>	bir=6,iki=6	Atama yapılır
	say = 6		bir=12,ikinci =6	bir=bir*2
	say = 6		bir=12,iki =7	iki=iki+1

Üçüncü fonksiyonda ilki gibi iki adet formal parametreye sahiptir. Bu formal parametrelerin birincisi referans parametresidir. Bu parametre üzerinde yapılan her işlem refere ettiği bellek adresindeki değişken üzerinde gerçekleşir. İkinci formal parametre ise değer parametresidir. Bu parametrenin barındırdığı değer fonksiyon içerisinde işlenir ve değeri fonksiyon dışından erişilebilir olmadığı gibi herhangi bir değişikliğe de neden olmaz.

Fonksiyon	Değişkenler	Fonksiyon	Değişkenler	Açıklama
main	say = 6	ikinci_kuvver	referans=6, deger=6	Atama yapılır
	say = 36		referans=36, deger=6	referans=referans*referans
	say = 36		referans=36, deger=7	deger=deger+1

## 5.8.Referans Parametreleri ve Değer Döndüren Fonksiyonlar

Değer döndüren fonksiyonlar ana fonksiyona sadece bir tek değer döndürmektedir. Eğer bir değer döndüren fonksiyon ile birden çok sayıda değerin asıl fonksiyona döndürülmesi istenirse bu durumda değer döndüren fonksiyon içerisinde referans parametreleri kullanılması ilk akla gelen çözüm olacaktır. Bu yaklaşım ile problemin çözülmesi uygun olmayacağından. Çünkü değer döndüren fonksiyon adından anlaşılacağı üzere sadece bir tek değeri ana fonksiyona döndürür. Eğer birden çok sayıdaki değerin döndürülmesinin gerektiği durumlarda değer döndürmeyen fonksiyonlar ile referans parametreleri kullanılarak değerler ana fonksiyona döndürülmelidir.

## 5.9.Değişkenlerin ve Tanımlayıcılarının Kapsamı

Bir değişken, bir program içerisinde farklı şekillerde tanımlanabilir. Bu bir fonksiyonun başlangıcında veya bir blok içerisinde tanımlanabilir. Bundan önceki bölümlerde bir çok değişken tanımlamasını görmüştük. Bunlara dayanarak şu soru akla gelebilir. **“Bir değişken tanımlamasına bağlı olarak program içerisinde farklı zamanlarda veya farklı kod blokların içerisinde ne şekilde erişilebilir?”**

Bu sorunun yanıtı ise değişkenin nasıl tanımlandığında bulunmaktadır. Bir değişken **yerel değişken** veya **genel değişken** olarak tanımlanabilir.

- **Yerel değişken:** Bir fonksiyon veya kod bloku içerisinde tanımlanmış olan değişkendir. Sadece tanımlandığı kod bloku veya fonksiyon içerisinde erişilebilir ve işlenebilir.
- **Genel değişken:** Bir programın başlangıcında tanımlanmış olan ve fonksiyon veya kod bloku içerisinde tanımlanmamış olan değişkendir. Program içerisinde herhangi bir kod blokundan veya fonksiyondan erişilebilir.

Ayrıca C++ Programlama Dili fonksiyonlar yazılmasında bir fonksiyonun içerisinde bir başka fonksiyonun yazılmasına izin vermemektedir. Diğer bir deyişle, bir fonksiyon içerisinde bir diğer fonksiyon yazılamayacağı anlamına gelir. Aşağıdaki kurallar bir C++ Programı içerisinde yerel ve genel değişkenlerin asıl yazılıp erişilebileceğini belirtmektedir:

1. Genel değişkenler, bir kod bloku veya fonksiyon tarafından ancak şu koşullar sağlanırsa erişilebilir durumdadır;
  - (a) Değişken fonksiyondan önce tanımlanmış ise,
  - (b) Fonksiyonun adı, değişkenin adından farklı ise,

- (c) Fonksiyonun tüm parametrelerine verilen isimler değişkenden farklı ise,  
(d) Tüm yerel değişkenlerin isimleri, değişkenin isminden farklı ise
2. İç içe geçmiş kod bloklarında tanımlanmış olan bir değişken ancak şu koşullar sağlanırsa erişilebilir durumdadır;
- (a) Sadece ve sadece tanımlandığı blok içerisinde erişilebilir,  
(b) Tanimlandığı kod bloku içerisinde başka alt kod blokları tanımlanmış ise, bu durumda blok içerisinde yer alan alt kod blokları tarafından erişilebilir,
3. Değişkenin adı, aynı şekilde değişkenlerin isimleri için tanımlanmış olan kurallara uymak durumundadır. Bir kod bloku dışında tanımlanmış olan bir değişken isimi nasıl erişilebilir durumda oluyorsa, benzer olarak da fonksiyon adı da aynı şekilde erişilebilir durumdadır.

Aşağıda verilen sembolik program kodunda yukarıda belirtilen kuralların nasıl uygulandı görülmektedir.

```
#include <iostream>
#include <locale>
using namespace std;
const double DEGER = 17.13;
int z;
double t;
void fonksiyon1(int x, char y);
void fonksiyon2(int a, int b, char x);
void fonksiyon3(int bir, double iki, int z);
int main()
{
    int say, birinci,
        double x, y, z;
    char isim, son;
    ....
    return0;
}
void fonksiyon1 (int x, char y)
{  
    ....
```

```

}

int w;

void fonksiyon2(int a, int b, char x)
{
    int sayac;

    .....

}

void fonksiyon3(int bir, double y, int z)
{
    char karakter;

    int a;

    // İç içe kod bloku başladı (4. kod bloku)
    {
        int x;
        char a;

        .....

    } // iç içe kod bloku sona erdi.

    .....

}

```

Bu örnek kodun bir programda kullanılmış olduğunu varsayılmı. Bu durumda fonksiyonların ve yerel ve genel değişkenlerin yukarıda açıklanan kurallara göre erişilebilirlik durumları aşağıda verilmiştir. 4. Kod bloku adlı kısımda “a” adlı bir değişken tanımlandığı için fonksiyon3 içerisinde aynı isime sahip olan bir değişken tanımlanması söz konusu değildir.

Burada genel değişkenler konusunda bazı noktaları vurgulamakta yarar bulunmaktadır:

1. Önceki bölümlerde C++ Programlama Dili'nin standardına göre bir değişken tanımlandığında bu değişkenin değeri de belirtilmemiş ise derleyici tarafından etkinleştirme işlemi otomatik olarak yapılmaz. Bu kural her derleyici için genel değişkenler söz konusu olduğunda geçerli değildir. Bazı derleyiciler genel değişkenin değeri tanımlanmamış olsa bile, otomatik olarak etkinleştirip var sayılan değerini atamaktadır. Örneğin int, double veya char veri tipinde tanımlanmış bir genel değişkenin ön tanımlı değeri sıfır olarak atanmaktadır.

- Eğer bir genel değişken programın başında tanımlanmış iken, program içerisinde sonradan tanımlanmış olan bir fonksiyonda yine aynı isime sahip olan bir değişken bulunuyor ise "**kapsam çözümleyici operatörü – scope resolution operator**" yani "::" kullanılarak aynı isimli genel değişkene erişim sağlanabilir. Bir sonraki program bunu göstermektedir.
- C++ Programlama Dili ayrıca bir genel değişken bir fonksiyondan sonra tanımlanması durumunda erişilebilir olmasını sağlamak için mekanizmaları da sunmaktadır. (C++ 11 standardında bu özellik eklenmiştir.) yukarıdaki örnekte fonksiyon1 tanımlanıp yazıldıktan sonra "w" adlı genel değişken tanımlanmıştır. Bu değişkene fonksiyon1 içeriisinden erişmek için aynı isime sahip olan bir değişken tanımlanması gereklidir. Ayrıca bu genel değişkenin erişilebilir olması için de "extern int w;" ifadesinin fonksiyon1 içerisinde yazılması gereklidir. Bu ifade "w" adlı değişkenin bir genel değişken olduğunu belirtmektedir.

Tanımayıcı/ Değişken	fonksiyon1	fonksiyon2	fonksiyon3	4.kod bloku	main
DEGER (main öncesi)	Evet	Evet	Evet	Evet	Evet
z (main öncesi)	Evet	Evet	Hayır	Hayır	Hayır
t (main öncesi)	Evet	Evet	Evet	Evet	Evet
main	Evet	Evet	Evet	Evet	Evet
main'deki yerel değişkenler	Hayır	Hayır	Hayır	Hayır	Evet
fonksiyon1 adı	Evet	Evet	Evet	Hayır	Evet
x (fonksiyon1 formal parametre)	Evet	Hayır	Hayır	Hayır	Hayır
y (fonksiyon1 formal parametre)	Evet	Hayır	Hayır	Hayır	Hayır
w (fonksiyon2 'den önce)	Hayır	Evet	Evet	Evet	Hayır
fonksiyon2 adı	Evet	Evet	Evet	Evet	Evet
a (fonksiyon2 formal parametre)	Hayır	Evet	Hayır	Hayır	Hayır
b (fonksiyon2 formal parametre)	Hayır	Evet	Hayır	Hayır	Hayır
x (fonksiyon2 formal parametre)	Hayır	Evet	Hayır	Hayır	Hayır
fonksiyon2'nin yerel değişkeni	Hayır	Evet	Hayır	Hayır	Hayır
fonksiyon3 adı	Evet	Evet	Evet	Evet	Evet
bir (fonksiyon3 formal parametre)	Hayır	Hayır	Evet	Evet	Hayır
y (fonksiyon3 formal parametre)	Hayır	Hayır	Evet	Evet	Hayır
z (fonksiyon3 formal parametre)	Hayır	Hayır	Evet	Evet	Hayır
karakter, fonksiyon3'ün yerel değişkeni	Hayır	Hayır	Evet	Hayır	Hayır
a, fonksiyon3 yerel değişkeni	Hayır	Hayır	Evet	Hayır	Hayır
X, 4.kod bloku yerel değişken	Hayır	Hayır	Hayır	Evet	Hayır
a, 4.kod bloku yerel değişken	Hayır	Hayır	Hayır	Evet	Hayır

## 5.10.Genel Değişkenler, Sabitler ve İstenmeyen Yan Etkiler

C++ Programlama dili ile yazılan bir programda değişkenlerin ve tanımlayıcılarının program içerisindeki fonksiyonların hangileri tarafından erişilebilir olduğunu bilmek durumunda kalmadan tüm değişkenlerin genel değişken olarak tanımlanması olası sorunların en başından ortadan kaldırılması anlamına gelebilir. Bu fikir, çekici görünse de istenmeyen yan etkilere neden olacağı için uygulamada kullanılmaması gereklidir. Çünkü tüm değişkenlerin genel değişken olarak tanımlanıldığı bir programda hangi fonksiyonun hangi değişkeni ne zaman değiştirdiğinin izini sürmek kolay değildir. Bu programda bir yazım ve kural hatasına değil bulunması ve düzeltilmesi çok zor olan mantık hatalarının kapasını aralamaktan başka bir getirişi olmayacağından emin olmayıacaktır.

Aşağıda tüm değişkenlerin genel değişken olarak tanımlandığı bir program görülmektedir.

```
/*
 * geneldegiskenproblemdir.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtım; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
```

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

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

int p;           // Genel değişken problemdir!

void fonksiyon (int& a);

int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    p = 15;
    cout << "Ana fonksiyondaki genel değişkenin değeri: " << p << endl;
    fonksiyon (p);
    cout << "Fonksiyon çalıştırıldıktan sonraki genel değişken:" << p << endl;
    return 0;
}

void fonksiyon ( int& a )
{
    cout << "a= " << a << " " << "p= " << p << endl;
    a = a + 12;
    cout << "a= " << a << " " << "p= " << p << endl;
```

```

a = a + 13;

cout << "a= " << a << " " << "p= " << p << endl;

cout << "ana fonksiyona geri dönülüyor" << endl;

}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./geneldegiskenproblem
```

Ana fonksiyondaki genel değişkenin değeri: 15

a= 15 p= 15

a= 27 p= 27

a= 40 p= 40

ana fonksiyona geri dönülüyor

Fonksiyon çalıştıktan sonraki genel değişken:40

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Program içerisinde tüm fonksiyonlardan önce tanımlanmış olan bir genel değişken bulunmaktadır. Bu genel değişken programda tanımlanmış olan tüm fonksiyonlar tarafından erişilebilir durumdadır. Önceki örneklerde de incelendiği üzere genel bir değişken eğer bir fonksiyonun formal parametrelerinde bulunan referans parametresi ile erişilebilir olduğunda bu genel değişkenin değeri üzerinde gerçekleştirilen her işlem sonunda değişecektir. Yukarıdaki örnek programda görüldüğü gibi program sona erdiğinde genel değişkenin değeri her fonksiyon çağrılığında değişecektir.

Ayrıca programlar içerisinde kullanılan sabitler ster tek bir fonksiyonda ister birden çok fonksiyon içerisinde kullanıldın, programın başlangıç bölümünde genel değişkenlerden önce tanımlanmalıdır. Sabitler, de önceki bölümlerde açıklandığı üzere bir değişken olmakla birlikte, değerleri programın çalışması süresice değiştirilemez. Bu nedenle genel değişkenlerin aksine beklenmedik yan etkiler ile karşılaşılmaz. Programda sabitler üzerinde yapılacak olan değişikliklerin bu yazım şekli ise hem bulunması hemde düzenlenmesi daha kolaydır.

Bunun gibi yazılan programlarda hataların bulunması son derece güçtür. Program derleyici tarafından sadece yazım kuralları ile anlamsal olarak kontrol edilir. Programın kötü yazılmış, hatalar davet çeken yapısı ise kalıcıdır. Bu tür program yazma tarzları deneyimli programcılar arasında hoş karşılanmaz. Yazılan kod “spagetti kodu” olarak isimlendirilir ve kısa zamanda internette en iyi tanınan isimlerden birisi olmanız da kaçınılmaz istenmeyen yan etkilerindendir.

**Örnek Program:** Aşağıda verilen program bir ikinci derecen polinomun çarpanlarına ayrılmış olarak yazmaktadır. İkinci dereceden bir polinomun çarpanlarına ayrılması Cebir derslerinin konusudur. Bir polinomun çarpanlarına ayrılması için bir çok yöntem bulunmakla birlikte burada programın yazılmasında kolaylık sağlama için polinom  $x^2+bx+c=(x\pm u)(x\pm v)$  biçiminde olduğu kabul edilecektir. Program çarpanlara ayırma işleminin daha kolay olması için b, c, u ve v paramet-

relerinin tam sayı olduğu kabul edilecektir. Örneğin  $x^2+5x+6=(x+3)(x+2)$  ve benzer olarak da  $x^2+10x-24=(x+12)(x-2)$  polinomları gösterilebilir.

Çarpanlara ayırma işleminde  $u$  ve parametrelerinin bulunabilmesi için ikinci dereceden denklemin köklerinin bulunmasında kullanılan formül kullanılabilir. Dikkat edilmesi gereken nokta ise formül vereceği değerlerin ayrık ve tam sayı olması gerektidir.

$$u = \frac{-b + \sqrt{b^2 - 4c}}{2} \quad v = \frac{-b - \sqrt{b^2 - 4c}}{2}$$

Bu bağıntılardan da görüleceği gibi diskriminantın negatif olması gerçek köklerin olmadığını belirtir. Aynı şekilde diskriminat sıfırdan büyük ve sonuç bir tam sayının karesi değilse  $u$  ve  $v$  gerçel sayılardır. Aynı zamanda diskriminatın kara kökü sıfırdan büyük,  $-b - \sqrt{b^2 - 4c}$  ve  $-b + \sqrt{b^2 - 4c}$  ikiye kalansız bölünemiyorsa ise kökler tam sayı olamaz. Eğer  $-b - \sqrt{b^2 - 4c}$  ve  $-b + \sqrt{b^2 - 4c}$  ikiye kalansız bölünüyorsa kökler tam sayı olabilir.

Bu koşulları kontrol eden bir fonksiyon yazılarak verilen polinomun çarpanlarına ayrılmıştır. Böylece ana fonksiyona gereken değerler doldurulabilir. Aşağıda programın kaynak kodu görülmektedir.

```
/*
 * carpanlara_ayir.cxx
 *
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 *
 * Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
```

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

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

void carpanlara_ayir(int b, int c, int& u1, int& v1, bool& carpanlar_var);

int main()
{
    int katsayı, sabit, u, v;
    bool carpanlar_var;
    cout << "x'in katsayısını girip enter tuşuna basınız: ";
    cin >> katsayı;
    cout << "Polinomun sabit değerini girip enter tuşuna basınız: ";
    cin >> sabit;
    carpanlara_ayir(katsayı, sabit, u, v, carpanlar_var);
```

```

if (carpanlar_var)
{
    cout << "x^2";
    if(katsayi > 0)
        cout << " + " << katsayi << "x";
    else if (katsayi < 0)
        cout << " - " << abs(katsayi) << "x";
    if (sabit > 0)
        cout << " + " << sabit;
    else if (sabit < 0)
        cout << " - " << abs(sabit);
    cout << " = (x";
    if (u >0)
        cout << " - " << u << ")(x";
    else if (u < 0)
        cout << " + " << abs(u) << ")(x";
    if (v>0)
        cout << " - " << v << ")" << endl;
    else if (v<0)
        cout << " + " << abs(v) << ")" << endl;
}
else
    cout << "Polinom çarpanlarına ayrılamaz." << endl;
return 0;
}

void carpanlara_ayir(int b, int c, int& u1, int& v1, bool& carpanlar_var)
{
    double diskriminant;
    int tmp;
    carpanlar_var = true;
    diskriminant = b * b - 4 * c;
}

```

```

if (diskriminant < 0)

    carpanlar_var = false;

else

{

    tmp = static_cast<int>(sqrt(diskriminant));

    if (tmp * tmp != diskriminant)

        carpanlar_var = false;

    else

    {

        if (((-b + tmp) % 2 != 0) || ((-b - tmp) % 2 != 0))

            carpanlar_var = false;

        else

        {

            u1 = (-b + tmp) / 2;

            v1 = (-b - tmp) / 2;

        }

    }

}

}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```

[goksin@tardis ~/projeler/C++_Notlar/]$ ./carpanlara_ayir

x'in katsayısını girip enter tuluna basınız: -1

polinomun sabit değerini girip enter tuluna basınız: -6

x^2 -1x - 6 = (x - 3)(x + 2)

[goksin@tardis ~/projeler/C++_Notlar/]$

```

Aşağıda yer verilen program ise uluslararası uzunluk ölçü birimleri olan metre ve santimetre cinsinden verilen bir uzunluğu Amerikan uzunluk ölçü birimlerine veya tersi şekilde çevirmektedir. Program kullanıcıya bir menu aracılığı ile seçenekler sunmaktadır. Ek olarak programın nasıl kullanılacağını açıklayan bir yardım bölümü de bulunmaktadır.

```

*
* uzunluk_cevir.cxx
*
```

\*

\* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

```

/*
#include <iostream>
#include <locale>
#include <cfloat>
using namespace std;
const double CM_CEVAR = 2.54;
const int INCH_FOOT = 12;
const int CM_M = 100;
void menu();
void inch_feet_cm_m_cevir(int f, int in, int& m, int& cm);
void cm_m_inch_feet_cevir(int m, int cm, int& f, int& in);
int main()
{
    int feet, inch, metre, cmetre, secim;
    do
    {
        menu();
        cin >> secim;
        switch (secim)
        {
            case 1:
                cout << "Uzunluğu feet ve inch cinsinden girip enter tuşuna basınız: ";
                cin >> feet >> inch;
                inch_feet_cm_m_cevir(feet, inch, metre, cmetre);
                cout << feet << " feet " << inch << " inch " << metre << " m " <<
cmetre << " cm eder." << endl;
                break;
            case 2:
                cout << "Uzunluğu m ve cm cinsinden girip enter tuşuna basınız: ";
                cin >> metre >> cmetre;
                cm_m_inch_feet_cevir(metre, cmetre, feet, inch);
        }
    }
}

```

```

        cout << metre << " m " << cmetre << " cm " << feet << " feet " <<
inch << " inch eder." << endl;

        break;

    case 9:

        break;

    default:

        cout << "Geçersiz giriş" << endl;

    }

} while (secim !=9);

return 0;

}

void menu()

{

    cout << "feet ve inch cinsinden verilen uzunluğu metre, cm çevirmek için \"1\" giriniz" <<
endl << endl;

    cout << "metre ve cm cinsinden verilen uzunluğu feet, inch çevirmek için \"2\" giriniz" <<
endl << endl;

    cout << "Programdan çıkmak için \"9\" giriniz" << endl << endl;

    cout << "Yukarıdaki seçeneklerden birisini girip enter tuşuna basınız: ";

    cout << endl;

}

void inch_feet_cm_m_cevir(int f, int in, int& m, int& cm)

{

    int inches;

    inches = f * INCH_FOOT + in;

    cm = static_cast<int>(inches * CEVIR);

    m = cm / CM_M;

    cm = cm % CM_M;

}

void cm_m_inch_feet_cevir(int m, int cm, int& f, int& in)

```

```
{  
    int santimetre;  
  
    santimetre = m * CM_M + santimetre;  
  
    in = static_cast<int>(santimetre/CEVIR);  
  
    f = in / INCH_FOOT;  
  
    in = in % INCH_FOOT;  
}
```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./uzunluk_cevir
```

feet ve inch cinsinden verilen uzunluğu metre, cm çevirmek için "1" giriniz

metre ve cm cinsinden verilen uzunluğu feet, inch çevirmek için "2" giriniz

Programdan çıkmak için "9" giriniz"

Yukarıdaki seçeneklerden birisini girip enter tuşuna basınız: 2

Uzunluğu m ve cm cinsinden girip enter tuşuna basınız: 6 28

6 m 28 cm 20 feet 7 inc eder.

feet ve inch cinsinden verilen uzunluğu metre, cm çevirmek için "1" giriniz

metre ve cm cinsinden verilen uzunluğu feet, inch çevirmek için "2" giriniz

Programdan çıkmak için "9" giriniz"

Yukarıdaki seçeneklerden birisini girip enter tuşuna basınız: 9

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Programın akışı menü ile kontrol edileceği için biw do .. while döngüsü kullanılmıştır. Döngüden çıkış koşulu gerçekleşmediği sürece program çalışmaya devam edecektir.

## 5.11.Statik ve Otomatik Değişkenler

Buraya kadar incelediğimiz örneklerde bir değişkenin şu iki kurala uymakta olduğunu gördük:

1. Genel değişkenlere ayrılan bellek alanı program çalıştığı sürece korunur.
2. Bir kod bloku içerisinde tanımlanan değişken için ayrılacak olan bellek alanı, ilgili kod bloku çalıştırıldığında işletim sistemi tarafından ayrılır. Söz konusu kod bloğunun işlenmesi sona erince de söz konusu bellek alanı geri alınır.

Yukarıda verilen kurallara uyan bir değişken için “**otomatik değişken – automatic variable**” olduğu söylenir. Benzer olarak bir değişken için ayrılmış olan bellek alanının programın işlediği süre boyunca korunması yani kalıcı olması durumu ise “**statik -değişken – static variable**” olarak isimlendirilir. Genel değişkenler statik değişkenlere örnek verilebilir. Bir fonksiyon veya kod bloku içerisinde tanımlanmış olan bir değişken ise bir otomatik değişkendir. Bir kod bloku veya fonksiyon içerisinde tanımlanmış olan bir değişkenin kalıcı olması isteniyorsa bunun için aşağıdaki tanımlamanın yapılması gereklidir.

static veritipi tanımlayıcı;

Yukarıdaki kuralın uygulanması ise aşağıdaki gibidir:

static int x;

Bir kd bloku veya fonksiyon içerisinde statik olarak tanımlanan bir değişken sadece söz konusu kod bloku veya değişken içerisinde erişilebilir, yani bu değişken bir yerel değişken olma özelliğini korur.

Bir çok derleyici, statik olarak tanımlanan değişkenler için eğer bir değer ataması yapılmamış ise bu durumda değişkenin ön tanımlı değerini atar. Bu değer int, float ve char veri tipleri için sıfırdır. Eğer vir statik değişkenin ilk tanımlandığında değerinin olması yani etkinleştirilmesi isteniyor ise bir değer atanmalıdır.

static int x = 0;

Aşağıdaki program sözü edilen statik ve otomatik değişkenlerin uygulanmasını göstermektedir.

```
*  
* statikotomatikdegiskencxx  
*  
*  
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>  
* *  
* Her hakkı saklıdır.  
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak  
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği  
* takdirde serbesttir:  
*
```

\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
```

```
#include <locale>
```

```
using namespace std;
```

```
void test();
```

```
int main()
```

```
{
```

```
    setlocale(LC_ALL, "tr_TR.UTF-8");
```

```
int sayac;
for (sayac = 1; sayac <= 5; sayac++)
    test();
return 0;
}
void test()
{
    static int x = 0;
    int y= 10;
    x = x + 2;
    y = y + 1;
    cout << "Test fonksiyonu çalışıyor..." << endl << endl;
    cout << "x = " << x << " " << "y = " << y << endl;
}
```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./statikotomatikdegisen
```

```
Test fonksiyonu çalışıyor...
```

```
x = 2 y = 11
```

```
Test fonksiyonu çalışıyor...
```

```
x = 4 y = 11
```

```
Test fonksiyonu çalışıyor...
```

```
x = 6 y = 11
```

```
Test fonksiyonu çalışıyor...
```

```
x = 8 y = 11
```

```
Test fonksiyonu çalışıyor...
```

```
x = 10 y = 11
```

Programın çalıştırılmasından da görüleceği üzere test fonksiyonu içerisinde statik olarak tanımlanmış olan x değişkenin etkinleştirildiğinde değeri sıfırdır. Fonksiyon içerisinde statik olarak tanımlanmış olan değişken fonksiyonun işlenip sona ermesi ile bellekten silinmemektedir. Aksine değişken bellekte kalmaktadır. Program sona erene kadar de bellekte varlığını koruyacaktır. Bu nedenle de her fonksiyon çağrılığında en son işlemden kalan değer değişkende yer alacağı için ifadeler işlendikçe x değişkenin değeri değişime uğrayacaktır. Aynı kod blokunda yer alan y değe ise bir otomatik değişkendir. Bu nedenle de fonksiyon çağrılığında bu değişken için bellekte yer ayrılmak, fonksiyonun işlenmesi sırasında değişkenin değeri değişecek ve fonksiyon sona erdiğinde de bellekten silinecektir. Programın içerisinde yer verilen cout ile yapılan yazdırma işlemlerinde bu durum her fonksiyon çağrıısı yapıldığında statik ve otomatik değişkenlerin barındırdıkları değerleri yazdırmaktadır.

Statik değişkenin işleyisi, genel değişkenlere benzer gibi görünse de değildir. Genel değişkenler bir program akışı içerisinde tüm fonksiyonlar ve kod blokları tarafından erişilebilir ve üzerinde işlem yapılabilirken, statik olarak tanımlanmış olan değişkenler ise sadece tanımlandığı kod bloku veya fonksiyon içerisinde erişilebilir durumdadır. Bu nedenle statik değişken tanımlamaları genel değişkenler gibi olmayıp kullanılması konusunda mantık hatalarına neden olabileceği dikkate alınmalıdır.

## 5.12.Fonksiyonların Ön Tanımlı Parametreleri

Fonksiyonlar konusundaki örneklerde bir fonksiyonun işleyeceği parametre sayısı ile o fonksiyonun işlemesi istenilen parametrelerin sayısının birbirine eşit olduğunu görmüştük. Öte yandan C++ Programlama Dili bu konuda esnek davranışlıdır. Bir fonksiyonun formal parametre sayısından az parametre ile işlem yapabilmektedir. Bir fonksiyonun bu şekilde işleyebilmesi için de fonksiyonun parametrelerine ön tanımlı değerlerin atanması gereklidir. Bu işlem ise bir fonksiyonun program içerisinde tanımlandığı ilk satırda yapılır. Burada bir prototip fonksiyon tanımlaması yapılmakta olduğundan her bir parametreye ait ön tanımlı değerler parantez içerisinde yazılabilir.

Bir fonksiyonda ön tanımlı değerlerin tanımlanmasında şu kurallara dikkat edilmesi gereklidir:

- Fonksiyondaki parametrenin ön tanımlı değeri tanımlanmaz ise, o parametre için ön tanımlı değer kullanılır.
- Fonksiyonda formal parametreler tanımlandıktan sonra bu parametrelere ait olan ön tanımlı değerler parantez içerisinde ve en sağda yazılmalıdır.
- Eğer bir fonksiyonun birden çok sayıda parametresi varsa ve bu parametrelere her birisi için bir ön tanımlı değer bulunabilir. Eğer fonksiyon çağrılığında, fonksiyona ait parametrelere en az birisi kullanılmıyor ise, bu parametreden sonra gelen diğer parametrelere de değer aktarımı yapılamaz.
- Ön tanımlı değerler sabitler, genel değişkenler veya fonksiyon çağrıları olabilir.
- Fonksiyonu çağrıran kod bloku veya diğer bir fonksiyon, ön tanımlı değer dışında bir değeri fonksiyona aktarabilir.

- Bir fonksiyonda bir sabiti, referans parametresi olarak atayamazsınız.

Aşağıdaki prototip fonksiyonda parametrelerin ön tanımlı değerleri tanımlanmıştır.

```
void fonksiyon_ORNK(int t, int u, double v, char w = 'A', int x = 42; char y = 'G', double z = 88.23);
```

Yukarıda verilen prototip fonksiyonun yedi adet parametresi vardır. Bu parametre tanımlarından görüleceği üzere, w, x, y ve z parametrelerinin ön tanımlı değerleri bulunmaktadır. Eğer bu fonksiyon çağrılığında ön tanımlı değerleri olan parametrelere herhangi bir değer aktarılmaz ise ön tanımlı değerleri kullanılır.

Örneğin bir programda aşağıdaki bildirimler yapılmış olsun.

```
int a, b;  
char karakter;  
double d;
```

Bu program içerisinde aşağıda gösterilen fonksiyon çağrıları yapılacak olursa, fonksiyonlar doğru çalışacaktır.

```
void fonksiyon_ORNK(a, b, d);  
void fonksiyon_ORNK(a, 15, 34.6, 'B', 87, karakter);  
void fonksiyon_ORNK(b, a, 14.56, 'D');
```

Yukarıda verilen ilk fonksiyon çağrısında w,x ve y parametrelerinin ön tanımlı olan değerleri kullanılmıştır. İkinci fonksiyon çağrısında w için verilen ön tanımlı değer 'B' ile; x için verilen ön tanımlı değer ise 86; y için verilen ön tanımlı değer karakter ile değiştirilmiştir. Son fonksiyon çağrısında w için belirlenmiş olan ön tanımlı değer 'D' ile değiştirilmiş, x,y ve z için verilen ön tanımlı değerler kullanılmıştır.

Aşağıda verilen fonksiyon çağrıları ise yanlışır.

```
void fonksiyon_ORNK(a, 15, 34.6, 46.7);  
void fonksiyon_ORNK(b, 25, 48.76, 'D', 4567, 78.34);
```

İlk fonksiyon çağrısında fonksiyon parametrelerinden olan "w" çıkarılmıştır. Bu durumda bu parametreyi izleyen tüm diğer parametrelere de değer aktarımı yapılamamalıdır.

Aşağıdaki fonksiyon çağrılarında ise prototip fonksiyonlar ön tanımlı değerler kullanılarak tanımlanmıştır. Bu şekilde yapılan fonksiyon tanımlamaları yanlışır.

```
void fonksiyon_1(int x, double z = 23.45, char karakter, int u = 45);  
void fonksiyon_2(int uzunluk = 1, int genislik, int yükseklik = 1);  
void_fonksiyon_3(int x, int& y = 16, double z = 34);
```

Birinci fonksiyon tanımlandığında ikinci parametre olan z bir ön tanımlı değere sahiptir. Bunu izleyen tüm parametrelerin ön tanımlı değerlere sahip olmalıdır. İkinci fonksiyon tanımlandığında ilk parametre ön tanımlı değere sahip ise, bunu izleyen tüm parametrelerin ön tanımlı değer-

leri olması gereklidir. Üçüncü fonksiyon tanımlamasında ise ikinci parametre olan y bir referans parametresidir. Bu durumda ön tanımlı değeri olamaz.

Aşağıdaki örnek program ön tanımlı değerleri olan fonksiyonların kullanımını göstermektedir.

```
/*
 * fonksiyonontanımlıdeger.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 *
 * Kaynak kodun yeniden dağıtım; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
```

```

* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
* SORUMLU DEĞİLLERDİR.

/*
*/
/*
#include <iostream>
#include <locale>
#include <iomanip>
using namespace std;

int hacim(int U = 1, int Y = 1, int G = 1);
void fonksiyon_bir (int&x, double y = 12.34, char z ='G');

int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    int a = 23;
    double b = 48.78;
    char k = 'Z';
    cout << fixed << showpoint << setprecision(2);
    cout << "a= " << a << " " << "b=" << b << " " << "k= " << k << endl;
    cout << "Hacim= " << hacim () << endl;
    cout << "Hacim= " << hacim (5,4) << endl;
    cout << "Hacim= " << hacim (4,5,6) << endl;
    fonksiyon_bir(a);
    fonksiyon_bir(a,42.68);
    fonksiyon_bir(a,24.65,'Q');
    cout << "a= " << a << " " << "b= " << b << " " << "k= " << k << endl;
    return 0;
}

```

```

int hacim(int U, int Y, int G)
{
    return U * G * Y;
}

void fonksiyon_bir (int& x, double y, char z)
{
    x = x * 2;
    cout << "x=" << x << " " << "y=" << y << " " << "z=" << z << endl;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./fonksiyonontanimlideger
```

```
a= 23 b=48.78 k= Z
```

```
Hacim= 1
```

```
Hacim= 20
```

```
Hacim= 120
```

```
x=46 y=12.34 z=G
```

```
x=92 y=42.68 z=G
```

```
x=184 y=24.65 z=Q
```

```
a= 184 b= 48.78 k=Z
```

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Yukarıdaki programda iki adet prototip fonksiyon tanımlanmıştır. Her bir fonksiyonda yukarıda yer verilen ön tanımlı değerlere ait kurallara uyularak yazılmıştır. İlk fonksiyon adında da anlaşılacağı üzere bir dörtgen prizmanın hacmini hesaplamaktadır. İkinci fonksiyon ise yine aynı şekilde ön tanımlı değerlere sahiptir. İkinci fonksiyonda referans parametreleri bulunmaktadır. Bu nedenle fonksiyonun ilk parametresi, referans parametresi olarak yazılmıştır. Böylece kendisinden sonra gelen parametrelere ön tanımlı değerler atanabilir. İkinci fonksiyon çağrıldığında aktarılan değerler parametrelere aktarılmış ve fonksiyon tarafından işlenmiştir.

Ana fonksiyonda hacim() fonksiyonuna değer aktarımı yapılmıştır. İlk çağrıda, ön tanımlı değerleri kullanılmıştır. İkinci ve son çağrıda ana fonksiyonda tanımlanmış olan a değişkenin değeri fonksiyona aktarılmış, ayrıca fonksiyona gereken diğer değerler de doğrudan aktarılıarak sonuç döndürülmüştür.

Diğer fonksiyonda ilk parametre referans parametresi olduğundan her fonksiyon çağrıda ilk parametre üzerinde yapılan işlemler sonraki fonksiyon çağrılarında yeniden

kullanılmaktadır. Diğer parametrelere aktarılan değerler, parametrenin değer parametresi olması nedeni ile aktarılan değer üzerinde tanımlanan işlemleri yaparak fonksiyon sonlanmaktadır.

## 5.13.Fonksiyonlarda Aşırı Yükleme – Overloading

Bir program içerisinde bazen bir fonksiyonu tanımlayan isim diğer fonksiyonlarda da kullanılabilir. Bu şekilde yapılan tanımlamalara “**fonksiyonların aşırı yüklenmesi – function overloading**” adı verilir. Bir fonksiyonun aşırı yüklenmesi ile ilgili kuralları belirtmeden önce şu tanımı yapmak yerinde olacaktır.

İki fonksiyonun **formal parametrelerinin farklı olması** için şu iki koşulun gerçekleşmesi gereklidir:

1. Fonksiyonların formal parametrelerinin sayıları birbirinden farklı olmalıdır.
2. Fonksiyonların formal parametrelerinin en az bir tanesinin veri tipi ve tanımlandığı yer diğer fonksiyondakilerden farklı olmalıdır.

Bu tanımlamaya dayanarak aşağıda gösterilen fonksiyonların parametrelerinin farklı olduğu söylenebilir.

```
void fonksiyon_f (int x);
void fonksiyon_g (int x, float y)
void fonksiyon_h (float y, int x);
void fonksiyon_j (char K, int x, float y)
void fonksiyon_k (char K, int x, string isim)
```

Bu fonksiyonların her birisinin formal parametreleri farklıdır. Buna ek olarak aşağıda verilen iki fonksiyonu incelediğimizde yukarıdaki tanıma uymadığını görüyoruz.

```
void fonksiyon_m (int x, double y, char Z);
void fonksiyon_n (int A, double B, char W);
```

Her iki fonksiyonun formal parametre sayısı eşittir. Ayrıca formal parametrelerin veri tipleri de aynıdır. Bu değerlendirmeye göre her iki fonksiyonun formal parametreleri aynıdır denebilir. Burada dikkat edilmesi gereken fonksiyondaki veri tipleri veya döndürülen değer incelenmemektedir.

İki veya daha çok sayıdaki aynı isimle yazıldığından ve böylece fonksiyon aşırı yüklenmesi gerçekleştiğinden söz edebilmemiz için fonksiyonların formal parametrelerinin farklı olması gereklidir. Bilgisayar dillerinde aşırı yükleme konusunda üzerinde hem fikir olunan konu, fonksiyon aşırı yüklemesi yapılabilmesi için **fonksiyonların imzası** esas alınır.

Bir fonksiyonun imzası, **fonksiyonun adı ile fonksiyonun formal parametreleri** bir araya gelmesinden oluşur. Aşağıda verilen örnek fonksiyonların isimleri aynı olmasına karşın fonksiyon aşırı yüklemesi olarak nitelendirilemez.

```
void fonksiyon_m (int x, double y, char Z);
```

```
int fonksiyon_n (int x, double y, char Z);
```

Yukarıdaki iki fonksiyonda yanı program içerisinde yer allığında derleyici, bu iki fonksiyon için yazım hatası yapıldığına dair hata mesajı döndürecektr.

O halde fonksiyon aşırı yüklemesi söz konusu olduğunda derleyici kaynak kodu makine diline çevirdiğinde hangi fonksiyonun işleneceği nasıl belirlenecektir?

Bu sorunun yanıtı basitçe aşırı yüklenen fonksiyonların bulunduğu bir programda hangi fonksiyonun işleneceğini belirleyen fonksiyonun formal parametreleridir.

Bir programda verilen iki veriyi karşılaştırıp hangisinin diğerine kıyasla “büyük” olduğunu bulan bir fonksiyon yazmanız gereken bir durumu düşünelim. Bu durumda programın ne tür veriyi karşılaştıracağını bilemeyeceğimiz için farklı veri tiplerine yönelik fonksiyonların yazılması gerekecektir. O halde fonksiyonlar sırası ile karakter, tam sayı, kayar noktalı sayı veya karakter dizisi üzerinde işlem yapacaktır. Böylece aşağıdakine benzer şekilde dört farklı fonksiyon yazılması gerekecektir.

```
chat charBuyuktur (char a, charb);  
int intBuyuktur (int a, intb);  
double doubleBuyuktur (double a, double b);  
string stringBuyuktur (string a, string b);
```

Yukarıdaki yazım şeklinin yanlış olan bir yanı yoktur. Sadece işin basitleştirilmesi adına fonksiyon aşırı yüklemesi kullanılabilir. Böylece yukarıdaki fonksiyonlar daha basit olarak aşağıdakiler gibi yazılabılır.

```
chat buyuktur (char a, charb);  
int buyuktur (int a, intb);  
double buyuktur (double a, double b);  
string buyuktur (string a, string b);
```

Böylece fonksiyon sayısı aynı kalırken, yapılacak olan fonksiyon çağrıları bire indirgenebilir. Kullanıcıdan alınan veri doğrudan aşağıda görüldüğü gibi kullanılabilir.

```
buyuktur (12, 33);
```

Bu fonksiyon çağrıları formal parametreleri esas alarak, doğrudan `int buyuktur(int a, int b)` fonksiyonunu çağıracaktır. Aynı şekilde aşağıdaki fonksiyon çağrıları da uygun fonksiyona parametre aktarımı yapılarak işlemin gerçekleşmesini sağlar. Aşağıdaki örnek `string buyuktur( string a, sitring b)` fonksiyonunu çağrıır.

```
buyuktur (“ali”, “veli”);
```

Fonksiyonların aşırı yüklenmesi, farklı veri kümeleri üzerinde aynı işlemlerin yapılması gereken durumlarda işe yaramaktadır. Bunun dışındakiler için uygun bir çözüm olmayıp yazılım yaşam döngüsünde hatalı işlemlerin yapılmasına neden olabilir. Bir programda fonksiyon aşırı yüklemesi yapılacak ise bunun için her bir fonksiyonun ayrıca tanımlanması gerekecektir.

**Örnek Program:** Kullanıcı tarafından girilen 20 adet tam sayı içerisindeki tek, çift ve sıfır olan sayıların kaç adet olduğunu bulan bir program yazınız.

**Çözüm:** Problemin çözümü için öncelikle problemin alt parçalara ayrılması yoluna gidilecektir. Böylece her bir problem kendi özeline ele alınıp daha sonra asıl problemin çözümünü elde etmek amacıyla ile diğer çözümler ile birleştirilerek kullanılacaktır.

Problem şu alt problemlere ayrılabilir:

- 1 Klavyeden girilecek olan sayıların okunması
- 2 Klavyeden girilen sayıların hangisinin tek olduğunu belirlenmesi ve tek sayı adedinin kayıt edilmesi
- 3 Klavyeden girilen sayıların hangisinin çift olduğunu belirlenmesi ve çift sayı adedinin belirlenmesi
- 4 Klavyeden girilen sayıların hangisinin sıfır olduğunu belirlenmesi ve sıfır adedinin kayıt edilmesi
- 5 Kullanıcıya sonuçların bildirilmesi

Bu alt problemlerin her biri fonksiyon olarak çözülebilir. Böylece ana fonksiyonun karmaşıklığı azaltılırken belirli işlevlerin etkin ve verimli bir şekilde gerçekleştirilmesi için özel kod blokları içerisinde çalışmak olanaklı olacaktır.

1. Klavyeden girilen sayıların okunması: Klavyeden girilecek olan sayı adedi yirmidir. Bu durumda bir döngü kullanılarak girilen her sayı okunabilir. Sayı girişinin sonlanması için aralarında boşluk bırakılarak giriş yapılması istenebilir. Böylece enter tuşuna basıldığında sayı girişi işlemi sona ermiş olacaktır. Bu işlem için tekrar sayısı bilindiğinden bir for() döngüsünün kullanılması daha uygundur.
2. Klavyeden girilen sayının tek sayı olup olmadığını belirlenmesi için girilen sayının 2'e bölümünden kalanın 1 veya -1 olması yeterlidir. Bu koşulun gerçekleşmesi durumunda tek sayı adedini kayıt etmek için tanımlanacak bir değişken kullanılabilir ve bu değişkenin değeri koşulun gerçekleşmesi durumunda arttırılır. Değişkenin başlangıç değeri sıfır alınmalıdır.
3. Klavyeden girilen sayının çift sayı olup olmadığını belirlenmesi için girilen sayının 2'e bölümünden kalanın 0 olmasıdır. Bu koşulun gerçekleşmesi durumunda çift sayı adedini kayıt etmek için tanımlanacak bir değişken kullanılabilir ve bu değişkenin değeri koşulun gerçekleşmesi durumunda artırılır. Değişkenin başlangıç değeri sıfır alınmalıdır.
4. Klavyeden girilen sayının sıfır olup olmadığını belirlenmesi için tek/çift değerlendirmesi dışında ayrıca sayının sıfır eşit olması durumu kontrol edilmelidir. Sıfır sayısını belirlemek için ilk değeri sıfır olan bir değişken tanımlanabilir ve her sıfır okunduğunda değeri bir artırlır.
5. Elde edilen sonuçlar, her bir koşulun gerçekleşme durumunda arttırılan değişkenler yardımı ile ekranaya yazdırılabilir.

Programın yazılması için yukarıda belirtilen alt problemler kendi özelinde çözülebilir. Sayının tek, çift veya sıfır olarak değerlendirilmesi tek bir fonksiyonda incelenebilir. Böylece her bir işlem için fonksiyon yazılması yerine tek bir fonksiyon içerisinde her üç koşul kontrol edilebilir ve ilgili değişkenler de fonksiyon tarafından güncellenebilir. Sayıların ayrıstırıldığı fonksiyonda çift tek koşulu için mantıksal yapı olarak switch ... case kullanılması ayrı if ... else if ... else yapılarının kullanılmasına göre daha hızlıdır. Yapılacak olan ayrıştırma ve sayımla işlemler için ortak kullanılacak değişkenler (referans parametreleri) kullanılması daha uygundur. Böylece her bir fonksiyon için ayrı değişkenlere ve raporlama araçlarına gerek kalmayacaktır.

Programın yukarıdaki algoritma ve tasarlanan yapıya göre yazılmış kaynak kodu aşağıda verilmiştir.

```
/*
 * tek_cift_sifir.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
```

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
#include <locale>
using namespace std;
const int ADET = 20;
// Fonksiyon prototipleri
void degiskenler(int& sifir_sayi, int& tek_sayi, int& cift_sayi);
void sayi_oku(int& sayi);
void grupla(int sayi, int& sifir_sayi, int& tek_sayi, int& cift_sayi);
void sonuc_yaz(int sifir_sayi, int tek_sayi, int cift_sayi);
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişken tanımlamaları
    int sayac, sayi, sifir, tek, cift;
    degiskenler(sifir, tek, cift);
    cout << "Bu program girilen "<< ADET << " adet tam sayının içerisinde kaç adet sıfır, tek ve
çift sayı bulunduğu belirler. Aralarında bir adet boşluk bırakarak " << ADET << " tam sayı giriniz
ve bitirince enter tuşuna basınız." << endl;
```

```

cout << "Sayiları giriniz: ";
for (sayac = 1; sayac <= ADET; sayac++)
{
    sayi_oku(sayi);
    cout << sayi << " ";
    grupla(sayi, sifir, tek, cift);
}
cout << endl;
sonuc_yaz(sifir, tek, cift);
return 0;
}

void degiskenler(int& sifir_sayi, int& tek_sayi, int& cift_sayi)
{
    sifir_sayi=0;
    tek_sayi=0;
    cift_sayi=0;
}

void sayi_oku(int& sayi)
{
    cin >> sayi;
}

void grupla(int sayi, int& sifir_sayi, int& tek_sayi, int& cift_sayi)
{
    switch (sayi % 2)
    {
        case 0:
            cift_sayi++;
            if (sayi==0)
                sifir_sayi++;
            break;
        case 1:

```

```

case -1:

    tek_sayı++;

}

}

void sonuc_yaz(int sıfır_sayı, int tek_sayı, int çift_sayı)

{

    cout << "Girilen sayılardan " << çift_sayı << " adedi çifttir." << endl;
    cout << "Girilen sayılardan " << tek_sayı << " adedi tektir." << endl;
    cout << "Girilen sayılardan " << sıfır_sayı << " adedi sıfırdır." << endl;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./tek_cift_sifir
```

Bu program girilen 20 adet tam sayının içerisinde kaç adet sıfır, tek ve çift sayı bulunduğu belirler. Aralarında bir adet boşluk bırakarak 20 tam sayı giriniz ve bitirince enter tuşuna basınız.

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

Girilen sayılardan 10 adedi çifttir.

Girilen sayılardan 10 adedi tektir.

Girilen sayılardan 1 adedi sıfırdır.

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

**Örnek Program:** Bir üniversitenin farklı programlarında kayıtlı olan öğrencilerin her bir programda ortak olanan derslerdeki dönem sonu başarı notlarını esas alarak ders özeline her bir programın başarısını diğerleri ile karşılaştırmak için çubuk grafiği olarak raporlayan bir program yazınız.

**Çözüm:** Programın işleyeceği veri ve bu veriden üreteceği sonuçlar şöyledir:

Her bir eğitim programına ait olan başarı notu verileri ayrı dosyalarda bulunmaktadır. Program bu dosyalardan verileri okuyarak her bir eğitim programına ait olan başarı notu toplamı bulunacaktır. Her bir eğitim programında dersi alan öğrenci sayıları belirlenip, derse ait ortalama elde edilecektir. Eğitim programındaki bir dersin başarı ortalaması da başarı notu toplamının öğrenci sayısına bölünmesi ile bulunacaktır. Program sadece tüm eğitim programlarındaki ortak dersler için raporlama yapacaktır. Eğer herhangi bir ders için en az bir tane eğitim programı verisi bulunamaması durumunda ise program hata mesajı döndürüp sona erecektir.

Problemin çözümünde şu algoritmadan yararlanılabilir:

- 1 Değişkenlerin tanımlanması ve etkinleştirilecektir.
- 2 Her bir programda ortak olan derslerin kodları okunacaktır.

- 3 Eğer ortak ders kodlarında uyumsuzluk belirlenirse, hata mesajı döndürülür ve program sona erecektir.
- 4 Her bir ortak ders verisine sahip olan dosyalardan yararlanılarak programa özel olan başarı notu ortalaması hesaplanacaktır.
- 5 Başarı durumunu belirten ortalama yazdırılacaktır.
- 6 İkiiden beşe kadar olan işlemler her bir ders için tekrarlanacaktır.
- 7 Sonuçlar rapor biçiminde yazdırılacaktır.
  - 7.1 Raporun başlık alanında “Ders Kodu, Program Adı, Program Ortalaması” yazılacaktır.
  - 7.2 Ders kodu sadece bir defa yazılacaktır.
  - 7.3 Ders kodu ardından program kodu yazdırılacaktır.
  - 7.4 Program kodunun ardından programa ait başarı notu ortalaması yazdırılacaktır.

Programın yapılması için ana fonksiyon, yapılacak olan işlemler özelinde fonksiyonlara iki bölünecektir. Birincisi fonksiyon ortalama hesaplayacak ve diğeri de sonuçları rapor biçiminde yazdıracaktır. Fonksiyonlarda parametre aktarımı kullanılarak ortak değişkenler üzerinden değer aktarılarak sonuçlar elde edilecektir. Ana fonksiyonda şu değişkenler kullanılacaktır:

```
string DersAdi1;  
string DersAdi2;  
int Ders_Sayisi=0;  
double ortalama1=0.0, ortalama2=0.0;  
double Program_1_Ortalama, Program_2_Ortalama;  
ifstream program1;  
ifstream program2;  
ofstream cikti;
```

İkinci aşama olarak kullanılacak olan fonksiyonlar tasarılanacaktır. Ortalamaların hesaplanması için **ortalama\_hesapla()** adlı fonksiyon kullanılacaktır. Fonksiyon, her bir programa ait olan ortak derslere ait başarı notlarının bulunduğu dosyadan dersi alan öğrencilerin dönem sonu başarı notlarını okuyacaktır. Okunan başarı notu değerleri toplanacak ve elde edilen sonuç dersi alan öğrenci sayısına bölünderek derse ait olan başarı notu ortalaması belirlenecektir. Dersi alan öğrenci sayısı bilinmediğinden dosyadan okuma yapılırken derse ait olan verilerin bittiğini belirtmek için “-1” kullanılacaktır. Bu değere gelindiğinde okuma sona erecektir.

```
double Basari_Not_Toplama = 0.0;  
int Ogrenci_Sayisi = 0;  
int Basari_Notu;
```

Her bir eğitim programındaki ilgili ortak derse ait veriler toplanacak ve ortalama belirlenecektir. Buna göre bu fonksiyonun algoritması aşağıdaki gibi olacaktır:

- 1 Değişkenlerin tanımlanması
- 2 Derse ait notların okunması
- 3 Not -1 olmadığı sürece;
  - 3.1  $\text{Basari\_Not\_Toplam} = \text{Basari\_Not\_Toplam} + \text{Basari\_Not}$
  - 3.2 Okuma işleminden sonra öğrenci sayısı değişkenini bir arttır
  - 3.3 Sıradaki notu oku
  - 3.4 Okunan değeri kullanarak kontrol değişkenini güncelle
- 4  $\text{Basari\_Not\_Toplam} / \text{Ogrenci\_sayisi}$  ile ortalama hesapla

Burada ortalama hesaplamak için kullanılacak **ortalama\_hesapla()** fonksiyonu aşağıdaki gibi olacaktır:

```
void ortalama_hesapla (ifstream& kaynak, double& DersOrtalama )  
{  
    double Basari_Not_Toplam = 0.0;  
    int Ogrenci_Sayisi = 0;  
    int Basari_Notu;  
    kaynak >> Basari_Notu;  
    while (Basari_Notu != -1)  
    {  
        Basari_Not_Toplam = Basari_Not_Toplam + Basari_Notu;  
        Ogrenci_Sayisi++;  
        kaynak >> Basari_Notu;  
    }  
    DersOrtalama = Basari_Not_Toplam / Ogrenci_Sayisi;  
}
```

Programda yer alan diğer fonksiyon ise elde edilen sonuçların bir tablo halinde önceden belirlenen bir dosyaya yazdırmaktadır. Programın yapması beklenen işlemler tanımlanırken çıktıının biçimi de belirlenmiştir. Buna göre her iki programın karşılaştırılması aynı ders özelinde yapılacaktır. Buna göre de bir dersin kodunun bir defa yazılması yeterlidir. Ayrıca her bir programın grup numarası yazılacaktır. Son olarak da programın o ders için başarı notu ortalaması yer alacaktır. Bütün sonuçlar dosyaya yazılacağı için fonksiyonda şu değişkenler olmalıdır.

- sonuçların yazılması için gerekli olan dosya yazma değişkeni

- başarı notu hesaplanan dersin adı
- programı tanımlayan kod veya grup numarası
- Hesaplanmış olan başarı notu ortalaması

Buna göre yazdırma işleminin tasalanması gereklidir. Aşağıda verilen “**sözde kod – pseudo code**” yazdırma işlemini yani algoritmasını ortaya koymaktadır.

```

if (GrupNo ==1)
    print (grup no == 1)
else
    print "boşluk"
print "grup no," başarı notu ortalaması"
```

Buna göre **sonuc\_yaz()** adlı fonksiyon aşağıdaki gibi olacaktır.

```

void sonuc_yaz (ofstream& cikti, string DersAdi, int grup, double ortalama)
{
    if (grup==1)
        cikti << " " << DersAdi << " ";
    else
        cikti << "      ";
    cikti << setw(8) << grup << setw(17) << ortalama << endl;
}
```

Buraya kadar programda yer alacak olan iki fonksiyon da tasarlanmış ve kaynak kodu yazılmıştır. Bu aşamada “**ana fonksiyonun – main()**” algoritması yazılabilir. Ancak algoritmayı yazmadın önce olası bir hata durumunu kontrol etmek gerekecektir. Elimizde iki adet programa ait olan ve tüm programlarda ortak olan derslerin yer aldığı iki dosya bulunmaktadır. Bu dosyaların içeriklerinin programın doğru çalışması için derslere ait olan verilerin dersin kodu ve notlar şeklinde yer alması gereklidir. Ayrıca ders kodlarının karşılaştırma yapılması için aynı satırda ve aynı şekilde yapılması zorunludur. Eğer kaynak dosyalarda, sıra değişik ise veya bit programda farklı bir derse ait veriler de bulunuyorsa bu durumda program uyarı mesajı görüntülemeli ve sona ermelidir.

Buna göre ana fonksiyon – main() algoritması aşağıdaki gibi olmalıdır:

- 1 Değişkenleri (yerel değişkenler) tanımla.
- 2 Kaynak verinin bulunduğu dosyayı aç.
- 3 Eğer kaynak dosyalara erişilemiyor ise, hata mesajı yaz ve programı sonlandır.
- 4 Sonucun yazılıacağı dosyayı aç.

- 5 Sonuçların kaya noktalı olacağı dikkate alınarak ondalık basamakların görüntülenmesini sağla ve ondalık basamak sayısını iki olarak belirle.
- 6 Birinci grup için ortalama 0.0 olsun.
- 7 İkinci grup için ortalama 0.0 olsun.
- 8 Ders sayısı 0 olsun.
- 9 Raporun başlık satırını yaz.
- 10 Birinci grup için ders kodunu oku.
- 11 İkinci grup için ders kodunu oku.
- 12 Her iki grup için de;
  - 12.1 Eğer ders kodları eşleşmiyor ise hata mesajı döndür ve programı sonlandır.
  - 12.2 Eşleşiyor ise
    - 12.2.1 Birinci grup için ortalamayı hesapla (ortalama\_hesapla() fonksiyonunu çağır ve gerekli parametreleri kullan)
    - 12.2.2 İkinci grup için ortalamayı hesapla (ortalama\_hesapla() fonksiyonunu çağır ve gerekli parametreleri kullan)
    - 12.2.3 Birinci grup için ortalamayı yaz (sonuc\_yaz() fonksiyonunu çağır ve gerekli parametreleri kullan)
    - 12.2.4 İkinci grup için ortalamayı yaz (sonuc\_yaz() fonksiyonunu çağır ve gerekli parametreleri kullan)
    - 12.2.5 Birinci grup için ortalamayı güncelle.
    - 12.2.6 İkinci grup için ortalamayı güncelle.
    - 12.2.7 Ders sayısını bir arttır.
  - 12.3 Birinci grup için ders adını oku.
  - 12.4 İkinci grup için ders adını oku.
- 13 Dosyadan okuma işlemini kontrol et
  - 13.1 Eğer ikinci grupta dosyadan okuma işlemi sona erdi ise uyarı mesajı yaz.
  - 13.2 Eğer birinci grupta dosyadan okuma işlemi sona erdi ise uyarı mesajı yaz.
  - 13.3 Her iki dosyada okuma işlemi aynı anda bitti ise sonuçları dosyaya yaz.
- 14 Açılan tüm dosyaları kapat
- 15 Programı sonlandır.

Programın yukarıdaki algoritma ve tasarlanan yapıya göre yazılmış kaynak kodu aşağıda verilmiştir.

```
/*
*raporla.cxx
*
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
* takdirde serbesttir:
*
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
*
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
*
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
* SONUCUNDA OLUSAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
```

```
* SORUMLU DEĞİLLERDİR.  
*  
*/  
/*  
  
#include <iostream>  
#include <locale>  
#include <iomanip>  
#include <fstream>  
#include <string>  
using namespace std;  
  
// Fonksiyon prototipleri  
void ortalama_hesapla (ifstream& kaynak, double& DersOrtalama );  
void sonuc_yaz (ofstream& cikti, string DersAdi, int grup, double ortalama);  
  
// Ana fonksiyon  
int main()  
{  
    // Dil ayarları  
    setlocale(LC_ALL, "tr_TR.UTF-8");  
    // Değişken tanımlamaları  
    string DersAdi1;  
    string DersAdi2;  
    int Ders_Sayisi=0;  
    double ortalama1=0.0, ortalama2=0.0;  
    double Program_1_Ortalama, Program_2_Ortalama;  
    ifstream program1;  
    ifstream program2;  
    ofstream cikti;  
    program1.open("program1.txt");  
    program2.open("program2.txt");  
    if (!program1 || !program2)  
    {
```

```

cout << "Verilerin alınacağı dosyalara erişilemedi. Program sonlanıyor. hata kodu: 1"
<< endl;

        return 1;

    }

cikti.open("durum.txt");

cikti << fixed << setprecision(2);

cikti << "Ders Kodu Program No    Program Ortalaması" << endl;

program1 >> DersAdi1;

program2 >> DersAdi2;

while (program1 && program2)

{

    if (DersAdi1 != DersAdi2)

    {

        cout << " Ders kodları eşleşmiyor. Program sonlanıyor. Hata kodu: 2" <<
endl;

        return 2;

    }

    else

    {

        ortalama_hesapla (program1, ortalama1);

        ortalama_hesapla (program2, ortalama2);

        sonuc_yaz (cikti, DersAdi1, 1, ortalama1);

        sonuc_yaz (cikti, DersAdi2, 2, ortalama2);

        Program_1_Ortalama = Program_1_Ortalama + ortalama1;

        Program_2_Ortalama = Program_2_Ortalama + ortalama2;

        cikti << endl;

        Ders_Sayisi++;

    }

    program1 >> DersAdi1;

    program2 >> DersAdi2;

}

```

```

if (program1 && !program2)
{
    cout << "İkinci programda verilerin okunması diğer programdan önce sona erdi." <<
endl;
}

else if (!program1 && program2)
{
    cout << "Birinci programda verilerin okunması diğer programdan önce sona erdi."
<< endl;
}

else
{
    cikti << "Birinci program için ortalama: " << Program_1_Ortalama / Ders_Sayisi <<
endl;

    cikti << "İkinci program için ortalama: " << Program_2_Ortalama / Ders_Sayisi <<
endl;

    cikti << endl;
}

program1.close();
program2.close();
cikti.close();

return 0;
}

void ortalama_hesapla (ifstream& kaynak, double& DersOrtalama )
{
    double Basari_Not_Toplam = 0.0;
    int OGRENCI_Sayisi = 0;
    int Basari_Notu;
    kaynak >> Basari_Notu;
    while (Basari_Notu != -1)
    {
        Basari_Not_Toplam = Basari_Not_Toplam + Basari_Notu;
    }
}

```

```

        Ogrenci_Sayisi++;
        kaynak >> Basari_Notu;
    }

    DersOrtalama = Basari_Not_Toplam / Ogrenci_Sayisi;
}

void sonuc_yaz (ofstream& cikti, string DersAdi, int grup, double ortalama)
{
    if (grup==1)
        cikti << " " << DersAdi << " ";
    else
        cikti << "      ";
    cikti << setw(8) << grup << setw(17) << ortalama << endl;
}

```

Programın işleyeceği veriler iki adet programa ait olduğu için program1.txt ve program2.txt dosyaları aşağıdaki gibidir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ cat program1.txt
TBT 80 100 70 80 72 90 89 100 83 70 90 73 85 90 -1
MAT 80 90 80 94 90 74 78 63 83 80 90 -1
TAR 90 70 80 70 90 50 89 83 90 68 90 60 80 -1
TUR 74 80 75 89 90 73 90 82 74 90 84 100 90 79 -1
TEK 100 83 93 80 63 78 88 89 75 -1

[goksin@tardis ~/projeler/C++_Notlar/]$ cat program2.txt
TBT 90 75 90 75 80 89 100 60 80 70 80 -1
MAT 80 80 70 68 70 78 80 90 90 76 -1
TAR 100 80 80 70 90 76 88 90 90 75 90 85 80 -1
TUR 80 85 85 92 90 90 74 90 83 65 72 90 84 100 -1
TEK 90 93 73 85 68 75 67 100 87 88 -1

[goksin@tardis ~/projeler/C++_Notlar/]$
```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./raporla
[goksin@tardis ~/projeler/C++_Notlar/]$ cat durum.txt
```

Ders Kodu	Program No	Program Ortalaması
TBT	1	83.71
	2	80.82
MAT	1	82.00
	2	78.20
TAR	1	77.69
	2	84.15
TUR	1	83.57
	2	84.29
TEK	1	83.22
	2	82.60

Birinci program için ortalama: 82.04  
 İkinci program için ortalama: 82.01

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

**Örnek Program:** Önceki programın verdiği sayısal sonuçların çubuk grafikler şeklinde raporlanması istenmektedir. Bunun için gereken değişiklikleri yapınız.

**Çözüm:** Programın ürettiği sayısal çıktıların grafik olarak yazdırılabilmesi için de programın raporlamayı yapan fonksiyonun yeniden yazılması gerekecektir. Grafik gösterimde işlemlerin basitleşmesi için yatay çubuklar kullanılacaktır. İki adet program karşılaştırıldığı için programlar farklı karakterler ile temsil edilmelidir. Birinci program için '\*' ve ikinci program için de '#' karakteri kullanılacaktır. Ayrıca program kodu tanımlaması sonuç sayfasının altına taşınacak ve grafikte kullanılan karakterler ile tanımlanacaktır.

Çubuk grafiklerin ortalamayı temsil etmesi için kullanılacak olan yaklaşım elde edilen ondalık basamaklı olarak hesaplanan ortalamanın tam sayıya çevrilmesi ile yapılabilir. Bunun için de yazdırılacak karakter sayısını belirlemek için aşağıdaki ifade kullanılabilir:

```
sayac = static_cast<int>(ortalama) / 2;
```

Bu ifade bir **for ...** döngüsü ile birlikte kullanılarak yatay çubuklar elde edilebilir. Bu durumsa her bir program için derse ait olan ortalamalar değil yatay çubuk grafikleri yazdırılır. Buna göre **sonuc\_yaz()** fonksiyonu aşağıdaki gibi yeniden yazılmalıdır.

```
void sonuc_yaz (ofstream& cikti, string DersAdi, int grup, double ortalama)
{
    int adet = 0, sayac;
```

```

if (grup==1)
    cikti << setw(4) << DersAdi << " ";
else
    cikti << " ";
adet = static_cast<int>(ortalama)/2;
if (grup==1)
    for (sayac=1; sayac <= adet; sayac++)
        cikti << '*';
else
    for (sayac=1; sayac <= adet; sayac++)
        cikti << '#';
cikti << endl;
}

```

Çubuk grafiklerinin okunması eğer bir ölçek kullanılıyorsa daha kolay olacaktır. Ancak bir önceki program sürümüğe ait olan kaynak kod içerisinde bir çıktı içerisinde bir ölçek için herhangi bir bildirim bulunmamaktadır. Bu durumda ölçegin yazdırılması için **ana fonkisyon – main()** içerisinde yeni bir fonksiyon olan **baslik\_yaz()** kullanılmıştır. Böylece raporlama yapılan dosyada çubuk grafiklerinin okunmasını kolaylaştıracak bir başlık satırı eklenebilir. Eklenecek olan **baslik\_yaz()** fonksiyonu aşağıdaki gibidir:

```

void baslik_yaz (ofstream& cikti)
{
    cikti << "Ders Kodu      Dersin ortalaması" << endl;
    cikti << "      10  20  30  40  50  60  70  80  90  100" << endl;
    cikti << "      ....|....|....|....|....|....|....|....|" << endl;
}

```

Bu yapılan düzenlemeler sonucunda programın kaynak kodu aşağıdaki gibi olacaktır:

```

/*
*raporla_grafik.cxx
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
* *

```

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
#include <locale>
```

```
#include <iomanip>
#include <fstream>
#include <string>
using namespace std;
// Fonksiyon prototipleri
void baslik_yaz (ofstream& cikti);
void ortalama_hesapla (ifstream& kaynak, double& DersOrtalama );
void sonuc_yaz (ofstream& cikti, string DersAdi, int grup, double ortalama);
// Ana fonksiyon
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişken tanımlamaları
    string DersAdi1;
    string DersAdi2;
    int Ders_Sayisi=0;
    double ortalama1=0.0, ortalama2=0.0;
    double Program_1_Ortalama, Program_2_Ortalama;
    ifstream program1;
    ifstream program2;
    ofstream cikti;
    program1.open("program1.txt");
    program2.open("program2.txt");
    if (!program1 || !program2)
    {
        cout << "Verilerin alınacağı dosyalara erişilemedi. Program sonlanıyor. hata kodu: 1"
<< endl;
        return 1;
    }
    cikti.open("durum.txt");
```

```

cikti << fixed << setprecision(2);
baslik_yaz (cikti);
program1 >> DersAdi1;
program2 >> DersAdi2;
while (program1 && program2)
{
    if (DersAdi1 != DersAdi2)
    {
        cout << "Ders kodları eşleşmiyor. Program sonlanıyor. Hata kodu: 2" <<
endl;
        return 2;
    }
    else
    {
        ortalama_hesapla (program1, ortalama1);
        ortalama_hesapla (program2, ortalama2);
        sonuc_yaz (cikti, DersAdi1, 1, ortalama1);
        sonuc_yaz (cikti, DersAdi2, 2, ortalama2);
        Program_1_Ortalama = Program_1_Ortalama + ortalama1;
        Program_2_Ortalama = Program_2_Ortalama + ortalama2;
        cikti << endl;
        Ders_Sayisi++;
    }
    program1 >> DersAdi1;
    program2 >> DersAdi2;
}
if (program1 && !program2)
{
    cout << "İkinci programda verilerin okunması diğer programdan önce sona erdi." <<
endl;
}

```

```

else if (!program1 && program2)
{
    cout << "Birinci programda verilerin okunması diğer programdan önce sona erdi."
<< endl;
}

else
{
    cikti << "Birinci program -> *****" << endl;
    cikti << "İkinci program -> #####" << endl << endl;
    cikti << "Birinci program için ortalama: " << Program_1_Ortalama / Ders_Sayisi <<
endl;
    cikti << "İkinci program için ortalama: " << Program_2_Ortalama / Ders_Sayisi <<
endl;
    cikti << endl;
}

program1.close();
program2.close();
cikti.close();
return 0;
}

void baslik_yaz (ofstream& cikti)
{
    cikti << "Ders Kodu      Dersin ortalaması" << endl;
    cikti << "      10  20  30  40  50  60  70  80  90  100" << endl;
    cikti << "      ....|....|....|....|....|....|....|....|" << endl;
}

void ortalama_hesapla (ifstream& kaynak, double& DersOrtalama )
{
    double Basari_Not_Toplama = 0.0;
    int Ogrenci_Sayisi = 0;
    int Basari_Notu;
}

```

```

kaynak >> Basari_Notu;
while (Basari_Notu != -1)
{
    Basari_Not_Toplam = Basari_Not_Toplam + Basari_Notu;
    Ogrenci_Sayisi++;
    kaynak >> Basari_Notu;
}
DersOrtalama = Basari_Not_Toplam / Ogrenci_Sayisi;
}

void sonuc_yaz (ofstream& cikti, string DersAdi, int grup, double ortalama)
{
    int adet = 0, sayac;
    if (grup==1)
        cikti << setw(4) << DersAdi << " ";
    else
        cikti << " ";
    adet = static_cast<int>(ortalama)/2;
    if (grup==1)
        for (sayac=1; sayac <= adet; sayac++)
            cikti << '*';
    else
        for (sayac=1; sayac <= adet; sayac++)
            cikti << '#';
    cikti << endl;
}

```

Program derlenip çalıştırılırsa sonuçlar dosyaya yazdırılacak ama sayısal sonuç yerine yatay çubuk grafikleri şeklinde gösterilecektir. Program derlenip çalıştırılırsa sonuçlar durum.txt adlı dosyada oluşur. “cat durum.txt” komutu ile dosyanın içeriği görüntülenebilir.



## 6.Programcı Tarafından Tanımlanan Basit Veri Tipleri, İsim Uzayları ve string Veri Tipi

Önceki bölümlerde C++ Programlama Dili'nin temel veri tiplerinin üye ayrıldığını ve bunların **basit ve yapılandırılmış veri tipleri** ile **işaretçiler** olarak grubalandığını belirtmiştık. Temel veri tipleri de kendi içerisinde mantıksal, karakter, tam sayı, kayar noktalı sayı ve karakter dizileri olarak yeniden grubalandırılmıştır. Bunlar içinde yaygın olarak bool, char, int, float, double ve string ile çalışılmıştır. Bunların yanında bir diğer veri tipi olan ve programcı tarafından tanımlanmış sıraya göre üzerinde işlemler yapılan veri tipi olan “**enumeration – sıralama veri tipi**” de bulunmaktadır. Bu ve yapılandırılmış veri tipleri tüm programlama dillerinde önemli bir yere sahip olup, programcılar bir çok problemi kolaylıkla çözmeyi sağlamaktadır. Ayrıca ANSI/ISO C++ standartlarını benimseyen tüm projelerde yazılan C++ programları “**isim uzayları – namespaces**” kullanılmaktadır. Bu bölümde isim uzayları ile karakter dizileri üzerinde gerçekleştirilen işlemler de inceleneciktir.

### 6.1.Sıralama – Enumeration – Veri Tipi

Bir bilgisayarın amacı bilgiyi depolamak ve bilgi ile işlem yapmaktır. Bunun gerçekleştirmesi için de bilgisayarın yapacağı eylemlerin bir bilgisayar programı ile tanımlanması gereklidir. Bilgisayar programlarını temel işlevi veri işlemek olduğundan veri tipleri tanımlanır. Bir veri tipi verinin nasıl ifade edileceği ile bu veri üzerinde gerçekleştirilecek olan işlemleri tanımlar. Bu nedenle de bilgisayar programlama dillerinde veri tipleri önemli bir yere sahiptir. Ancak bilgisayar ile problem çözmenin doğası gereği, problemlerin çözümünde kullanılacak olan veri bir programlama dilinin yapısında tanımlanmış olan standart veri tipleri ile her zaman çözülemez. Bu nedenle de bir programlama dilinin esnekliği de programcı tarafından tanımlanabilen veri tiplerinin varlığında gözlenebilir.

C++ Programlama Dili’nde bu şekilde programcı tarafından tanımlanabilen veri tipleri arasında “sıralama veri tipi – neumeration data type”da yer alır. Bu veri tipinin kullanılması için şu koşulların sağlanması gereklidir:

1. Veri tipine bir sim verilerek tanımlanmış olması
2. Veri tipi ile kullanılacak olan verilerin tanımlanmış olması,
3. Bu veriler üzerinde gerçekleştirilecek olan işlemler

C++ Programlama Dili, gerektiğinde programcılar yeni veri tipleri tanımlamasına izin vermektedir. Bunu yapmak için veri tipinin adı ile değerlerini tanımlanması yeterlidir. Öte yandan bu veriler üzerinde yapılacak olan işlemler önceden tanımlı olduğu için programcının yeni işlemler tanımlamasının önüne geçilmektedir. Bunu tercih edilmesini nedeni, olası hataların tüm sistem üzerinde olabilecek yıkıcı etkilerinin önüne geçilmesidir.

Veri tipinde kullanılan değerler, veri tipi için tanımlamalar olmak durumundadır.

enum İsim (değer, değer, değer ,değer, ... )

Bu tanımlamada yer verilen değerler, “**enumerator – sıralayıcı**” adı verilen tanımlayıcılardır. Unutulmaması gereken “enum” programlama diline ait olan bir sözcüktür. Başka şekillerde programlarda kullanılamaz.

Sıralanmış bir ve itipi tanımlaması yapıldığında ilk değerden başlamak üzere her bir değer bir sonrakinden küçük olacak şekilde sıralanır. Buna göre değer1 < değer2 < değer3 < değer4 < ... olarak sıralama gerçekleşir. Ayrıca sıralamada ilk değer indis olarak 0 ile başlar. Bunun anlamı değer1 indis 0, değer2 indis 1, değer 3 indis, ... olarak tanımlanmaktadır. Bu sıralama ön tanımlı olmakla birlikte programcı tarafından gerekiğinde yeniden kurgulanabilir. Sıralama veri tipinde dikkat edilmesi gereken bir diğer önemli nokta da tanımlanan verilerin değişken olmadıklarıdır.

Aşağıda sıralama veri tipi tanımlamaları görülmektedir. Bunlar arasında geçeli olanlar ile geçersiz olanlara yer verilmiştir.

`enum donemler (YY1, YY2, YY3, YY4, YY5, YY6, YY7, YY8);`

Bu örnekte “donemler” adı verilen bir veri tipi tanımlanmıştır. Değerleri YY1, YY2, YY3, YY4, YY5, YY6, YY7, YY8 olarak tanımlanmıştır.

`enum siniflar (bir, iki, uc, dort);`

Bu örnekte “siniflar” adı verilen bir veri tipi tanımlanmıştır. Değerleri bir, iki, uc ve dort olarak tanımlanmıştır.

Bir sıralama veri tipi tanımlandığında verilerin mutlaka bir tanımlayıcı olması gereklidir. Verilen tanımlanmasında ise C++ Programlama Dili’nde geçerli olan tanımlama kuralları geçerlidir. Aşağıda verilen örneklerdeki sıralama veri tipleri geçersizdir.

`enum siniflar ('A', 'B', 'C', 'A', );`

Benze şekilde aşağıdaki sıralama veri tipi tanımlaması da geçersizdir.

`enum sira (1, 2, 3, 4, 5);`

Ayrıca bir sıralama veri tipi tanımlaması yapıldığında, kullanılan veriler bir diğer sıralama veri tipi tanımlamasında kullanılmaz. Sıralama veri tipi tanımlamasında her bir tanımlama için veriler tekil olmak durumundadır. Aşağıdaki örneklerde bu kurala uyulmadığı için geçersizdir.

`enum etkinlik_salonlar (A1, A2, A3, B2, B3, E2);`

`enum salonlar (A1, A2, A3, B1, B2, B3, E1, E2, E3);`

### 6.1.1.Değişkenlerin Tanımlanması

Bir veri tipi tanımlandığında bu veri tipinde değişken tanımlamaları yapılabilir. Bir değişkenin tanımlanması için aşağıdaki şablon kullanılır

`veritipi tanımayıcı, tanımlayıcı, ... ;`

Buna göre sıralam veri tipi tanımlandıktan sonra değişken tanımlamaları aşağıdaki gibi yapılabilir.

`enum dersler (BTP101, BTP103, ELO103, ING187, MAT125, TAR165, TEK107, TUR125);`

Yukarıdaki ifade dersler adlı bir sıralama veri tipi ve b veri tipine ait olan veriler tanımlanmıştır. Bu veri tipinde değişkenlerin tanımlanması ise aşağıdaki gibidir.

```
dersler mesleki_ders, ortak_ders;
```

Bu değişken tanımlaması ise dersler sıralı veri tipi ile ilişkili olan mesleki\_dersler ve ortak\_dersler adlı iki değişken tanımlanmıştır.

### 6.1.2. Atamalar

Bir değişken tanımlandığında, bu değişkende belirtilen veri tipine ait veri barındırılabilir. Yukarıda yapılan tanımlamaları esas alarak değişkenlere aşağıda gösterildiği şekilde veri ataması yapılabilir.

```
mesleki_ders = BTP101;
```

```
ortak_ders = TUR125;
```

İlk ifade mesleki\_ders adlı değişkene BTP101, ve ikinci değişken olan ortak\_ders ise TUR125 verisi atanmıştır.

### 6.1.3. Sıralama Veri Tipi Üzerinde Gerçekleştirilen İşlemler

Sıralama veri tipi üzerinde herhangi bir aritmetik işlem gerçekleştirilemez. Bu nedenle de aşağıda verilen ifadeler geçersizdir.

```
mesleki_dersler = ortak_ders +1;
```

```
ortak_ders = mesleki_ders * 3 ;
```

Benzer şekilde artım ve azaltım işlemleri de yapılamaz. Bu nedenle de aşağıda verilen ifadeler geçersizdir.

```
mesleki_dersler ++;
```

```
ortak_ders --;
```

Bir sıralama veri tipinde tanımlanmış olan bir değişkenin değerinin artım veya azaltım yapılması gerekiyor ise bu durumda programcı **static\_cast<>()** kullanmak durumundadır. Derleyici çalıştırıldığında programcının burada ne yapmak istediğini bildiği var sayilarak derleyici kaynak kodu makine diline çevirecektir. Bu duruma aşağıda verilen kaynak kod derlenebilir.

```
mesleki_ders = static_cast<dersler>(mesleki_ders + 1);
```

Bu ifadenin işlenmesi sonucunda mesleki\_ders adlı değişkenin değeri 1 artar. Aşağıdaki örnekte bu durum gösterilmiştir.

```
mesleki_ders = BTP101;
```

```
mesleki_ders = static_cast<dersler>(mesleki_ders + 1);
```

Bu ifadenin işlenmesi ile mesleki\_ders adlı değişkenin değeri BTP101 iken BTP103 olacaktır. Aynı şekilde aşağıdaki ifadenin işlenmesi ide eski duruma dönülmüş olur.

```
mesleki_ders = static_cast<dersler>(mesleki_ders - 1);
```

#### 6.1.4. Sıralama Veri Tipi Üzerinde İlişkisel Operatör İşlemleri

Sıralama veri tipi, bir indise göre sırası belirli olan nesnelerin bir kümesidir. Bu nedenle de ilişkisel operatörler bu veri tipi ile ilgili işlemler yapmak için kullanılabilir. Aşağıdaki örneklerde ilişkisel operatörler ve sonuçları gösterilmiştir.

```
enum dersler (BTP101, BTP103, ELO103, ING187, MAT125, TAR165, TEK107, TUR125);  
dersler mesleki_ders1, mesleki_ders2, ortak_ders1, ortak_ders2;  
mesleki_ders1 = BTP101;  
mesleki_ders2 = BTP103  
ortak_ders1 = TAR165;  
ortak_ders2 = TAR165;
```

Yukarıdaki tanımlamalar yapıldıktan sonra aşağıdaki ifadelerin mantık sonuçları “**doğru – true**” olacaktır.

```
mesleki_ders1 <= mesleki_ders2  
ELO103 > BTP103  
TUR125 > BTP101
```

Aşağıda verilen şu ifadelerin mantıksal sonuçları ise “**yansı – false**” olacaktır.

```
ortak_ders1 <= mesleki_ders2  
ING187 > BTP103  
TUR125 < MAT125
```

Sıralama veri tipi özünde “**sıralı küme – ordered set**” olarak tanımlanmış olduğu için bu durumda ilişkisel işlemler ile programcı tarafından yapılacak olan veri tipi dönüşümü işlemleri de yapılabilir. Bu nedenle sıralı veri tipi döngüler içinde kullanım alanı bulacaktır. Aşağıdaki örnekte verilen **for ...** döngüsü işlenecektir.

```
for(mesleki_ders1 = BTP101; mesleki_ders1 <= TUR125; mesleki_ders1 =  
static_cast(dersler)(mesleki_ders1 +1)  
...  
Bu döngü yedi defa çalışacaktır.
```

#### 6.1.5. Sıralama Veri Tipi ile Girdi/Cıktı İşlemleri

C++ Programlama Dili’nde girdi ve çıktı işlemleri sadece dahili veri tipleri için yani int, char, double, float ve string için tanımlıdır. Bu nedenle sıralama veri tipi tanımlandığında bir girdi/çıktı işlemi için doğrudan kullanılamaz. Ancak dolaylı yoldan kullanılabilir. Aşağıdaki örnek kaynak kodda bu dolaylı girdi/çıktı işlemi gösterilmiştir.

Örneğin bir üniversitenin ders yılının her bir yarı yılında öğrencilerinin ders seçimi zamanında kullanılan bir programın bir bölümü görülmektedir.

```

enum dersler {BTP101, BTP103, ELO103, ING187, MAT125, TAR165, TEK107,
TUR125};

string ders_kodu, kayit_olunan;

cin >> ders_kodu;

if (ders_kodu == "BTP101")
    kayit_olunan = "BTP101";
else if (ders_kodu == "BTP103")
    kayit_olunan = "BTP103";
else if (ders_kodu == "ELO103")
    kayit_olunan = "ELO103";
else if (ders_kodu == "ING187")
    kayit_olunan = "ING187";
else if (ders_kodu == "MAT125")
    kayit_olunan = "MAT125";
else if (ders_kodu == "TAR125")
    kayit_olunan = "TAR125";
else if (ders_kodu == "TEK107")
    kayit_olunan = "TEK107";
else if (ders_kodu == "TUR125")
    kayit_olunan = "TUR125";
else
    cout << "Geçersiz seçim yaptınız." << endl;

```

Benzer şekilde **switch()** ... **case** ile de aynı işlem yapılabilir.

```

enum dersler {BTP101, BTP103, ELO103, ING187, MAT125, TAR165, TEK107,
TUR125};

string ders_kodu, kayit_olunan;

cin >> ders_kodu;

switch(kayit_olunan)
{
    case BTP101:
        kayit_olunan = "BTP101";

```

```

        break;

    case BTP103:
        kayit_olunan = "BTP103";
        break;

    case ELO103:
        kayit_olunan = "ELO103";
        break;

    case ING187:
        kayit_olunan = "ING187";
        break;

    case MAT125:
        kayit_olunan = "MAT125";
        break;

    case TAR125:
        kayit_olunan = "TAR125";
        break;

    case TEK107:
        kayit_olunan = "TEK107";
        break;

    case TUR125:
        kayit_olunan = "TUR125";
default:
        cout << "Geçersiz seçim yaptınız." << endl;
}

```

Yukarıdaki örneklerde sıralama veri tipinde tanımlana veriler, dolaylı olarak girdi/çıktı işlemlerinde kullanılmıştır. Eğer kayit\_olunan adlı değişkenin değeri eğer TAR125 olarak girilmiş ise aşağıdaki kaynak kod örneğinin işlenmesi ile bu veriye ait olan indis numarası elde edilir.

```
cout << kayit_olunan << endl;
```

Bu bildirimin sonucu “5” olacaktır. Aynı şekilde aşağıdaki ifadede aynı sonucu döndürür.

```
cout << TAR125 << endl;
```

## 6.1.6.Sıralama Veri Tipi ve Fonksiyonlar

Fonksiyonlarda değer veya parametre aktarımı şeklinde, bir sıralama veri tipi verisini fonksiyona aktarabiliriz. Benzer şekilde bir fonksiyondan da sıralama veri tipinde tanımlı olan bir değişken değer olarak döndürülebilir veya parametre olarak da aktarılabilir. Böylece fonksiyonlar sıralama veri tipi için girdi ve çıktı işlemleri için kullanılabilirler. Aşağıdaki örnek kaynak kod içe-risinde klavyeden sadece ilk üç harfi girilen sebzelerin ana fonksiyona değer olarak döndürülmesi görülmektedir. Öncelikle manav adlı bir sıralı veri tipi tanımlanmış ve değerleri belirtilmiştir.

```
enum manav{pazı, pırasa, pancar, soğan, salatalık, biber, bamya, brokoli, karnabahar, kereviz, ayva, armut, avokado, elma}
```

Fonksiyon aşağıda verilmiştir.

```
enum manav{pazı, pırasa, pancar, sogan, salatalık, biber, bamya, brokoli, karnabahar, kereviz, ayva, armut, avakado, elma};
```

```
string urun;  
char karakter1, karakter2, karakter3;  
cout << "Aldığınız manav ürünlerinin ilk üç harfini yazıp enter tuşuna basınız";  
cin >> karakter1 >> karakter2 >> karakter3;  
switch (karakter1)  
{  
    case 'p':  
        if (karakter2 == 'a' && karakter3 == 'z')  
            urun = pazı;  
        else if (karakter2 == 'a' && karakter3 == 'n')  
            urun = pancar;  
    case 's':  
        if (karakter2 == 'o' && karakter3 == 'g')  
            urun = sogan;  
        else if (karakter2 == 'a' && karakter3 == 'l')  
            urun = salatalık;  
    case 'b':  
        if (karakter2 == 'i' && karakter3 == 'b')  
            urun = biber;  
        else if (karakter2 == 'a' && karakter3 == 'm')  
            urun = bamya;
```

```

else
    urun = brokoli;

case 'k':
    if (karakter2 == 'a' && karakter3 == 'r')
        urun = karnabahar;
    else if (karakter2 == 'e' && karakter3 == 'r')
        urun = kereviz;

case 'a':
    if (karakter2 == 'y' && karakter3 == 'v')
        urun = ayva;
    else if (karakter2 == 'r' && karakter3 == 'm')
        urun = armut;
    else
        urun = avakado;

default:
    urun = elma;
}

return urun;
}

```

### 6.1.7. Sıralama Veri Tipi Kullanarak Değişken Tanımlamaları

Yukarıdaki örneklerde sıralama veri tipi tanımlanmasının ardından değişkenlerin tanımlanlığını belirtmiştir. Ancak C++ Programlama Dili bu konuda bir kolaylık sağlayarak değişken tanımlamalarının sıralama veri tipi tanımı ile birlikte yapılmasına olanak vermektedir. Örneğin aşağıdaki bildirimde bu uygulama görülmektedir.

```
enum harf_notu (DZ, FF, DD, DC, CD, CC, CB, BC, BB, BA, AB, AA) ders_harf_notu;
```

Yukarıdaki ifade aşağıdaki ifadeler ile aynıdır.

```
enum harf_notu (DZ, FF, DD, DC, CD, CC, CB, BC, BB, BA, AB, AA);
```

```
harf_notu ders_harf_notu;
```

### 6.1.8. Anonim Veri Tipleri

Sıralama veri tipi olarak tanımlanan bir veri tipinde eğer veri tipi için herhangi bir isim tanımlanmadan doğrudan değerlerin yazılıp ardından değişkenin veya değişkenlerin tanımlanması

ile oluşturulula veri tipi tanımlamasına anonim veri tipi adı verilir. Aşağıdaki örnekte bir anonim veri tipi tanımlamasını göstermektedir.

```
enum (A1, A2, A3, B1, B2, B3, C1, C2, D1, E1, E2) salonlar
```

Bu veri tipi tanımlamasının iki olumsuz yanı vardır: Birincisi, bu veri tipi tanımlaması herhangi bir isime sahip olmadığı için bir fonksiyona değer veya parametre olarak aktarılması olanaklı değildir. AYNI şekilde de bir fonksiyon de anonim veri tipini sonuç veya parametre olarak döndürmez. İkincisi de anonim veri tipi tanımlaması bir diğer anonim veri tipi tanımlamasında kullanılabilir. Örneğin aşağıdaki ki adet anonim veri tipi tanımlaması yapılmıştır.

```
enum (A, B, C, D, E, F) notlar;
```

```
enum (A, B, C, D, E, F) kategoriler;
```

Yukarıdaki tanımlamaya göre iki anonim veri tipinin değerleri aynı gibi görünse de derleyici açısından bunlar farklı veri tipi tanımlamalarıdır. Dolayısı ile de bu veri tipleri ile tanımlanan değişkenlerde farklıdır. Bu nedenle de aşağıdaki ifade geçersiz bir ifadedir.

```
notlar = kategoriler;
```

Bu anonim veri tipi tanımlamaları nesne yönelimli programlama paradigması içerisinde kendisine yer bulmaktadır. Yapısal programlama paradigması içerisinde sıralama veritipleri tanımlanırken bir isim verilmesi ve değişkenlerinde bu veri tipi ile birlikte tanımlanması doğru bir yaklaşımındır.

### 6.1.9. **typedef** Bildirimi

C++ Programlama Dili’nde önceki bölümlerde tanımlanmış olan veri tiplerini tanımlamak için programcı tarafından tanımlana bilen takma isimler kullanılabilir. Bunun için **typedef** kullanılır. Aşağıda kullanım şekli görülmektedir.

```
typedef VarOlanVeriTipi yeni_isim;
```

Aşağıdaki örnekte bazı veri tipleri için tanımlanmış olan takma isimlerin bildirilmesi görülmektedir.

```
typedef int TamSayi;
```

```
typedef double Ondalik;
```

```
typedef boolean Mantiksal;
```

Yukarıdaki örneklerden de görüleceği üzere bazı veri tipleri için takma isimler tanımlanmıştır.

## 6.2. İsim Uzayları – Namespaces

İsim uzayları C++ Programlama Dili’nin ilk olarak 1988 yılında kabul edilmiş olan ANSI/ISO standarı ile tanımlanmıştır. İsim uzaylarının kullanım alanı, bir programın yapısında kullanılacak olan ön işlemci yönergeleri veya bir proje için geliştirilen harici kütüphanelerin programa eklenmesi için kullanılması durumunda ortay çıkabilecek olan çakışmaların önüne geçilmesi için

tanımlanmış olan bir mekanizmadır. Bunun nedeni standarda göre derleyici ve bağlayıcının makine diline çevrilmiş olan koda eklenmesi gereken dış kaynakların tanımlanması “**alt çizgi – underscore**” kullanılması gerekliliğinden kaynaklanmaktadır. Bu durumdan dolayı bir derleyici eğer aynı şekilde tanımlanmış olan birden çok sayıda kütüphane vb eklemek istediğiinde önceden tanımlanmış olduğuna dair hata mesajı döndürecektir.

Hata mesajının önüne geçirilmesi ile isim uzayları mekanizmasının kullanılması ile olanaklıdır. Çünkü isim uzayı mekanizmasına göre bir isim uzayı tanımlandığında, sadece kendi kapsamı içerisinde tanımlanmış olan üyeleri bu isim uzayı içerisinde var olacaklardır. Isim uzayının dışından erişim ise söz konusu değildir. Ancak gerekli tanımlamaların yapılması ile bir isim uzayı içerisinde tanımlanmış olan üyeleri dışarıdan erişilebilir.

ANSI/ISO standardına göre bir isim uzayının tanımlanabilmesi için aşağıdaki yapının kullanılması gereklidir.

```
namespace isim_uzayının_adı
{
    üyeleri
}
```

Tanımlama içerisinde isim uzayının üyeleri olacak olan sabitler, değişkenler, bildirimler ve fonksiyonlar ile diğer isim uzayları da yer alabilir. Aşağıdaki örnekte “genel” adı verilen bir isim uzayı tanımlanmış ve yapısında bulunan üyeleri belirtilmiştir.

```
namespace genel
{
    const int G = 3;
    const double ORAN = 0.18;
    int adet = 0;
    void vergi_hesapla();
}
```

Yukarıdaki tanımlaya göre söz konusu üyeleri sadece genel adlı isim uzayı içerisinde erişilebilir durumdadır. Eğer bir isim uzayının bir üyesinin, çağrılmaması gerekiyor ise bu durumda aşağıdaki çağrı biçimini kullanılmalıdır.

```
isim_uzayının_adı :: ÜYE
```

Yukarıdaki örnek tanımlamanın bir programda kullanıldığını var sayarsak, aşağıdaki çağrıların yapılması durumunda söz konusu üyenin tanımlı olduğu isim uzayının dışından erişebilirilmiş oluruz.

```
genel::ORAN;
```

benzer şekilde fonksiyonun erişilebilir olması için de,

```
genel:: vergi_hesapla()
```

ANSI/ISO standardına göre isim uzayının tüm üyelerinin dışarıdan erişilebilir olması için aşağıdaki tanımlamanın yapılması yeterlidir.

```
using namespace genel;
```

Eğer sadece belirli üyelerin erişilebilir olmasını istiyor isek,

```
using namespace genel::ORAN;
```

Eğer yukarıdaki tanımlamayı bir program içerisinde kullanmak ve programın tüm bileşenlerinin bu isim uzayında tanımlanmış olan üyelerle erişilebilir kılınmasını istersek aşağıdaki kod blokundaki gibi tanımlamaların yapılması gereklidir.

```
namespace genel
```

```
{
```

```
    const int G = 3;
```

```
    const double ORAN = 0.18;
```

```
    int adet = 0;
```

```
    void vergi_hesapla();
```

```
}
```

```
using namespace genel;
```

Bir program içerisinde bir isim uzayının genel olarak erişime açılması için “**using namespace ...**” kullanılabilir. Burada dikkat edilmesi gereken nokta ise isim uzayı ile program içerisinde tanımlanmış bir genel bildirimin aynı isime sahip olması durumunda isim uzayının üyelerine erişim için “**namespace isim:: üye**” bildirimini kullanması gereklidir. Benzer şekilde bir genel değişken veya genel olarak tanımlanmış bir ifadeyi barındıran bildirim veya kod bloku, bir isim uzayının içerisinde kullanılacak olursa “**namespace isim:: ...**” bildirimi kullanılmalıdır. Aşağıdaki örneklerde bu konu açıklanmıştır.

```
#include <iostream>
```

```
using namespace std;
```

```
...
```

```
int main()
```

```
{
```

```
...
```

```
}
```

Bu örnek kaynak kodda iostream kütüphanesinin üyeleri olan cin, cout, ve endl kullanılırken öncelerine “std::” yazılmasına gerek yoktur. Doğrudan kullanılabilir. Bu konuda daha önceden yer verilen örneklerden bilinmektedir.

```

#include <iostream>
# include <cmath>
...
int main()
{
    double a = 4.6, b = 3.0, y;
    y = std:: pow(4.6 , 3.0);
    std::cout << y << std::endl;
}

```

Yukarıdaki örnekte ise, isim uzayı için herhangi bir tanımlama yapılmamıştır. Bu nedenle de söz konusu cmath ve iostream kütüphanelerine ait üyelerin kullanılabilmesi için “std::” ön eki kullanılmıştır.

```

#include <iostream>
# include <cmath>
...
int main()
{
    using namespace std;
    ...
}

```

Yukarıdaki örnekte isim uzayı ana fonksiyon – main() içerisinde tanımlanmıştır .Böylelikle ana fonksiyon içerisindeki tüm kod blokları ve fonksiyonlar iostream ve cmath kütüphanelerine erişirken “**std::**” ön ekini kullanmak durumunda olmayacağındır. Ancak tanımlama yapılmadan önce yazılan tüm kod blokları ve fonksiyonlar için ise bu durum geçerli değildir.

### **6.3.string – Karakter Katarı Veri Tipi**

Karakter katarları adında anlaşılacağı üzere birden çok sayıdaki karakterin bir araya getirilmesi ile oluşan karakter kümeleridir. Bu karakter kümeleri C++ Programlama Dili’nde “**string data type - karakter katarı veri tipi**” olarak tanımlanır. C++ Programlama Dili’nin ISO/ANSI Standartlaştırılması öncesinde dile dahil edilmemiş olduğu için C++ derleyicileri yazan taraflar tarafından harici kütüphaneler olarak sunulmuştur. Buda farklı kütüphanelerin ve farklı derleyicilerin farklı çalışan programlar/yazılmasına neden olmuştur. Standartlaşmanın sonrasında ise bu veri tipi C++ Programlama Dili’nin “**dahili – integral type**” veri tipi olarak yer bulamamış olsa da standart C++ kütüphanesi içerisinde standartlaşmıştır. Bugün karakter katarları üzerinde yapılacak olan işlemlerin neler olduğu, nasıl yapılacağı kütüphanede tanımlanmıştır. Karakter katarları üzerinde

yapılacak olan işlemler için “**string**” önişlemci komutunun programın başlangıç bölümünde yazılıması gereklidir.

```
#include <string>
```

Karakter katarları konusunda daha ilerlemeden önce önceki bölümlerde karakter katarları üzerinde işlemlerin yapıldığı önceki bölümlerin tekrar gözden geçirilmesinde yarar bulunmaktadır. Ayrıca karakter katarlarının birden çok sayıdaki karakterin ardışık olarak bellekte yer olması ve son karakterden sonra ise “**null terminator - \n**” yer alarak karakter katarının sonlandırıldığı her zaman için aklıda tutulmalıdır.

Aşağıdaki bir karakter katarı tanımlanmıştır.

```
string isim = "Baturalp Dinçdari";
```

Tanımlanmış olan isim adlı karakter dizisine bir karakter katarı atanmıştır. Karakter kataralarının diğer dahili veri tiplerinde olduğu gibi büyülük olarak herhangi bir sınırı bulunmaktadır ancak bu sınır işlemciye ve derleyicilere göre değişkenlik gösterebilir. Pratik bir kural olarak bir karakter katarının uzunluğunun bir tam sayı olduğu ve işaretsiz tam sayı olarak ifade edildiğini burada belirtmekte yarar var. Bunların ışığında karakter katarının büyülüğünün pratik olarak işretsiz tam sayının üst sınırı olacağının tahmin edilebilir.

Karakter katarları üzerinde önceki bölümlerde yer verilen örneklerde görülen işlemler dışında ayrıca bir den çok sayıdaki karakter katarı birleştirilebilir veya bir karakter katarının elemanları üzerinde de çeşitli işlemler gerçekleştirilebilir. Aşağıdaki örnekte iki karakter katarının birleştirilmesi gösterilmiştir.

```
string isim = "Baturalp Dinçdari";
string karakterkatari1, karakterkatari2, karakterkatari3;
cout << isim << endl;
karakterkatari1 = "merhaba";
karakterkatari2 = karakterkatari1 + " " + isim;
cout << karakterkatari2;
karakterkatari3 = isim;
cout << karakterkatari3 << endl;
```

İlk karakter katarı tanımlaması “isim” adlı değişkende yapılmıştır. İkinci olarak karakterkatari1, karakterkatari2 ve karakterkatari3 adlı değişkenler karakter katarı veri tipinde tanımlanmıştır. İkinci satırda tanımlanmış olan karakterkatari1 adlı değişkene “merhaba” karakter katarı atanmıştır. Sonrasında karakterkatari2 adlı değişkene karakterkatari1 ve isim değişkenleri ile birlikte “ ” yani boşluk eklenmiştir. Bu işlemden sonra çıktı ise “merhaba Baturalp Dinçdari” olacaktır. Sonraki satırda geçildiğinde karakterkatari3 adlı değişken isim adlı değişken ile aynı veriyi barındıracaktır.

Yukarıdaki örnekte kullanılan **aritmetik toplama operatörü** olan “+” karakter katarları ile kullanıldığında iki karakter katarının birleştirilip tek bir karakter katarına dönüştürülmesini sağlar.

Karakter katarları üzerinde yapılabilecek olan işlemlerde karakter katarının uzunluğu yani eleman sayısı önemlidir. C++ Programlama Dili ile diğer bir çok dil için bir verinin bellekteki başlangıç adresi, veri tipi tanımı gereği bellekte her bir tekil verinin kapsadığı alanın büyüklüğü eşit olduğundan her bir verinin bellek üzerindeki konumu ilk verinin bellek adresi referans alınarak bu adrese olan uzaklığa ile bulunabilir. Böylece ilk verinin konumu sıfır olarak kabul edilir ve sırası ile izleyen her verinin konu bir artarak devam eder. Karakter katarları üzerinde bir karakterin konumu o katar içerisindeki konumunu belirten **indis** ile tanımlanmaktadır. İndisin kullanılması gereken durumlarda indis “[ ]” ile belirtilir.

Yukarıda verilen isim adlı değişkeni esas alırsak şunu görürüz.

<b>B</b>	<b>a</b>	<b>t</b>	<b>u</b>	<b>r</b>	<b>a</b>	<b>l</b>	<b>p</b>		<b>D</b>	<b>i</b>	<b>n</b>	<b>ç</b>	<b>d</b>	<b>a</b>	<b>r</b>	<b>i</b>
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Karakter katarının 8. elemanı boşluktur. Yukarıda belirtilen isim adlı değişkende 12. eleman ‘c’ ile değiştirilebilir.

isim[12] = ‘c’;

Bu işlem sonrası ise isim adlı değişken aşağıdaki gibi olur.

<b>B</b>	<b>a</b>	<b>t</b>	<b>u</b>	<b>r</b>	<b>a</b>	<b>l</b>	<b>p</b>		<b>D</b>	<b>i</b>	<b>n</b>	<b>c</b>	<b>d</b>	<b>a</b>	<b>r</b>	<b>i</b>
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Aşağıdaki örnek programda yukarıda belirtilen karakter katarı işlemlerini gerçekleştirmektedir.

```
/*
*karakter_katar.cxx
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
* takdirde serbesttir:
*
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
*
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
```

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen  
\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.  
\*  
\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA  
\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
```

```
#include <locale>
```

```
#include <string>
```

```
using namespace std;
```

```
// Ana fonksiyon
```

```
int main()
```

```
{
```

```
    // Dil ayarları
```

```
    setlocale(LC_ALL, "tr_TR.UTF-8");
```

```
    // Değişken tanımlamaları
```

```
    string isim;
```

```

string karakterkatar1, karakterkatar2, karakterkatar3, karakterkatar4;
char k;

cout << "Merhaba Baturalp Dinçdari" << endl;
cout << "Sen Baturalp değilsin. Adın nedir? ";
getline(cin, isim);
cout << "Merhaba " << isim << endl;
karakterkatar1 = "Pearl Jam";

cout << "Sevdiğim rock gruplarından birisi " << karakterkatar1 << "\dir." << "Favori par-
çamda I'm Still Alive." << endl;

cout << "Senin sevdiğin müzik grubunun adı nedir? ";
getline(cin, karakterkatar2);

karakterkatar3 = karakterkatar2 + " bende dinlemek isterim./";

cout << "Sevdiğin müzik grubunun adı " << karakterkatar2 << " imiş." << karakterkatar3 <<
endl;

cout << "Bir sözcük gir ve enter tuşuna bas. ";

cin >> karakterkatar4;

cin.get(k);

karakterkatar4[1] ='A';

cout << "Yazdığınızın ikinci harfini değiştirdim: " << karakterkatar4 << endl;

return 0;
}

```

### 6.3.1.Karakter Katarı Üzerinde Gerçekleştirilen Diğer İşlemler

Aşağıdaki tabloda bazı sık kullanılan karakter katarı veri tipi üzerinde gerçekleştirilen işlemler gösterilmiştir. **karakter\_katarı\_değişken** ile temsil edilen değişken karakter katarı veri tipinde tanımlanmıştır. Bu veri tipi üzerinde gerçekleştirilen işlemlere ilişkin fonksiyon ise “.” sonra gelmektedir.

<b>Bazı string – karakter katarı fonksiyonları</b>	
karakter_katarı_değişken[indis]	“indis” ile belirtilmiş konumdaki karakter döndürülür.
karakter_katarı_değişken.at(indis)	“indis” ile belirtilmiş konumdaki karakter döndürülür.
karakter_katarı_değişken.length()	string::size_type değişkeninde barındırılan karakter katarının uzunluğunu döndürür.

karakter_katari_değişken.size()	string::size_type değişkeninde barındırılan karakter katarının uzunluğunu döndürür.
karakter_katari_değişken.empty()	karakter_katari_değişken adlı değişkende herhangi bir veri bulunmuyor ise <b>true – doğru</b> değerini aksi durumda ise <b>false – yanlış</b> değerini döndürür
karakter_katari_değişken.erase()	karakter_katari_değişken adlı değişkende barındırılan veriyi siler.
karakter_katari_değişken.clear()	karakter_katari_değişken adlı değişkende barındırılan veriyi siler.
karakter_katari_değişken.erase(indis, n)	karakter_katari_değişken adlı değişkende barındırılan veriyi, belirtilen indisten başlayarak n adedini siler.
karakter_katari_değişken.append(karakter_katari)	karakter_katari_değişken adlı değişkendeki verinin sonuna karakter_katari adlı veriyi ekler.
karakter_katari_değişken.append(n, karakter)	karakter_katari_değişken adlı değişkendeki verinin sonuna karakter veri tipinde tanımlanmış olan karakter adlı veriyi n adet yineleyerek ekler.
karakter_katari_değişken.insert(konum, karakter_katari)	karakter_katari_değişken adlı değişkendeki verinin belirtilen konumuna karakter_katari adlı değişkendeki karakter katarını ekler.
karakter_katari_değişken.insert(konum, n, karakter)	karakter_katari_değişken adlı değişkendeki verinin belirtilen konumuna karakter adlı değişkendeki karakteri n defa yineleyerek ekler.
karakter_katari_değişken.replece(konum, n, karakter_katari)	karakter_katari_değişken adlı değişkendeki verinin belirtilen konumundan başlayarak karakter_katari adlı değişkendeki karakter katarını yazar. Eğer karakter_katari adlı değişkendeki veri karakter_katari_değişkeni adlı değişkendeki dizinin kalan kısmından büyük ise yazılan veri sonucunda karakter_katari_değişken adlı değişkenin büyüğünü büyür.
karakter_katari_değişken.substr(konum, büyükük)	karakter_katari_değişken adlı değişkende belirtilen indisten başlayarak, belirtilen kadar karakteri döndürür. Eğer büyükük değeri belirtilen indisten itibaren sayıldığında kalan karakter sayısından büyük ise bu durumda karakter katarının kalan tüm elemanlarını döndürür.

karakter_katari_değişken.find(karakter_katarı)	karakter_katari_değişken adlı değişkende barındırılan karakter katarı içerisinde, karakter_katarı ile belirtilen elemanların ilk bulunduğu konumu döndürür. Eğer bulunmaz ise karakter_katari_değişken adlı değişkenin büyülüğünü döndürür.
karakter_katari_değişken.find(karakter_katarı, konum)	karakter_katari_değişken adlı değişkende barındırılan karakter katarı içerisinde, karakter_katarı ile belirtilen elemanları belirtilen konumdan itibaren aramaya başlar.
karakter_katari_değişken.find_first_of(karakter_katarı, konum)	karakter_katari_değişken adlı değişkende barındırılan karakter katarı içerisinde, karakter_katarı ile belirtilen elemanları belirtilen konumdan itibaren aramaya başlar ama bu kez karakter_katarı değişkenindeki karakterlerden ilk eşleşeni bulur.
karakter_katari_değişken.find_first_not_of(karakter_katarı, konum)	karakter_katari_değişken adlı değişkende barındırılan karakter katarı içerisinde, karakter_katarı ile belirtilen elemanları belirtilen konumdan itibaren aramaya başlar ama bu kez karakter_katarı değişkenindeki karakterlerden olmayanlar arasından ilk eşleşeni bulur.
karakter_katari_değişken.swap(karakter_katarı)	Karakter_katarı_değişkeni adlı değişken ile karakter_katarı adlı değişkenin içeriği yer değiştir.

Eğer bir karakter katarına ait olan karakter sayısının büyülüğü yukarıdaki fonksiyonlar dışında doğrudan standart kütüphane ile döndürmek istenirse “**string:: size\_type**” fonksiyonu kullanılır. Bu fonksiyon işaretsiz tam sayı veri tipindedir. Genellikle de karakter katarının büyülüğünü herhangi bir veri tipi dönüşümüne gerek olmadan döndürmek için kullanılır. Aynı şekilde katarı veri tipinin üst sınırının belirlenmesi için “**string::npos**” fonksiyonu kullanılır.

Aşağıdaki örnek programda yukarıda açıklanmış olan fonksiyonlar kullanılmıştır.

```
/*
*karakterkatar_fonksiyon00.cxx
*
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
```

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtım; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

#include <iostream>

#include <locale>

#include <string>

using namespace std;

```

// Ana fonksiyon

int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");

    // Değişken tanımlamaları
    string isim = "Tayfur";
    string hava_durumu = "Her zamanki gibi, ne olsun.";
    string karakter_katar1 = "";
    string karakter_katar2 = "Bilgisayar Programcılığı";
    string karakter_katar3 = isim + " " + karakter_katar2 + " öğrencisidir.";
    string::size_type uzunluk;

    //Fonksiyonların kullanımı
    cout << "isim: " << isim << endl;
    cout << karakter_katar1.empty() << endl;
    cout << karakter_katar2.empty() << endl;
    cout << karakter_katar3.empty() << endl;
    cout << karakter_katar3 << endl;
    cout << "\"isim\" adlı karakter katarı temizlendikten sonra: " << isim.erase() << endl;
    cout << "karakter_katar2 adlı değişkende son 14 karakter silindikten sonra: " << karakter_katar2.erase(10,16) << endl;
    uzunluk = hava_durumu.length();
    cout << "hava_durumu adlı değişkeninin karakter olarak büyütüğü: " <<
static_cast<unsigned int>(uzunluk) << endl;
    uzunluk = karakter_katar2.length();
    cout << "karakter_katar2 değişkeninin karakter olarak büyütüğü: " << static_cast<unsigned int>(uzunluk) << endl;
    cout << "karakter_katar3 adlı değişkenin karakter olarak büyütüğü: " <<
karakter_katar3.size() << endl;
    cout << static_cast<unsigned int>(string::npos) << endl;
    return 0;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./karakterkatar_fonksiyon00
```

isim: Tayfur

```
1  
0  
0
```

Tayfur Bilgisayar Programcılığı öğrencisidir.

"isim" adlı karakter katarı temizlendikten sonra:

karakter\_katar2 adlı değişkende son 14 karakter silindikten sonra: Bilgisayarı

hava\_durumu adlı değişkeninin karakter olarak büyülüğu: 27

karakter\_katar2 değişkeninin karakter olarak büyülüğu: 12

karakter\_katar3 adlı değişkenin karakter olarak büyülüğu: 51

4294967295

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Son satırlarda karakter katarlarının uzunlıklarının döndürülmesinde kullanılan fonksiyonların özelliğine göre elde edilecek olan değerler yukarıda belirtilen işaretsiz tam sayı değerinin üst sınırı kadar olabilir veya daha büyük sayılar döndürülebilir. Bunun nedeni bazı derleyicilerin, söz konusu **string::size\_type** ve **string::npos** fonksiyonlarının makine diline çevrilmede farklı şekilde işlem yapmalarının önüne geçilmesi içindir. Veri tipi dönüşümü yapmadan doğrudan fonksiyonun döndürdüğü değerin kullanılması sonucunda elde edilecek olan çıktı ve sonuçların değerlendirilmesi öğrenciye bireysel çalışma olarak bırakılmıştır.

Aşağıdaki programda bir karakter katarı içerisinde arama yapılmasını sağlayan **find** fonksiyonu kullanılmıştır.

```
/*  
*karakterkatar_fonksiyon01.cxx  
  
*  
  
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>  
* *  
  
* Her hakkı saklıdır.  
  
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak  
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği  
* takdirde serbesttir:
```

\*

\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
```

```
#include <locale>
```

```
#include <string>
```

```
using namespace std;
```

```
// Ana fonksiyon
```

```
int main()
```

```

{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");

    // Değişken tanımlamaları
    string metin = "El feneri alt katta koridordaki dolapta, biriniz inip getiriverin./";

    string aranan;

    string::size_type konum;

    // Fonksiyonların kullanımı
    cout << "Üzerinde işlem yapılacak olan metin: " << metin << endl;

    cout << "\"El\" sözcüğünün tümce içerisindeki konumu: " << static_cast<unsigned int>(metin.find("El"))<< endl;

    cout << "\"fener\" sözcüğünün tümce içerisindeki konumu: " << static_cast<unsigned int>(metin.find("fener"))<< endl;

    cout << "\"alt\" sözcüğünün tümce içerisindeki konumu: " << static_cast<unsigned int>(metin.find("alt"))<< endl;

    cout << "\"kat\" sözcüğünün tümce içerisindeki konumu: " << static_cast<unsigned int>(metin.find("kat"))<< endl;

    cout << "\"koridor\" sözcüğünün tümce içerisindeki konumu: " << static_cast<unsigned int>(metin.find("koridor"))<< endl;

    cout << "\"dolap\" sözcüğünün tümce içerisindeki konumu: " << static_cast<unsigned int>(metin.find("dolap"))<< endl;

    cout << "\"bir\" sözcüğünün tümce içerisindeki konumu: " << static_cast<unsigned int>(metin.find("bir"))<< endl;

    cout << "\"inip\" sözcüğünün tümce içerisindeki konumu: " << static_cast<unsigned int>(metin.find("inip"))<< endl;

    cout << "\"getir\" sözcüğünün tümce içerisindeki konumu: " << static_cast<unsigned int>(metin.find("getir"))<< endl;

    cout << "Aranacak olan metini giriniz: ";

    getline(cin, aranan);

    konum = static_cast<unsigned int>(metin.find(aranan));

    cout << "Aranan metinin konumu: " << konum << endl;

    return 0;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./karakterkatar_fonksiyon01
```

Üzerinde işlem yapılacak olan metin: El feneri alt katta koridordaki dolapta, biriniz inip getiriverin.

"El" sözcüğünün tümce içerisindeki konumu: 0

"fener" sözcüğünün tümce içerisindeki konumu: 3

"alt" sözcüğünün tümce içerisindeki konumu: 10

"kat" sözcüğünün tümce içerisindeki konumu: 14

"koridor" sözcüğünün tümce içerisindeki konumu: 20

"dolap" sözcüğünün tümce içerisindeki konumu: 32

"bir" sözcüğünün tümce içerisindeki konumu: 41

"inip" sözcüğünün tümce içerisindeki konumu: 49

"getir" sözcüğünün tümce içerisindeki konumu: 54

Aranacak olan metini giriniz: araba

Aranan metinin konumu: 4294967295

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Aşağıdaki programda bir karakter katarı içerisinde **insert** ve **replace** fonksiyonları fonksiyonu kullanılmıştır.

```
/*
*karakterkatar_fonksiyon02.cxx
*
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
* takdirde serbesttir:
*
* Kaynak kodun yeniden dağıtımlı; yukarıdaki telif hakkı uyarısını,
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
*
```

```

* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

/*
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
* SORUMLU DEĞİLLERDİR.

*/
/*
#include <iostream>
#include <locale>
#include <string>
using namespace std;
// Ana fonksiyon
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişken tanımlamaları
}

```

```

string metin1 = "Hava bulutlu.";
string metin2 = "Hava soğuk. ";
string metin3 = "Samim BTP101 dersini alıyor.";
string isim1 = "Baturalp";
//Fonksiyonların kullanımı
cout << "Birinci metin budur: " << metin1 << endl;
cout << "İkinci metin budur: " << metin2 << endl;
metin1.insert(0, metin2);
cout << "Birinci metine ekleme yapıldıktan sonra: " << metin1 << endl;
cout << "Üçüncü metin budur: " << metin3 << endl;
cout << "Bir isim, " << isim1 << endl;
metin3.replace(0, 5, isim1);
cout << "Üçüncü metinde değişiklik yapıldıktan sonra: " << metin3 << endl;
return 0;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./karakterkatar_fonksiyon02
```

Birinci metin budur: Hava bulutlu.

İkinci metin budur: Hava soğuk.

Birinci metine ekleme yapıldıktan sonra: Hava soğuk. Hava bulutlu.

Üçüncü metin budur: Samim BTP101 dersini alıyor.

Bir isim, Baturalp

Üçüncü metinde değişiklik yapıldıktan sonra: Baturalp BTP101 dersini alıyor.

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Aşağıdaki programda bir karakter katarı içerisinde **substr** fonksiyonları fonksiyonu kullanılmıştır.

```
/*
*karakterkatar_fonksiyon03.cxx
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
```

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

#include <iostream>

```

#include <locale>
#include <string>
using namespace std;
// Ana fonksiyon
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişken tanımlamaları
    string metin = "El feneri alt katta koridordaki dolapta, biriniz inip getiriverin.";
    string karakter_katar = "sin";
    //Fonksiyonların kullanımı
    cout << "Bir metinden yol çıkarak yaratılan dialog örneği." << endl << endl;
    cout << "Kaynak metin: " << metin << endl << endl;
    cout << "Soru: El feneri nerede?" << endl;
    cout << "Yanıt: " << metin.substr(0,19) << endl;
    cout << "Soru: El feneri alt katta nerede?"<< endl;
    cout << "Yanıt: " << metin.substr(0,9) << " " << metin.substr(20, 19) << endl;
    cout << "Soru: Oraya baktım ama yoktu."<< endl;
    cout << "Yanıt: " << metin.substr(0,9) << "ni" << metin.substr(40, 26) << endl;
    cout << "Soru: Orada aradım ama bulamadım." << endl;
    cout << "Yanıt: " << metin.substr(0,10) << metin.substr(20,19) << metin.substr(40,8) <<
metin.substr(53,6) << karakter_katar << endl;
    cout << "Soru: Bu katta dolap mı var?" << endl;
    cout << "Yanıt: " << metin << endl;
    return 0;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./karakterkatar_fonksiyon03
```

Bir metinden yol çıkarak yaratılan dialog örneği.

```
Kaynak metin: El feneri alt katta koridordaki dolapta, biriniz inip getiriverin.
```

Soru: El feneri nerede?

Yanıt: El feneri alt katta

Soru: El feneri alt katta nerede?

Yanıt: El feneri koridordaki dolapta

Soru: Oraya baktım ama yoktu.

Yanıt: El fenerini biriniz inip getiriverin.

Soru: Orada aradım ama bulamadım.

Yanıt: El feneri koridordaki dolapta biriniz getirsün

Soru: Bu katta dolap mı var?

Yanıt: El feneri alt katta koridordaki dolapta, biriniz inip getiriverin.

[goksin@tardis ~/projeler/C++\_Notlar/]\$

Yukarıdakiler ek olarak diğer karakter katarı fonksiyonları aşağıdaki tabloda verilmiştir.

### **Diger string – karakter katarı fonksiyonları**

getline(#girdi\_akım\_değişkeni, karakter\_katar\_değişkeni);  
girdi\_akım\_değişkeni, istream veya ifstream değişkeni olurken, yeni satır karakterine kadar okunan tüm karakterler karakter\_katar\_değişkeninde barındırılır. Yeni satır karakteri okunur ama göz ardı edilir ve bu fonksiyonun döndürdüğü değer dikkate alınmaz.

karakter\_katar\_değişkeni.append(karakter\_katar, n); Karakter katarı bir karakter dizisi ise, ilk n adet karakter karakter\_katar değişkeninin sonuna eklenir.

karakter\_katar\_değişken.c\_str() karakter\_katar\_değişken barındırılan ve null terminator kullanılarak sonlandırılmış olan bir C tipi karakter kata-rının bellekteki taban adresini döndürür..

karakter\_katar\_değişken.capacity() karakter\_kata\_değişken adlı değişkenin bellekte tükettiği alanın büyüğünü verir.

karakter\_kata\_değişken.erase(konum); karakter\_katar\_değişken adlı değişkendeki verinin belirtilen konumdan başlayarak sonraki tüm karakterleri siler. Burada konum string::size\_type için bir parametredir.

karakter\_katar\_değişken.resize(n, karakter); karakter\_katar\_değişken adlı değişkenin bellekte tükettiği alanı n kadar yapar. Eğer n ilk durumdaki bellek miktarından büyük ise bu alana sığan veri kalır ve diğerleri erişilemez olur. Aksi durumda ise boş kalan alan olacaktır ve bu alan karakter değişkeni ile belirtilen

veri ile doldurulur.

## 7.Diziler ve string Veri Tipi

Önceki bölümlerde C++ Programlama Dili'nin yapısında basit, yapılandırılmış ve işaretçiler olarak adlandırılan üç adet veri tipi olduğunu belirtmiştik. Bu ve işleyen bir kaç bölüm boyunca yapılandırılmış veri tiplerini inceleyeceğiz.

Önceki bölümlerde basit veri tipleri (bazı kaynaklarda primitifler olarak da tanımlanır) yapısı gereği tek bir değişken ile ilişkilendirilip tek bir değeri barındırmakta idi. **Yapılandırılmış veri tipleri – structured data types** – ise basit veri tiplerinin aksine yapılandırılmış veri tipleri birden çok veriyi yapısında barındırır. Yapılandırılmış veri tiplerinin ilk örneği **dizilerdir – arrays**. Diziler teknik olarak aynı veri tipin tanımlanmış olan çok sayıdaki değerin bellek üzerinde ardışık adresler üzerinde barındırılmasını sağlayan bir veri yapısıdır.

Bu tanımdan da anlaşılacağı üzere bir dizi sadece belirli bir veri tipindeki birden çok sayıdaki değişkende barındırılacak olan veriyi tek bir tanımlama yapılarak barındırılmasını sağlar. Önceki bölümlerde verilen bazı örneklerle burada yeniden dönmekte yarar var. Örneğin 20 adet çok sayıdaki tam sayının ortalamasını bulmak için farklı algoritmalar kullanılabilir. İlk algoritma örneği aşağıda verilmiştir:

- 1 Başla
- 2 Ortalama, toplam, sayaç ve sayı adlı değişkenleri tanımla.
- 3 Ortalama, toplam ve sayaç adı değişkenin değerleri sıfır olsun.
- 4 Sayıyı oku ve sayı adlı değişken ata.
- 5 Toplam = toplam + sayı hesapla.
- 6 Sayaç = sayaç + 1 hesapla
- 7 Ortalama = (ortalama+ toplam)/sayaç hesapla.
- 8 Sayaç = 20 kontrol et.
  - 8.1 Eğer sayaç 20'e eşit ise 8 adıma git
  - 8.2 Eğer sayaç 20'e eşit değilse 4 adıma git.
- 9 Ortalamayı yazdır
- 10 Son

Bu algoritma sadece ortalama sonucunu verecektir. Bu algoritmayı ortalamadan başka girilen en büyük sayı ve en küçük sayıyı bulacak şekilde değiştirmek istersek bu durumda algoritmaya en büyük ve ne küçük adlı iki değişken daha eklemek gerekecektir. Bu aşamadan sonra algoritmayı en büyükten başlayarak en küçüğe doğru girilen sayıları sırlamayacak şekilde de geliştirmek isteyecek olursak bu durumda en az yirmibir adet daha değişken tanımlamak gerekecektir.

Bu ve benzeri bazı problemlerin çözümünde temel veri tiplerinin kullanılması ile yapılacak olan programlarda algoritmalar karmaşıklaşmaktadır. Bu karmaşıklığın nedeni değişken sayılarının artması, bu artışa bağlı olarak da yapılacak olan işlemlerin de artmaka oluşudur. Ayrıca yapılacak

işlemler de sadece aritmetik işlemler olmakla kalmayıp aynı zamanda döngüler ve karar yapılarının da kullanılmasını gerektirecektir. Bu nedenle basit veri tipleri karmaşık programların çözümü için uygun olmamaktadır.

## 7.1.Diziler

Bir dizi aynı veri tipinde olan birden çok sayıdaki veriyi barındıraman ve büyülüğu de yapısında bulunan elaman sayısının toplam büyüklüğüne eşit olan bir veri tipidir. Bir dizi tanımlandığında bellekte ardışık adresler kullanılarak dizi tanımlanır. Dizinin elemanlarının tamamı aynı veri tipinde olduğu için her bir tekil elemanın bellekte kapladığı alan sabittir. Bu nedenle de bir dizinin elemanlarına erişim için gerekli olan işlem sayısı ve işlem süresi sabittir ve değişmez. Bir bellek alanı üzerinde ardışık bellek adreslerinde değerleri barındıran dizilere de **tek boyutlu diziler – one dimensinal arrays** – adı verilir.

Bir dizini tanımlanabilmesi için aşağıdaki bildirimin yapılması yeterlidir.

**veri\_tipi dizi\_adı [işretsiz\_tamsayı olarak eleman sayısı];**

Dizinin barındıracağı veri tipinin tanımlanmış, dizinin adının belirtilmesi ve elaman sayısını belirten işaretsiz tam sayı değerinin belirtilmesi gereklidir. Bu tanımlama ile bellekte dizi elemanları için yer ayrılacaktır. Aşağıdaki örnek tanımlamada 10 elamanlı bir dizi tanımlanmıştır.

**int elemanlar [10];**

Bu bildirim ile on adet tam sayı veri tipinde elemana sahip olan elemanlar adlı dizi oluşturulmuş olacaktır. Diziler üzerindeki işlemlerin gösteriminde kolaylık olması için diziler aşağıdaki gibi gösterilir.

elemanlar											
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	

Bu şekildeki gösterim ile bir dizin elemanları ile dizi içerisindeki konumu belirtilmiş olur.

### 7.1.1.Dizi Elemanlarına Erişim

Bir dizinin elemanları ardışık bellek adreslerinde barındırıldığı için bir dizi elemanına erişilebilmesi için dizi elemanın bellek adresinin başlangıç adres ise bu başlangıç adresine olan uzaklığının bilinmesi gereklidir. Bu işlemler programlama dilinin yapısındaki komutlar ile otomatik olarak gerçekleştiriliyor. Programcının yapması gereken ise sadece erişmek istediği dizi elemanın dizi içerisindeki konumunu yani **indisini** tanımlamasıdır. Buda aşağıdaki bildirim ile yapılır.

**dizi\_adı [indis]**

Dizideki elemanların sayısının işaretsiz tam sayı olduğu dikkate alındığında indis bildiriminin de aynı şekilde işaretsiz tam sayı olması gereği görülecektir. C++ Programlama Dili yapısında dizilerde barındırlan veriye ulaşmak için dizi adı ile birlikte kullanılan **[ ] dizi indis operatörü – array subscripting operator** olarak adlandırılır. C++ programlama Dili'nde dizi indis operatörünün başlangıç değeri sıfırdır.

Bir dizinin tanımlanmasında yukarıda belirtilen bildirim yapısı kullanılır. Dizinin elemanlarına erişmek için kullanılan erişim bildirimi aynı zamanda bir dizinin belirtilen sıradaki elmanın erişilmesini sağlarken aynı zamanda dizinin belirtilen indis numarasındaki elemanı üzerinde işlem yapılmasını sağlar.

Bir dizi tanımlanırken veri tipi, dizinin adı ve elaman sayı ile yapılan tanımlamada dizinin herhangi bir elemanı tanımlanmamış olacaktır. Bu durumda dizinin elamanlarına erişim için kullanılan bildirim dizisi eleman atamak için de kullanılabilir. Yukarıda örnek verile diziye geri dönelim.

<b>elemanlar</b>											
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	

Bu dizinin herhangi bir elmanı bulunmamaktadır. Aşağıdaki bildirimler ile dizinin belirtilen indise sahip elemanlarına atama yapabiliriz.

elemanlar [1] = 2;

elemanlar [3] = 3;

elemanlar [4] = elemanlar[1] + elemanlar [3];

Bu bildirimlerden sonra dizinin durumu aşağıdaki gibi olacaktır.

<b>elemanlar</b>		2		3	5						
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	

Yukarıdaki bildirimlerden görüleceği üzere dizi elamanları aynı veri tipinde olduğundan bu veri tipi üzerinde yapılabilecek olan tüm işlemler de belirlidir. Dolayısı ile de bir dizi elamanı sayısal veri tipinde ise diğer dizi elamanları da var olan dizi elamanları üzerinde yapılabilecek olan işlemler ile tanımlanabilir. Dizilerin elamanlarının tanımlanması için dizi elamanlarının her birinin tekil olarak tanımlanması elaman sayısının çok olduğu durumlarda verimli bir çözüm olmayacağı.

Bir matematik problemi için tekil ve çift sayılardan yirmi adedini barındıran iki adet dizi gerektiğini var sayalım. Bu iki dizi için başlangıç elemanları tanımlanmış ve geri kalan on dokuz adet dizi elamanının ise program tarafından hesaplanarak dizideki ilgili yerlerine atanması yapılacak olsun. Çift ayılar için ilk elaman 2 ve tek sayılar için de ilk eleman 1 olarak verilsin. Bu dizi elemanları programın başında tanımlanarak ilk atamaların yapılmasını sağlayalım.

unsigned int tek [20];

unsigned int cift [20];

tek[0] = 1;

cift [0] = 2;

Bu bildirimlerden sonra dizinin durumu aşağıdaki gibi olacaktır.

<b>tek</b>	1																		
<b>cift</b>	2																		

Bu aşamadan sonra diğer elamanların tanımlanıp diziye atanması için bir döngü kullanılabilir.

```
unsigned int tek [20];  
unsigned int cift [20];  
tek[0] = 1;  
cift [0] = 2;  
for (i = 1; i; i <=19)  
{  
    tek [i] = i * 2 +1;  
    cift [i] = i * 2;  
}
```

Döngünün çalıştırılıp sonlanmasından sonra dizilerin son durumu aşağıdaki gibi olacaktır.

<b>tek</b>	1	3	5	7	9	11	13	14	15	16	17	19	21	23	25	27	29	31	33	35
<b>cift</b>	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]

Yukarıdaki örneklerden de görüleceği üzere bir dizinin tanımlanması için gerekli olan parametrelerin önceden belirlenmiş olması gereklidir. Dizinin elaman sayının ne kadar olacağının bilinmediği durumlarda ise  **işaretçiler – pointer** – kullanılarak diziler oluşturulabilir. Bu konu daha sonraki bölümlerde ele alınacağı için burada yer verilmemiştir.

### 7.1.2.Tek Boyutlu Diziler Üzerinde İşlemler

Tek boyutlu dizilerde sık yapılan işlemler, dizin tanımlanması, diziye eleman atanması, dizide barındırılan verinin okunması, dizi elemanlarının kendi aralarında sırlanması ve bu elamanlar arasından en büyük ve en küçük olanlarının bulunması gibi işlemlerdir. Eğer dizide barındırılan elemanlar sayısal veri tiplerinden herhangi birisi ise bu durumda aritmetik ve mantıksal işlemler de diziler üzerinde kolaylıkla gerçekleştirilebilir. Aşağıdaki örnek problemde tek boyutlu diziler kullanılmıştır. Dizide elemanların en büyüğü ile en küçüğü bulunmuştur.

**Örnek Problem:** Bir perakende satış zincirine ait olan mağazaların yıl boyunca yaptıkları satışlara ait olan ciro değerleri bilinmektedir. Bu zincirdeki mağazaların yıllık ciro toplamı, en yüksek ve en düşük ciro miktarları ile ortalama cironun büyülüğünün belirlenmesi istenmektedir. Mağazaların isimleri yerine sıra numaraları kullanılacaktır. Her bir mağazaya ait olan veri klavyeden girilecektir. Veri giriş işlemi bittikten sonra girilen verilerin kullanıcı tarafından doğrulanması istenecektir. Hatalı girilen veri var ise bu düzeltilecektir.

**Problemin çözümü:** Mağazaların sayısı bilinmektedir. Veri girişinde hatalı veri girişi olup olmadığı kontrol edilecektir. Bu nedenle de girilen veriler bir dizide saklanacaktır. Mağazaların ciro ortalamasının bulunması için tüm mağazalara ait olan ciro verisi toplanıp mağaza sayısına bölünecektir. En yüksek cironun bulunması için dizideki elamanların en yüksek değer sahip olanı bulunmalıdır.

Bunun içinde dizideki her bir elaman sırası ile okunacaktır. En büyük değeri bulmak için ilk okunan veri en yüksek değer olarak kabul edilecektir. Bu verinin indis değeri de en yüksek ciorya sahip mağaza olarak kabul edilecektir. Sırası ile okuna her bir veri en yüksek değer olarak kabul edilen veri ile karşılaşılacaktır. Eğer okunan veri en yüksek kabul edilen veriden büyük ise en yüksek ciro yeni okunan veri olarak kabul edilecek ve önceden saklanan veri ile değiştirilecektir. En büyük olarak okunan değer olarak yeni değerin okunduğu sıraya ait indis değeri kayıt edilecektir. Bu işlem tüm verilerin karşılaştırması yapılmaya kadar devam edecektir. En düşük ciro içinde yanı işlem basamakları bu sefer en büyük olanı değil en küçük olanı bulmak için aynı şekilde ama karşılaşılan iki değerden küçük olanı belirleyecek şekilde düzenlenecektir.

Programın kaynak kodu aşağıdaki gibidir:

```
/*
 * ciro.cxx
 *
 *
 * Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
```

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
#include <locale>
#include <iomanip>
using namespace std;
// Ana fonksiyon
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişken tanımlamaları
    char kontrol;
    unsigned int en_yuksek_sira = 0, en_dusuk_sira = 0;
    int i=0, j=0, k=0, l=0;
    double en_yuksek_ciro, en_dusuk_ciro;
    double ciro_toplam = 0.0, ciro_ortalama = 0.0;
    // Dizinin tanımlanması
    double ciro [10];
    // Dizinin etkinleştirilmesi
    for (i=0; i<10; i++)
```

```

{
    ciro[i]=0.0;
}

// Veri girişi

for (j=0; j<10; j++)
{
    cout << j+1 << ". mağazanın cirosunu giriniz: ";
    cin >> ciro[j];
}

// Girilen verilerin kontrolü

cout << "Giriş yapılan verileri kontrol ediniz. Hatalı girilen veri var ise düzeltmek için yanıt satırına Y veya y girip enter tuşuna basınız aksi durumda ise E veya e girip enter tuşuna basınız. Hatalı veriler için sizden doğru olan veriyi girip enter tuşuna basmanız istenecektir.." << endl;

cout << fixed << setprecision(2);

for (k=0; k < 10; k++)
{
    cout << k+1 << ". mağazanın cirosu: " << ciro[k] << " TL'dir." << endl;
    cout << "Girilen veri doğru mudur? Doğru ise E\|e, yanlış ise Y\|y girip enter tuşuna basınız: ";
    cin >> kontrol;

    if ((kontrol != 'Y') && (kontrol != 'y') && (kontrol !='E') && (kontrol != 'e'))
    {
        cout << "Hatalı giriş yapıldı program sonlanıyor.";
        return 1;
    }
    else if ((kontrol == 'Y') || (kontrol == 'y'))
    {
        cout << k+1 << ". mağazanın cirosunu giriniz: ";
        cin >> ciro[k];
    }
}

```

```

// tek döngüde en büyük ciro, toplam ve ortalama belirlenir.

en_dusuk_ciro = ciro[0];
en_yuksek_ciro = ciro[0];
for (l=0; l<10; l++)
{
    ciro_toplam = ciro_toplam + ciro[l];
    if (ciro[l] > en_yuksek_ciro)
    {
        en_yuksek_ciro = ciro[l];
        en_yuksek_sira = l;
    }
    if (ciro[l] <= en_dusuk_ciro)
    {
        en_dusuk_ciro = ciro[l];
        en_dusuk_sira = l;
    }
}
ciro_ortalama = ciro_toplam / 10.0;
cout << "Yıllık ciro ortalaması: " << ciro_ortalama << endl;
cout << "En yüksek ciro: " << en_yuksek_ciro << endl;
cout << "En yük ciroya sahip olan mağaza: " << en_yuksek_sira + 1 << endl;
cout << "En düşük ciro: " << en_dusuk_ciro << endl;
cout << "En düşük ciroya sahip olan mağaza: " << en_dusuk_sira + 1 << endl;
return 0;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./ciro
```

1. Mağazanın cirosunu giriniz: 11
2. Mağazanın cirosunu giriniz: 22
3. Mağazanın cirosunu giriniz: 33
4. Mağazanın cirosunu giriniz: 44

5. Mağazanın cirosunu giriniz: 55
6. Mağazanın cirosunu giriniz: 66
7. Mağazanın cirosunu giriniz: 77
8. Mağazanın cirosunu giriniz: 88
9. Mağazanın cirosunu giriniz: 99
10. Mağazanın cirosunu giriniz: 10

Girişi yapılan verileri kontrol ediniz. Hatalı girilen veri var ise düzeltmek için yanıt satırına Y veya y girip enter tuşuna basınız aksi durumda ise E veya e girip enter tuşuna basınız. Hatalı veriler için sizden doğru olan veriyi girip enter tuşuna basmanız istenecektir.

1. mağazanın cirosu: 11.00 TL'dir.

Girilen veri doğru mudur? Doğru ise E\e, yanlış ise Y\y girip enter tuşuna basınız: e

2. mağazanın cirosu: 22.00 TL'dir.

Girilen veri doğru mudur? Doğru ise E\e, yanlış ise Y\y girip enter tuşuna basınız: e

3. mağazanın cirosu: 33.00 TL'dir.

Girilen veri doğru mudur? Doğru ise E\e, yanlış ise Y\y girip enter tuşuna basınız: e

4. mağazanın cirosu: 44.00 TL'dir.

Girilen veri doğru mudur? Doğru ise E\e, yanlış ise Y\y girip enter tuşuna basınız: e

5. mağazanın cirosu: 55.00 TL'dir.

Girilen veri doğru mudur? Doğru ise E\e, yanlış ise Y\y girip enter tuşuna basınız: e

6. mağazanın cirosu: 66.00 TL'dir.

Girilen veri doğru mudur? Doğru ise E\e, yanlış ise Y\y girip enter tuşuna basınız: e

7. mağazanın cirosu: 77.00 TL'dir.

Girilen veri doğru mudur? Doğru ise E\e, yanlış ise Y\y girip enter tuşuna basınız: e

8. mağazanın cirosu: 88.00 TL'dir.

Girilen veri doğru mudur? Doğru ise E\e, yanlış ise Y\y girip enter tuşuna basınız: 9. mağazanın cirosu: 99.00 TL'dir.

Girilen veri doğru mudur? Doğru ise E\e, yanlış ise Y\y girip enter tuşuna basınız: e

10. mağazanın cirosu: 10.00 TL'dir.

Girilen veri doğru mudur? Doğru ise E\e, yanlış ise Y\y girip enter tuşuna basınız: Yıllık ciro ortalaması: 50.50

En yüksek ciro: 99.00

En yük ciroya sahip olan mağaza: 9

```
En düşük ciro: 10.00
```

```
En düşük ciroya sahip olan mağaza: 10
```

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

### 7.1.3.Dizilerde Sınır Kontrolü

Diziler özelikleri gereği ilk elamanın indis değeri her zaman için sıfırdır. Son elamanın indis değeri ise dizinin eleman sayısının bir eksigidir. Bunun anlamı dizinin son elemanın indis numarasının her zaman için dizinin elaman sayısından bir eksik olacağıdır.

Buna göre bir dizide indis değerleri  $0 \leq \text{indis} \leq \text{eleman\_sayısı} - 1$  aralığında olacaktır. Programlama dilleri programcıların ne yaptığını bildiği varsayılarak geliştirilir. Bu nedenle programcının bazı hataları yapmayacağı düşünülür. Örneğin dizinin eleman sayısının dikkate alınarak diziler üzerinde işlemler yapılacağı, programcının bu sınırlar konusunda dikkatli davranışları düşünülür. Bu nedenle de dizilerin sınır değerlerinin yani  $0$  ile  $\text{eleman\_sayısı} - 1$  aralığının dışına çıkmayacağı kabul edilir.

Bellek korumasına sahip olan işletim sistemlerinde bir dizinin sınır değerlerinin dışındaki kalan bellek alanları üzerinden okuma yapılmasına neden olan hatalı yazılmış programlarda sınır koruması neden ile program çalışmayacak veya işletim sistemi hata döndürülecektir. Bu nedenle bir dizi üzerinde işlem yapılacak ise programcının dizinin sınır değerlerinin aşılmasına neden olacak olan hatalı kod yazımı eylemlerinden kaçınması gereklidir.

Dizinin alt ve üst sınırları içerisinde kalan işlemler her zaman **in bounds – sınır dahilinde** olarak tanımlanırken sınırların dışına taşan işlemlerde her zaman **out of bounds – sınırlar dışında** olarak değerlendirilir.

### 7.1.4.Dizilerin Etkinleştirilmesi

Yukarıdaki programda dizi kullanılmış ve dizinin tanımlanmasının ardından bir döngü ile dizinin elamanlarının ilk değerleri atanarak dizi etkinleştirilmiştir. Burada dizileri etkinleştirilmesi işlemi aslında bir C++ programında değişkenlerin etkinleştirilmesi için izlenen yöntemlerden farklı değildir.

Dizinin tanımlanması sırasında dizinin elaman sayısının belirtilmesine gerek duyulmadan da dizi tanımlanabilir. Aşağıdaki örnek bildirimde dizinin elaman sayısı verilmemiş ancak dizinin elemanları tanımlanmış olduğu için dizinin tanımlanması ve etkinleştirilmesi işlemi gerçekleşmiştir.

```
double olcumler [] = {12.3, 15.6, 16.8, 16.9, 17.1, 17.4, 17.7, 18.2, 18.8};
```

Aynı bildirim ise dizinin elaman sayısının ve elamanlarının da tanımlanması ile yapılabilir.

```
double olcumler [9] = {12.3, 15.6, 16.8, 16.9, 17.1, 17.4, 17.7, 18.2, 18.8};
```

Yukarıdaki ikinci örnek, programcılık açısından bakıldığından daha doğru bir eylemdir. Dizilerin elaman sayısının tanımlanması konusunda önceki bölümde yapılan uyarılara dikkate alınarak dizi tanımlanmıştır. Böylece dizinin tanımlanması ile etkinleştirilmesi işlemleri tek bildirimde yapılmış olmaktadır.

Öte yandan dizinin elemanları da gerekiyorsa dizi tanımlandıktan sonra indis numaraları kullanılarak tanımlanabilir. Aşağıdaki bildirimde bu gösterilmiştir.

```
double olcumler [9];  
double olcumler [0] = 12.3;  
double olcumler [1] = 15.6;  
double olcumler [2] = 16.8;  
double olcumler [3] = 16.9  
double olcumler [4] = 17.1  
double olcumler [5] = 17.4;  
double olcumler [6] = 17.7;  
double olcumler [7] = 18.2;  
double olcumler [8] = 18.8;
```

### 7.1.5.Dizilerin Kısmental Etkinleştirilmesi

Dizilerin etkinleştirilmesi için tüm elamanlarının dizi tanımlandığında veya bir döngü yardımı ile elamanların tanımlanarak etkinleştirilmesi bir zorunluluk değildir. Dizinin elamanlarının bir bölümü tanımlanabilir ve bu tanımlama ile tanımlanmış olan elamanlar etkinleştirilebilir. Bu tanımlama biçimine **dizinin tanımlanırken kısmental etkinleştirilmesi – partial initialization of an array during declaration** adı verilir.

Dizinin kısmental etkinleştirilmesine ilişkin örnekler aşağıda verilmiştir.

```
int liste [10] = {0};
```

Bu tanımlama biçiminde dizinin veri tipi, adı ve eleman sayısı tanımlanmış ve ardından tüm değerlerinin sıfır eşit olduğu belirtilmiştir. Bu şekilde yapılan tanımlamada dizi elamanlarının tümüne aynı değer atanır. Eğer dizi elamanlarının sadece kısmental tanımlanması yeterli ise ilk indisten başlayarak belirli bir sayıdaki elemanı tanımlanabilir. Yukarıdaki örnek aşağıdaki gibi de tanımlanabilir.

```
int liste [10] = {1, 2, 3};
```

Bu durumda dizinin veri tipi, adı ve eleman sayısı ile ilk üç elemanı tanımlanıp etkinleştirilmiştir. Diğer dizi elamanları ise sıfır değerine sahiptir.

Önceki örneklerde olduğu gibi eğer dizinin veri tipi, dizinin adı belirtilip dizi elamanları tanımlanırsa, bu durumda dizi sadece tanımlanan elaman sayısı kadar elamanlı bir dizi olarak tanımlanıp etkinleştirilir. Aşağıdaki örnek bu durumu göstermektedir.

```
int liste [] = {1, 2, 3};
```

Dizinin tanımlanmasında bazı derleyiciler yukarıdaki tanımlama biçimleri arasından aşağıdakini dizinin tüm elamanlarını sıfır kabul ederek hem tanımlama hem de etkinleştirme işlemini

gerçekleştirmektedir. Bu durum ise tüm derleyiciler için geçerli değildir. Kullandığınız derleyicinin belgelerine başvurmanız ve bundan sonra aşağıdaki tanımlamayı kullanmanız doğru olacaktır.

```
int liste [10] = {};
```

Dizilerin tanımlanmasında ve etkinleştirilmesinde dikkat edilmesi gereken diğer bir noktada dizilerin tanımlanmasında ve etkinleştirilmesinde dizi eleman sayının belirtilmeden elamanların ardışık olmayan sırada tanımlanmasıdır. Bu yanlış bir tanımlama biçimidir. Bu hata aşağıda gösterilmiştir.

```
int liste [10] = {1, 2, 3, , , 5}; // YANLIŞ
```

Yukarıdaki tanımlamada yanlış olan fikir, bir n adet elaman sahip dizinin ilk ardışık k adet elamanın tanımlanmasının ardından gelen elemanlar eğer tanımlanmamış ise bunlar sıfır olarak tanımlandığı standardın bir gereğidir. Ayıca dizi tanımlandıktan ve etkinleştirildikten sonra sıfır olan elamanlar indis numarası tanımlanarak yeni bir değer atanabilmektedir. Bu iki işlem standardın gereği doğru olurken ikisinin birlikte kullanılabileceği fikri yanlıştır. Çünkü standarda göre bu yaklaşım hatalıdır. Eğer dizinin elemanları tanımlanırken elaman sayısından daha az sayıda olan elamanın tanımlanıp etkinleştirilmesi durumunda son elamanı izleyen diğer elemanlar de boş bırakılmalıdır. Arada bir veya çok sayıda elamanın boş bırakılıp izleyen elemanların tanımlanması söz konusu değildir.

### 7.1.6.Diziler İle İlgili Bazı Kısıtlamalar

Dizilerin tanımlanması ve etkinleştirilmesi konusunda standartta belirtilen sınırlamalar dışında diziler üzerinde bazı işlemlerin yapılması konusunda da sınırlamalar vardır. Örneğin değişkenlerin tanımlanmasında ve değişkenlere değerlerin atanmasında kullanılan kurallar diziler için geçerli değildir. Örneğin aşağıda yukarıdaki örnek programlarda sık kullanılan atama işlemi gösterilmiştir.

```
toplam = toplam + girdi; // DOĞRU
```

```
sonuc = toplam // DOĞRU
```

Bu yazım şekli C++ Programlama Dili'nin standına göre doğrudur. Atama operatörü işlemleri sağdan sola doğru gerçekleşmektedir. Bu durumda sağdaki ifade işlenecek ve selde edilen sonuç ise değişkene atanacaktır.

Yukarıdaki kural ise diziler için geçerli değildir. Bir dizinin elemanlarının değişkenlerdeki gibi diğer bir diziye atanması söz konusu değildir. Dolayısı ile de aşağıdaki bildirim yanlıştır.

```
int elemanlar[10] = {0,1,2,3,4,5,6,7,8,9}; // DOĞRU
```

```
int bir_diger_dizi[10] ={}; // DOĞRU
```

```
bir_diger_dizi = elemanlar; // YANLIŞ
```

Dizilerin yapısında bulunan veri ancak indis numarası ve dizi adı belirtilerek erişilebilir durumdadır. Dolayısı ile de bir dizinin elamanlarının doğrudan diğer bir diziye kopyalanması da yapılması tanımsız bir işlemidir. C++ programlama Dili için böyle bir işlem söz konusu olmamakla

birlikte bu işlem bir döngü aracılığı bir diziden okunan verinin bir diğer diziye aktarılması söz konusudur.

```
for (i=0; i<10; i++)
    elemanlar [i] = bir_diger_dizi [i];
```

Eğer döngü kullanılmayacak ise doğrudan belirli bir dizi elemanın okunup diğer dizideki bir başka konuma kopyalanması da söz konusu olabilir veya dizinin ilgilil elamanları doğrudan bire bire olarak da diğer diziye kopyalanabilir.

```
elemanlar [0] = bir_diger_dizi [2];
elemanlar [1] = bir_diger_dizi [3];
elemanlar [2] = bir_diger_dizi [4];
elemanlar [5] = bir_diger_dizi [5];
```

### 7.1.7.Fonksiyonlarda Dizilerin Parametre Olarak Kullanımı

C++ Programlama Dili'nde fonksiyonlar formal parametreleri ve ismi ile diğerlerinden ayrılır. Bir fonksiyonun formal parametreleri içerisinde bir diğer fonksiyon kullanılması söz konusu değilken bir dizi kullanılabilir. Dizilerin formal parametre olarak tanımlanmasında dikkat edilmesi gereken nokta dizinin bir referans parametresi olarak tanımlanması gerektidir. Bu yapılrken değişkenlerde olduğu gibi veri tipinin sonuna "&" işaretini yazılmalıdır. Fonksiyon formal parametresi olarak dizi tanımlanmış ise, dizinin eleman sayısı dikkate alınmaz. Derleyici bunu doğrudan ihmal ederek programı derleyecektir.

Bir dizinin bir fonksiyonda formal parametre olarak tanımlanması aşağıdaki bildirimde görüldüğü gibi yapılmalıdır.

```
void fonksiyon_adı (veri_tipi dizi_adı [], veri_tipi dizi_adı [], diğer-parametreler);
```

Bir fonksiyonun yapısında formal parametre olarak tanımlanan dizilerde dizinin eleman sayısı dikkate alınmadığı için dizi üzerinde yapılacak olan işlemlerde bu durum programcı tarafından dikkate alınmalıdır. Aşağıdaki örnek kod içerisinde fonksiyon içerisinde tanımlanan dizi üzerinde yapılacak olan işlemler için ikinci formal parametre sınır kontrolü için kullanılmıştır.

```
// Kontenjan negatif olamaz!
unsigned int kontenjan = 200;
// Dizi tanımla ve etkinleştir.
unsigned int derse_kayit_olanlar[200] = {};
// kayıt olan öğrenci sayısı negatif olamaz
unsigned int ders_kayit;
...
void dersi_alanlar_listesi (unsigned int kayit_olanlar [], unsigned int ders_kayit);
```

```

int main()
{
    ...
}

void dersi_alanlar_listesi (unsigned int derse_kayit_olanlar [], unsigned int ders_kayit);
{
    for ( i =0; i<= ders_kayit; i++)
    {
        cout << kayit_olanlar[i] << endl;
    }
}

```

Bir fonksiyonda herhangi bir parametre referans parametresi olarak kullanılırsa, hem fonksiyon hem de fonksiyon dışında ana fonksiyon veya diğer fonksiyonlar tarafından da üzerinde işlem yapılarak içeriği değiştirilebilir. Bu durumda programın üzerinde işlem yaptığı bellek adresi farklı zamanlarda değişikliğe uğrayacaktır. Bunun önüne geçilmesi için değişkenlerin sabit olarak tanımlanması ile değiştirilemez olmasının yapılması sağlanabilir. Bir diğer olası çözüm ise sabit olarak tanımlanan değişkenin değerinin program akışı içerisinde global veya yerel değişken olarak tanımlanmış değişkenlere değerinin kopyalanarak üzerinde işlem yapılması sonuçların döndürülmesi ama asıl verinin korunmasıdır.

Referans parametrelerde değişkenin bellek üzerinde barındırıldığı bellek adresinin döndürülecek, buradan okunarak üzerinde işlem yapılması nedeni ile, bir fonksiyonda dizilerin parametre olarak tanımlanması durumunda yukarıda belirtilen yaklaşımlar ile asıl verinin bulunduğu dizi ve içeriği korunabilir. Asıl verinin bulunduğu dizinin içeriğinin bir kopyasının bir diğer dizeye aktarılması ile bu yeni dizi üzerinde işlemler yapılabilir ve ana dizi korunur. Ancak bir dizini içeriğinin kopyalanması için tüm dizi elemanlarının teker teker kopyalanmak durumunda olması nedeni ile ortaya çıkacak zaman ve bellek kayıpları ile olası hatalar programın hatalı sonuçlar döndürmesine neden olur. Bunun önüne geçilmesi için asıl kaynak olarak kullanılacak ve referans olarak okuna dizinin sabit olarak tanımlanması sorunu ortadan kaldıracaktır. Aşağıdaki örnek prototip fonksiyon tanımında derse\_kayit\_olanlar [] adlı dizi sabit olarak tanımlanmıştır. Bunun için C++ Programlama Dili'nin saklı sözcüklerinden olan **const** kullanılmıştır. Böylece söz konusu dizi her ne kadar referans parametresi olsa, fonksiyon tarafından işlem yapılrken değiştirilemez durumdadır.

```

// Kontenjan negatif olamaz!
unsigned int kontenjan = 200;
// Dizi tanımla ve etkinleştir.
unsigned int derse_kayit_olanlar[200]={};
double basari_notu [ ];

```

```

// kayıt olan öğrenci sayısı negatif olamaz
unsigned int ders_kayit;

...
void dersi_alanlar_listesi (unsigned int kayit_olanlar [], unsigned int ders_kayit, double basari_notu []);

int main()
{
    ...
}

void dersi_alanlar_listesi (const unsigned int derse_kayit_olanlar [], unsigned int ders_kayit,
double basari_notu []);

{
    ...
}

```

Aşağıdaki örneklerde bir program içerisinde fonksiyonlarda formal parametre olarak dizilerin kullanım şekilleri gösterilmiştir.

Dizilerin tanımlanmasında gerekli olan koşullardan birisi dizinin eleman sayısıdır. Eğer üzerinde işlem yapılacak olan verinin sayısı programın başlangıcında bildirileceği var sayılmıştır.

```

void dizi_etkinlestir( int elemanlar [], unsigned eleman_sayisi)
{
    for (int i = 0; i < eleman_sayisi; i++)
    {
        elemanlar [i] = 0;
    }
}

```

Veriler okunduktan sonra bir dizi içerisinde barındırılacak ise dizinin eleman sayısının bilinmesi v okuma işleminin eleman sayısı kadar tekrar etmesi gereklidir. Dizinin eleman sayısının bilindiği ve okuma işleminin dizi eleman sayısından daha fazla olmaması kontrol edilmelidir.

```

void verileri_oku_sakla ( int elemanlar [], unsigned eleman_sayisi)
{
    for (int i=0; i < eleman_sayisi; i++)
    {

```

```

        cout << i +1 << "Değeri giriniz:"
        cin >> elemanlar [i];
    }
}

```

Veriler okunup dizide barındırılmaktadır. Barındırılan diziler sırası ile yazdırılır. Yazdırma işlemi için dizinin eleman sayısı parametre olarak fonksiyona aktarılmaktadır. Döngünün gerçekleştüğü çevrim sayısı dizinin eleman sayısın kadar tekrar etmesi gereklidir. Dizinin eleman sayısının bilindiği ve okuma işleminin dizi eleman sayısından daha fazla olmaması kontrol edilmelidir.

```

void verileri_yaz ( int elemanlar [], unsigned eleman_sayisi)
{
    for (int i = 0; i < eleman_sayisi; i++)
    {
        cout << i +1 << ". olarak okunmuş değer: " << elemanlar [i];
    }
}

```

Dizide barındırılan sayısal değerlerin toplamı bulunup ortalama hesaplanacaktır. Bu nedenle fonksiyona formal parametre olarak sabit dizi tanımlanmış ve dizin eleman sayısı da diğer formal parametre olarak tanımlanmıştır. Bu dizi değer döndürmekte olduğu için veri tipi tanımlaması yapılmıştır.

```

double verilerin_ortalama ( int elemanlar [], unsigned eleman_sayisi)
{
    double toplam = 0;
    double ortalama = 0.0;
    for (int i = 0; i < eleman_sayisi; i++)
    {
        toplam = toplam + elemanlar [i];
    }
    ortalama = toplam / eleman_sayisi
    return ortalama;
}

```

Aşağıdaki örnek ise bir dizinin elemanlarının bir diğer diziye kopyalanmasını göstermektedir. Her iki dizinin eleman sayısının büyüklükleri bilinmekte olduğu ve kopyalama işlemi sırasında herhangi bir sınır aşılması durumunun söz konusu olmaması için programcının bu

işlem için gereken kontrol mekanizmalarını kurması ve işletmesi gerekiği unutulmamalıdır. İkinci diziye yapılan kopyalama işleminde ilk dizinin ilk elamanında başlanırken, ikinci dizide ise kullanıcısı tarafından belirtilen bir sırasından kopyalama işlemine başlanmaktadır.

```
void kopyala ( int elemanlar [], unsigned eleman_sayisi, unsigned int konum, unsigned int
hedef_konum, int kopyalar [], unsigned int kopyalar_eleman_sayisi, unsigned kopyalanacak)
{
    for (i = konum; konum < hedef_konum + kopyalanacak; konum++)
    {
        kopyalar[hedef_konum] = elemanlar[konum];
    }
}
```

### 7.1.8.Dizilerin Taban Adresi – Base Address

Intel i386 işlemcisinin tasarlarken o zaman için önemli bir problem olan uygulamaların artan bellek gereksinimlerini karşılamak için yeni bir bellek erişim modeli geliştirmiştir. Buna **kesimli bellek modeli – segmented memory model** adı verilmiştir. Kesimli bellek modelinin üstünlüğü işlemcinin bir seferde işleyebileceği verinin büyüklüğünün i386 için adresleyebildiği bellek miktarına eşit olmasıdır. Bu da 4GB karşılık gelmektedir. Ancak işlemcinin üretiminde kullanılan mimari ve teknolojisi bir işlemcinin üzerinde, sadece işlemci tarafından erişilip kullanılabilenek olan belleğin büyüklüğünün 4GB erişmesini maliyet nedeni olanaksız kılmaktadır. Bu nedenle de i386 kullanmakta olduğu kesimli bellek modeli işlemci tarafından işlenecek olan veriyi çok daha küçük parçalara ayırıp bunlar üzerinde işlem yapmasını sağlamaktadır. Bunun için de kullanılan teknoloji, bir verinin birçok parçaaya ayrılarak öncelikle gereksinim duyulanların RAM üzerinde barındırılmasını ve öncelikli olmayan verilerin ise ise ikincil depolama birimleri üzerinde yani sabit disk üzerindeki takas dosyası üzerinde barındırılmasını sağlamaktadır. Bu model, ile işlemci tarafında işlenecek olan verinin bellek üzerinde bulunduğu adresin başlangıç değerinin ve işlenecek olan verinin başlangıç adresine olan uzaklığının bilinmesini sağlamaktadır.

Bu modelin işleyişini dizilerde gözlemlenir. Program içerisinde bir dizi tanımlandığında, dizin adı, dizinin barındıldığı ver tipi ile dizinin bellekteki başlangıç adresi kayıt altına alınır. Dizinin bellekteki başlangıç adresi **taban adresi – base address** olarak tanımlanır.

```
int tam_say_dizi [10] ={};
```

Bu tanımlama yapıldığında on elemanlı bir dizi için bellekte yer ayrılır. Ayrılan yer tam sayı veri tipi olduğu için her bir dizi elemanı için 4 byte olmak üzere toplam da 40 byte yer ayrılacaktır. Dizide barındırılan bir veriye erişilmesi için gerekli olan dizinin adı ile temsil edilen ve dizinin bellekte barındırıldığı bölümün başlangıç adresi, veri tipi ile dizi içerisindeki konumu belirten indis gerekektir. Bunlar kullanılarak bellekten istenilen veri elde edilebilir.

Aşağıdaki temsili gösterimde kesimli bellek modelinin kullanımı gösterilmiştir. Bellek üzerinde dizinin başlangıç adresi 1024 olarak var sayılmıştır. Dizi üzerinde işaretli tam sayılar barındırılmaktadır. Bu nedenle de her bir dizi elemanı için 4 byte yer ayrılmıştır. Bu da bir elamanın bel-

lekte barındırıldığı adresden itibaren 4 byte sonra diğer bellek verisin bulunduğu anlamına gelmektedir.

<b>tam_say_dizi</b>	1	2	3	4	5	6	7	8	9	0
<b>indis</b>	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
<b>Bellek adresi</b>	1024	1028	1032	1036	1040	1044	1048	1052	1056	1060
<b>Taban adresi</b>	<b>1024</b>									

Eğer bir dizi, bir fonksiyonda formal parametrelerden birisi olarak tanımlanmış ise bu durumda, dizinin taban adresi fonksiyona aktarılır.

### 7.1.9.Dahili Veri Tipleri ve Dizinin İndisi

C++ Programlama Dili'nin önemli özelliklerinden bir tanesi dahili veri tiplerinin dizilerde indis olarak kullanılabilmesidir. Bunun için “typedef” ve “enum” doğru şekilde kullanılırsa “enum” veri tipinin indis olarak kullanılmasını sağlarken, dizi tanımlamaları için de “typedef” ile tanımlamalar yapılması olanaklıdır. Aşağıdaki örnek bunu göstermektedir.

```
enum konut_tipleri (ev1, ev2, ev3, ev4, ev5, ev6);  
double konut_satis[6];  
konut_tipleri konut;
```

Bu tanımlama ile satış verilerinin barındırılacağı dizide indis numarası olarak doğrudan sıralama veri tipinde tanımlanmış olan elemanlar dizi için indis durumuna geçmektedir. Örneğin aşağıdaki kod bu özelliği kullanmaktadır.

```
for (konut = ev1; konut <=ev6; konut = static_cast<konut_tipleri>(konut+1))  
    konut_satis[konut] = 0.0;
```

Örneğin ev3 ile tanımlanan konutun satış gelirinin artırılması için aşağıdaki kod kullanılmıştır.

```
konut_satis [ev3] = konut_satis [v3] + 1250000.00;
```

Bu tür yazılan kodlar özellikle program içerisinde sabit olacak verilerin kullanılmasının daha uygun olduğu durumlarda kullanılır. Sıralama veri tiplerinin indis numarasının sıfırdan başladığı unutulmamalıdır.

### 7.1.10.Dizilerin Tanımlanması İçin Kullanılan Diğer Yöntemler

Dizilerin tanımlanmasında farklı yöntemlerden yararlanılmaktadır. Bunlardan bir diğer ide “typedef” ile “const” kullanılmıştır. Eğer programda aynı sayıda elemana sahip birden çok dizinin kullanımı söz konusu ise bu durumda dizilerin tanımlanmasında kolaylık sağlamak için aşağıdaki gibi bir tanımlama kullanılabilir.

```
const unsigned int ELEMAN_SAYI = 100;  
typedef double liste[ELEMAN_SAYI];
```

...

liste veri;

liste islem\_sonuclar;

Yukarıdaki kod bloklarında görüleceği üzere dizinin eleman sayısı 100 adet olarak sabit bir değerdir. Benzer olarak liste adlı bir tanımlama yapılmış ve tanımlamanın aslında double veri tipinde ve dizinin eleman sayısının da önceden belirtilen sayıda olduğu tanımlanmıştır. Yukarıdaki kod blokundaki son iki bildirim dizi tanımlamakta kullanılmaktadır.

## 7.2.Dizilerde Arama ve Sıralama İşlemleri

Diziler üzerinde en yaygın olarak yapılan işlemler arasında dizi içerisinde bir verinin aranması veya dizinin barındırdığı verilerin belirli bir ölçüte göre sıralanmasıdır. Aram işlemlerinde kullanılan temel algoritmaların bir tanesi **sıralı arama – sequential search** veya **doğrusal arama -linear search** algoritmasıdır. Bu algoritma adında da anlaşılacağı üzere bir dizi içerisinde barındırılan veri kümelerindeki elamanların ilk elamanından başlayarak son elaman'a kadar karşılaştırma yaparak aramaktadır. Eğer aranan veri dizi elamanları arasında bulunuyor ise bu işlem başarılı kabul edilir, aksi durumda ise başarısız olarak kabul edilir.

Aşağıdaki örnekte algoritmanın işleyişi görülmektedir.

<b>tam_say_dizi</b>	-1	3	-7	0	5	2	9	12	-98	-6
<b>indis</b>	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Bu dizi içerisinde 12 aradığımızı var sayalım. Aramaya ilk dizi elemanından başlanılır. Buna göre tam\_say\_dizi[0] elemanı okunur ve aranılan veri ile karşılaştırılır. Eğer aranılan veri bulunmuş ise arama işlemi sona erer ve döngüden çıkıştır, aksi durumda ise sıradaki dizi elamanına geçilir ve karşılaştırma işlemi tekrarlanır. Yukarıdaki örnekte aranan veri olan 12 dizi içerisinde 8. elemandır. Bu da arama işleminin sekiz defa tekrarlanacağı anlamına gelir.

Aşağıdaki programda verilen bir sayı dizisinde kullanıcı tarafından belirtilen bir sayı doğrusal arama algoritması kullanılarak aranmakta ve işlem sonucu döndürülmektedir.

```
/*
 * arama_yap.cxx
 *
 *
 * Copyright 2022 Gökşin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
```

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,

\* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

/\*

```
#include <iostream>
```

```
#include <locale>
```

```
using namespace std;
```

```
// Ana fonksiyon
```

```
#include <iostream>
```

```

#include <locale>
#include <iomanip>
using namespace std;
// Sabitlerin tanımlanması
const unsigned int BOYUT = 20;    // Dizinin eleman sayısı
// Prototip fonksiyon - doğrusal arama
int DogrusalArama (const int liste[BOYUT], int eleman_say, int aranan);
// Ana fonksiyon
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişkenlerin-dizilerin tanımlanması
    int veriler[BOYUT];           // verilerin saklanacağı dizi
    int sayı = 0;                 // aranacak olan veri
    int konum = 0;                // aranan verinin sırası
    cout << BOYUT << " adet sayıyı aralarında boşluk bırakarak giriniz. " << endl;
    for (unsigned int indis = 0; indis < BOYUT; indis++)
        cin >> veriler[indis];
    cout << endl;
    cout << "Aranacak sayıyı giriniz: ";
    cin >> sayı;
    cout << endl;
    konum = DogrusalArama (veriler, BOYUT, sayı);
    if (konum != -1)
        cout << "Aranan veri " << konum+1 << " sırada bulunmuştur." << endl;
    else
        cout << "Aranan veri bulunamamıştır." << endl;
    return 0;
}

int DogrusalArama (const int liste[BOYUT], int eleman_say, int aranan)

```

```

{
    // Değişkenlerin tanımlanması

    int konum = 0;

    bool bulundu = false;

    // Arama için kullanılacak olan döngü

    while (konum < eleman_say && !bulundu)

    {
        if (liste[konum] == aranan)

            bulundu = true;

        else

            konum++;

    }

    if (bulundu)

        return konum;

    else

        return -1;
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./arama_yap
```

20 adet sayıyı aralarında boşluk bırakarak giriniz.

```
-2 -3 -5 1 9 34 19 22 3 10 11 12 13 14 15 16 17 18 19 20
```

Aranacak sayıyı giriniz: 9

Aranan veri 5. sırada bulunmuştur.

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

### 7.2.1.Seçerek Sıralama

Bir dizi de barındırılan elemanlar üzerinde gerçekleştirilen arama işlemleri için doğrusal arama algoritmasını gördük. Bu algoritma bir diziyi baştan sona tarayarak aranılan elemanın dizide bulunup bulunmadığını belirlemekte idi. Bu bölümde ise benzer bir yaklaşım ile bir dizideki elemanların taranarak büyükten küçüğe veya küçükten büyüğe sıralanmasını sağlayan seçerek sıralama algoritmasını inceleyeceğiz.

Seçerek sıralama algoritması bir dizi içerisinde barındırılan elamanları aksi belirtilmekçe küçükten büyüğe doğru sıralar. Bunun için öncelikle dizideki en küçük elaman aranır. Bulunan en küçük elaman dizinin ilk elemanı ile yer değiştirir. Böylece ilk sıraya en küçük eleman gelmiş olur. Bu aşamadan sonra döngü kalan elemanlar için tekrarlanır. Böylece döngü tüm elemanlar sırası ile yer değiştirdiğinde sona erer.

Aşağıdaki dizide seçerek sıralama algoritmasının işleyişi örnek bir dizi üzerinde gösterilmektedir. Öncelikle dizideki en küçük elemanın bulunması için tüm elemanlar taranır.

<b>tam_say_dizi</b>	-1	3	-7	0	5	2	9	12	-98	-6
<b>indis</b>	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
<b>Sıralanmamış dizi</b>										

Dizideki en küçük eleman -98 ve indis'i ise 8'dir. Bu durumda ilk eleman yani indis numarası 0 olan eleman ile yer değiştirecektir.

<b>tam_say_dizi</b>	<b>-98</b>	3	-7	0	5	2	9	12	<b>-1</b>	-6
<b>indis</b>	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
<b>Sıralanmamış dizi</b>										

Bu işlemden sonra dizi ilk eleman hariç olmak üzere yeniden en küçük elemanın bulunması için tekrar taranır. Bir diğer deyişle n elemanlı dizide en küçük elemanın bulunmasından sonra kalan n-1 eleman tekrar en küçük eleman için taranacaktır. Bu taramadan sonra en küçük eleman ise dizideki indis numarası 2 olan -7'dir. Bu eleman, kalan elemanlardan ilk olan ve indis numarası 1 olan 3 ile yer değiştirir.

<b>tam_say_dizi</b>	<b>-98</b>	<b>-7</b>	<b>3</b>	0	5	2	9	12	-1	-6
<b>indis</b>	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
<b>Sıralanmamış dizi</b>										

Bu işlemden sonra dizi ilk iki eleman hariç olmak üzere yeniden en küçük elemanın bulunması için tekrar taranır. Bir diğer deyişle n elemanlı dizide en küçük elemanın bulunmasından sonra kalan n-2 eleman tekrar en küçük eleman için taranacaktır. Bu taramadan sonra en küçük eleman ise dizideki indis numarası 9 olan -6'dır. Bu eleman, kalan elemanlardan ilk olan ve indis numarası 3 olan 3 ile yer değiştirir.

<b>tam_say_dizi</b>	<b>-98</b>	<b>-7</b>	<b>-6</b>	0	5	2	9	12	-1	<b>3</b>
<b>indis</b>	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
<b>Sıralanmamış dizi</b>										

Seçimli sıralama algoritmasının işleyişi yukarıdaki örnekten de görüleceği gibi aşağıda gösterilen sözde kod ile ifade edilebilir:

```
for (indis = 0; indis < eleman_sayısı; indis++)
```

{

- 1.[indis] .... [eleman\_sayısı – 1] arasında en küçük elemanı ve bu elamanın konumunu, ayrıca en küçük [indis] değerini bul.
2. Dizideki en küçük [indis] değerine sahip olan eleman ile yer değiştir.

}

Buradaki sözde kod ile temsil edilen döngü yukarıda verilen örnekteki işlemler göstermektedir. Döngünün ilk çalışması sırasında dizindeki yani dizi[0] ... dizi[eleman\_sayısı – 1] içerisindeki en küçük eleman bulunmaktadır. Ardından en küçük eleman ilk sıraya yani dizi[0] yerleştirilmektedir. Buradaki elemanda en küçük elamanın yerine yerleştirilmektedir. Döngünün ikinci tekrarında dizideki yani dizi[1] ... dizi[eleman\_sayısı – 1] içerisindeki en küçük eleman bulunmaktadır. Bu bulunan eleman dizi[1] eleman ile yer değiştirmektedir. Döngü son dizi elemanı ile dizi[eleman\_sayısı – 2] ile dizi[eleman\_sayısı – 1] yer değiştirip tamamlanıncaya kadar tekrar edecektir.

Seçerek sıralama algoritması görüleceği üzere iki aşamalı bir algoritmadır. Birinci aşama her zaman için sıralanmamış dizi içerisindeki en küçük elemanı bulmak için kullanılmaktadır. Aşağıda görülen örnek kod bu işlemi yapmaktadır.

```
// ilk sıradaki eleman en küçük varsayılsın.  
EnKucukEleman = indis;  
for (konum = indis + 1; konum < eleman_sayisi; konum ++)  
{  
    // dizide işlenen eleman, varsayıldan daha küçük ise en küçük elaman güncellenir.  
    if (dizi[konum] < dizi[EnKucukEleman])  
        EnKucukEleman = konum;  
}
```

Bu aşamadan sonra ise algoritmada belirtilen ikinci aşamaya geçilir. Bu aşamada en küçük elaman ile döngü gereği olarak ilk sıradaki eleman yer değiştirecektir. Yani dizi[ilk\_sira] elamanı ile dizi[konum] yer değiştirmelidir. Bu yer değiştirme işlemi için bir geçici olarak dizi elemanını barındıracak olan bir değişkene gereksinim olacaktır.

```
// okunan elemanı barındıracak olan geçici değişken  
gecici_saklama = dizi[EnKucukEleman];  
dizi[EnKucukEleman] = dizi[indis];  
dizi[indis] = gecici_saklama
```

Yukarıdaki kod bloku dizi elemanlarının yer değiştirilmesinde kullanılır. İki verinin yer değiştirebilmesi için bunlardan bir tanesini geçici olarak barındıracak bir değişken gereksinim vardır, yani iki değişkenin yer değiştirmesi için üç değişken gereklidir. Aşağıda secimli\_sıralama fonksiyonu kaynak kodu görülmektedir.

```
/*  
* secmeli_siralama.cxx  
*  
*  
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>  
* *  
* Her hakkı saklıdır.  
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak  
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği  
* takdirde serbesttir:  
*  
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,  
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.  
*  
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,  
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen  
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.  
*  
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA  
* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
* SONUCUNDA OLUSAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
```

```

* SORUMLU DEĞİLLERDİR.

*
*/
/*
#include <iostream>
#include <locale>
using namespace std;
// Sabitlerin tanımlanması
const int BOYUT = 10;      // Dizinin eleman sayısı
typedef int liste[BOYUT];
// Prototip fonksiyon - seçmeli sıralama
void SecmeliSiralama (int liste[], int elaman_sayisi);

// Ana fonksiyon
int main()
{
    // Dil ayarları
    setlocale(LC_ALL, "tr_TR.UTF-8");
    // Değişkenlerin tanımlanması
    int sayac;                      // ilk döngü için gerekli olan sayaç
    int kopya;                       // dizileri kopyalamak için gerekli değişken
    // dizilerin tanımlanması
    liste veriler;                  // asıl verilerin olacağı dizi
    liste islem;                    // işlem yapılacak olan dizi
    cout << BOYUT << " adet sayıyı aralarında boşluk bırakarak giriniz. " << endl;
    // Klavyeden girilen sayılar okunur, veriler dizisine atanır.
    for (sayac = 0; sayac < BOYUT; sayac++)
    {
        cin >> kopya;
        veriler[sayac] = kopya;
        islem[sayac] = kopya;
    }
}

```

```

    }

    cout << endl;

    // Seçerek sıralama algoritmasını uygulayan fonksiyon çağrılp çalıştırılır.

    SecerekSiralama (islem, BOYUT);

    // İşlem sonucu yazdırılır

    cout << "Sıralama öncesi verilerin durumu: ";

    for (int sayac = 0; sayac < BOYUT; sayac++)

    {

        cout << veriler[sayac] << " ";

    }

    cout << endl << "Sıralama sonrası verilerin durumu: ";

    for (int sayac = 0; sayac < BOYUT; sayac++)

    {

        cout << islem[sayac] << " ";

    }

    cout << endl;

    return 0;
}

void SecerekSiralama (int liste[], int eleman_sayisi)

{

    // Değişkenlerin tanımlanması

    int indis;                                // dizinin başlangıcı

    int EnKucukEleman;                         // dizideki en küçük elamanın konumu

    int konum;                                 // dizide işlenecek oaln sıradaki eleman

    int gecici_saklama;                        // takas için kullanılacak değişken

    for (indis = 0; indis < eleman_sayisi-1; indis++ )

    {

        // Algoritmanın ilk aşaması

        // ilk sıradaki elemanın en küçük olduğu varsayılr.

        EnKucukEleman = indis;

        for (konum = indis + 1; konum < eleman_sayisi; konum ++)

}

```

```

{
    // dizide işlenen eleman, varsayılandan daha küçük ise en küçük elaman güncellenir.

    if (liste[konum] < liste[EnKucukEleman])
        EnKucukEleman = konum;

    // Algoritmanın ikinci aşaması

    // En küçük elaman ilk sıraya taşınabilmesi için yer değiştirme işlemi yapılır.
    gecici_saklama = liste[EnKucukEleman];
    liste [EnKucukEleman] = liste[indis];
    liste [indis] = gecici_saklama;
}

}
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./secerek_siralama
```

10 adet sayıyı aralarında boşluk bırakarak giriniz.

```
-100 100 1 -3 8 16 -2 -9 18 42
```

Sıralama öncesi verilerin durumu: -100 100 1 -3 8 16 -2 -9 18 42

İşlem sonrası verilerin durumu: -100 -9 -3 -2 1 8 16 18 42

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

Seçerek sıralama algoritmasının incelenmesinden görüleceği her sıralama işlemi için dizi-deki ilk elamandan başlanarak n adet elemanlı bir dizide n – 1 adet karşılaştırma yapılmaktadır. İkinci döngüde n – 1 eleman işleme girmekte ve n – 2 adet eleman ile karşılaştırılmaktadır. Bu işlemlerin adet olarak aşağıdaki serİYE karşılık gelecektir.

$$\sum_{i=1}^n (i_{i-1} \times (n-i)) = i_0 \times (n-1) + (i_1) \times (n-2) + \dots$$

Bu serinin toplamı Gauss toplam formüle ile bulunabilir. Buna göre karşılaştırma döngüsü için n elemanlı bir dizi için toplam aşağıdaki gibidir:

$$\sum_{i=1}^n (i_{i-1} \times (n-i)) = \frac{n \times (n-1)}{2}$$

Döngünün içerisinde yer alan ikinci döngü ile iki elamanın yer değiştirmesi için 3 adet işlem yapılmaktadır. Bu durumda son elaman hariç olmak üzere elaman sayısının bir eksiği kadar değişken ataması de yapılacaktır. Bu işlemlerin sayısı da dizideki elaman sayına bağlı olarak aşağıdaki gibi ifade edilebilir:

$$3 \times (n-1)$$

Bu bağıntıların yorumlanması sonucunda dizinin elaman sayısı arttıkça seçerek sıralama algoritmasının yaklaşık olarak zaman yapacağı işlem sayısının üstel olarak arttığı söylenebilir. Dizinin elaman sayısı 10000 adet olarak seçilmesi durumunda, sıralama işlemleri 49995000 ve atama işlemleri 29997 adet olacaktır. Eğer ifadelerde basitleştirme yapılacak olursa karşılaştırma işlem sayısı ile atama işlem sayısının sıra ile aşağıdaki gibi olduğu söylenebilir:

$$\frac{n^2}{2} \text{ ve } 3n$$

### 7.3.for Döngülerinde Döngü Sayısının Otomatik Belirlenmesi

Bilgisayar programlarında değişkenlerin veri tiplerinin tanımlanması doğal bir eylem olduğu kadar programcının bir sorumluluğudur da. C++ Programlama Dili bu konuda programcılar işlenini kolaylaştırmak adına “**auto**” özel sözcüğü kullanılır. Aşağıda yer verilen kullanım örneği bu sözcüğün verinin tipini tanımlamaya gerek olmadan bu işlemin otomatik olarak yapılmasını sağlamaktadır.

```
auto deger = 15;
```

Bu bildirim ile “deger” adlı değişkenin ilk değerinin tam sayı veri tipinde olduğunu ve değerinin de 15 olduğunu tanımlamasını sağlar.

Dizilerin elemanları üzerinde gerçekleştirilecek olan işlemler için “for ... ” döngüleri kullanmasında döngünün kaç defa işleyeceğini belirtmek için programın başında yapılan sabit değer tanımlamaları kullanılabilir. Yukarıdaki örnek programlarda bu yaklaşım kullanılmıştır.

```
for ( int i = 0; i < eleman_sayisi; i++)
    // işlemler
```

Dizilerin elemanları üzerinde gerçekleştirilecek olan işlemler için “for ... ” döngüleri kullanmasında döngünün kaç defa işleyeceğini belirtmek için programın başında yapılan sabit değer tanımlamaları kullanılabilir. Yukarıdaki örnek programlarda bu yaklaşım kullanılmıştır. Öte yandan C++ Programlama Dili ile döngülerin çevrim sayısının tanımlanmasında yukarıdaki yaklaşım yanında kod yazımını kısaltan bazı özellikleri de sunmaktadır. “for ... ” döngüleri için kapsam çözümleyici operatörüne benzer şekilde dizi üzerinde yapılacak işlemler için dizinin veri tipi ile aynı veri tipinde tanımlanacak değişkenler kullanılarak dizinin adının tanımlanması ile gerekli olan tanımlamaların hızlıca ve otomatik olarak yapılması sağlanabilir. Aşağıdaki kullanım şekli bu yapının nasıl bir bildirimde kullanılacağını göstermektedir.

```
for ( veritipi değişken : diziAdı)
    // işlemler
```

Bu şekilde yapılan döngü tanımlamasına **boyut tabanlı for döngüsü – range-based for loops** adı verilir. Diziler üzerindeki döngü ile gerçekleştirilen işlemlerde kullanılır. Aşağıdaki örnek kod blokunda uygulaması gösterilmiştir.

```
int liste[20];
int toplam = 0;
```

...

```
for (int sayı : liste)  
    toplam = toplam + sayı;
```

Yukarıdaki kod blokunda liste adlı dizinin tam sayı veri tipinde verileri barındırdığı görülmektedir. Döngünün ilk çevriminde sayı adlı değişken dizideki ilk elamanın yani indis değeri 0 olan veriye eşittir. İkinci çevrimde ise aynı değişken dizideki sıradaki elaman olan indis değeri 1 olan veriye eşittir. Döngü bu şekilde tüm dizi elamanları işlenene kadar devam eder.

Aşağıdaki örnekte ise bölümün başında tanımlanan “auto” özel sözcüğü kullanılarak bir döngü kurulmuştur. Döngü dizi içerisindeki en büyük elemanı bulmaktadır.

```
for (auto sayı : liste)  
{  
    if (en_buyuk < sayı)  
        en_buyuk = sayı;  
}
```

Yukarıdaki kullanım şekilleri fonksiyonlar için geçerli değildir. Daha önce de belirtildiği üzere, fonksiyonlar da diziler formal parametre olarak kullanılrsa, bu durumda dizinin taban adresi yani başlangıç adresi fonksiyona parametre olarak aktarılmış olmaktadır. Yukarıdaki kullanım örneklerinin kullanılması için fonksiyon içerisinde kullanılmaması gereklidir. Çünkü yukarıdaki örneklerde otomatik tanımlama için dizinin ilk elamanı ilse son elamanına ait bilgiler yani indis değerleri kullanılır. Fonksiyonlardaki formal parametrelerde ise sadece tek bir veri aktarılmaktadır.

## 7.4.Karater Katarı Dizileri – C-Strings

Bir çok programlama kitabında veri tipleri genel olarak tek bir bölümde ele alınır, büyük-lükleri ile barındırabilecekleri verinin özellikleri açıklanır ve konu sona erer. Bu yaklaşım bu ders notlarında bilinçli olarak uygulanmamıştır. Bunun nedeni ise C++ Programlama Dili’nde iki farklı karakter katarı veri tipinin bulunmakta olmasıdır. Önceki bölümlerde incelendiği üzere tek bir karakteri barındıran veri tipi için tek turnak kullanılacağı belirtilmiştir. Birden çok sayıdaki karakterin bir araya geldiği karakter katarı – string veri tipinde ise veriler çift turnak arasında yazılmak durumunda idi. Karakter katarı dizileri, karakter katarı veri tipinde olduğu gibi çift turnak arasında tanımlanır. Ancak farklı olarak karakter katarı veri tipi ardışık olarak bellekte yer kaplayan karakterlerden oluşurken, karakter katarı dizisi ise ardışık olarak bellekte bulunan karakterlerin sonunda yer alan sonlandırma karakteri – null terminator ile sonlanır. Sonlandırma karakteri bellekte bir byte alan kaplar ve yazdırılamaz bir karakterdir. Aşağıdaki bildirimde son adlı değişken null terminator barındırmaktadır.

```
char son = '\n';
```

Örneğin değişken adı olarak isim ve bu değişkene atanın veri de “Veli” olsun. Eğer isim adlı değişken karakter katarı – string olarak tanımlanmış ise bellekte 4 byte yer kaplar. Eğer aynı değişken karakter katarı dizisi – C-string olarak tanımlanmış ise bellekte 5 byte yer kaplar.

Karakter katarı dizileri – C-string tanımlanırken aşağıdaki yapı kullanılır:

```
char isim[10];  
isim = "Veli";
```

Yukarıda gösterilen tanımlama şekli bir karakter katarı dizisi – C-string tanımlamasıdır. Veri tipi olarak karakterlerden oluşan ve yapısında 20 adet karakter barındıran bir dizi olarak tanımlanır. Bu tanımlamada karakter katarı dizinin eleman sayısı 20 olsa da, son elamanın null terminator olması nedeni ile ancak 19 karakter olacaktır. Eğer bu değişken tanımlaması yapıldıktan sonra bu değişkene "Veli" değeri atanacak olursa aşağıdaki durum ortaya çıkacaktır.

isim	V	e	I	i	\0					
indis	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

İsim adlı değişken 4 karakter büyüğünde bir veriyi barındırıyor gibi görünse de null – terminator nedeni ile beş karakter saklanacak ve geri kalan 5 karakter de boş kalacaktır. Yapılan tanımlama yukarıda belirtilen yapı dışında aşağıdaki gibi de olabilir.

```
char isim[10] = {'V', 'e', 'l', 'i', '\0'};
```

C++ Programlama Dili'nde diziler ile ilgili olan kurallar, karakter katarı dizileri için de geçerlidir. Buna göre bir karakter katarının tanımlanması için dizinin eleman sayısının tanımlanmasına gerek yoktur. Doğrudan dizi elemanlarının tanımlanması yeterlidir.

```
char isim[] = "Veli";
```

Bir diğer dikkat edilmesi gereken nokta da, dizilerde atama işlemlerinin eleman bazında yapılmakta olmasıdır. Böylece, bir karakter katarı dizisine atam yapılması gerekn durumlarda değişken adının yazılarak değişkenlere yapılan atama işleminin karakter katarı dizilerinde kullanılması olanaklı değildir.

```
char isim[10];  
  
char soyad[10];  
  
isim = "Mahmut"; // Bu işlem yapılamaz!  
soyad = isim; // Bu işlem yapılamaz.
```

Aşağıdaki tabloda karakter katarı dizisi fonksiyonları gösterilmiştir.

Bazı C-string – karakter katarı dizisi fonksiyonları		
Fonksiyon Adı	Fonksiyon Parametreleri	Fonksiyonun Döndürdüğü Değer
strcat(hedef, kaynak)	hedef ve kaynak null terminator ile sonlandırılmış karakter katarı dizisi olmak	Hedef karakter katarı dizisinin taban adresi döndürülür. Kaynak karakter katarı dizisi null terminator de dahil

	zorundadır. Hedefin büyük-lüğü kaynağı da alabilecek kadar büyük olmalıdır.	olmak üzere hedefin sonuna eklenir.
strncat(#hedef,kaynak,limit)	hedef ve kaynak null terminator ile sonlandırılmış karakter katarı dizisi olmak zorundadır. Hedefin büyük-lüğü kaynağı da alabilecek kadar büyük olmalıdır. Son parametre olan limit ise negatif olmayan bir tam sayı değeridir.	Önceki fonksiyon ile aynıdır, ama farklı olarak belirtilen limit değeri ile sınırlı karakter hedefin sonuna eklenir.
strcmp(str1, str2)	str1 ve str2 null terminator ile sonlandırılmış karakter katarı dizileridir.	Fonksiyon, iki karakter katarı dizisini karşılaştırır. Eğer str1, str2'den küçük ise negatif değer, dğer str1, str2 eşit ise sıfır, eğer str1, str2'den büyük ise pozitif değer değer döndürür.
strncmp(#str1, str2, limit)	str1 ve str2 null terminator ile sonlandırılmış karakter katarı dizileridir. Son parametre olan limit ise negatif olmayan bir tam sayı değeridir.	Önceki fonksiyon ile aynıdır, ama belirtilen limit değeri ile sınırlı karakter katarı dizisi elemanları karşılaştırılır.
strcpy(#hedef,kaynak)	str1 ve str2 null terminator ile sonlandırılmış karakter katarı dizileridir.	Hedef karakter katarı dizisinin taban adresi döndürülür. Kaynak karakter katarı dizisi null terminator de dahil olmak üzere hedefe kopyalanır.
strncpy(#hedef, kaynak, limit)	str1 ve str2 null terminator ile sonlandırılmış karakter katarı dizileridir. Son parametre olan limit ise negatif olmayan bir tam sayı değeridir.	Önceki fonksiyon ile aynıdır, ama belirtilen limit değeri ile sınırlı karakter katarı dizisi elemanları kopyalanır.
strlen(str)	str null terminator ile sonlandırılmış karakter katarı dizisidir.	Null terminator hariç olmak üzere karakter katarı dizisinin eleman sayısı döndürülür. Boş karakter katarı dizisi için sıfır değeri döndürür.

#### 7.4.1. İki Dizinin Karşılaştırılması

C++ programlama Dili’nde karakter katarı dizilerinin karşılaştırılması işlemi dilin tasarıımı gereği ilk elamandan başlayarak ve gerekiyorsa sıra ile diğer elamanların karşılaştırılıp sonucun döndürülmesi ile yapılmaktadır. Burada basitlik sağlanması amacı ile ASCII karakter kodu kullanıldığı kabul edilmiştir.

1. İlk karakter katarı dizisinde barındırılan veri “Ali” ve ikinci karakter katarı dizisinde barındırılan veri “Bora” olsun. Bu ikisinin karşılaştırılması durumunda ilk harfler üzerinde işlem yapılır. ASCII tablosuna göre ‘A’, ‘B’ küçük olduğu döndürülür.
2. İlk karakter katarı dizisinde barındırılan veri “Ahmet” ve ikinci karakter katarı dizisinde barındırılan veri “Ali” olsun. Bu ikisinin karşılaştırılması durumunda ilk harfler aynı olduğundan ikinci harfler üzerinde işlem yapılır. ASCII tablosuna göre ‘h’, ‘l’ küçük olduğu döndürülür.
3. İlk karakter katarı dizisinde barındırılan veri “Bora” ve ikinci karakter katarı dizisinde barındırılan veri “Boran” olsun. Bu ikisinin karşılaştırılması durumunda ilk harfler aynı olduğundan ikinci harfler üzerinde işlem yapılır. ASCII tablosuna göre ‘\0’, ‘n’ küçük olduğu döndürülür.
4. İlk karakter katarı dizisinde barındırılan veri “merhaba” ve ikinci karakter katarı dizisinde barındırılan veri “Merhaba” olsun. Bu ikisinin karşılaştırılması durumunda ilk harfler üzerinde işlem yapılır. ASCII tablosuna göre ‘m’, ‘M’ küçük olduğu döndürülür.

Aşağıdaki örnek kod blokunda karakter katarı dizisi fonksiyonlarının kullanımı gösterilmiştir.

```
char isim1[20];
char isim2[20];
char isim3[20];
int uzunluk;
strcpy (isim1, "Mahmut Tuncer");           // isim1 = "Mahmut Tuncer"
strlen (isim1);                            // 13 değerini döndürür.
uzunluk = strlen("Mahmut Tuncer")          // uzunluk değişkeni 13 barındırmaktadır.
strcpy(isim2, "Yusuf")                     // isim2 değişkeni Yusuf barındırmaktadır.
strcpy(isim3, isim2)                       // isim3 değişkeni Yusuf barındırmaktadır.
strcmp(Ali, Veli)                          // 0> değer döndürür.
```

## 7.4.2.Karakter Katarı Dizilerinin Okunması ve Yazılması

C++ programlama Dili’nde karakter katarı dizileri teknik olarak bir dizi olduğu için önceki bölümlerde değişkenler üzerinde yapılan bazı işlemlerin diziler üzerinde yapılmasının olanaklı olmadığı, böylece karakter katarı dizilerinin de bu kurala uymak durumunda olduğu belirtilmiştir. Bir dizinin elamanları üzerinde gerçekleştirilen tüm işlemler tekil eleman düzeyinde gerçekleştirilmekte olmasına rağmen bir istisna olarak tüm karakter katarı dizisinin okuma ve yazma işlemle-rinde bütün olarak gerçekleşir. Aşağıdaki karakter katarı dizisi tanımlaması izleyen bölümlerde kul-lanılacaktır.

```
char isim[22];
```

## 7.4.3.Karakter Katarı Dizilerinde Girdi İşlemleri

Yukarıdaki tanımlamayı kullandığımız bir programda eğer girdi akım değişkeni olan cin ile girdinin okunması gerekiyor ise kural olarak girdinin karakter olarak uzunluğunun 29 karaktere eşit veya az olması gereklidir. Bunun nedeni karakter katarı dizisi veri tipinde son elamanın ‘\n’ olma-sıdır. Girdi okunurken girdi akım değişkeni karakter olarak girilen veriyi kontrol etmez. Girdinin ayrılan bellek alanından büyük olması durumunda eğer işletim sistemi bellek koruması desteği sunmuyor ise, bu durumda program değişken için ayrılan bellek alanının ötesine geçerek sıradaki bellek alanındaki verinin üzerine yazacaktır. Bu durum ciddi problemlerin ortaya çıkmasına neden olabilir.

```
char isim[22];
```

```
cin >> isim;
```

Aynı girdi akım ayrıştırma operatörü “>>” okuma işlemini başlangıçta yer alan beyaz boş-lukları dikkate almadan girdiyi ayırtırıp, sonda yer alan beyaz boşluk veya geçersiz veri ile karşı-laşana işlemeye devam etmektedir. Bu nedenle karakter katarı dizisi veri tipi, içerisinde beyaz boşluk bulunan verilerin okunmasında kullanılamaz.

Bu tür verilerin okunmasında ise “get” kullanılabilir. Ancak get sadece tek bir veri okumakta olduğu için hem karakter katarı hem de karakter katarı dizisi veri tiplerini okuyacak şekilde yer verilmelidir. Bunun için de “get” iki adet parametre ile birlikte kullanılmalıdır. Aşağıdaki örnekte bu yapı gösterilmiştir.

```
char isim[22];
```

```
cin.get(isim, 22);
```

Yukarıdaki kod blokunda get iki adet parametre ile kullanılmaktadır. Birinci parametre karakter katarı dizisi, ikinci parametrede okunacak olan karakter sayısını belirtmektedir. Karakter sayısı okunurken enter tuşuna basılarak girdi işlemi sonlandırılacağı için bu durumda enter tuşuna basılarak karakter katarının sonuna eklenecek olan yeni satır karakteri olan ‘\n’ okunmayacağı için örnekte belirtildiği gibi 22 karakter değil 21 karakter okunur ve son olarak da sona null terminator eklenir.

Eğer bu kod blokunun kullanıldığı bir programda klavyeden girilen veri “Merhaba, ben Mahmut Tuncer” girilirse tanımlanmış olan uzunluk aşılmış olacaktır. Girdi 26 karakter uzunluğundadır. “isim” adlı değişkende barındırılacak olan veri ise “ Merhaba, ben Mahmut T” olacaktır.

Ayrıca klavyeden girilen verinin tamamı ön bellekte barındırılacağı için okunmayan kısım halen var olacaktır. Bu nedenle de girilen verinin yeni satır karakterine kadar kesintisiz okunmasını sağlamak için yeteri kadar büyük bir bellek alanının ayrılması yerinde olacaktır. Yukarıda verilen örnek metinin kesintisi olarak saklanması istiyorsak bu durumda karakter katarı dizisi veri tipinde yapılan tanımlamalarda dizi isimlerinin uzun metinleri temsil edecek biçimde seçilmesi programcı(lar arasın)dan kaynaklanan yanlış anlaşılmalarдан kaynaklı hataların önüne geçilmesini sağlayacaktır. Yukarıdaki tanımlamanın doğru şekli aşağıdaki gibi olmalıdır.

```
char metin1[255];
cin.get(metin1, 255);
```

Bu tanımlama ile 254 karakter uzunluğundaki metinler okunabilir ve saklanabilir.

#### 7.4.4.Karakter Katarı Dizilerinde Çıktı İşlemleri

Karakter katarları dizisinde barındırılan bir veri doğrudan G/C akım değişkenlerinden “cout <<” ile yazdırılabilir. Önceki bölümler yapılan tanımlamalar gereği çıktı operatörü işlem yaparken yeni satır değişkenine gelene kadar okuma yapacağı için karakter katarı veri tipinde yapılan tanımlamalar ile karakter katarı dizisi veri tipindeki tanımlamaların çıktıları farklı olacaktır. Aşağıdaki örnek kod blokunda veri tipi tanımlamalarının hatalı yapılması nedeni ile farklı çıktılar elde edilir.

```
include <iostream>
include <locale>
using namespace std;
int main()
{
    setlocale(LC_ALL, "tr_TR.UTF-8");
    char isim[4] = {'V', 'e', 'l', 'i'};
    int sayi1 = -5;
    int sayi2 = 11;
    cout << isim << endl;
}
```

Bu kod blokunun çalıştırılması sonucunda isim adlı değişkenin yazdırılmasının ardından bellek alanı okunmaya devam edecktir. Bunun sonucunda Veli yazdırıldıktan sonra bellek alanında sıradaki bellek adresleri okunmaya devam edeceği için çıktıda garip şeyler görülmeye olağandır. Bu tür hataların önüne geçilmesi için doğru veri tiplerinin ve ön işlemci komutlarının tanımlanması bir zorunluluktur.

#### **7.4.5. Çalışma Zamanında G/C İşlemleri için Dosya Kullanımı**

Önceki bölümlerde dosyadan okuma işlemleri için dosya akım değişkenlerinin nasıl tanımlandığı ve verinin okunması için “open” kullanımı görmüştür. Tek bir değişken ile kullanılması durumunda veri program içerisinde her zaman için tanımlanan değişkenin belirttiği dosya adından okuma yapacaktır. Bu bazı özel durumlarda, örneğin programın işleyişinin izlenmesi veya bazı işlemlere ait bilgilerin bir kayıt dosyasında saklanması gibi, uygun bir çözümüdür. Ancak kullanıcı ile etkileşimde bulunulması istenen programlarda veriler birden çok sayıdaki dosyalardan okunabilir veya yazılabilir. Bunu içinde dosya adının kullanıcı tarafından tanımlanması istenebilir. Aşağıdaki örnek programda dosya isimlerinin kullanıcıdan alınması gösterilmiştir.

```
ifstream dosya_oku;  
ofstream dosya_yaz;  
  
cout << "Verilerin okunacağı dosyanın adını yazıp enter tuşuna basınız: ";  
  
cin >> kaynak_veri_dosyasi;  
  
dosya_oku.open(kaynak_veri_dosyasi);  
  
...  
  
cout << "Verilerin yazılacağı dosyanın adını yazıp enter tuşuna basınız: ";  
  
cin >> sonuc_veri_dosyasi;  
  
dosya_yaz.open(sonuc_veri_dosyasi);
```

#### **7.4.6. Çalışma Zamanında G/C İşlemleri için Dosya İsimlerinde Karakter Katarı Kullanımı**

Önceki bölümde dosya isimlerinin kullanıcı tarafından girilmesini sağlayacak olan kod blokuna yer vermiştık. Kod blokunda dikkat edildiyse, kullanılan isimlerin veri tipi olarak herhangi bir tanımlama yapılmamıştır. Önceki bölümlerde veri tiplerinin tanımlanmasının bir zorunluluk olduğun belirtilmiştir ve dolayısı ile de yukarıdaki kod blokunda kullanılacak olan dosya isimleri için veri tipi olarak ilk akla gelecek olan karakter katarı olacaktır. C++ Programlama Dili'nin standartı gereği dosya isimleri karakter katarı olamaz. Dosya isimleri sonunda null terminator bulunan karakter katarı dizisi veri tipinde tanımlanmış olmak durumundadır.

Bu zorunluluğun gereği olarak programlarda iki çözüm yönteminden birisi seçilebilir. Birinci yaklaşım, karakter katarı dizisi veri tipinin tün karakter katarları için kullanılmasıdır. Bu durumda tüm karakter katarları üzerinde yapılacak olan işlemler buna göre kurgulanmalıdır.

İkinci yaklaşım ise karakter katarı veri tipi kullanılırken, gerektiğinde de kullanmak üzere karakter katarı dizisi veri tipi tanımlanabilir. Bir veri tipinden diğerine dönüşüm işlemi programcı tarafından gerçekleştirilir. Bu yaklaşımın iyi tarafı her iki veri tipinin sunduğu fonksiyonların program akışı içerisinde gerektiğinde kullanılabilmesinin olağan sunmasıdır.

Aşağıdaki örnek kod blokunda kullanıcı tarafından girilecek olan dosya isimleri karakter veri tipinde tanımlandıktan sonra karakter katarı dizisi veri tipine dönüştürilmektedir.

```
string kaynak_veri_dosyasi, sonuc_veri_dosyasi  
ifstream dosya_oku;  
ofstream dosya_yaz;  
cout << "Verilerin okunacağı dosyanın adını yazıp enter tuşuna basınız: ";  
cin >> kaynak_veri_dosyasi;  
dosya_oku.open(kaynak_veri_dosyasi.c_str());  
...  
cout << "Verilerin yazılacağı dosyanın adını yazıp enter tuşuna basınız: ";  
cin >> sonuc_veri_dosyasi;  
dosya_yaz.open(sonuc_veri_dosyasi.c_str());
```

Yukarıdaki örnek kod blokunda bir önceki kod bloku kullanılmıştır. Dosya okuma ve yazma işlemleri için gerekli olan karakter katarı dizisi veri tipi dönüşümü için aşağıdaki fonksiyon kullanılmıştır. Bu işlemler için programda ön işlemci yönergesi olarak string kullanılmalıdır.

```
include <string>  
...  
karakter_katarı_değişkeni.c_str();
```

## 7.5.Paralel Diziler

İki veya daha fazla sayıdaki dizilerde barındırılan değişkenler arasında ilişki bulunuyor ise bu tür dizilere **paralel diziler – parallel arrays** adı verilir. Dizilerde saklanan verilerin ilişkisi her bir dizideki karşılıklı olarak aynı indis numarasına sahip değişkenlerin birlikte işleneceği durumlarda yararlı olmaktadır.

Önceki bölümlerde örnek verilen öğrenci dönem içi değerlendirme etkinlerinde aldığı notlarının işlenerek her bir öğrenciye ait harf notlarının hesaplandığı programa geri dönecek olursak, öğrenci adı, öğrenciye ait notlar ile harf notundan oluşan bir yapı ile çalışıldığını gözlemleriz. Burada öğrenci adlarını kullanmak yerine öğrencilere ait numaraların kullanılması durumunda verilerin okunarak sadece öğrenci numarası ile buna karşılık gelen harf notundan oluşan bir işlem sonucunun elde edilmesi olanaklıdır.

Bu tasarımda, öğrenci numaraları ve harf notları bir dosyada barındırılmakta olduğunda, ilk dizinin öğrenci numaraları ve ikinci dizinin de öğrenci harf notlarını barındırması yeterlidir. Ancak her bir öğrenci numarası ile harf notunun doğru bir şekilde ilişkilendirilebilmesi için karşılıklı olarak her bir değişkenin indis numaralarının aynı olması gerekecektir.

Öğreğin ilk dizi öğrenci numarasını barındıran OgrenciNo[] ve ikinci dizi de ilgili öğrenciye ait olan harf notunu barındıran OgrenciHarfNot[] olsun. Bu durumda her iki veri kümelerinin dosyadaki durumu aşağıdaki gibi olur:

202201001 AA

202201002	CC
202201003	DC
202201004	BA
202201005	FF
202201006	AB

Eğer bu dosyadaki veriler yukarıda belirtilen dizilerde barındırılıyor olursa, iki dizi arasındaki ilişki şu şekilde gösterilebilir

OgrenciNo[0] = 202201001	OgreniHarfNot[0] = AA
OgrenciNo[1] = 202201002	OgreniHarfNot[1] = CC
OgrenciNo[2] = 202201003	OgreniHarfNot[2] = DC
OgrenciNo[3] = 202201004	OgreniHarfNot[3] = BA
OgrenciNo[4] = 202201005	OgreniHarfNot[4] = FF
OgrenciNo[5] = 202201006	OgreniHarfNot[5] = AB

Paralel dizilerin zayıf yanı ise bir dizide yapılan işlemin diğer dizi ile olan ilişkisinin dikkate alınmaması durumunda geçersiz işlem sonuçlarının elde edilebilecek olmasıdır. Bu nedenle bir dizi üzerinde işlem yapılrken diğer dizinin de aynı indis numarasına sahi olan verinin uygun şekilde işlenmesi gereklidir. Örneğin Harf notuna göre sıralama yapılacak olursa yukarıdaki dizilerin son şekli aşağıdaki gibi olmalıdır.

OgrenciNo[0] = 202201001	OgreniHarfNot[0] = AA
OgrenciNo[1] = 202201006	OgreniHarfNot[1] = AB
OgrenciNo[2] = 202201004	OgreniHarfNot[2] = BA
OgrenciNo[3] = 202201002	OgreniHarfNot[3] = CC
OgrenciNo[4] = 202201003	OgreniHarfNot[4] = DC
OgrenciNo[5] = 202201005	OgreniHarfNot[5] = FF

## 7.6.İki ve Üzeri Boyutlu Diziler

Bir programın işleyeceği veri liste biçiminde sunulmuş ise veya sunulabiliyor ise diziler bu veri üzerinde işlem yapılması uygun olacaktır. Özellikle de önceki bölümde incelene tek boyutlu diziler kullanılıyor ise. Tek boyutlu diziler her veri için uygulanamayacağı için liste biçiminde sunulan verilerin bazen çok boyutlu diziler tarafından işlenmesi daha uygun olacaktır.

Aşağıdaki örnekte İstanbul'un ilçeleri bulunduğu yaka, nüfusu, mahalle sayısı bilgileri bir iki boyutlu dizide verilmiştir. İstanbul'un ilçelerinin idari olarak Anadolu Yakası ve Avrupa Yakası olarak gruplandığı için Listedede Anadolu Yakası 1 ve Avrupa Yakası da 2 ile temsil edilmektedir.

istanbul	Bulunduğu Yaka	Nüfus	Mahalle Sayısı
----------	----------------	-------	----------------

Adalar	1	16033	5
Arnavutköy	2	296709	38
Ataşehir	1	422594	17
Avcılar	2	436897	10
Bağcılar	2	737206	22
Bahçelievler	2	592371	11
Bakırköy	2	226229	15
Başakşehir	2	469924	10
Bayrampaşa	2	269950	11
Beşiktaş	2	176513	23
Beykoz	2	246110	45
Beylikdüzü	2	365572	10
Beyoğlu	2	226396	45
Büyükçekmece	2	257362	24
Çatalca	2	74975	39
Çekmeköy	1	273658	21
Esenler	2	446276	17
Esenyurt	2	957398	43
Eyüpsultan	2	405845	29
Fatih	2	396594	57
Gaziemanpaşa	2	487778	16
Güngören	2	280299	11
Kadıköy	1	481983	21
Kağıthane	2	442415	19
Kartal	1	474514	20
Küçükçekmece	2	789633	21
Maltepe	2	515021	18
Pendik	1	726481	36
Sancaktepe	1	456861	19
Sarıyer	2	335298	38
Siliviri	2	200215	35
Sultanbeyli	1	343318	15
Sultangazi	2	537488	15
Şile	1	37904	62
Şişli	2	266793	25

Tuzla	1	273608	17
Ümraniye	1	713803	38
Üsküdar	1	520771	33
Zeytinburnu	2	283657	13

Görüleceği üzere veriler tablo biçiminde sunulmuştur. Tabloda  $39 \times 3 = 117$  adet veri bulunmaktadır. Bu tablo temel olarak tek boyutlu ve veri tipi de int olarak tanımlanana bir dizide barındırılabilir. Ancak bu durumda örneğin diziden Anadolu yakasında olanların bulunması veya hem Anadolu yakasında olan hem de nüfusu 200000 büyük olanları bulmak gibi işlemler için tek boyutlu diziler uygun olmayacağıdır. Her ne kadar bunu tek boyutlu bir dizi ile yapmak olsak olsa da karmaşık sorguların yanıtlarının doğru bir şekilde döndürülmesi için kullanılacak olan algoritma karmaşıklaşacağı gibi aynı zamanda programın kaynak kodu da aynı ölçüde karmaşık olacaktır. Bunun gibi karmaşık verilerin sunulmasında ve işlenmesinde tek boyutlu diziler pratik olmaktan uzaktır. Bu nedenle iki veya çok boyutlu diziler bu tür verilerin işlenmesi ve sunulması için daha uygundur.

**İki boyutlu diziler**, verinin bir satırlar ve sütunlar şeklinde bir tablo olarak düzenlenmesi ile kurulur. Tek boyutlu dizilerden farklı olarak iki boyutlu dizilerde dizinin satır ve sütun sayısı belirtilmelidir.

```
veritipi iki_boyutlu_dizi_adi [int sayı][int sayı];
```

İki boyutlu diziler de tek boyutlu dizilere göre önemli ölçüde farklıdır. Aradaki fark iki boyutlu dizilerde ilk tanımlata satır sayısını, ikinci tanımlama da sütun sayısını temsil etmektedir.

### 7.6.1.İki Boyutlu Dizilerde Elemanlara Erişim

Çok boyutlu dizilerde ister iki boyutlu ister daha fazla boyutlu olsun dizideki bir eleman erişmek için dizinin boyutu kadar indis kullanılmalıdır. İlk örnek olarak iki boyutlu bir dizi örneğini inceleyelim. Örnek olarak otomobil bayisinin aylık satılan araba sayısını barındıracak bir iki boyutlu dizi tanımlayalım.

```
double satilan_adet[12][5];
```

Bu dizideki ilk pozitif sayı satır sayısı ve ikinci pozitif sayı sütun sayısıdır. Satır sayısı ayları ve sütun sayısı da model çeşidini barındırmaktadır. Bu dizide eğer Mart ayında yapılan satış sayıları arasından ilk modele ait olanı diziye atamak istersek aşağıdaki şekilde yapılması gerekecektir.

```
satilan_adet[1][1] = 125456.78;
```

Bu şekilde istenen dizi elamanın değeri dizide atanabilir. Bu işlemlerin yapılması için dizinin indis değerleri kullanılacağı için indis değerleri birer değişken ile tanımlanıp kullanılabilir.

```
i = 1, j = 1;
```

```
...
```

```
satilan_adet[i][j] = 125456.78;
```

## 7.6.2.İki Boyutlu Dizilerin Etkinleştirilmesi

Tek boyutlu dizilerde olduğu gibi iki boyutlu dizilerde tanımlandığında etkinleştirilebilir. Aşağıdaki örnekte bu özellik gösterilmiştir.

```
dizi[5][2] = {{0,0},{0,0},{0,0},{0,0},{0,0}}
```

Dizinin tüm elemanları “0” olarak tanımlanmış ve etkinleştirilmiştir. Bu örnekte olduğu gibi iki ve daha fazla boyutlu dizilerin tanımlandığında etkinleştirilmesi sürecinde aşağıdaki kurallar geçerlidir:

1. Her satırın elemanları “{ }” içerisinde yazılır ve elemanlar arasında „,” yer alır.
2. Tüm elamanlar “{ }” yazılılığı gibi dizinin tüm elamanları “{ }” arasında yazılır.
3. Eğer dizide sayısal veriler yer alacak ve herhangi bir satırın elemanlarının tamamının sıfır olarak tanımlanması istenirse, sadece satırın ilk elamanı sıfır olarak tanımlanmalıdır.

## 7.6.3.İki ve Üzeri Boyutlu Dizilerde Sıralama Veri Tipi Kullanımı

Sıralama veri tipi, iki boyutlu dizilerde indis olarak kullanılabilir. Böylelikle indislerin değerlerinin sayısal olarak tanımlanması yerine doğrudan sıralama veri tipinde tanımlanan veri indis olarak kullanılabilir.

Örneğin, bir önceki dizi örneğinde yer verilen İstanbul ili Adalar İlçesi’ne ait mahalleler deki satılık veya kiralık olan arsa, konut ve işyeri sayısını bir dizide barındırmak için aşağıdaki gibi bir tanımlama yapılabilir.

```
const int satir_sayisi = 5;  
const int sutun_sayisi = 5;  
  
enum mahalleler = {Burgazada, Heybeliada, Kinalıada, Maden, Nizam};  
  
enum durum = {kiralikkonut, kiraliksiyeri, satılıkarsa, satılıkkonut, satılıkisiyeri};  
  
int gayrimenkuller [satir_sayisi][sutun_sayisi];
```

Yukarıdaki kod bloku ile **gayrimenkuller** adlı bir dizi tanımlanmıştır. Satır ve sütunların tanımlanması için **sıralama veri tipi** kullanılmıştır. Sıralama veri tipi ise **mahalleler** ve **durum** olmak üzere iki adettir. Bu dizi **5 satır ve 5 sütundan** oluşmaktadır.

Bu dizi için yapılan tanımlamalardan da anlaşılacağı üzere bir emlak ofisinin portföyünde bulunan gayri menkullerin sayısı tutulacaktır. Çok boyutlu dizilerde dizi elamanlarının tanımlanması ve erişilmesi için indislerin kullanılması gereklidir. Bu durum indislerin sayısal olarak kullanılmasına göre sıralama veri tipinin kullanımı daha pratik olmaktadır. Eğer sıralama veri tipi kullanılmamış olsaydı örneğin satılık arsaların sayısının atanması istendiğinde sütunu temsil eden indis olarak 2 kullanılması, yine aynı şekilde mahallenin ismine karşılık gelen indis numarasının belirtilmesi zorunlu olacaktır. Bunun yerinde doğrudan mahallenin adı ve gayrimenkulün durumu indis olarak kullanılacaktır.

**gayrimenkuller   kiralikkonut   kiraliksiyeri   satılıkarasa   satılıkkonut   satılıkisiyeri**

<b>Burgazada</b>				
<b>Heybeliada</b>				
<b>Kinalıada</b>				
<b>Maden</b>				
<b>Nizam</b>				

Yukarıdaki diziyi kullanarak Burgazada Mahallesi’ndeki satılık konut sayısını tanımlamak için aşağıdaki bildirim kullanılmalıdır.

gayrimenkuller[0][4]=12;

Bu bildirim yerine aşağıdaki de yazılabilir:

gayrimenkuller[Burgazda][satılıkkonut]=12;

Her iki bildiriminde işlenmesinin ardından dizin,n durumu aşağıdaki gibi olacaktır.

gayrimenkuller	kiralikkonut	kiralıkşyeri	satılıkkarasa	satılıkkonut	satılıkşyeri
<b>Burgazada</b>				15	
<b>Heybeliada</b>					
<b>Kinalıada</b>					
<b>Maden</b>					
<b>Nizam</b>					

Sıralama veri tipleri özellikle verinin sıklıkla değişmediği durumlarda yapılacak işlemler için daha kolay okunan ve yönetilebilir kaynak kod yazmak için uygun bir yaklaşım olacaktır.

İki veya üzeri boyutlu dizilerde sık yapılan dört temel işlem bulunmaktadır. Bunlar:

1. Dizinin elamanından bir tanesinin işlenmesi
2. Dizinin tüm elamanlarının işlenmesi
3. Dizinin belirli bir satırında yer alan elamanların işlenmesi
4. Dizinin belirli bir sütununda yer alan elemanların işlenmesi

Dizinin bir adet elamanın işlenmesi ile bir program içerisindeki bir değişkenin işlenmesi arasında herhangi bir fark bulunmamaktadır. İki veya üzeri boyutlu dizilerde yapılan etkinleştirme ve dizi elamanlarının yazdırılması işlemleri dizinin tamamı üzerinde yapılan işlemlerdir. Bir satırda/veya sütünde en büyük elamanının bulunması veya satırın/sütunun toplamının elde edilmesi gibi işlemler satır/sütun işlemlerine verilebilecek olan örneklerdir.

Yukarıdaki örnek üzerinden devam edelim ve Nizam Mahallesi’nde bulunan gayri menkul-lerin sayılarını atayalım. Söz konusu satır dizinin sona satırıdır. Buna göre satır ait indis numarası sabit olacak ve her bir sütun için indis sıfırdan başlayarak devam edecektir.

gayrimenkuller[5][0]=0;

```
gayrimenkuller[5][1]=0;  
gayrimenkuller[5][2]=0;  
gayrimenkuller[5][3]=0;  
gayrimenkuller[5][4]=0;  
gayrimenkuller[5][5]=0;
```

Yukarıda verilen bildirim ile bu işlemler yapılabilir ancak bunu tekil girdiler olarak tanımlamak yerine bir “**for ...**” döngüsü ile yapmak daha kolaydır.

```
for (sutun=0; sutun < SUTUN; sutun ++)  
    gayrimenkuller[5][sutun] = 0;
```

Benzer şekilde belirli bir sütundaki veriler üzerinde işlem yapmak için aşağıdaki döngü kullanılabılır. Örnekte 3 sütun üzerindeki değerlere 0 atanmıştır.

```
for (satir=0; satir < SATIR; satir ++)  
    gayrimenkuller[satir][2] = 0;
```

Bundan sonra ise yukarıda tanımlanan işlemlere ait olan algoritmalar üzerinde durulacaktır.

#### 7.6.4. İki Boyutlu Dizilerde Etkinleştirme

Dizilerde etkinleştirme işlemlerinin gerçekleştirilmesi dizinin boyutundan bağımsız bir işlemdir. Yukarıda görülen örnek kaynak kodu kullanarak bu işlemleri gerçekleştirebiliriz. Örneğin dizinin bir satırının tamamının etkinleştirilmesi örneğin ilk satırın tüm elemanlarının 0 atanması için aşağıdaki döngü kullanılabilir.

```
for (sutun=0; sutun < SUTUN; sutun ++)  
    gayrimenkuller[0][sutun] = 0;
```

Dizinin tüm elemanlarının etkinleştirilmesi için ise önce ilk satır ve sonra da izleyen tüm satırlardaki elemanlar 0 olarak atanmalıdır. Bunun için de iç içe kurgulanmış olan “**for ...**” döngüleri kullanılır.

```
SATIR = 5, SUTUN = 5;  
  
for (satir=0; satir < SATIR; satir ++)  
{  
    for (sutun=0; sutun < sutun; sutun ++)  
        gayrimenkuller[satir][sutun] = 0;  
}
```

### **7.6.5.İki Boyutlu Dizilerde Elemanların Yazdırılması**

İki veya üzeri boyutlu dizilerde bir dizi elemanın yazdırılması için elemanın indis numaraları ile çağrılmazı gereklidir. Yukarıda verilen örneklerdeki kaynak kod kullanılarak yapılabilir: Emlak örneğini kullanarak yukarıda verilen tablodan bir verinin yazdırılması aşağıdaki örneklerde gösterilmiştir:

Tüm dizi elamanlarının yazdırılması için:

```
for (satir=0; satır < SATIR; satır ++)  
{  
    for (sutun=0; sutun < sutun; sutun ++)  
        cout << gayrimenkuller[satir][sutun] << endl;  
}
```

Dizi içerisinde sadece belirli bir elemanın yazdırılması için:

```
cout << gayrimenkuller[3][1] << endl;
```

Sıralama veri tipi kullanılarak yazdırılabilir:

```
cout << gayrimenkuller[Nizam][kiralikkonut] << endl;
```

### **7.6.6.İki Boyutlu Dizilerde Elemanların Atanması**

İki veya üzeri boyutlu dizilerde bir dizi elemanlarına atam yapılması yazdırma işlemi ile aynı olmak birlikte cout yerine cin kullanılır.

Tüm dizi elamanlarına atama yapmak için:

```
for (satir=0; satır < SATIR; satır ++)  
{  
    for (sutun=0; sutun < sutun; sutun ++)  
        cin >> gayrimenkuller[satir][sutun];  
}
```

Dizi içerisinde sadece belirli bir elemana atama yapmak için:

```
cout >> gayrimenkuller[3][1];
```

Sıralama veri tipi kullanılarak atama yapılabilir:

```
cout >> gayrimenkuller[Nizam][kiralikkonut];
```

### **7.6.7.İki Boyutlu Dizilerde Satırların Toplanması**

İki veya üzeri boyutlu dizilerde herhangi bir satırdaki veriler ancak dizi sayısal veri içeriyor ise toplanabilir. Bu toplama işleminin aritmetik toplama işlemi olduğu unutulmamalıdır. Dizinin herhangi bir satırının toplamını elde etmek için:

```

satir_toplam = 0;
for (sutun=0; sutun < sutun; sutun++)
    satir_toplam = satir_toplam + dizi[satir][sutun];

```

Dizi içerisindeki tüm satırların toplamının ayrı ayrı bulunması içinde:

```

for (i=0; i < satir ; satir++)
{
    satir_toplam = 0;
    for (j=0; j < sutun; << sutun++)
        satir_toplam = satir_toplam + dizi [i][j];
    cout << satır toplamı: " << satir_toplam << endl;
}

```

### 7.6.8.İki Boyutlu Dizilerde Sütunların Toplanması

İki veya üzeri boyutlu dizilerde herhangi bir sütundaki veriler ancak dizi sayısal veri içeriyor ise toplanabilir. Bu toplama işleminin aritmetik toplama işlemi olduğu unutulmamalıdır. Dizinin herhangi bir sütunun toplamını elde etmek için:

```

sutun_toplam = 0;
for (j=0; j < sutun; j ++)
    sutun_toplam = sutun_toplam + dizi[satir][sutun];

```

Dizi içerisindeki tüm sütunların toplamının ayrı ayrı bulunması içinde:

```

for (i=0; i < satir ; satir++)
{
    for (j=0; j < sutun; << sutun++)
    {
        sutun_toplam = 0;
        sutun_toplam = sutun_toplam + dizi [i][j];
        cout << sütun toplamı: " << sutun_toplam << endl;
    }
}

```

### 7.6.9.İki Boyutlu Dizilerde Satır ve/veya Sütunlardaki En Büyük Değerin Bulunması

İki boyutlu dizilerde bir satırındaki ve/veya sütündeki en büyük elemanın bulunması yaygın olarak yapılan işlemlerden birisidir. Bu problemin çözümü, eğer iki boyutlu dizinin satır veya sütün düzeyindeki dizi elamanları olarak problem yaklaşımıında, aslında bir dizideki en büyük veya en küçük elamanın bulunması probleminden farklı olmadığı görülür. Önceki bölümlerde incelenen

doğrusal arama ve seçerek sıralama algoritmaları bu tür problemlerin çözümünde temel alınabilecek olan algoritmalarıdır. Konuyu daha iyi açıklamak için 5 satır ve 6 sütundan oluşan ve MATRIS adlı bir iki boyutlu dizi üzerinde işlemlerin gerçekleştiğini var sayalım. Buna göre algoritmaların hazırlanacağı ve uygulanacağı dizi aşağıdaki gibi olacaktır.

MATRIS	0	1	2	3	4	5
0						
1						
2						
3						
4						

İlk olarak ikinci satır yani indis numarası 1 olan satırda en büyük elemanı aramak içim satır elemanları taranır. İlk eleman en büyük eleman olarak kabul edilir ve sırası ile kalan elamanlar en büyük eleman ile karşılaştırılır. Eğer en büyük elamandan büyük olan bir diz elemanı bulunursa bu en büyük eleman olarak kabul edilere taramaya devam edilir. Tüm elemanlar tarandıktan sonra işlem sona erer.

MATRIS	0	1	2	3	4	5
0						
1	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]
2						
3						
4						

```

satir = 1;
en_buyuk = MATRIS[1][0];
for (i=0; i < sutun ; sutun ++)
{
    if (MATRIS[satir][sutun] > en_buyuk)
        en_buyuk = MATRIS[satir][sutun];
}

```

Eğer tek bir satır değil de tüm satırlar içerisinde yer alan tüm satırlar içerisindeki en büyük elamanın bulunması istenirse aşağıdaki şekilde arama yapılır.

```

for (i = 0; i < SATIR_SAYISI; i++)
{
    en_buyuk = MATRIS[i][0];
}

```

```

for (j=1; j < SUTUN SAYISI; j++)
{
    if (MATRIS[i][j] > en_buyuk)
        en_buyuk = MATRIS[i][j];
}
cout << "Tüm satırlar içerisinde yer alan en büyük eleman: " << en_buyuk << endl;

```

Eğer tek bir sütun değil de tüm sütunlar içerisinde yer alan tüm sütunlar içerisindeki en büyük elamanın bulunması istenirse aşağıdaki şekilde arama yapılır.

MATRIS	0	1	2	3	4	5
0	[0][1]					
1	[0][2]					
2	[0][3]					
3	[0][4]					
4	[0][5]					

```

for (i = 0; i < SUTUN SAYISI; i++)
{
    en_buyuk = MATRIS[0][i];
    for (j=1; j < SATIR SAYISI; j++)
    {
        if (MATRIS[i][j] > en_buyuk)
            en_buyuk = MATRIS[i][j];
    }
}
cout << "Tüm sütunlar içerisinde yer alan en büyük eleman: " << en_buyuk << endl;

```

### 7.6.10. İki Boyutlu Dizilerin Etkinleştirilmesi İçin Diğer Yöntemler

İki boyutlu dizilerin kullanılacağı durumlarda, bir dizinin tanımlanması ve etkinleştirilmesi için öncekinden farklı olarak dizi boyutunun bilinmesi ve bunun bir sabit değer ile `typedef` kullanılarak tanımlanması olanaklıdır. Örneğin bir iki boyutlu dizinin satır ve sütun sayısının üst sınırının 20 olduğunu bildiğimizi var sayalım. Bu dizinin yapısında barındıracağı verinin kayar noktalı olacağı da ayrıca biliniyor olsun. Bu durumda diziye ilişkin gerekli tanımlamalar yapıldığında, dizi

program içerisinde tanımlanan değişkenler ve parametreler aracılığı ile tanımlanabilir ve aynı şekilde etkinleştirilebilir. Aşağıdaki örnekte bu uygulama gösterilmiştir.

```
// Dizinin satır sayısı  
cont int SATIR_SAYISI = 20;  
  
// Dizinin sütün sayısı  
cont int SUTUN_SAYISI = 20;  
  
// dizinin barındıracağı ver tipi ve dizi tanımlaması  
typedef double Liste_Tablo[SATIR_SAYISI][SUTUN_SAYISI];
```

Bu kod bloku dizinin tanımlanması için gerekli olan tüm bilgileri içermektedir. Bu aşamadan sonra ise ana fonksiyon içerisinde gerekli bildirimler kullanılarak fonksiyon tanımlanabilir. Dizi için isim olarak veriler kullanılmıştır.

Liste\_Tablo veriler;

Dizinin yapısında verileri barındırabilmesi için etkinleştirme işleminin yapılması gereklidir. Bunun için aşağıdaki fonksiyondan yararlanılır.

```
void DiziEtkinlestir(Liste_Tablo isim)  
{  
    for (int satir = 0; satir < SATIR_SAYISI; satir++)  
        for (int sutun = 0; sutun < SUTUN_SAYISI; sutun++)  
            isim[satir][sutun] = 0;  
}
```

Bu yaklaşım C++ Programlama Dili yapısında iki boyutlu dizilerin bir fonksiyona parametre olarak aktarılması durumunda dikkat edilmesi gereken bazı unsurları kullanmaya gerek kalmadan doğrudan fonksiyonun adının parametre olarak aktarılıp gereken işlemin yapılmasını sağlamaktadır. Bunun işlevsel olmasını nedeni ise gerekli veri tipi tanımlamalarının önceden yapılmış olmasıdır.

### 7.6.11. İki Boyutlu Dizilerin Fonksiyonlara Parametre Olarak Aktarılması

C++ Programlama Dili’nde kural olarak fonksiyonlara, dizi parametre olarak aktarılabilir. Bu işlem ise dizinin taban adresinin fonksiyona aktarılması ile gerçekleşir. Tek boyutlu dizilerde bu durum herhangi bir özel işlem gerektirmeden doğrudan yapılabilir. Ancak iki boyutlu dizilerde ise dizinin satır ve sütün sayısının tanımlanması gerekmektedir. Bunu nedeni ise C++ programlama Dili’nde iki boyutlu dizilerin bellekte nasıl barındırıldığından yatkınlıkta. İki boyutlu dizilerde dizilerin satır sayısı yani satırların sırası esas alınarak bellek üzerinde veri yerleştirilmektedir. Buna da **sıralama biçimini – order form** adı verilir.

İki boyutlu dizinin referans olarak fonksiyona aktarılması durumunda dizi boyutuna ait olan ilk bilgi olan satır sayısı dikkate alınmayabilir. Ancak sütün sayısının kesinlikle tanımlanması gereklidir.

Örneğin aşağıdaki program örneğinde iki boyutlu olarak tanımlanan bir dizinin yapısında bulunan ve önceki bölümlerde yer verdiğimiz bazı işlemelere ait olan algoritmalar uygulanmıştır.

```
/*
* iki_boyutlu_dizi_islemleri.cxx
*
*
* Copyright 2022 Goksin Akdeniz <goksin.akdeniz@gmail.com>
* *
* Her hakkı saklıdır.
* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
* takdirde serbesttir:
*
* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
*
* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
*
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
* VERİ VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
```

```
* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
* SORUMLU DEĞİLLERDİR.

/*
*/
/*
```

// İki boyutlu dizi parametre olarak fonksiyona aktarılmaktadır.

```
#include <iostream>
#include <locale>
#include <iomanip>
using namespace std;
const int SATIR_SAYISI = 6;
const int SUTUN_SAYISI = 5;
void Yazdir(int iki_boyutlu_dizi[][SUTUN_SAYISI],int SATIR_SAYISI);
void SatirToplam(int iki_boyutlu_dizi[][SUTUN_SAYISI],int SATIR_SAYISI);
void SatirEnBuyukEleman(int iki_boyutlu_dizi[][SUTUN_SAYISI],int SATIR_SAYISI);
int main()
{
    // sıralama düzeni anlaşılması için bu şekilde yazılmıştır.
    int tablo[SATIR_SAYISI][SUTUN_SAYISI] ={{17, 8, 24, 10, 28},
                                                {9, 20, 16, 55, 90},
                                                {25, 45, 35, 8, 78},
                                                {5, 0, 96, 45, 38},
                                                {76, 30, 8, 14, 28},
                                                {9, 60, 55, 62, 10}};

    Yazdir(tablo, SATIR_SAYISI);
    cout << endl;
    SatirToplam(tablo, SATIR_SAYISI);
    cout << endl;
    SatirEnBuyukEleman(tablo, SATIR_SAYISI);
```

```

    return 0;
}

void Yazdir(int iki_boyutlu_dizi[][SUTUN_SAYISI], int satir_sayisi)
{
    for (int satir = 0; satir < satir_sayisi; satir++)
    {
        for (int sutun = 0; sutun < SUTUN_SAYISI; sutun++)
            cout << setw(5) << iki_boyutlu_dizi[satir][sutun] << " ";
        cout << endl;
    }
}

void SatirToplam(int iki_boyutlu_dizi[][SUTUN_SAYISI], int satir_sayisi)
{
    int toplam = 0;
    // Her bir satirin toplamı
    for (int satir = 0; satir < satir_sayisi; satir++)
    {
        for (int sutun = 0; sutun < SUTUN_SAYISI; sutun++)
            toplam = toplam + iki_boyutlu_dizi[satir][sutun];
        cout << (satir + 1) << " Satırındaki elemanların toplamı = " << toplam << endl;
    }
}

void SatirEnBuyukEleman(int iki_boyutlu_dizi[][SUTUN_SAYISI], int satir_sayisi)
{
    int largest;
    // Her bir satirdaki en büyük eleman
    for (int satir = 0; satir < satir_sayisi; satir++)
    {
        en_buyuk = iki_boyutlu_dizi[satir][0];
        for (int sutun = 1; sutun < SUTUN_SAYISI; sutun++)
            if (en_buyuk < iki_boyutlu_dizi[satir][sutun])

```

```

        en_buyuk = iki_boyutlu_dizi[satir][sutun];
        cout << (satir + 1) << " Satırındaki en büyük eleman = " << en_buyuk << endl;
    }
}

```

Program derlenip çalıştırılırsa aşağıdaki gibi bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/]$ ./secerek_siralama
```

```

17  8   24  10  28
 9  20  16  55  90
25  45  35  8   78
 5  0   96  45  38
76  30   8   14  28
 9  60  55  62  10

```

1 Satırındaki elemanların toplamı = 87

2 Satırındaki elemanların toplamı = 277

3 Satırındaki elemanların toplamı = 468

4 Satırındaki elemanların toplamı = 652

5 Satırındaki elemanların toplamı = 808

6 Satırındaki elemanların toplamı = 1004

1 Satırındaki en büyük eleman = 28

2 Satırındaki en büyük eleman = 90

3 Satırındaki en büyük eleman = 78

4 Satırındaki en büyük eleman = 96

5 Satırındaki en büyük eleman = 76

6 Satırındaki en büyük eleman = 62

```
[goksin@tardis ~/projeler/C++_Notlar/]$
```

## 7.6.12.Karakter Dizileri

Karakter dizileri üzerinde yapılacak olan bir çok işlem söz konusudur. Bunlardan en yaygın olarak kullanılanı, karakter dizilerinin alfabetik olarak sıralanmasıdır. Bu işlemler için karakter dizilerinin kullanılması özellikle de karakter dizisi fonksiyonları ile birleştirildiğinde algoritmanın

hazırlanması ve programın yazılmasını çok olaylaştırmaktadır. Bunu hem karakter katarı – string hem de karakter katarı dizisi – C-string olarak yapabiliriz.

### 7.6.13.Karakter Dizileri ve Dizi Kullanımı

Eğer veri tipi karakter katarı olarak tanımlanmış ise bu durumda karakter katarını yapısında bulunan karakter sayısına eşit büyüklükte bir dizi tanımlayarak bu dizide tanımlayabiliriz. Burada dikkat edilmesi gereken katar katarının sonunda null terminator bulunmalıdır. Eğer verinin büyülüğu yani karakter katarının büyülüğu 50 adet olsun. Bu durumda bu karakter katarını barındırmak için 50 elamanlı bir dizi tanımlanabilir.

```
string veri[50];
```

Bu tanımlama veri adlı dizinin karakter katarı veri tipinde ve 50 adet elemanı bulunduğu bildirilmiştir. Bu şekilde tanımlanan bir dizide karakter katarlarında gerçekleştirilen atama, girdi ve çıktı işlemleri aynı şekilde yapılabilir. Dizi tek boyutlu bir dizi olduğu için, tek boyutlu dizilerde olduğu gibi işlenebilir.

### 7.6.14.Karakter Katarı Dizileri ve Dizi Kullanımı

Eğer veri tipi karakter katarı dizisi olarak tanımlanmış ise bu durumda tek boyutlu dizi işlemlerinin yapılabilmesi yanında dizilerde iki boyutlu dizilerde bu yapıda kullanılabilir. Aşağıda gösterilen tanımlamada veriler adlı dizi karakter veri tipinde tanımlanmış, 15 sütun ve 10 satırdan oluşmaktadır.

```
char veriler[15][5];
```

Bu tanımlamada dikkat edilmesi gereken nokta, karakter katarı dizilerinin tanımlama manluğunun bir metinin tek satır ve çok sayıda sütundan oluşmakta olduğunu. Dolayısı ile her bir satır tekil bir dizi olarak kabul edilerek işlem yapılabilir.

veriler	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
veri[0]															
veri[1]															
veri[2]															
veri[3]															
veri[4]															

Bu veri yapısının ilk bileşeni olan veri[0] atama yapmak için aşağıdaki bildirim kullanılır.

```
strcpy(list[0], "Mahmut Tuncer")
```

Bu bildirimin işlenmesinin ardından veri yapısının durumu aşağıdaki gibidir.

veriler	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
veri[0]	M	a	h	m	u	t		T	u	n	c	e	r	\0	
veri[1]															
veri[2]															

veri[3]												
veri[4]												

Yukarıdakine benzer şekilde karakter katarı dizisi veri yapısı kullanılarak her bir satırda bir metin atanması için bir “**for ...**” döngüsü kullanılabilir. Aşağıdaki örnek kod bloku kullanılabilir.

```
for {int i =0; i < 15; i ++)
    cin.get(veriler[i], 15);
```

Yukarıdaki kod bloku, en çok 15 karakter uzunluğunda olan karakter katarı dizisi veri tipinde tanımlanmış olan veriyi yukarıda belirtilen örnek veri yapısı içerisinde barındırılmasını sağlar. Eğer bu veri yapısının içinde barındırılan verileri okumak ve yazdırmak için ise aşağıdaki kod bloku kullanılabilir.

```
for {int j =0; j < 15; j ++)
    cout << veriler[j] << endl;
```

Yukarı verilen örnekleri daha da ileri götürmek ve döngünün sadece karakter katarı dizisinin eleman sayısı kadar okuma yapmasını ve böylece sınır aşımı olmadan işlemlerin gerçekleşmesini sağlamak için **strcmp** ve **strlen** fonksiyonları da kullanılabilir.

### 7.6.15.Çok Boyutlu Diziler

Bu bölümün başında dizileri aynı veri tipinde olan birden çok sayıdaki elamanın topluluğu olarak tanımlamıştır. Dizler tek boyutlu ve iki boyutlu olarak incelendi. Ama dizilerin teknik olarak iki boyuttan daha fazla boyut sahip olması olanaklıdır. Örneğin analitik geometri problemleri kartezyen koordinat sistemi üzerinde incelenirken problem iki boyutlu diziler kullanılarak çözülebilir. Ancak problem üç boyutlu uzayda ele alınacak olursa üç boyutlu dizilerin kullanılması daha uygun olacaktır. Aşağıda bildirim yapısında çok boyutlu dizilerin yani dizinin boyut sayısının birden fazla olması durumunda dizinin tanımlanması görülmektedir.

```
veri_tipi_bildirim dizi_adı [eleman_sayıısı1][eleman_sayıısı2] ... [eleman_sayıısıN];
```

Benzer şekilde bir çok boyutlu dizin herhangi bir elemanına erişmek için de aşağıdaki bildirim yapısı kullanılabilir.

```
dizi_adı [indis1][indis2] ... [indisN];
```

Çok boyutlu dizilerde gerçekleştirilen işlemler, bir iki boyutlu dizilerde gerçekleştirilen işlemlerden farklı değildir. Değişen sadece boyut sayısının değişmesi ile birlikte işlemler için geliştirilecek olan algoritmaların ve yapılacak olan kaynak kodun boyut sayısına ve ilgili boyutun elaman sayısının aşılmadan işlemleri gerçekleştirilmesi gerekmekte olduğudur.

Çok boyutlu dizileri bir fonksiyona formal parametre olarak tanılanması durumunda iki boyutlu dizilerdeki gibi işlemler yapılmalıdır. Dizinin ilk boyutu ihmali edilemeyecek olsa, diğer boyutları kesinlikle tanımlanmalıdır. Ayrıca diziler fonksiyonlara referans parametresi olarak tanımlandığı için bir dizi üzerinde işlem yapan fonksiyonun dizinin veri tipini değer olarak döndürmeyeceği unutulmamalıdır.



## 8.Birden Çok Çeşitteki Veriyi Barındıran Yapılar (struct)

Önceki bölümlerde birden çok sayıdaki verinin tümünün veri tipinin aynı olması durumunda tanımlanan bir dizi içerisinde barındırılabilceğini ve üzerinde işlemler yapılabileceğini görmüştük. Böylece veri tipi aynı olan verilerin dizide barındırılması ile üzerinde işlem yapılması kolaylaşmaktadır. Ancak bu yaklaşım programcayı kısıtladığı nokta veri tipinin tüm veriler için aynı olması zorunluluğudur. Bunu aşmak için ise paralel ve çok boyutlu diziler kullanılabilecek olsa da dizinin eleman sayısına bağlı olarak yapılandırılmış olması problemlere yol açabilmektedir.

Bu olası problemlerin ortadan kaldırılması için, bir problemde üzerinde işlem yapılacak olan verinin farklı veri tiplerinde tanımlanmış olan bileşenlere ayrılarak bunların bir yapı içerisinde tanımlanması uygun olacaktır. Örneğin bir öğrencinin kimlik bilgileri (adı, soyadı, doğum tarihi, ikametgah bilgileri vb), aldığı dersler, derslerde uygulanan ölçme ve değerlendirme etkinliklerinde aldığı notlar ile bunlara bağlı olarak hesaplanan başarı notu ve harf notu, devamsızlık durumu gibi bir çok farklı bilginin işlenebilmesi için bunların tek bir yapı içerisinde toplanması, hepsinin farklı dosyalarda ve farklı veri tiplerinde tanımlanarak saklanması göre daha az hataya açık bir durumdur.

C++ Programlama Dili’nde bu tür farklı veri tiplerinde tanımlanabilecek çeşitli bilgilerin birlikte tek bir yapılandırılmış veri tipinde tanımlanması için **struct** ile yapılır. Bazı programlam dillerinde bu terimin karşılığı **record** olarak yer alır. C Programlama Dili’nde de benzer olarak aynı şekilde **struct** kullanılır. Burada ele alınan konu algoritmalar konusundan sonra gelecek olan bir diğer yapılandırılmış veri tipi olan **class** için alt yapı oluşturmaktadır.

### 8.1.struct (record)

Önceki bölümde örnek verilen öğrenci verilerini işleyecek olan bir program yazılacağını düşünelim. Öğrenci verisi, öğrencinin adı, öğrencinin soyadı, öğrencinin numarası, öğrenciye ait olan kimlik bilgileri, öğrencinin kayıtlı olduğu program, öğrencinin aldığı dersler, öğrencinin başarı durumu vb bir çok farklı veriden oluşmaktadır. Bu durumda tek bir dizi tanımlayarak bu farklı veri tiplerini tek bir yapı içerisinde barındırmak olanaklı değildir. Bunun yerine öğrenci verisi kapsamında yer alan tüm verilerin farklı veri tiplerinde tanımlanarak bir yapı içerisinde barındırılması daha uygundur. Bu yapı özellikle de farklı veri tiplerinde tanımlanan verinin işlenmesinde fonksiyonların kullanılması söz konusu olduğunda çok daha etkin ve verimli çalışılmasını sağlamaktadır.

Burada bir tanım yapılması konunun daha iyi anlaşılması için gereklidir.

**struct;** belirli sayıdaki bileşenlerin bir araya getirilmesi ile oluşan, her bir bileşenin tanımlandığı adı ile erişilebildiği ve üzerinde tanımlandığı veri tipine bağlı olarak gerçekleştirilecek olan işlemlerin bilindiği kurgulanmış yapıdır.

Yukarıdaki tanımadan anlaşılacağı üzere **struct** bir yapılandırılmış veri tipidir. Yapısında birden çok farklı veri tiplerinin bir arada kullanılmaktadır. Bu özelliği nedeni ile karmaşık veri işlemleri için uygundur. C++ Programlama Dili’nde **struct** özel sözcükler arasında yer alır.

C++ Programlama Dili’nde yapılandırılmış veri tipi olan **struct** aşağıdaki görüldüğü gibi tanımlanır.

```

struct Kurgulanmis_Veritipi_Adı
{
    veri tipi 0    tanımlayıcı0;
    veri tipi 1    tanımlayıcı1;
    veri tipi 2    tanımlayıcı2;
    ....
    veri tipi n    tanımlayıcıN;
};

```

Yukarıdaki yapıda görüleceği üzere tanımlama her ne kadar “{ }” arasında yer alan bildirimlerden oluşan bir kod blokunu andırmakta olsa da aslında bir bileşik ifade olarak değerlendirilmez. Bu nedenle de son ”}” ardından ”;” konulmaktadır.

Aşağıdaki örnekte farklı özelliklere sahip bilgisayarları tanımlamak için kurgulanmış olan bir veri yapısı tanımlaması görülmektedir.

```

struct BilgisayarModelleri
{
    string islemci_ureticisi;                      //1. eleman
    string islemci_modeli;                          //2. eleman
    int     islemci_sayisi;                         //3. eleman
    int     islemci_cekirdek_sayisi;                 //4. eleman
    string ekran_karti_ureticisi;                   //5. eleman
    string ekran_karti_modeli;                     //6. eleman
    int     bellek_miktari;                        //7. eleman
    string sabit_disk_ureticisi;                   //8. eleman
    int     sabit_disk_kapasitesi;                  //9. eleman
    string anakart_ureticisi;                      //10. eleman
    string anakart_modeli;                         //11. eleman
    int     sata_port_hizi;                        //12. eleman
    int     sata_port_sayisi;                      //13. eleman
    double usb_port1_tipi;                        //14. eleman
    double usb_port2_tipi;                        //15. eleman
    int     usb_port1_sayisi;                      //16. eleman
};

```

```

        int    usb_port2_sayisi;           //18. eleman
        string Nic_yonga;                //19. eleman
        int    Nic_hizi;                 //20. eleman
    };

```

Yukarıdaki örnekte BilgisayarModelleri adlı yapıda üye sayısı 20 adettir. Yapıda yer alan her bir üyenin veri tipi tanımlanmıştır. Örneğin Nic\_hizi adlı üye int veri tipinde tanımlanmış iken, sabit\_disk\_ureticisi adlı üye ise string veri tipinde tanımlıdır.

C++ Programlama Dili ile yazılan her programda yapılan her değişken tanımlamasında olduğu gibi **struct**'ta bir tanımlamadır ve bir bildirim değildir. Bunun anlamlı **struct** ile yapılan tanımlama ile bellekte herhangi bir alan ayrılmaz. Bir veri tipi tanımlandığında bu veri tipinde değişken tanımlaması yapılabilir. Aşağıdaki örnekte yukarıdaki yapı esas alınarak birden çok sayıda değişken tanımlaması yapılmıştır.

```

// Değişken tanımlaması

BilgisayarModelleri DesktopModel;
BilgisayarModelleri WorkStationModel;
BilgisayarModelleri SunucuModel;

```

Bu işlemin ardından bellekte **struct** ile tanımlanan veri tipleri ile tanımlar için yeterli bellek ayrılacaktır.

### **DesktopModel**

islemci_ureticisi	
islemci_modeli	
islemci_sayisi	
islemci_cekirdek_sayisi	
ekran_karti_ureticisi	
ekran_karti_modeli	
bellek_miktari	
sabit_disk_ureticisi	
sabit_disk_kapasitesi	
anakart_ureticisi	
anakart_modeli	
sata_port_hizi	
sata_port_sayisi	
usb_port1_tipi	
usb_port2_tipi	
usb_port1_sayisi	

usb_port2_sayisi	
Nic_yonga	
Nic_hizi	

Sıralama veri tipinde olduğu gibi değişken tanımlaması yapılması için önce struct ile gerekli bildirimin yapılması ve en sonraki “}” sonuna ise ilgili değişkenin tanımlanması da yeterlidir. Bu gösterim şeklinde ilgili yapıya ait tek bir değişken tanımlanabileceği unutulmamalıdır. Eğer aynı yapıya ait birden çok sayıda değişken tanımlanacak ise yukarıdaki örnek kullanılmalıdır.

### 8.1.1.struct ile Tanımlanan Yapıdaki Üyelere Erişim

Dizilerde bir elemana erişim için dizinin adı ile dizideki göreceli konum yani indis bilgisi bir dizi elemanına erişmek için yeterli olmaktadır. Öte yandan struct ile kurgulanmış bir yapıda bir üyeye erişmek için yapı ile ilişkili değişken adı ile üyenin adı gereklidir. Değişken ile üye arasında ise “.” yer alır.

```
struct_Değişken_Adı.üyenin_adı
```

Bu değişken tanımlamasının diğer değişken tanımlamaları ile karşılaştırıldığında herhangi bir farkı olmadığı görülebilir. Bunun anlamı diğer değişken tanımlamalarında olduğu gibi aynı kuralların geçerli olduğunu göstermektedir. Ayrıca diğer değişkenler üzerinde yapılan tüm işlemler burada da aynı şekilde yapılabilir.

C++ Programlama Dili’nde “.” üye erişim operatörü – member access operator – olarak tanımlanır.

Aşağıdaki örnekte bir öğrencinin kayı olduğu bir derste aldığı notları işleyerek başarı notunu hesaplayan bir programda kullanılan yapı gösterilmiştir.

```
struct BasariNotu
{
    int      Ogrenci_No;
    string  Ogrenci_Adi;
    string  Ogrenci_Soyadi;
    double   Ara_Sinav_Not;
    double   Kisa_Sinav1_Not;
    double   Kisa_Sinav2_Not;
    double   Odev_Not;
    double   Donem_Sonu_Not;
    double   Basari_Notu;
    string  Harf_Notu;
```

```

};

// Değişken tanımlamaları

BasariNotu Kayitli_Ogrenci;

BasariNotu Secilen_Ogrenci;

```

Bu örnek yapıyı kullanarak bir öğrenci bilgilerinin dosyaya yazılması durumunu düşünelim. Örnek olarak öğrenci numarası 2202022 ve adı ile soyadı Baturalp Kovan olan bir öğrenciye ait olan verileri girmek için aşağıdaki gibi işlem yapılacaktır.

```

cin >> Secilen_Ogrenci.Ogrenci_No;
cin >> Secilen_Ogrenci.Ogrenci_Adi;
cin >> Secilen_Ogrenci.Ogrenci_Soyadi;

```

Bu kod bloku çalıştırıldığında okunan veri ilgili değişkenlerde belirtilen alanlarda saklanır.

### **Secilen\_Ogrenci**

Ogrenci_No	2202022
Ogrenci_Adi	Baturalp
Ogrenci_Soyadi	Kovan
Ara_Sinav_Not	
Kisa_Sinav1_Not	
Kisa_Sinav2_Not	
Odev_Not	
Donem_Sonu_Not	
Basari_Notu	
Harf_Notu	

Diğer üyelere henüz atama yapılmadığı için ilgili alanlar boş durumdadır.

### **8.1.2.struct ile Tanımlanan Yapıdaki Üyelere Atama Yapmak**

Burada **struct** ile tanımlanmış olan yapının üyelerine yukarıdaki örnekte görüldüğü gibi doğrudan atama yapabiliriz.

Bu değişken tanımlamasının diğer değişken tanımlamaları ile karşılaştırıldığında herhangi bir farkı olmadığı görülebilir. Bunun anlamı diğer değişken tanımlamalarında olduğu gibi aynı kuralların geçerli olduğunu söylemektedir. Ayrıca diğer değişkenler üzerinde yapılan tüm işlemler burada da aynı şekilde yapılabilir. Örneğin Secilen\_Ogrenci adlı değişkende barındırılan verileri Kayitli\_Ogrenci adlı değişken doğrudan kopyalanabilir.

```
Kayitli_Ogrenci = Secilen_Ogrenci
```

Yukarıdaki bildirim aşağıdaki bildirimler ile aynıdır.

```
Kayitli_Ogrenci.Ogrenci_No = Secilen_Ogrenci.Ogrenci_No;  
Kayitli_Ogrenci.Ogrenci_Adi = Secilen_Ogrenci.Ogrenci_Adi;  
Kayitli_Ogrenci.Ogrenci_Soyadi = Secilen_Ogrenci.Ogrenci_Soyadi;  
Kayitli_Ogrenci.Ara_Sinav_Not = Secilen_Ogrenci.Ara_Sinav_Not;  
Kayitli_Ogrenci.Kisa_Sinav1_Not = Secilen_Ogrenci.Kisa_Sinav1_Not;  
Kayitli_Ogrenci.Kisa_Sinav2_Not = Secilen_Ogrenci.Kisa_Sinav2_Not;  
Kayitli_Ogrenci.Odev_Not = Secilen_Ogrenci.Odev_Not;  
Kayitli_Ogrenci.Donem_Sonu_Not = Secilen_Ogrenci.Donem_Sonu_Not;  
Kayitli_Ogrenci.Basari_Notu = Secilen_Ogrenci.Basari_Notu;  
Kayitli_Ogrenci.Harf_Notu = Secilen_Ogrenci.Harf_Notu;
```

### 8.1.3.struct ile Tanımlanan Yapıdaki Üyeler Arasında İlişkisel Operatör İşlemleri

Bir yapı ile tanımlanmış olan değişkenler arasında ilişkisel operatörler ile işlem yapılabilir. Ancak ilişkisel operatörlerin kullanılabilmesi için atama konusunda olduğu gibi doğrudan yapılamaz. İlişkisel operatörler ile yapılacak olan işlemler için yapıyı oluşturan üyeler arasında yapılabilir. Bu işlemler yapılırken üyeleri arasındaki ilişkisel operatörün mantık olarak doğru üyeleri karşılaştırması doğrudur. Örneğin yukarıdaki öğrenci örneğinden hareketle bir öğrencinin dersi alıp olmadığı kontrol edilmesi örneğini inceleyelim. Öğrencinin dersi alan öğrenciler arasında olup olmadığı kontrol edilmesi için aranan öğrencinin adının ve soyadının kayıtlı olan öğrencilerin adları ve soyadları arasında olup olmadığı kontrol edilmesi gereklidir. Bu işlemin daha kolay yolu ise öğrenci numaralarından yararlanmaktır. İlk örnek öğrenci numaralarını kullanmaktadır.

```
if (Secilen_Ogrenci.Ogrenci_No == Kayitli_Ogrenci.Ogrenci_No)
```

```
...
```

Eğer öğrenci numarası yerine öğrenci adı ve soyadını kullanacak olursak yukarıdaki kod bloku aşağıdaki gibi olacaktır.

```
if (Secilen_Ogrenci.Ogrenci_Adi == Kayitli_Ogrenci.Ogrenci_Adi &&  
Secilen_Ogrenci.Ogrenci_Soyadi == Kayitli_Ogrenci.Ogrenci_Soyadi)  
...
```

Yukarıda verilen örnek kod blokunu kısaltmak amaçlı olarak aşağıda gösterilen kodun kullanılması hatalıdır. C++ Programlama Dili’nde aşağıdaki gibi bir ilişkisel operatör işlemi **struct** ile kurgulanan yapılarda gerçekleştirilemez.

```
if (Secilen_Ogrenci == Kayitli_Ogrenci) // Geçersiz işlem!
```

### 8.1.4. struct ile Tanımlanan Yapıda Girdi/Çıktı İşlemleri

Yukarıda yer verilen örnekler ile **struct** yapılarının işleyişine ait olan açıklamalardan da anlaşılıcağı üzere bir yapıdaki üyelere her koşulda teker teker veri ataması yapılabilir. Benzer olarak da bir üyenin yazdırılması da yanı şekilde gerçekleşir.

```
cin >> Secilen_Ogrenci.Ogrenci_No;  
cin >> Secilen_Ogrenci.Ogrenci_Adı;  
cin >> Secilen_Ogrenci.Ogrenci_Soyadı;
```

Benzer şekilde çıktı işlemleri de aşağıdaki gibi yapılır.

```
cout << Kayitli_Ogrenci.Ogrenci_No << " " << Kayitli_Ogrenci.Ogrenci_Adı << " " <<  
Kayitli_Ogrenci.Ogrenci_Soyadı << endl;  
  
cout << "Ara sınav notu: "<< Kayitli_Ogrenci.Ara_Sinav_Not << endl;  
  
cout << "Kısa sınav notları: "<< Kayitli_Ogrenci.Kisa_Sinav1_Not << " " << Kayitli_O-  
grenci.Kisa_Sinav2_Not << endl;  
  
cout << "Dönem sonu sınavı notu: "<< Kayitli_Ogrenci.Donem_Sonu_Not << endl;  
  
cout << "Başarı notu: "<< Kayitli_Ogrenci.Basari_Notu << endl;  
cout << "Harf notu: "<< Kayitli_Ogrenci.Harf_Notu << endl;
```

### 8.1.5. struct ile Tanımlanan Yapıda Değişkenler ve Fonksiyonlar

Fonksiyonlar içerisinde dizilere yer verildiğinde dizilere erişimin referans verilerek yapıldığını görmüştük. Dolayısı ile de bir fonksiyon bir dizi üzerinde işlem yaptığında değer döndürmesi söz konusu olmamakta idi. Bu kural benzer şekilde **struct** ile kurgulanan yapılar içinde geçerlidir. Ancak burada **struct** işleyışı dikkate alındığında iki durum söz konusu olmaktadır.

- Yapı içerisinde yer alan bir değişkenin barındıldığı veri fonksiyona parametre olarak veya doğrudan referans olarak aktarılabilir.
- Fonksiyon, işlem sonucunda **struct** ile tanımlanmış olan yapıya bir değer döndürebilir.

Yukarıda yer verilen kod blokunu aşağıda fonksiyonda kullanılmıştır.

```
void HarfNotuHesapla(Kayitli_Ogrenci& ogrenci)  
{  
    int hesaplanan_not;  
  
    cin >> ogrenci.Ogrenci_Adı >> ogrenci.Ogranci_Soyadi;  
    cin >> ogrenci.Kisa_Sinav1_Not >> ogrenci.Ara_Sinav_Not;  
    cin >> ogrenci.Kisa_Sinav1_Not >> ogrenci.Kisa_Sinav2_Not;  
    cin >> ogrenci.Kisa_Sinav1_Not >> ogrenci.Kisa_Sinav2_Not;
```

```

cin >> Donem_Sonu_Not;

hesaplanan_not = (K1 * Kisla_Sinav1_Not) + (K2 * Kisla_Sinav2_Not) + (K3 *
Ara_Sinav1_Not) + (K4 * Donem_Sonu_Not)

if (hesaplanan_not >= 90)
    ogrenci.Harf_Notu = 'AA';
else if (hesaplanan_not >=85)
    ogrenci.Harf_Notu = 'AB';
else if (hesaplanan_not >=75)
    ogrenci.Harf_Notu = 'BA';
else if (hesaplanan_not >=65)
    ogrenci.Harf_Notu = 'BB';
else if (hesaplanan_not >=60)
    ogrenci.Harf_Notu = 'CB';
else if (hesaplanan_not >=55)
    ogrenci.Harf_Notu = 'BC';
else if (hesaplanan_not >=50)
    ogrenci.Harf_Notu = 'CC';
else if (hesaplanan_not >=45)
    ogrenci.Harf_Notu = 'CD';
else if (hesaplanan_not >=40)
    ogrenci.Harf_Notu = 'DC';
else if (hesaplanan_not >=35)
    ogrenci.Harf_Notu = 'DD';
else if (hesaplanan_not >=0)
    ogrenci.Harf_Notu = 'FF';
}

```

Yukarıdaki kod bloku HarfNotuHesapla adlı fonksiyona aittir. Bu fonksiyonun çağrılması için aşağıdaki bildirimin kullanılması gereklidir.

**HarfNotuHesapla(ogrenci);**

Fonksiyon çağrıldığında gerekli olan bilgi ogrenci adlı değişkende barındırılmaktadır.

## 8.1.6.struct ile Tanımlanan Yapı ve Dizilerin Karşılaştırılması

Önceki bölümlerde yer verilen örnekler incelendiğinde struct ile tanımlanan bir yapının bir dizi karşılaştırıldığında aralarında benzerlikler olduğu gibi farklılıklar da olduğu da görülmektedir. Bu farklılıklar ve benzerlikler aşağıda gösterilmiştir.

İşlemin çeşidi	Dizi	struct
Aritmetik	Yapılamaz	Yapılamaz
Atama	Yapılamaz	Yapılabilir
Girdi/Çıktı	Yapılamaz (karakter katarları hariç)	Yapılamaz
Kıyaslama	Yapılamaz	Yapılamaz
Parametre aktarımı	Referans ile	Referans veya değer ile
Değer döndüren fonksiyon	Yapılamaz	Yapılabilir

## 8.1.7.struct ile Tanımlanan Yapıda Dizilerin Kullanılması

Bir dizi, aynı veri tipindeki bir veya daha çok sayıdaki elemanı barındıran bir veri yapısıdır. Dolayısı ile de bir diziden söz ederken dizinin barındırdığı elemanlar (veri) ile elaman sayısı ilk olarak akla gelir. Bir dizide hem veri tipi hemde eleman sayısı dizi ile dolaysız olarak ilişkili olduğu için struct ile tanımlanan bir yapıda bu iki özellik kullanılabilir.

Aşağıdaki örnek kodda, ListeTipi ad verilen bir yapı kurgulanmıştır.

```
const int dizi_eleman_sayisi = 100;  
  
struct ListeTipi  
{  
    int liste_eleman[dizi_eleman_sayisi];  
    int liste_indis;  
};
```

Aşağıda verilen bildirimde **struct** ile tanımlanan yapı kullanılarak TamSayiListesi adlı bir değişken tanımlanmıştır.

ListeTipi TamSayiListesi;

**TamSayiListesi**

liste_eleman	

Bu kod blokunun çalıştırılması ile söz konusu değişken adlı ile bellekte 100 adet tam sayı veri tipinde sayı barındıracak bir alan ayrılmaktadır.

Bu yapıda iki adet üye bulunmaktadır. Birincisi liste\_eleman adlı dizidir. Bu dizinin 100 adet elemanı bulunmaktadır. Dizi tam sayı veri tipinde tanımlanmıştır. Ayrıca dizinin boyutunu belirten liste\_boyut adlı üyenin de veri tipi tam sayı veri tipidir.

Bu yapı bellekte ilk oluşturulduğu anda aşağıdaki görüldüğü gibi bir alan ayrılacaktır.

Bu yapıya veri ataması yapılabilmesi için aşağıdakine benzer şekilde tanımlamalar yapılabilir.

```
TamSayiListesi.liste_indis = 0;  
TamSayiListesi.liste_eleman[0] = 1;  
TamSayiListesi.liste_indis++;  
TamSayiListesi.liste_eleman[1] = 2;  
TamSayiListesi.liste_indis++;  
TamSayiListesi.liste_eleman[2] = 3;
```

Yukarıdaki kod blokunda yer alan bildirimler çalıştırıldığında söz konusu yapının elemanlarına erişilebilir ve değerler atanabilir. Kod blokunun işlenmesi sonucunda yapı aşağıdaki duruma gelecektir.

**TamSayiListesi**

liste_eleman[0]	1
liste_eleman[1]	2
liste_eleman[2]	3
liste_eleman[3]	4

Kod blokunun ilk satırı işlendiğinde liste\_indis değişkeni dizinin ilk elemanını gösterecek şekilde “0” olarak tanımlanmıştır. Bu indis değeri kullanılarak ikinci satırındaki bildirim işlendiğinde ilgili alana “1” değeri atanmıştır. Ardından gelen bildirimde liste\_indis değişkeni bir attırılmış sıradaki dizi alanını gösterecek şekilde güncellenmiştir. Bu işlemin ardından bu alana “2” değeri atanmıştır. Bu işlemlerin tekrarlanması ile TamSayiListe yapısına veri atanabilir.

Bu yapıda dizi kullanıldığı için sıralı arama algoritması kullanılarak yapı içerisinde arama yapılabilir.

```
int SiraliArama(const ListeTipi& TamSayiListesi, int aranan_veri)  
{
```

```

int konum;

bool bulundu = false;

for (konum = 0; konum < TamSayiListe.liste_indis; konum++)

    if (TamSayiListe.liste_eleman[konum] == aranan_veri)

    {

        bulundu = true;

        break;

    }

    if (bulundu)

        return konum;

else

    retrun -1;

}

```

Yukarıda yazılan sıralı arama algoritmasının uygulandığı algoritma da liste\_indis TamSayiListe yapısının üyesi olduğundan TamSayiListe.liste\_indis kullanılarak yapıya erişilebilmektedir. Böylece yapı içerisindeki bir eleman aynı şekilde TamSayiListe.liste\_eleman[konum] ile erişilebilir.

Yazılan fonksiyonda kullanılan değişken, fonksiyona atanacak olan değeri barındıran fonksiyonun formal parametresidir. Formal parametre sabit olarak tanımlanarak, bu değişken üzerinde herhangi bir işlem yapılmasıının önüne geçilmiştir. Bu şekilde TamSayiListe yapısı fonksiyon tarafından atanınan değeri okuyabilecek iken bu değer üzerinde herhangi bir değişiklik yapması olanaklı olmayacağından emin olmaktadır.

Bir fonksiyonun formal parametresine bir değişken kullanılarak değer atama yapıldığında formal parametre aktarılan değeri kopyalatarak kullanmaktadır. Böylece asıl değer korunurken fonksiyon kendi içerisinde bu değer üzerinde işlem yapabilmekte ve değer değiştirilebilmektedir.

Burada dikkat edilmesi gereken **struct** ile kurgulanmış bir yapının üye sayısının sayıca fazla olması durumunda ilgili üyelerde barındırılan verilerin bellekte saklanması için gereken alan doğrusal olarak artacaktır. Böyle kurgulanmış yapıda bir üyeye veri aktarılması için de doğru değişkenin kullanılması gerekecektir. Ayrıca fonksiyonun formal parametresine karşılık gelen veri de fonksiyona kopyalanacaktır. Kopyalama işlemi için program derlenirken derleyicinin bu işlem için ayrıca bellek ayırması gerekecektir. Bu işlem ise programın çalıştırılması sırasında daha çok bellek kullanımını ve kopyalama işleminin de zaman olarak daha uzun sürede gerçekleşmesine neden olacaktır.

Öte yandan bir değişkenin değeri referans olarak aktarılırsa, fonksiyonun formal parametresi ilgili verinin bellek adresini okuyarak işlem yapacaktır. Dolayısı ile referans kullanımı daha verimli olacaktır. Ayrıca referans olarak aktarılmadan dolayı asıl parametre üzerinde yapılan işlemler doğrudan referansın belirttiği bellek adresindeki değerinde işlenmesi yani değiştirmesini sağlayacaktır.

Bazı durumlarda ise programcı, referans yolu ile veri aktarımını kullanırken, referansın bu değişime açık olan yapısından etkilenmek istemeyecektir. Referans edilerek aktarılan asıl değerin korunması gereklili ise yukarıdaki gibi ilgili referans değişken **const** ile tanımlanabilir. Bu şekilde C++ Programlama Dili’nde yazılan bir fonksiyonda veri referans olarak aktarılırken değeri de korunmuş olacaktır. Bu şekilde sıralama ve liste biçiminde verilen verilerin işlenmesi için yazılan fonksiyonlar yazılabılır.

### 8.1.8.Dizilerde struct ile Tanımlanan Yapı Kullanımı

Bir işletmede tam zamanlı çalışan personel sayısının 100 olduğunu var sayalım. Personelin ay sonunda ücretlerin hesaplanacak olsun. Ayrıca yıl başından günümüze kadar yapılan ödemeleri de personel özelinde raporlamak isteyelim.

Bunu yapacak programı yazmadan önce ilk olarak her personele ait kayıtların tutulacağı bir veri yapısı tasarlayalım.

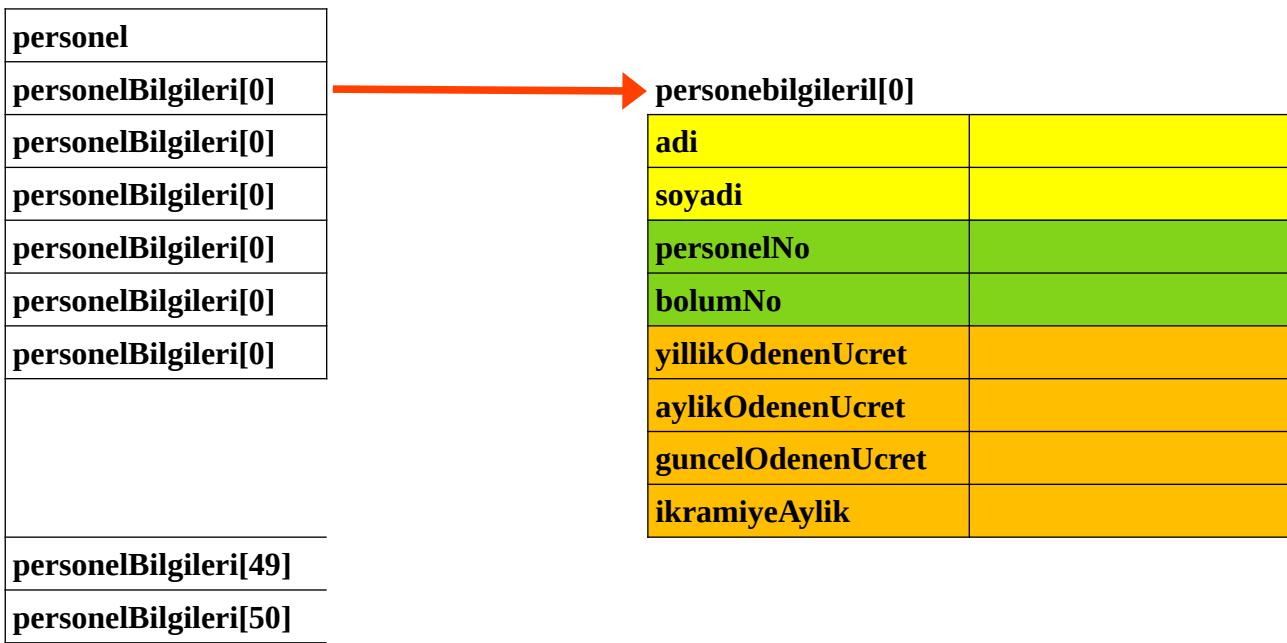
```
struct personelBilgileri
{
    string adi;
    string soyadi;
    int personelNo
    int bolumNo
    double yillikOdenenUcret;
    double aylıkOdenenUcret;
    double guncelOdenenUcret;
    double ikramiyeAylık;
};
```

Personel ilişkin bilgiler şu üyelikler (bileşenler) tarafından tanımlanmıştır: personelin adı, soyadı, personel numarası, çalıştığı bölümün numarası, yıllık ödenen ücret, aylık ödenen ücret, bugüne kadar ödenen ücret ile aldığı aylık ikramiye.

Her bir personel için bu bilgiler kullanılacaktır. Personele ait veriler tümü için aynı olduğundan 50 elemanlı bir dizi tanımlayarak bunu kullanabiliriz.

```
personelBilgileri personel[50];
```

Yukarıdaki tanımlama yapıldığında her bir tekil personel kaydı yapısında karakter katarı, tam sayı ve double veri tipinde alanlara sahip olan alt bellek alanlarına sahip olacaktır. Bu bilgilerin bir dosyada bulunacağını kabul edelim. Dosyada bulunan verilerin okunup ilgili yapılandırılmış veri tipinde saklanabilmesi için verileri dosyadan okuyacak ve ilgili bellek adreslerine yazacak bir okuma ve yazma kod blokuna gereksinim olacaktır.



Verilerin bulunduğu dosyadan okuyan kod bloku bir döngü ile kullanılarak verileri okumakta ve ilgili alanlara veriyi yazmaktadır.

```
ifstream dosyaoku; // dosyadan okuma işlemi için değişken
                    // dosya adı personel.vr olarak seçilmiştir.
                    // dosya açıldı ve okuma işlemi yapılıyor

// dosyadan okuma için döngü kullanılır. Eleman sayısı 50'dir.

for (int sayac = 0; sayac < 50; sayac++)
{
    dosyaoku >> personelBilgileri[sayac].adi
                >> personelBilgileri[sayac].soyadi
                >> personelBilgileri[sayac].personelNo
                >> personelBilgileri[sayac].bolumNo
                >> personelBilgileri[sayac].aylikOdenenUcret
                >> personelBilgileri[sayac].yillikOdenenUcret =
elBilgileri[sayac].aylikOdenenUcret * 12
                >> personelBilgileri[sayac].guncelOdenenUcret = 0.0
                >> personelBilgileri[sayac].aylikOdenenUcret = 0.0
                >> personelBilgileri[sayac].ikramiyeAylık = 0.0
}
```

Ayrıca personele ait bilgilerin bulunduğu dosyada ödenecek olan ikramiye miktarının da hazır bulunduğu var sayalım. Bu durumda her bir personele ay sonunda ödenecek olan ücretin miktarının hesaplanması gerekecektir. Aşağıdaki kod bloku ise söz konusu aya ait ücret ödemelerini hesaplamaktadır. Bu işlem için bir döngü kullanılmıştır.

```
double OdenecekAylikUcret          // hesaplanan ücrette ait olan değişken

// Hesaplama için döngü kullanılır. Eleman sayısı 50'dir.

for (int sayac = 0; sayac < 50; sayac++)
{
    cout << personelBilgileri[sayac].adi << " "
        << personelBilgileri[sayac].soyadi << " "

    OdenecekAylikUcret = personelBilgileri[sayac].aylikOdenenUcret
        + personelBilgileri[sayac].ikramiyeAylik;

    personelBilgileri[sayac].guncelOdenenUcret = OdenecekAylikUcret;
    cout << setprecision(2) << OdenecekAylikUcret << endl;
}
```

### 8.1.9. struct ile Tanımlanan Yapıların İçerisinde struct ile Yapı Tanımlaması ve Kullanımı

Yukarıdaki örneklerde **struct** ile dizi veri yapısının birlikte kullanılarak veri depolanabileceğini görmüş oldu. Ayrıca **struct** ile bir dizi tanımlanıp, dizinin de **struct** ile tanımlanabileceğini görmüş olduk. Burada ise bazı durumlarda verinin farklı gruplara bölünerek yeniden organize edilmesinde **struct** ile bu işlemin nasıl gerçekleştirileceğini göreceğiz.

Yukarıda işletmenin elamanları ile ilişkili olan bazı verilerin düzenlenmiş hali ile nasıl kullanıldığına dair örnek kod bloklarını kullanarak ama personele ait olan bilgilerin çeşidini arttırmış önceki örneklerde olduğu gibi bunları tek bir veri yapısında barındırmayı inceleyelim.

```
struct personelBilgileri
{
    string birinci_adi;
    string ikinci_adi;
    string soyadi;
    string personel_kimlik
    string personel_adres1;
    string personel_adres2;
```

```
    string personel_ilce;
    string personel_il;
    string personel_posta_kodu;
    int personel_ise_giris_yil;
    int personel_ise_giris_ay;
    int personel_ise_giris_gun
    int personel_isten_ayrilma_yil;
    int personel_isten_ayrilma_ay;
    int personel_isten_ayrilma_gün;
    string personel_cep_telefon_no;
    string personel_ev_telefon_no;
    string personel_fax_no;
    string personel_eposta_adresi;
    string personel_bolum;
    double personel_aylik_ucret;
};


```

Yukarıdaki kod blokundan da görüleceği üzere **struct** ile kurgulanan yapıda 21 adet üye bulunmaktadır. Bu üyelerden bazılarına daha sık erişilebileceği öngörlülebilir. Ayrıca bazı üyelerin kendi aralarında yakın ilişkileri olacağı görülebilir. Ek olarak bazı üyelerin veri tipi de aynıdır. Bu yapıyı kendi içerisinde yeniden organize ederek farklı yapılara bölebiliriz.

```
struct personel_isim_bilgileri
{
    string birinci_adi;
    string ikinci_adi;
    string soyadi;
};

struct personel_adres_bilgileri
{
    string personel_adres1;
    string personel_adres2;
    string personel_ilce;
};
```

```

        string personel_il;
        string personel_posta_kodu;
    };

    struct personel_tarihsel_bilgiler;
    {

        int personel_ise_giris_yil;
        int personel_ise_giris_ay;
        int personel_ise_giris_gun
        int personel_isten_ayrilma_yil;
        int personel_isten_ayrilma_ay;
        int personel_isten_ayrilma_gün;
    };

    struct personel_iletisim_bilgileri;
    {

        string personel_cep_telefon_no;
        string personel_ev_telefon_no;
        string personel_fax_no;
        string personel_eposta_adresi;
    };

    string personel_kimlik
    string personel_bolum;
    double personel_aylik_ucret;
};


```

Yukarıdaki yapılanmada ilişkili olan ve gereksinim olduğunda erişilecek olan ve birbiri ile ilişkili veriler ortak yapılar içerisinde yer almaktadır. Bu aşamadan sonra personel bilgileri için ilk kurguladığımı yapıyı yeniden düzenleyebiliriz.

```

struct personelBilgileri
{
    personel_isim_bilgileri isim;
    personel_adres_bilgileri adres;
    personel_tarihsel_bilgiler baslama_tarih;

```

```

    personel_tarihsel_bilgiler ayrılma_tarih;
    personel_iletisim_bilgileri iletisim;
    string personel_kimlik;
    string personel_bolum;
    double personel_aylik_ucret;
};


```

Bu yapı ile çalışmak ilk tasarlanan yapı ile karşılaştırıldığında daha kolaydır. Böylece bu yapı kullanılarak gereksinimlere uygun yeni yapılandırılmış veri tipleri kurgulanabilir. Örneğin müşterilere ait bilgilerin kullanılacağı bir programda yukarıdaki yapıyı kullanarak, örneğin isim\_bilgileri, adres\_bilgileri ve iletisim\_bilgileri gibi yapılar asıl işlevlerin programlanmasında gerekli verinin barındırılacağı yeni vari yapıları tasarlabilir.

Burada yukarıda kurgulanan yapılandırılmış veri yapısını kullanarak bu verilere üyeliği kullanarak nasıl erişebileceğimizi görelim. Bu işlem için aşağıdaki bildirimde kullanılmıştır.

```
personelBilgileri yeni_personel;
```

Önceki kod blokunda tanımlanan yapılandırılmış veri yapısı olan “personelBilgileri” ve değişken olarak “yeni\_personel” bildirimde kullanılmıştır.

#### **personelBilgileri**

personel_isim_bilgileri	birinci_adi	
	ikinci_adi	
	soyadi	
personel_adres_bilgileri	adres1	
	adres2	
	ilce	
	il	
	posta_kodu	
personel_tarihsel_bilgiler	personel_ise_giris_yil	
	personel_ise_giris_ay	
	personel_ise_giris_gun	
	personel_isten_ayrilma_yil	
	personel_isten_ayrilma_ay	
	personel_isten_ayrilma_gün	
personel_iletisim_bilgileri	personel_cep_telefon_no	
	personel_ev_telefon_no	
	personel_fax_no	
	personel_eposta_adresi	

personel_kimlik	
personel_bolum	
personel_aylik_ucret	

Yukarıdaki yapıyı kullandığımızı ve aşağıdaki bildirimlerin yapıldığını var sayalım.

yenি\_personel.personel\_isim\_bilgileri.birinci\_adi = “Yusuf”;

yenি\_personel.personel\_isim\_bilgileri.ikinci\_adi = “Mahmut”;

yenি\_personel.personel\_isim\_bilgileri.soyadi = “Tuncer”;

yenি\_personel.personel\_aylik\_ucret = 18000.00;

Yukarıdaki bildirimler incelenecək olursa personele ait isim bilgilerinin üç farklı üyelik ile erişildiği görülecektir. Buna karşılık ücret bilgisinin barındırıldığı bilgiler ise tek bir üyeliğe sahiptir. Program yazılrken bu özellik dikkate alınmalıdır.

Atamalar konusunda belirtilen özellikler burada da geçerlidir.

**Örnek Problem:** Bir işletmenin pazarlama bölümüne bağlı olarak çalışan adet bayisi bulunmaktadır. Bu bayilerin her ay gerçekleştirdikleri satışlara ait veriler bir dosyada kayıt altına alınmaktadır. Her ayın sonunda bayilerin gerçekleştirdikleri satışlar raporlanmaktadır.

Yıllık Ciro Raporu					
Bayi No	Dönem1	Dönem2	Dönem3	Dönem4	Toplam
12345	2608.00	1873.00	2646.00	2423.00	9550.00
32214	1178.00	1326.00	1148.00	996.00	4648.00
23422	1397.00	1007.00	1316.00	1662.00	5382.00
57373	1856.00	2291.00	1473.00	1678.00	7298.00
35864	2881.00	1960.00	2424.00	2419.00	9684.00
54654	1128.00	1560.00	1551.00	1405.00	5644.00
Toplam	11048.00	10017.00	10558.00	10583.00	

En büyük yıllık ciroya sahip olan bayının kimliği = 35864, Miktar = 9684.00 TL  
 En büyük cironun elde edildiği dönem = 1, Miktar = 11048.00 TL

Raporda bir yıl boyunca gerçekleşen toplam satış geliri üç aylık dönemler olarak raporlanmaktadır. Raporda belirtilen D1, D2, D3 ve D4 başlıklarını yılın aylarının üçerli gruplamalarına karşılık gelmektedir. Dönem1 Ocak, Şubat ve Mart; Dönem2 Nisan, Mayıs ve Haziran; Dönem3 Temmuz, Ağustos, Eylül ve Dönem4 Ekim, Kasım ve Aralık aylarına karşılık gelmektedir.

Bayilerin isimleri yerine, onları tanımlamak için kullanılan özel numaralar kullanılmaktadır. Satış verileri ile bayilerin bilgileri ayrı dosyalarda saklanmaktadır. Bayilerin bilgileri aşağıda gösterildiği gibidir:

2601001

2601002

...

Satış bilgileri nin dosyadaki yerleri için özel bir sıralama bulunmamaktadır. Satış bilgileri ise aşağıda gösterildiği gibidir:

2601001 1 218765

2601001 2 185578

2601002 3 234897

1101001 3 248956

0601001 1 266902

0602001 4 345098

0602002 6 768043

...

Yapılması gereken bu satış verilerinin örnek çıktıda olduğu gibi raporlanması sağlayacak bir programın yazılması istenmektedir. Programın işleyeceği veriler iki ayrı dosyadan okunacaktır. Birinci dosya bayileri tanımlamak için kullanılan numaraların bulunduğu dosyadır. Programın üzerinde işlem yapacağı diğer veri ise bayilerin bir yıllık zaman diliminde her ay sonunda rapor ettikleri satış verilerinin yer aldığı dosyadır. Program bu dosyalardan okuduğu verileri işleyerek yukarıda görüldüğü şekilde raporlayarak dosyaya yazacaktır. Burada raporun yazılacağı dosyanın adı da kullanıcından istenecektir. Burada programın test edilmesi amacı ile bayi verilerinin bulunduğu dosya olarak **bayiler.dat** adlı dosya ve bayilerin aylık satış verilerinin yer aldığı dosya olarak **bayi\_ciro.dat** adlı dosyalar kullanılmıştır. Dosya uzantısı sadece verileri barındırdığını belirtmek amaçlı “dat” olarak seçilmiştir. Dosyalar teknik olarak basit metin dosyalarıdır.

### Problemin Çözümlenmesi ve Algoritma Tasarımı:

Raporun hazırlanabilmesi için iki ayrı dosyada yer alan verilerin okunması ve bunların doğru şekilde eşleştirilerek işlenmesi gereklidir. Veriler farklı yapılara sahip oldukları için bunların gruplanması problemin çözümü için daha uygun olacaktır. Bu nedenle **struct** kullanılarak bayilere ait veriler tek bir yapıda toplanarak işlenebilir.

```
struct bayiVerileri
{
    // Bayi kimlik numarası
    string bayiNo;
    // Dönemlere ait ciro verisinin barındırılacağı dizi
    double donemCiroVerisi[4];
    // Tekil bayi ciro verisi
    double bayiCiro;
};
```

Şimdilik sadece altı adet bayi bulunduğu için alt adet elemana sahip olan bir dizinin kullanılması yeterlidir.

```
bayiVerileri bayiVerileriListesi[BAYI_SAYISI];
```

Yukarıdaki bildirimde **BAYI\_SAYISI** adlı değişkenin değeri 6 olarak alınacaktır. Örnek rapor çıktısında her bayi için ilgili dönem ciroları ile bayinin yıllık cirosu yer alırken aynı zamanda şirketin ilgili dönemdeki toplam satışı ile şirketin yıl sonu itibarı ile cirosu da yer almaktadır. Bu nedenle de dört elemanlı bir dizi tanımlanması yeterlidir. Bu dizi ilgili dönemler arasında en büyük olan cironun yapıldığı dönemin belirlenmesinde kullanılacaktır. Bu nedenle de programda aşağıdaki veri yapısı yani dizi kullanılacaktır.

```
double donemCiroVerisi[4];
```

Burada C++ programlama Dili’nde dizilerin indislerinin “0” başladığını anımsamakta yarar var. Dolayısı ile de dizinin ilk elamanın indis sıfır olurken ilk üç aylık dönemi, ikinci elemanın indis 1 olurken, ikinci üç aylık dönemi ve sırası ile her bir dizi elamanının karşılık gelen üç aylık dönemin verisini barındıracığı görülecektir.

bayiVeriler	bayiNo	donemCiroVerisi[0,1,2,3]				bayiCiro
bayilerListesi[0]						
bayilerListesi[1]						
bayilerListesi[2]						
bayilerListesi[3]						
bayilerListesi[4]						
bayilerListesi[5]						

Programın ilk olarak yapması gereken **bayiNo** bilgilerini **bayiler.dat** adlı dosyadan okuyarak diziye atamasıdır. Ardından her bir bayi için **donemCiroVerisi[]** ve **bayiCiro** değerlerinin etkinleştirip değer sıfır ataması işlemini gerçekleştirecektir. İşlemin ardından aşağıdaki durum oluşacaktır.

bayiVeriler	bayiNo	donemCiroVerisi[0,1,2,3]				bayiCiro
bayilerListesi[0]	2601002			0.0	0.0	0.0
bayilerListesi[1]	1101001	0.0	0.0	0.0	0.0	0.0
bayilerListesi[2]	0601001	0.0	0.0	0.0	0.0	0.0
bayilerListesi[3]	0602001	0.0	0.0	0.0	0.0	0.0
bayilerListesi[4]	0601002	0.0	0.0	0.0	0.0	0.0
bayilerListesi[5]		0.0	0.0	0.0	0.0	0.0

Bu işlemin ardından program verileri işlenmesi aşamasına geçecektir. Verilerin işlenmesi aşaması aşağıdaki işlem adımlarından oluşacaktır.

- 1 Her bir bayinin **bayiNo** numarasını, ilgili olduğu ay ve o aya ait olan satış verisini **bayi\_ciro.dat** dosyasından oku.
- 2 İlgili veriyi yerine yazmak için **donemCiroVerisi[]** adlı dizi üzerinde arama yap.
- 3 Okunan verinin hangi döneme ait olduğunu belirle.
- 4 İlgili dönem için ciro verisini oku, dosyadan okunan veri ile toplayarak ilgili dönem için ciro verisini güncelle.

Bayilerin satış verilerinin bulunduğu dosya okunup işlendikten sonra aşağıdaki işlemler yapılacaktır:

- 1 Her bir bayi için bir yıllık ciro toplamını hesapla.
- 2 Her bir dönem için gerçekleşen toplam ciroyu hesapla.
- 3 Raporu yaz.

Yukarıda tanılanan işlem basamaklarının algoritma olarak ifadesi aşağıdaki gibidir:

1. **donemCiroVerisi[]** adlı diziyi etkinleştir.
2. Satış verilerini işle.
3. Her bir dönem için ciro hesapla.
4. Her bir dönem için her bir bayiye ait ciroyu hesapla.
5. Raporu yazdır.
6. En büyük ciroya sahip olan bayiyi yazdır.
7. Cironun en büyük olduğu dönemi yazdır.

Bu işlem basamaklarının tamamı tek bir **main()** fonksiyonu içerisinde yazılabilir. Ancak programın basitleşmesi ve yaşam döngüsü içerisinde gereki güncellemelerin daha kolay yapılabilmesi için her bir basamak bir fonksiyona dönüştürülecektir.

### **etkinleştir Fonksiyonu**

Bu fonksiyon **bayiler.dat** adlı dosyadan bayilere ait numaraları okuyacaktır. Okunan veri **bayiVerileriListesi[]** adlı diziye yazılacaktır. Ayrıca söz konusu dizideki ilgili alanlara sıfır yarık tüm dizinin etkinleştirilmesini gerçekleştirecektir.

```
void etkinleştir(ifstream& veriOku, bayiVerileri list[], int listeBuyukluk)
{
    for (int indis = 0; indis < listeBuyukluk; indis++)
    {
        veriOku >> list[indis].bayiNo; // Bayi kimlik bilgisini oku
        for (int donem = 0; donem < 4; donem++)
            list[indis].donemCiroVerisi[donem] = 0.0;
    }
}
```

```

        list[indis].bayiCiro = 0.0;
    }

} // etkinlestir fonksiyonu sona erdi

```

### **verileriAl Fonksiyonu**

Bu fonksiyon **bayi\_ciro.dat** adlı dosyadan bayilere ait aylık satış verilerini okuyacaktır. Okunan veri, **bayiVerileriListesi[]** adlı dizide barındırılacaktır. Bu fonksiyonun algoritması aşağıdaki gibi olacaktır.

1. Bayinin tanımlama bilgisini yani **bayiNo**, **ay** ve o aya ait **satış miktarı verisini** oku.
2. İlgili veriyi yerine yazmak için **bayiVerileriListesi[]** adlı dizi üzerinde arama yap. Bunun yapılmasını nedeni bayilere ait olan satış verilerinin düzensiz olarak toplanmakta olmasıdır. Eğer bayilerin satış verileri önceden işlenerek sıralanmış olsa ide bu işleme gerek olmaya- caktı. Bu nedenle dizide her bir bayi için ilgili veri okunduktan sonra dizindeki doğru alan kayıt edilmesi için sıralı arama algoritması kullanmak durumundayız.
3. İlgili ayan bulunduğu dönemi belirle.
4. İlgili dönem için bayinin cirosunu güncelle.

Örneğin dosyadaki kaydın aşağıdaki gibi olduğunu varsayılmı.

1101001 8 209065

Bu veriye göre bayi\_ID 261001, 8. aya ait olduğu, bayinin o aydaki cirosunun 209065 olduğu anlaşılmaktadır. Programın verileri okumakta olduğunu ve aşağıdaki gibi olduğunu var sayalım.

<b>bayiVerileri</b>	<b>bayiNo</b>	<b>donemCiroVerisi[0,1,2,3]</b>			<b>bayiCiro</b>
bayilerListesi[0]	2601001	125645.0	86700.0	0.0	0.0
bayilerListesi[1]	2601002	28097.0	2345987.0	0.0	0.0
bayilerListesi[2]	1101001	0.0	12345.0	0.0	0.0
bayilerListesi[3]	0601001	120670.0	0.0	0.0	0.0
bayilerListesi[4]	0602001	145670.0	103109.0	0.0	0.0
bayilerListesi[5]	0601002	0.0	230765.0	0.0	0.0

Yukarı belirtilen girdi okunduğunda bayilerListesi[2] adlı dizi satırına ait olduğu belirlene- cektir. Verideki ay 8 olduğundan bu ise donemCiroverisi[2] karşılık gelmektedir. Söz konusu dizi elemanın değeri ise 0.0 olduğundan önceden var olan veri ile okunan veri toplanarak elde edilen sonuç söz konusu dizi elamanın güncel değeri olarak atanacaktır.

bayiVerileri	bayiNo	donemCiroVerisi[0,1,2,3]				bayiCiro
bayilerListesi[0]	2601001	125645.0	86700.0	0.0	0.0	0.0
bayilerListesi[1]	2601002	28097.0	2345987.0	0.0	0.0	0.0
bayilerListesi[2]	1101001	0.0	12345.0	209065.0	0.0	0.0
bayilerListesi[3]	0601001	120670.0	0.0	0.0	0.0	0.0
bayilerListesi[4]	0602001	145670.0	103109.0	0.0	0.0	0.0
bayilerListesi[5]	0601002	0.0	230765.0	0.0	0.0	0.0

Yukarıda verilenlere göre fonksiyon aşağıdaki gibi yazılır.

```
void verileriAl(ifstream& dosyaAc, bayiVerileri list[], int listeBuyukluk)
```

```
{
    int indis;
    int donem;
    string bID;
    int ay;
    double tutar;

    // Bayi kimlik bilgisini oku
    dosyaAc >> bID;
    while (dosyaAc)
    {
        // Takvim ayını ve o aya ait olan tutarı oku
        dosyaAc >> ay >> tutar;
        for (indis = 0; indis < listeBuyukluk; indis++)
            if (bID == list[indis].bayiNo)
                break;
            if (1 <= ay && ay <= 3)
                donem = 0;
            else if (4 <= ay && ay <= 6)
                donem = 1;
            else if (7 <= ay && ay <= 9)
                donem = 2;
            else

```

```

        donem = 3;

        if (indis < listeBuyukluk)

            list[indis].donemCiroVerisi[donem] += tutar;

        else

            cout << "Geçersiz bayi kimlik numarası tespit edildi." << endl;

            dosyaAc >> bID;

    } // while döngüsü sona erdi

} // verileriAl fonksiyonu sona erdi

```

### **donemToplamCiro Fonksiyonu**

Bu fonksiyon işletmenin ilgili dönemi için bayileri aracılığı ile elde ettiği cironun büyülü-ğünü hesaplamaktadır. Her bir döneme ait olan ciroyu hesaplayabilmek için her dönem ait olan bayi ciro verileri okunup toplanmaktadır. Bunun için de **bayiVerileriListesi** dizisine ve **dönemToplam** dizisine erişmesi gereklidir. Bu fonksiyona aynı zamanda her bir dizideki sütun sayısında tanımlamak gereklidir. Böylelikle bu fonksiyonun üç adet parametresi olmaktadır. Fonksiyon aşağıdaki gibidir.

```

void donemToplamCiro(bayiVerileri list[], int listeBuyukluk, double donemCiroToplam[])
{
    for (int donem = 0; donem < 4; donem++)
        donemCiroToplam[donem] = 0.0;

    for (int donem = 0; donem < 4; donem++)
        for (int indis = 0; indis < listeBuyukluk; indis++)
            donemCiroToplam[donem] += list[indis].donemCiroVerisi[donem];
}

```

### **bayiYillikCiro Fonksiyonu**

Bu fonksiyon işletmenin bayilerinin byıl sonunda elde ettikleri toplam ciroyu hesaplamak-tadır. Bayinin yıllık cirosunun hesaplanabilmesi için her bir bayının dönemlik cirolarının okunup toplanması gereklidir. Bu fonksiyonun **bayiVerileriListesi** dizisine erişmesi ve dizinin boyutunu kullanabilmesi gereklidir. Böylelikle bu fonksiyonun iki adet parametresi olmaktadır. Fonksiyon aşağıdaki gibidir.

```

void bayiYillikCiro(bayiVerileri list[], int listeBuyukluk)
{
    for (int indis = 0; indis < listeBuyukluk; indis++)

```

```

        for (int donem = 0; donem < 4; donem++)
            list[indis].bayiCiro += list[indis].donemCiroVerisi[donem];
    } // bayiYillikCiro fonksiyonu sona erdi

```

### **raporYaz Fonksiyonu**

Bu fonksiyon yukarıda yapılan işlemlerin rapor biçiminde ekrana ve dosyaya yazdırılmasını sağlamaktadır. Algoritması aşağıdaki gibidir:

1. Raporun başlığını yaz.
2. Her bir bayi için var olan veriyi yaz.
3. Raporun sonunu yaz.

Bu algoritmada bulunan son satır ise iki ayrı fonksiyon tarafından hazırlanmaktadır.

Bu fonksiyonun da adından anlaşılacağı üzere, **bayiVerileriListesi** ve **donemToplam** dizilerine erişebilmesi gereklidir. Çıktı dosyaya yazdırılacağı için bu fonksiyonun aynı zamanda **ofstream** değişkenine erişebilir olması gereklidir. Bu fonksiyonun dört adet parametresi bulunmaktadır; **bayiVerileriListesi** dizisine erişim için gerekli parametre, **donemToplam** dizisine erişim için gerekli parametre, ilgili dizilerin büyüklüğünü tanımayan parametre ve **ofstream** değişkenine erişim için gerekli olan parametre. Fonksiyon aşağıdaki gibidir:

```

void    raporYaz(ofstream&    dosyaYaz,    bayiVerileri    list[],int    listeBuyukluk,    double
donemCiroVerisi[])
{
    dosyaYaz << "----- Yıllık Ciro Raporu -----" << endl;
    dosyaYaz << endl;
    dosyaYaz << "Bayi No    Dönem1    Dönem2    Dönem3    Dönem4    Toplam" <<
endl;
    dosyaYaz
    _____
    for (int indis = 0; indis < listeBuyukluk; indis++)
    {
        dosyaYaz << list[indis].bayiNo << " ";
        for (int donem = 0; donem < 4; donem++)
            dosyaYaz << setw(12) << list[indis].donemCiroVerisi[donem];
        dosyaYaz << setw(12) << list[indis].bayiCiro << endl;
    }
    dosyaYaz << "Toplam";
}

```

```
for (int donem = 0; donem < 4; donem++)  
    dosyaYaz << setw(12) << donemCiroVerisi[donem];  
    dosyaYaz << endl << endl;  
} //raporYaz fonksiyonu sona erdi
```

### enBuyukCirosuOlanBayi Fonksiyonu

Bu fonksiyon tüm bayiler arasında en yüksek ciroyu elde etmiş olan bayinin belirlenmesi için kullanılmaktadır. Fonksiyon tüm satış verilerini tarayarak en yüksek satış değerine ulaşmış olan bayinin numarasını döndürmektedir. Bunun için de **bayiVerileri** dizisine erişmek durumundadır. İşlem sonucu dosyaya yazılacağı için aynı zamanda **ofstream** değişkenine de erişebilmelidir. Bu nedenle bu fonksiyonun üç adet parametresi olmaktadır. Birincisi ilgili diziyi tanımlayan parametre, ikincisi dizinin boyutunu tanımlayan parametre ve üçüncüsi de ofstream değişkenin tanımlayan parametredir.

Burada kullanılan algoritma daha önce bir dizideki en büyük elemanı bulmak için kullanılan algoritma ile aynıdır. Fonksiyon aşağıdaki gibidir.

```
void enBuyukCirosuOlanBayi(ofstream& veriYaz, bayiVerileri list[], int listeBuyukluk)  
{  
    int indisMax = 0;  
    for (int indis = 1; indis < listeBuyukluk; indis++)  
        if (list[indisMax].bayiCiro < list[indis].bayiCiro)  
            indisMax = indis;  
    veriYaz << "En büyük yıllık ciroya sahip olan bayının kimliği = " << list[indisMax].bayiNo  
<< ", Miktar = " << list[indisMax].bayiCiro << " TL" << endl;  
} // enBuyukCirosuOlanBayi fonksiyonu sona erdi
```

### enBuyukCirosuEldeEdilenDonem Fonksiyonu

Bu fonksiyon en büyük cironun elde edildiği dönemi belirlemektedir. Her bir döneme ait olan veri **donemToplam** adlı dizide barındırıldığı için bu dizide erişmesi gereklidir. Ayrıca fonksiyonun sonucunun dosyaya yazılacağı da dikkate alındığında **ofstream** değişkenine de erişmesi gereklidir. Bunda dolayı bu fonksiyonun ik adet parametresi olacaktır. Birinci parametre **donemToplam** dizisine erişim, ikincisi de **ofstream** değişkenine erişim için kullanılan parametredir.

Burada kullanılan algoritma daha önce bir dizideki en büyük elemanı bulmak için kullanılan algoritma ile aynıdır. Fonksiyon aşağıdaki gibidir.

```
void enBuyukCiroEldeEdilenDonem(ofstream& veriYaz, double donemCiroVerisi[])  
{  
    int indisMax = 0;  
    for (int donem = 0; donem < 4; donem++)
```

```

if (donemCiroVerisi[indisMax] < donemCiroVerisi[donem])

    indisMax = donem;

    veriYaz << "En büyük cironun elde edildiği dönem = " << indisMax + 1 << ", Miktar = "
<< donemCiroVerisi[indisMax] << " TL" << endl;

} // enBuyukCiroEldeEdilenDonem fonksiyonu sona erdi

```

## main Fonksiyonu

Ana fonksiyonun algoritması aşağıdaki gibi olacaktır.

1. Değişkenleri tanımla
2. Kullanıcı tarafından programın işlem yaparken kullanacağı bayi kimlik numaralarının bulunduğu dosyasının adını iste.
3. Dosyanın adını oku.
4. Okumak için dosyayı aç.
5. Eğer okunacak olan dosya bulunmuyor ise programı sonlandır.
6. **bayiVerileri** dizinini etkinleştir ve **etkinleştir** fonksiyonunu çağır.
7. Bayi kimlik verilerinin bulunduğu dosyayı kapat.
8. Kullanıcı tarafından programın işlem yaparken kullanacağı bayilere ait satış verilerinin bulunduğu dosyanın adını iste.
9. Dosyanın adını oku.
10. Okumak için dosyayı aç.
11. Eğer okunacak olan dosya bulunmuyor ise programı sonlandır.
12. Kullanıcı tarafından programın üreteceği sonuçları yazacağı dosyanın adını iste.
13. Dosyanın adını oku.
14. Dosyayı aç.
15. Çıktıda sayıların ondalık bamağın sayısının iki basamak ile sınırlamak için fixed ve showpoint kullan ve sonuçların da aynı şekilde iki ondalık basamak ile sınırla.
16. **verilarıAl** fonksiyonunu çağır ve satış verilerini işle.
17. **donemToplamCiro** fonksiyonu çağır ve her bir döneme ait olan toplam satış değerini hesapla.
18. **bayiYillikCiro** fonksiyonu çağır ve her bir bayi için bir yıllık ciroyu hesapla.
19. **raporYaz** fonksiyonunu çağır ve sonuçları dosyaya yaz.
20. **enBuyukCirosuOlanBayi** fonksiyonunu çağır ve bir yılda en çok satış yapan bayiyi belirle.

21. **enBuyukCiroEldeEdilenDonem** fonksiyonunu çağır ve bir dönemde en çok satış yapan bayiyi belirle.
22. Dosyaları kapat
23. Programı sonlandır.

Burada kullanılan algoritmanın kaynak kodu aşağıdaki gibidir:

```
/*
 * bayi_rapor.cxx
 *
 *
 * Copyright 2023 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
```

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
VERİ  
\* VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include <fstream>
#include <iomanip>
#include <string>
using namespace std;
const int BAYI_SAYISI = 6;
struct bayiVerileri
{
    // Bayi kimlik numarası
    string bayiNo;
    // Dönemlere ait ciro verisinin barındırılacağı dizi
    double donemCiroVerisi[4];
    // Tekil bayi ciro verisi
    double bayiCiro;
};

// Prototip fonksiyonlar
void etkinlestir(ifstream& veriOku, bayiVerileri list[], int listeBuyukluk);
void verileriAl(ifstream& dosyaAc, bayiVerileri list[], int listeBuyukluk);
void donemToplamCiro(bayiVerileri list[], int listeBuyukluk, double donemCiroToplam[]);
void bayiYillikCiro(bayiVerileri list[], int listeBuyukluk);
```

```
void raporYaz(ofstream& dosyaYaz, bayiVerileri list[], int listeBuyukluk, double donemCiroVerisi[]);

void enBuyukCirosuOlanBayi(ofstream& veriYaz, bayiVerileri list[], int listeBuyukluk);
void enBuyukCiroEldeEdilenDonem(ofstream& veriYaz, double donemCiroVerisi[]);

int main()
{
    // Adım 1

    // Verilerin okunması için değişken tanımlaması
    ifstream dosyaAc;

    // Verilerin yazılması için değişken tanımlaması
    ofstream dosyaYaz;

    // Verilerin okunacağı dosya adının barındırıldığı değişken
    string verilerinOkunacakDosya;

    // Verilerin yazılacağı dosya adının barındırıldığı değişken
    string verilerinYazilacakDosya;

    // Her döneme ait toplam ciro bilgisinin barındırılacağı dizi
    double donemToplam[4];

    // Bayilerin satış verilerinin barındırılacağı dizi
    bayiVerileri bayiVerileriListesi[BAYI_SAYISI];

    // Adım 2

    cout << "Bayi kimlik numaralarının bulunduğu dosyanın adını giriniz: ";

    // Adım 3

    cin >> verilerinOkunacakDosya;

    cout << endl;

    // Adım 4

    dosyaAc.open(verilerinOkunacakDosya.c_str());

    // Adım 5

    if (!dosyaAc)

    {
        cout << "Dosya bulunamadı, program sonlanıyor." << endl;
    }
}
```

```
        return 1;  
    }  
  
    // Adım 6  
    etkinlestir(dosyaAc, bayiVerileriListesi, BAYI_SAYISI);  
  
    // Adım 7  
    dosyaAc.close();  
    dosyaAc.clear();  
  
    // Adım 8  
    cout << "Bayilerin ciro kayıtlarının bulunduğu dosyanın adını giriniz: ";  
    cin >> verilerinOkunacakDosya;  
  
    // Adım 9  
    cout << endl;  
  
    // Adım 10  
    dosyaAc.open(verilerinOkunacakDosya.c_str());  
  
    // Adım 11  
    if (!dosyaAc)  
    {  
        cout << "Dosya bulunamadı, program sonlanıyor." << endl;  
        return 1;  
    }  
  
    // Adım 12  
    cout << "Raporun yazılacağı dosyanın adını giriniz: ";  
  
    // Adım 13  
    cin >> verilerinYazilacakDosya;  
    cout << endl;  
  
    // Adım 14  
    dosyaYaz.open(verilerinYazilacakDosya.c_str());  
  
    // Adım 15  
    dosyaYaz << fixed << showpoint << setprecision(2);  
  
    // Adım 16  
    verileriAl(dosyaAc, bayiVerileriListesi, BAYI_SAYISI);
```

```

// Adım 17
donemToplamCiro(bayiVerileriListesi,BAYI_SAYISI,donemToplam);

// Adım 18
bayiYillikCiro(bayiVerileriListesi, BAYI_SAYISI);

// Adım 19
raporYaz(dosyaYaz, bayiVerileriListesi,BAYI_SAYISI,donemToplam);

// Adım 20
enBuyukCirosuOlanBayi(dosyaYaz, bayiVerileriListesi,BAYI_SAYISI);

// Adım 21
enBuyukCiroEldeEdilenDonem(dosyaYaz, donemToplam);

// Adım 22
dosyaAc.close();
dosyaYaz.close();

return 0;

}

/* Aşağıda yukarıda yer verilen prototip fonksiyonlar bulunmaktadır.

*
* etkinlestir
* verileriAl
* donemToplamCiro
* bayiYillikCiro
* raporYaz
* enBuyukCirosuOlanBayi
* enBuyukCiroEldeEdilenDonem
*
*/
void etkinlestir(ifstream& veriOku, bayiVerileri list[],int listeBuyukluk)
{
    for (int indis = 0; indis < listeBuyukluk; indis++)
    {
        veriOku >> list[indis].bayiNo; // Bayi kimlik bilgisini oku

```

```

        for (int donem = 0; donem < 4; donem++)
            list[indis].donemCiroVerisi[donem] = 0.0;
        list[indis].bayiCiro = 0.0;
    }

} // etkinlestir fonksiyonu sona erdi

void verileriAl(ifstream& dosyaAc, bayiVerileri list[], int listeBuyukluk)
{
    int indis;
    int donem;
    string bID;
    int ay;
    double tutar;
    // Bayi kimlik bilgisini oku
    dosyaAc >> bID;
    while (dosyaAc)
    {
        // Takvim ayını ve o aya ait olan tutarı oku
        dosyaAc >> ay >> tutar;
        for (indis = 0; indis < listeBuyukluk; indis++)
            if (bID == list[indis].bayiNo)
                break;
        if (1 <= ay && ay <= 3)
            donem = 0;
        else if (4 <= ay && ay <= 6)
            donem = 1;
        else if (7 <= ay && ay <= 9)
            donem = 2;
        else
            donem = 3;
        if (indis < listeBuyukluk)
            list[indis].donemCiroVerisi[donem] += tutar;
    }
}

```

```

else
    cout << "Geçersiz bayi kimlik numarası tespit edildi." << endl;
    dosyaAc >> bID;
} // while döngüsü sona erdi
} // verileriAl fonksiyonu sona erdi
void donemToplamCiro(bayiVerileri list[], int listeBuyukluk, double donemCiroToplam[])
{
    for (int donem = 0; donem < 4; donem++)
        donemCiroToplam[donem] = 0.0;
    for (int donem = 0; donem < 4; donem++)
        for (int indis = 0; indis < listeBuyukluk; indis++)
            donemCiroToplam[donem] += list[indis].donemCiroVerisi[donem];
} // donemToplamCiro fonksiyonu sona erdi
void bayiYillikCiro(bayiVerileri list[], int listeBuyukluk)
{
    for (int indis = 0; indis < listeBuyukluk; indis++)
        for (int donem = 0; donem < 4; donem++)
            list[indis].bayiCiro += list[indis].donemCiroVerisi[donem];
} // bayiYillikCiro fonksiyonu sona erdi
void raporYaz(ofstream& dosyaYaz, bayiVerileri list[], int listeBuyukluk, double donemCiroVerisi[])
{
    dosyaYaz << "----- Yıllık Ciro Raporu -----" << endl;
    dosyaYaz << endl;
    dosyaYaz << "Bayi No   Dönem1   Dönem2   Dönem3   Dönem4   Toplam" << endl;
    dosyaYaz
    for (int indis = 0; indis < listeBuyukluk; indis++)
    {
        dosyaYaz << list[indis].bayiNo << " ";

```

```

for (int donem = 0; donem < 4; donem++)
    dosyaYaz << setw(12) << list[indis].donemCiroVerisi[donem];
    dosyaYaz << setw(12) << list[indis].bayiCiro << endl;
}

dosyaYaz << "Toplam";
for (int donem = 0; donem < 4; donem++)
    dosyaYaz << setw(12) << donemCiroVerisi[donem];
    dosyaYaz << endl << endl;
} //raporYaz fonksiyonu sona erdi

void enBuyukCirosuOlanBayi(ofstream& veriYaz, bayiVerileri list[], int listeBuyukluk)
{
    int indisMax = 0;
    for (int indis = 1; indis < listeBuyukluk; indis++)
        if (list[indisMax].bayiCiro < list[indis].bayiCiro)
            indisMax = indis;

    veriYaz << "En büyük yıllık ciroya sahip olan bayinin kimliği = " << list[indisMax].bayiNo
    << ", Miktar = " << list[indisMax].bayiCiro << " TL" << endl;
} // enBuyukCirosuOlanBayi fonksiyonu sona erdi

void enBuyukCiroEldeEdilenDonem(ofstream& veriYaz, double donemCiroVerisi[])
{
    int indisMax = 0;
    for (int donem = 0; donem < 4; donem++)
        if (donemCiroVerisi[indisMax] < donemCiroVerisi[donem])
            indisMax = donem;

    veriYaz << "En büyük cironun elde edildiği dönem = " << indisMax + 1 << ", Miktar = "
    << donemCiroVerisi[indisMax] << " TL" << endl;
} // enBuyukCiroEldeEdilenDonem fonksiyonu sona erdi

```

Program kullanılan bayiler.dat dosyasının içeriği aşağıdaki gibidir:

```

12345
32214
23422

```

57373

35864

54654

Program kullanılan bayi\_ciro.dat dosyasının içeriği aşağıdaki gibidir:

12345 1 896

32214 1 456

23422 1 245

57373 1 566

35864 1 987

54654 1 234

12345 2 906

32214 2 461

23422 2 876

57373 2 900

35864 2 987

54654 2 354

12345 3 806

32214 3 261

23422 3 276

57373 3 390

35864 3 907

54654 3 540

12345 4 609

32214 4 613

23422 4 480

57373 4 890

35864 4 707

54654 4 510

12345 5 703

32214 5 345

23422 5 267

57373 5 903  
35864 5 479  
54654 5 509  
12345 6 561  
32214 6 368  
23422 6 260  
57373 6 498  
35864 6 774  
54654 6 541  
12345 7 863  
32214 7 266  
23422 7 476  
57373 7 592  
35864 7 887  
54654 7 543  
12345 8 916  
32214 8 616  
23422 8 264  
57373 8 391  
35864 8 970  
54654 8 507  
12345 9 867  
32214 9 266  
23422 9 576  
57373 9 490  
35864 9 567  
54654 9 501  
12345 10 862  
32214 10 263  
23422 10 476  
57373 10 790

35864 10 677

54654 10 347

12345 11 800

32214 11 366

23422 11 609

57373 11 396

35864 11 771

54654 11 509

12345 12 761

32214 12 367

23422 12 577

57373 12 492

35864 12 971

54654 12 549

## 9.Sınıflar ve Veri Soyutlama

Önceki bölümde farklı veri tiplerinde tanımlanan verilerin birlikte kullanılarak yapılandırılmış veri tiplerinin nasıl edildiği gösterilmiştir. Bunun için de **struct** kullanılmıştır. C++ Programlama Dili’nde **struct** için verilen tanım aynı zamanda C programlam Dili’ndeki tanıma son derece yakındır. C++ Programlama Dili’nde **struct**, C programlam Dili’ndeki **struct** ile karşılaşıldığında yapılandırılmış veri tipinin sadece standart veri tiplerinin bir araya getirilmesinden değil aynı zamanda fonksiyonlarında kullanılabilirliğine olanak sağlamakta olması ile ayrılmaktadır.

Bundan başka C++ Programlama Dili bir diğer yapılandırılmış veri tipi sunmaktadır. Bu veri tipi belirli bir grup veri üzerinde gerçekleştirilen işlemlerden oluşan veri tipi olan **sınıf/class**’tir. Bu bölüm **class/sınıf** nasıl kurgulduğunu ve kullanıldığını öğretirken ayrıca **struct** ile ile benzerlikleri ve ayırdıkları noktalar üzerinde durmaktadır.

### 9.1.Sınıflar

İlk bölümde bilgisayar ile problemlerin çözülmesinde kullanılan yöntemlerden birisi olan **nesne yönelimli programlama/object oriented programming** kavramından söz edilmiştir. Bu kavramın veriler ile bu veriler üzerinde gerçekleştirilen işlemlerden olduğu belirtilmiştir. Nesne yönelimli programlama paradigmına göre öncelikle sınıfın bileşenleri olan **nesnelerin/objects** tanımlaması gereklidir. **Bir nesne temel olarak veri ile bu veri üzerinde gerçekleşen işlemleri bir araya getirilmesi ile oluşan bir birimdir.**

Bu tanımdan da anlaşılacağı üzere, bir sınıf/class, belirli bir sayıdaki bileşenden oluşur. Her sınıfın/class bileşenleri de bu **sınıfin üyeleri/members of the class** olarak tanımlanır.

C++ programlama Dili’nde bir sınıfın tanımlanması aşağıda gösterildiği gibi yapılır.

```
class sınıfınTanımlanması
{
    sınıfınÜyeleri
}
```

Yukarıdaki gösterimde yer verilen **sınıfinÜyeleri**, değişken tanımlamları ve/veya fonksiyonlar olabilir. Buna göre bir değişken bir sınıfın üyesi ise bu sınıfın yapısında gerçekleştirilecek işlemlerde kullanılacak olan bir veri barındırmaktadır. Yanı şekilde de bir fonksiyon bir sınıfın yapısında tanımlandığında ilgili veri üzerinde işlem yapacağı anlaşılmaktadır.

Aşağıdaki örnekte acilanDers adlı bir sınıf tanımlanmıştır. Bu sınıfın yapısında bulunan değişkenler ile fonksiyonlar tanımlanmıştır.

```
class acilanDers
{
public:
    void dersBilgileriniDuzenle(string acilanDersAdi, acilanDersKodu, int kredi);
```

```

    void yazdir() const;

    int dersKredisiOku();

    string dersAdiOku();

private:

    string dersAdi;

    string dersKodu;

    int aktsKredisi;

};


```

Yukarıdaki tanımlamayı esas alarak bir sınıfın tanımlanması ile ilgili olarak şu noktaları vurgulamakta yarar var:

- Bir değişken bir sınıfın üyesi ise, diğer değişkenlerin tanımlandığı gibi tanımlanır.
- Bir fonksiyon bir sınıfın üyesi ise, fonksiyon prototip fonksiyon olarak tanımlanır ve yazılır.
- Bir fonksiyon bir sınıfın üyesi ise, fonksiyon sınıfın tüm üyelerine ve bu sınıfın üyesi olan değişkenlere doğrudan erişebilir. Bir başka deyişle, fonksiyon üyesi olduğu sınıfın üyelerinin verilerine erişebilir, bunlar üzerinde işlem yapabilir ve daha önemlisi de fonksiyonun işlem yapabilmesi için herhangi bir referans veya parametre tanımı yapılmasına gerek yoktur. Tek gerekli ve yeter koşul, gerekli olan tanımlamaların önceden yapılmış olması zorunluluğu vardır.

C++ Programlam Dili’nde “**class**” özel/ayrılmış bir sözcüktür. Bir veri tipi tanımlaması yapar ve bu nedenle de bellekte özel bir alan ayrılmaz. Ayrıca tanımlamanın sonunda “;” kullanılması zorunludur. Eğer “;” kullanılmaz ise derleyici yazım hatası mesajı döndürecektr.

Bir sınıfın üyeleri üç grupta toplanır: **Genel/public**, **özel/private** ve **korumalı/protected**. Bu bölümde **genel/public** ve **özel/private** üyeleri üzerinde durulacaktır. C++ Programlam Dili’nde **genel/public**, **özel/private** ve **korumalı/protected** özel/ayrılmış sözcüklerdir ve **üyelik erişim tanımlayıcıları/member access specifiers** olarak standartta tanımlanır.

Aşağıda genel/public ve özel/private oalrak tanımlanan sınıf üyeleri ile çalışan programcının dikkat etmesi gereken önemli noktalardan bazılarıdır.

- Ön tanımlı olarak bir sınıfın tüm üyeleri **özel/private** olarak tanımlanır.
- Eğer bir sınıfın üyesi **özel/private** olarak tanımlanmış ise, sınıfın dışından erişilmesi olanağlı değildir.
- Eğer bir sınıfın üyesi **genel/public** olarak tanımlanmış ise, sınıfın dışından erişilebilir
- Eğer bir sınıfın üyesi **genel/public** olarak tanımlanacak ise “**public:**” olarak belirtilmesi gereklidir.

Gün işinden daha fazla tatarlanmak için yapılan zaman ayarlamalarının yararlılığı tartışılı dursun bunu uygulayan ülkelerde yaz ve kış saati uygulamaları arasındaki geçiş otomatik olarak

yapılır. Ancak bir çok aygit üzerinde zaman dilimi veya yerel zaman ayarını düzenlemek için bir uygulama bulunur. Aşağıdaki örnekte aygitin belirttiği zamanı saat, dakika ve saniye cinsinden okuyabilen, uygulamanın ise yapısında zaman bilgisini kullanıcının belirttiği şekilde uygulayan, gerektiğinde de kullanıcı tarafından belirtilen zaman bilgisini uygulan bir uygulamaya yer verilmiştir.

Bu uygulamada söz konusu olan değişkenler saat, dakika ve saniyedir. Buna göre şu değişkenler tanımlanmalıdır.

```
int saat;  
int dakika;  
int saniye;
```

Ayrıca bu program ile şu işlemleri gerçekleştirecek olalım.

- Uygulamanın saat, dakika ve saniye ayarlarını yapmak
- Aygitin saatinden zaman verisini okumak (Burada zaman dilimi UTC olarak alınmaktadır)
- Saat, dakika ve saniye yazdırma
- Saniyeyi birer artırmak veya azaltmak
- Dakikayı birer artırmak veya azaltmak
- Saati birer artırmak veya azaltmak
- İki farklı zaman bilgisini karşılaştırmak

Sözü edilen işlemlerin yapılabilmesi için yedi adet fonksiyonun yazılması gereklidir. Bunlar zamanAyarla, zamanOku, zamanYaz, saniyeDegistir, dakikaDegistir, saatDegistir ve zamanKontrol.

Bu göre saatUygulaması adlı sınıfın on adet üyesi olacaktır, bunların yedi tanesi fonksiyon olurken üç tanesi de değişkendir. Bu sınıfın bazı üyeleri özel ve diğerleri genel olarak tanımlanacaktır. Burada bu tanımlamalar yapılmadan önce hangi üyelerin genel ve hangilerinin özel olacağıın ayrılmının yapılması gereklidir. Genel olarak tanımlanan üyeler sınıfın dışından da erişilebilecek olanlar olurken, özel üyeler ise sadece sınıfın içerisinde erişilebilecektir. Örneğin zamanAyarla ve zamanYaz adlı fonksiyonlar kullanıcı tarafından çağrılp çalıştırılabilceği için genel olarak tanımlanmalıdır. Benzer olarak saatı dakika ve saniye düzenlemeleri yapan fonksiyonlar ile aygit ve uygulama arasındaki zaman eşgündümünü kontrol edecek olan fonksiyonlar da genel olarak tanımlanmak durumundadır. Aşağıda saatUygulaması sınıfı tanımlanmıştır.

```
class saatUygulaması  
{  
public:  
    void zamanAyarla(int, int, int);  
    void getTime(int&, int&, int&) const;  
    void zamanYaz() const;
```

```

void saniyeDegistir();

void dakikaDegistir();

void saatDegistir();

bool zamanKontrol(const saatUygulaması&) const;

private:

    int sa;

    int dk;

    int sn;

};

```

Yukarıda yapılan sınıf bildiriminde şu tanımlamalar yapılmıştır:

- saatUygulaması sınıfında yedi adet üye fonksiyon bulunmaktadır: "zamanAyarla, zamanOku, zamanYaz, saniyeDegistir, dakikaDegistir, saatDegistir ve zamanKontrol. Ayrıca üç adet tam sayı veri tipinde değişken tanımlanmıştır; saniye, dakika ve saat.
- Tanımlanan değişkenler sınıfa özeldir ve dışarıdan erişilemez.
- Sınıfta yer alan fonksiyonlar genel olarak tanımlanmıştır ve sınıfın dışından da erişilebilir durumdadırlar. Ayrıca bu fonksiyonlar doğrudan özel olarak tanımlanmış olan değişkenlere erişebilmektedir. Bir sınıf bildirimini yapıldığında ve bu sınıfın üyesi olan genel ve özel fonksiyonlar ile değişkenler tanımlandığında, fonksiyonların bu değişkenler üzerinden işlem yapabilmesi için parametre olarak tanımlanmasına gerek yoktur.
- zamanKontrol fonksiyonunda kullanılan formal parametre, sabit bir referanstır. Bu fonksiyon çağrıldığında, asıl parametrenin bulunduğu bellek adresini okuyarak veriye erişmektedir. Burada formal parametre, referans edilen parametrenin değeri üzerinde işlem yapmamaktadır. Eğer formal parametre referans olarak değil de değer olarak tanımlanmış olsaydı, formal parametrenin asıl işlem yapacağı veriyi ilgili bellek adresinden kopyalayarak üzerinde işlem yapması gerekecekti. (Bu konu ilerleyen bölümlerde ayrıntılı olarak açıklanacaktır.) Bu işlemler performansa olumsuz etki edecektir.
- zamanOku, zamanYaz ve zamanKontrol fonksiyonlarının sonunda yer alan const terimi, bu fonksiyonların işlem yaptıkları üye değişkenlerin değerlerini değiştiremeyeceğini belirtmektedir.

zamanAyarla adlı fonksiyonun üç adet üye değişken üzerinde işlem yapmaktadır. Bunlar saat, dakika ve saniyedir. Söz konusu değişkenler parametre olarak tanımlanmış ve fonksiyon böylükle bu değişkenler üzerinde işlem yapabilmektedir. zamanYaz fonksiyonu ise zamanı saat, dakika ve saniye olarak yazdırmaktadır. saniyeDegistir, dakikaDegistir ve saatDegistir fonksiyonları kullanıcısı tarafından verilen değerlere göre uygulamada gösterilen saatı düzenlemektedir. zamanKontrol adlı fonksiyon ise aygıtın ve uygulamanın zaman bilgilerini karşılaştırmaktadır. Burada dikkat edilecek olursa bu fonksiyonun sadece tek bir değişkeni olduğu görülecektir. Ancak fonksi-

yonun karşılaştırma işlevini yapabilmesi için iki adet değişken gereksinimi vardır. Bu konu ilerleyen bölümlerde tekrar ele alınarak açıklanacaktır.

### 9.1.1.UML İle Sınıf Diagramları

Nesne yönelimli programlama paradigmada sınıflar, üyeleri ve bunlar arasındaki ilişkiler grafik olarak ifade edilebilir. Bunun için **UML – Unified Modelling Language** kullanılır. Aşağıdaki görsel de saatUygulaması adlı sınıfı ait olan UML diagramı görülmektedir.

saatUygulaması
-sa: int
-dk:int
-sn:int
+zamanAyarla(int, int, int): void
+zamanOku(int&, int&, int&) const: void
+ zamanYaz() const: void
+saniyeDegistir(): void
+dakikaDegistir(): void
+saatDegistir(): void
+zamanKontrol( cont saatUygulaması) const: bool

Şeklin en üst kısmında sınıfın adı yer almaktadır. Ortadaki bölümde değişkenler ve bu değişkenlerin veri tipleri tanımlanmıştır. Son bölümde ise sınıfın üyesi olan fonksiyonlar tanımlanmıştır. Fonksiyonun adı parametreleri ve değer döndürüp döndürmediği gösterilmiştir. Grafikte görülen “+” işaretü üyenin **genel** olarak tanımlandığını, “-” işaret ise üyenin **özel** olarak tanımlandığını belirtmektedir. Eğer üyenin adı önünde “#” işaretü bulunuyorsa, **korumalı** olarak tanımlandığını belirtmektedir.

### 9.1.2.Nesnelerin (Değişkenlerin) Tanımlanması

Bir sınıf tanımlandığında, bu sınıfın belirttiği değişkenler yani nesneler tanımlanabilir. C++ Programlam Dili’nde bir sınıf yapısında tanımlanmış olan bir değişkene **sınıfın nesnesi/class object** veya basitçe **object/nesne** adı verilir.

Bir nesnenin tanımlanması ile bir değişkenin tanımlanması arasında biçim olarak herhangi bir fark yoktur. Aşağıda yer verilen iki ifade saatUygulaması sınıfı için iki adet değişken tanımlaması yapılmıştır.

saatUygulaması uygulamaSaatı

saatUygulaması kullaniciSaatı

uygulamaSaatı	sa	14
	dk	32
	sn	45

kullaniciSaatı	sa	17
	dk	32
	sn	45

Yukarıda yapılan bildirimler ile oluşturulan iki değişkeninde on adet elemanı bulunmaktadır. Her iki nesne için saat, dakika ve saniye adlı değişkenler için ayrı bellek adresleri ayrılır. Derleyici burada sınıfın üyesi olan fonksiyonun bir fiziksel kopyasını oluşturur. Program çalıştırıldığında ise, her nesne için aynı fonksiyon kopyası çalıştırılır. Bu nedenle bir UML diyagramı hazırlandığında sadece üye değişkenler gösterilir. Yukarıda görülen grafikte yukarıda tanımlanmış olan iki adet nesneye ait değerler temsili olarak gösterilmektedir.

### 9.1.3.Sınıf Üyelerine Erişim

Bir sınıfın ait olan bir nesne tanımlandığında, nesne ait olduğu sınıfın üyelerine erişebilir. Erişimin sağlanmasını sağlayan yazım şekli ise aşağıdaki gibidir.

```
sınıfNesnesininAdı.uyeninAdı
```

Bu tanımlamanın yaniltıcı olan tarafı, bir sınıfta yaratılan nesnenin o sınıfa ait olan tüm üyelerere erişebileceği şeklindeki düşüncedir. Tersine olarak nesne, sınıf içerisinde nerede yaratılmış olduğuna bağlı olarak bazı sınıf üyelerine erişebilir.

- Eğer nesne, üye fonksiyonlarının içerisinde tanımlanmış ise, bu durumda nesne hem genel hem de özel üyelerde erişebilir. (Bunu Üye Fonksiyonların Uygulanması konusunda ayrıntılı olarak ele alacağız.)
- Eğer nesne başka bir yerde yaratılmış ise, örneğin başka bir uygulama içerisinde yaratılmış ise bu durumda sadece genel üyelerde erişebilir.

Burada bir kez daha tekrar anımsamakta yarar var. C++ Programlama Dili’nde **“.” üye erişim operatörü – member access operator** olarak tanımlanır. Aşağıdaki örnekte yukarıda tanımlanmış olan nesnelerin örnek bir program içerisinde kullanımı görülmektedir.

```
saatUygulaması uygulamaSaati
```

```
saatUygulaması kullaniciSaati
```

```
uygulamaSaati.zamanAyarla(18,3,4);
```

```
uygulamaSaati.zamanYaz();
```

```
// aşağıda belirtilen değişkenlerin int veri tipinde tanımlanmış olduğunu var sayalım.
```

```
kullaniciSaati.zamanAyarla( a, b, c);
```

```
if(uygulamaSaati.zamanKontrol(kullaniciSaati))
```

```
....
```

Örnek kaynak kodda görüleceği üzere, yukarıda yazılan kod yazılım kurallarına göre doğrudur. İlk ifadede, uygulamaSaati.zamanAyarla(18,3,4) ile zamanAyarla() üye fonksiyonu çağrılmış ve çalıştırılmıştır. Bu fonksiyon da üye değişkenler olan saat, dakika ve saniye değişkenlerini işlemiştir. İkinci ifade benzer şekilde, zamanYaz üye fonksiyonunu çağrılmış ve fonksiyonun çalış-

tırılıp sonucun yazdırılmasını sağlamıştır. Üçüncü ifadede ise, kullanıcıZamanı nesnesine üç adet değer ataması yapılmıştır. Dördüncü ifadede zamanAyarla() fonksiyonu için üç adet değişkeni iki farklı nesne için karşılaştırılmaktadır. Söz konusu fonksiyon uygulamaSaati ve kullanıcıSaati nesnelerinin de üyesi olduğu için her iki nesnedeki değişkenlere de erişebilmektedir. Bu nedenle önceki kısımlarda yapılan bildirimde iki nesnenin tanımlanmasına gerek duyulmamıştır, bundan dolayı da tek parametreye sahiptir.

Yukarıda tanımlanmış olan saatUygulaması sınıfının yapısında iki adet nesne bulunmaktadır. Bu nesneler, uygulamaSaati ve kullanıcıSaati nesneleri olup, söz konusu olan sınıfın sadece genel olarak tanımlanmış üyelerine erişilebilmektedirler. Bu nedenle aşağıda gösterilen bildirimler, teknik olarak geçersizdir. Çünkü özel olarak tanımlanmış sınıf üyelerine erişim tanımlamaktadırlar.

uygulamaSaati.sa = 9;	// Geçersizdir
-----------------------	----------------

uygulamaSaati.dk = kullanıcıSaati.dk;	// Geçersizdir.
---------------------------------------	-----------------

#### **9.1.4. Sınıflar Üzerinde İşlem Yapan Hazır Fonksiyonlar**

C++ Programlama Dili’nde yerleşik olan bir çok fonksiyon ile sınıflar üzerinde işlem yapılması olanaklı değildir. (Eğer fonksiyonların aşırı yüklenmesi söz konusu değil ise. Bu konu daha sonraki bölümlerde ele alınacaktır.) Sınıflar üzerinde aritmetik işlemler yapılamaz, iki sınıfa ait olan nesneler birleştirilemez, hatta iki ayrı sınıf veya bu sınıflara ait olan ensemeler arasında karşılaştırma yapmak için ilişkisel operatörler kullanılamaz.

Sınıflar üzerinde yapılabilecek olan işlemleri barındıran hazır fonksiyonlar iki adettir. Birincisi üyelik erişim(.) fonksiyonu ve ikincisi de atama(=) fonksiyonudur. Önceki bölümlerde üyelik erişim fonksiyonunun erişim için nasıl kullanıldığını görmüştük.

#### **9.1.5. Sınıflar Üzerinde Hazır Fonksiyonlar İle Atama İşlemleri**

Önceki kısımlarda tanımlanmış olan saatUygulaması adlı sınıfa ait olan uygulamaSaati ve kullanıcıSaati adlı iki nesnenin olduğunu varsayıyalım. İlk nesnenin adında anlaşılacağı gibi aygıtta zaman bilgisini okuyup bu bilgileri uygulamanın zaman bilgisi olarak kullanıcıSaati nesnesine atasın. İlk şekilde atam işlemi öncesi durum ve ikinci şekilde de atama yapıldıktan sonraki durum gösterilmektedir.

uygulamaSaati	sa	14
	dk	32
	sn	45

kullaniciSaati	sa	17
	dk	32
	sn	45

Atama Öncesi durum

uygulamaSaati	sa	14
	dk	32
	sn	45

kullaniciSaati	sa	14
	dk	32
	sn	45

Atama sonrası durum

Yukarıd gösterilen işlemin gerçekleşmesi için aşağıdaki ifadenin işlenmesi gereklidir.

uygulamaSaati =kullaniciSaati;

Bu ifade işlendiğinde şu işlemler gerçekleşir.

- uygulamaSaati.sa verisi kullaniciSaati.sa atanır.
- uygulamaSaati.dk verisi kullaniciSaati.dk atanır.
- uygulamaSaati.sn verisi kullaniciSaati.sn atanır.

Basitçe söyleyecek olursak, uygulamaSaati nesnesinin üyesi olan üç değişkenin değeri kullaniciSaati adlı nesnenin üyesi olan üç değişkene sırası ile atanmış olmaktadır. Bu atama işlemi değişken özelinde gerçekleşmektedir. Atamalar bire bir karşılık oldukları değişkenler arasında gerçekleşmektedir.

### 9.1.5.Sınıfın Kapsamı

Sınıflar otomatik veya statik bir yapıda olabilir. Eğer otomatik yapıda olan bir sınıf söz konusu olduğunda, bir program içerisinde sınıfın tanımlandığı kod bloku işlenmeye başladığında sınıf ve yapısında tanımlanmış olan nesneler yaratılır ve söz konusu kod bloğunun işlenmesi sona erdiğinde de yok edilir. Eğer statik yapıda olan bir sınıf söz konusu olduğunda, bir program içerisinde sınıfın tanımlandığı kod bloku işlenmeye başladığında sınıf ve yapısında tanımlanmış olan nesneler yaratılır ve söz konusu kod bloğunun işlenmesi sona erdiğinde de yok edilmez. Sınıf ve yapısında tanımlanmış olan nesneler ancak program sonlandırıldığında yok edilir.

Ayrıca nesneler, aynı zamanda, bir dizi gibi de tanımlanabilir. Bir sınıfın yapısında tanımlanmış olan nesneler diğer değişkenlerde olduğu gibi nesne ile aynı kapsama uymaktadır. Bu açıdan bakıldığından **struct** kullanılarak kurgulanmış bir yapının kapsamı ile ilgili sınıfın içerisinde tanımlanmış bir nesnenin veya nesnelerin de kapsamı aynı olduğu görülür. Her iki yapıda da bir üyeye erişmek için **üye erişim operatörü – member access operator** kullanılır.

### 9.1.6.Fonksiyonlar ve Sınıflar

Aşağıda sınıflar ile fonksiyonlar arasındaki ilişkiler ile bu ilişkileri düzenleyen kurallar verilmiştir:

- Sınıfların nesneleri bir fonksiyona parametre olarak aktarılabilir ve fonksiyonun işlem sonucu da değer olarak döndürülür.
- Sınıf nesneleri fonksiyonlara parametre veya referans olarak aktarılabilir.
- Bir sınıfın nesnesi bir fonksiyona parametre olarak aktarılırsa, sınıfın üyesi olan değişkenlerin barındırdığı veriler fonksiyonun tanımında belirtilen parametrelerden hangisine karşılık geliyor ise ona kopyalanarak aktarılır.

### 9.1.7.Sınıfların Nesneleri ve Referans Parametreleri

Bir değişken değer olarak bir fonksiyona atandığında, fonksiyonun parametresi, asıl değişkenin değerini ilgili bellek adresinden okuyarak ayrılmış olan bellek alanına kopyalanmasını gerçekleştirir. Bir sınıfın nesnesi de benzer şekilde bir fonksiyonun parametresine değer olarak benzer şekilde kopyalanır.

Bir programda bir sınıfın bir çok sayıda değişken üyesi olduğunu var sayalım. Bu durumda her bir değişken için veri tipine göre bellekte alan ayrılması gerekecektir. Bu değişkenleri bir fonksiyona değer olarak atanması gerekiğinde, derleyici, bu işlem için ayrıca ilgili fonksiyonun erişebileceği ve bu değerleri okuyup yazabileceği ek bir alanın da ayrılmasını sağlamalıdır. Bu işlem, öncelikle ilgili bellek alından her bir verinin okunmasını, ardından da verilerin kopyalanıp yeni bir bellek alanına atanmasını gerektirmektedir. Aynı verinin iki kopyası olacağı için yapılan işlemde bağımsız olarak verinin iki katı büyülüüğünde bir bellek alanı ayrılması ile bu iki alan arasındaki kopyalama ve yazma işlemlerinin sayısı da düşünüldüğünde bunun verimsiz ve etkin olmayan bir işlem olduğu kolaylıkla görülebilir.

Aynı durum bu sefer değer olarak atamak yerine, fonksiyona referans olarak aktarılması durumunda ise, fonksiyon doğrudan ilgili verinin bulunduğu bellek adresini okuyarak buradan veriye erişerek işlem yapacaktır. Çok sayıdaki veri üzerinde yapılacak olan işlemler için referans aktarımı, doğrudan diğer aktarımı göre daha etkili ve verimli bir yöntemdir. Çünkü asıl veri değiştiğinde referansın belirttiği bellek adresi sabit kalırken, içeriği değişmektedir. Fonksiyon ise sadece ilgili bellek adresinde değeri okuyarak işlem yapmaktadır. İşlem sonucu doğrudan referans olarak aktarılan bellek adresine yazılmaktadır.

Bazen de program içerisinde bir verinin referans olarak bir fonksiyona aktarılırak işlem yapılması ancak işlem sonucunun ise söz konusu kaynak verinin bellek adresine yazılmaması istenildir. Bunu yapabilmek için C++ Programlam Dili’nde **const** özel sözcüğü kullanılır. Burada const formal parametre tanımlanırken kullanılır. Aşağıdaki örnekte bu durumu görebiliriz.

```
void zamanKontrol(const saatUygulaması&, saatBilgisi)
{
    saatUygulaması aSaat;
    ...
}
```

Yukarıda gösterilen zamanKontrol fonksiyonunda yer alan digerSaat adlı parametre referans parametresidir. Aynı zamanda da **const** özel sözcüğü ile digerSaat içeriğinin değiştirilemeyeceği belirtilmiştir. zamanKontrol adlı fonksiyon sadece söz konusu referans parametresine aktarılan veriyi okuyabilir, bu veri üzerinde işlem yapabilir ama içeriğini değiştiremez. Eğer söz konusu fonksiyona aşağıdaki gibi bir tanımlama yapılacak olursa, derleyici bu bildirimi işlerken hata mesajı döndürecektr.

```
zamanKontrol(uygulamaSaati);
```

Eğer bir sınıfın üyesi olan nesnenin bir değer atanabilen bir nesne olarak tanımlanması isteniyor ise, bir referans parametresi olarak tanımlanması gereklidir. Bunun için de **const** özel sözcüğü ile yukarıda gösterildiği gibi tanımlanmalıdır.

Burada bir noktayı tekrar anımsamakta yarar var: Bir fonksiyonun tanımlanırken belirtilen bir formal parametreye değer atanabiliyor ise, atama ifadesi kullanılarak bu fonksiyonun formal parametresinin değeri değiştirilebilir. Ancak bu formal parametre bir referans parametresi olarak tanımlanmış ise, bu durumda söz konusu parametrenin işaret ettiği bellek adresindeki veriyi oku-

yabilir. Bu da bu parametrenin değerinin ne programcı tarafından nede bir başka fonksiyon aracılığı ile veya bir atama ifadesi kullanılarak düzenlenmeyeceği anlamına gelir. Fonksiyon içerisinde bu parametrenin değeri okunduktan sonra değiştirilemez. Yukarıdan belirtilen zamanKontrol fonksiyonunda tanımlanmış olan digerSaat adlı nesnenin değeri değiştirilemez. Aşağıdaki örnekte bu durum gösterilmiştir.

```
saatBilgisi.zamanAyarla(17,0,0); //Geçersiz işlem  
saatBilgisi = aSaat; //Geçersiz işlem
```

### 9.1.8. Üyelik Fonksiyonlarının Uygulanması

Yukarıda **saatUygulaması** adlı sınıfı tanımladığımızda, sadece üye fonksiyonlar için prototip fonksiyon tanımlamalarını yapmıştık. Bu fonksiyonların etkin ve verimli bir şekilde çalışması için öncelikle algoritmalarının hazırlanması gereklidir. Bunun yapılabilmesi için uygulanabilecek ilk yöntem, bu prototip fonksiyonların yerine fonksiyonun açık olarak tanımlanması yoludur. Bu ise kaynak kodu okuyan bir programcı için incelemekte olduğu sınıf kaynak kodunun uzaması ve karmaşıklaşmasına neden olurken, anlaşılmasıının da zorlaşacağı anlamına gelecektir. Burada sınıfın içerisinde fonksiyonun açık olarak yazılması yerine prototip olarak yazılması veri üzerinde gerçekleşecek olan işlemlerin nesne yönelimli programlama paradigmاسının dışına çıkması ve yazılan özel, genel ve korumalı ayrımlarının da başından etkisiz hale gelmesi gibi bir riski beraberinde getirmesi durumunu yaratacaktır. Bu konu ilerleyen bölümlerde tekrar ele alınacaktır.

```
class saatUygulaması  
{  
public:  
    void zamanAyarla(int, int, int);  
    void zamanOku(int&, int&, int&) const;  
    void zamanYaz() const;  
    void saniyeDegistir();  
    void dakikaDegistir();  
    void saatDegistir();  
    bool zamanKontrol(const saatUygulaması&) const;  
  
private:  
    int sa;  
    int dk;  
    int sn;  
};
```

Yukarıdaki kaynak kod içerisinde yer alan prototip fonksiyonlarının yazılması gereklidir. İlk olarak **zamanAyarla** fonksiyonu yazılacaktır. Sınıfın üyesi olan fonksiyonlar, doğası gereği sadece

sınıf içerisinde erişilebilirler. Bu nedenle bir fonksiyonun sınıf dışından da çağrılabilmesi için **kapsam çözümleme operatörü – scope resolution operator** kullanılması gereklidir. Bu operatörün kullanılabilmesi için öncelikle fonksiyonun adının sınıfı nadı ile aynı olması, “::” yazılıdıktan sonra da sınıfın üyesi olan fonksiyonun adının yazılması gereklidir. Aşağıda görülen **zamanAyarla** fonksiyonun sınıf dışından kapsam çözümleme operatörü kullanılarak tanımlandığı görülmektedir.

```
void saatUygulaması::zamanAyarla(int saat, int dakika, int saniye)
{
    if (0 <= saat && saat < 24)
        sa = saat;
    else
        sa=0;
    if (0<= dakika && dakika < 60)
        dk = dakika;
    else
        dk = 0;
    if (0 <= saniye && saniye < 60)
        sn = saniye;
    else
        sn = 0;
}
```

Yazılan **zamanAyarla** fonksiyon kullanıcı tarafından girilen saat, dakika ve saniye değerlerinin geçerli aralıkta olup olmadığını kontrol etmektedir. Eğer verilen değerler geçerli aralığın dışında ise değişkenlerin değerleri sıfır olarak atanmaktadır. Burada **saatUygulaması** sınıfının nesneleri tarafından **zamanAyarla** fonksiyonuna nasıl erişilip işlendiğini görelim.

**zamanAyarla** fonksiyonunun üç adet formal parametresi bulunmaktadır. Buna göre:

- Fonksiyonun çağrımması için tek bir yönerge yeterlidir.
- Fonksiyonun çağrımması için ayrıca üç adet parametreye de değer atanmış olması zorunludur.

Burada dikkat edilmesi gereken **zamanAyarla** fonksiyonun **saatUygulaması** sınıfının bir üyesi olduğu için bu sınıfın üyesi olan değişkenlere erişebilecek olmasıdır. Burada, **uygulamaSaati** adlı nesnenin **saatUygulaması** sınıfının üyesi olduğunu düşünelim. Bu nesnenin üç adet üye değişkeni bulunduğu için aşağıdaki durumu inceleyelim.

Program çalışırken aşağıda verilen bildirimin çalıştırıldığını düşünelim.

```
uygulamaSaati.zamanAyarla(2,14,2);
```

Bu ifade ile **zamanAyarla fonksiyonu** 2, 14 ve 2 parametreleri çağrılmaktır. Bu durumda fonksiyonun formal parametreleri olan saat, dakika, saniye sırası ile yukarı belirtilen değerleri alacaktır. Ardından **uygulamaSaatı.ZamanAyarla(2, 14, 2)** ifadesinde yer alan **zamanAyarla()** fonksiyonu **uygulamaSaatı** nesnesine erişecektir. Burada **zamanAyarla** fonksiyonunun parametreleri aynı zamanda **uygulamaZamani** nesnesinin de üyeleri durumundadır. Söz konusu fonksiyon çağrılp çalıştırıldığında, fonksiyonun parametreleri **uygulamaSaatı** nesnesine kopyalanacaktır. Daha açık olarak söyleyecek olursak, saatı barındıran parametre **uygulamaSaatı.sa**, dakikayı barındıran parametre **uygulamaSaatı.dk** ve saniyeyi barındıran parametre **de uygulamaSaatı.sn** kopyalanmış olacaktır. Fonksiyonun çalıştırılması ilse nesnenin durumu yukarıda görülen **(b)** durumundaki gibi olacaktır.

<b>uygulamaSaatı</b>	<b>sa</b>	
	<b>dk</b>	
	<b>sn</b>	

**(a)**

<b>uygulamaSaatı</b>	<b>sa</b>	2
	<b>dk</b>	14
	<b>sn</b>	2

**(b)**

Yukarıda tanımlanan üye fonksiyonlarının diğerlerini de sırası ile yazalım.

```
void saatUygulaması::zamanOku(int& st, int& dk, int& sn) const
{
    saat = st;
    dakika = dk;
    saniye = sn;
}

void saatUygulaması::zamanYaz(int& st, int& dk, int& sn) const
{
    if (st < 10)
        cout >> "0"
    cout >> st << ":";

    if (dk < 10)
        cout >> "0"
    cout >> dk << ":";

    if (sn < 10)
        cout >> "0"
    cout >> sn;
}

void saatUygulaması::void saniyeDegistir()
```

```

{
    sn++;
    if (sn > 59)
    {
        sn = 0;
        dakikaDegistir();
    }
}

void saatUygulamasi::void dakikaDegistir()
{
    dk++;
    if (dk > 59)
    {
        dk = 0;
        saatDegistir();
    }
}

void saatUygulamasi::void saatDegistir()
{
    st++;
    if (st > 23)
    {
        st = 0;
    }
}

```

Yukarıda gösterilen fonksiyonların tanımlanmasında da görüleceği üzere bir sınıfın üyesi olan bir fonksiyon, yine aynı sınıfın üyesi olan bir başka fonksiyonu çağrılabılır.

**zamanKontrol** fonksiyonu aşağıda verilmiştir.

```

bool saatUygulamasi::zamanKontrol(const saatUygulamasi& digerZaman) const
{
    return (sa == digerZaman.sa && dk == digerZaman.dk && sn == digerZaman.sn);
}

```

}

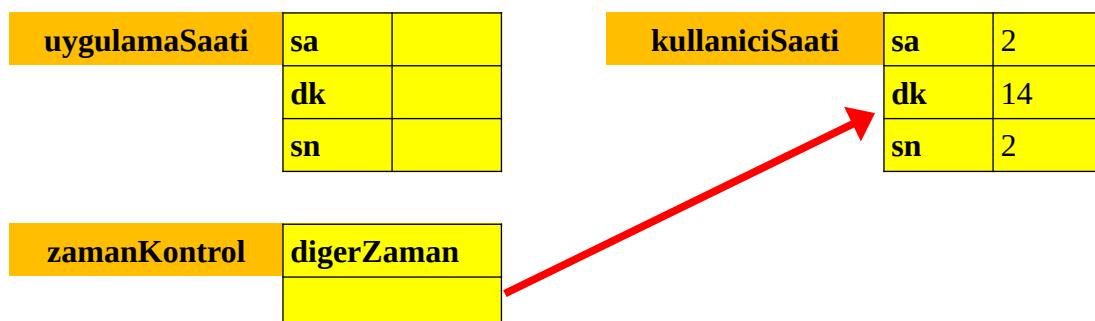
Yukarıda görülen **zamanKontrol** fonksiyonu **saatUygulaması** nesnesin üyesidir. Bu nesne program içerisinde aşağıda görüldüğü gibi çağrılığını düşünelim.

```
if (uygulamaSati.zamanKontrol(kullaniciSati))
```

....

Burada **uygulamaSati** nesnesi, **zamanKontrol** üye fonksiyonu tarafından erişilebilmektedir. Halbuki **zamanKontrol** fonksiyonda **digerZaman** referans parametresi olarak tanımlanmıştır. Bu durumda, **zamanKontrol** fonksiyonu, **kullaniciSati** adlı nesnenin adres bilgisini **digerZaman** adlı formal parametreye aktarmaktadır.

**uygulamaZamani** nesnesinin değişkenleri sırası 2, 14 ve 2 değerlerini barındırmaktadır. Söz konusu olan **zamanKontrol** fonksiyonu çalıştırıldığında ise karşılık gelen değişkenlerin değerleri **kullaniciSati** nesnesinin değerlerinin kopyası olacaktır. Ayrıca **zamanKontrol** fonksiyonu aynı zamanda **uygulamaZamani** nesnesinin de üyesi olduğundan, **zamanKontrol** fonksiyonu yapısındaki **digerZaman** nesnesinde barındırılmakta olan değerleri **uygulamaSati** nesnesinde barındırılan değişkenlerin değerleri ile karşılaştıracaktır. Burada tekrardan görüleceği üzere **zamanKontrol** nesnesinin sadece tek bir formal parametresi olması olağandır. Başka bir parametre gereksinim yoktur.



Bir defa daha **zamanKontrol** fonksiyonunu inceleyelim. Fonksiyonun tanımlanmasında yer verilmiş olan **digerZaman** nesnesi, üye değişkenler olan saat, dakika ve saniye erişebilmektedir. Ancak bu değişkenler **özel** olarak tanımlanmıştır. Burada şu soru sorulabilir: "Bir sınıfın üyeleri özel olarak tanımlandığında, bir diğer nesne tarafından erişilebilmesi hatalı olmaz mı?" Bu sorunun yanıt basitçe "Hayır"dır. Çünkü **saatUygulaması** sınıfının üyeleri arasında **zamanKontrol** fonksiyonunda da yer almaktadır. Ayrıca **digerZaman** adlı nesne de aynı sınıfın üyesidir. Böylece **digerZaman** nesnesi **zamanKontrol** fonksiyonu ile sınıfın **özel** üyelerine erişebilmektedir.

Bu erişim özelliği bir sınıfın tüm üyeleri için geçerlidir. Bunu bir örnek ile açıklayalım. Programın yapısında **ornekSınıf** adlı bir nesne tanımlanmış olsun. Bu sınıfın içerisinde **ornekSınıfFonksiyonu** adlı bir fonksiyon tanımlayalım. Ayrıca bu sınıfın üzerinde işlem yapacağı **ornekSınıfNesnesi** adlı bir de nesne tanımlanmış olsun. Böylece programın akışı içerisinde **ornekSınıf** adlı sınıfın içerisinde tanımlı olan **ornekSınıfFonksiyonu**, **ornekSınıfNesnesi** ile sınıfın tüm özel, genel ve korulamalı değişkenlerine erişebilir.

Bir sınıfın bir program içerisinde kullanılabilmesi için, öncelikle sınıfın doğru şekilde tanımlanmış ve uygulanmış olması gereklidir. Böylece program içerisinde sınıfın yapısında tanım-

lanmış olan nesnelere erişebilen ve bunları kullanabilen her öğe bu sınıfın bir **istemcisi** olarak tanımlanabilir. Bu tanımlama sadece bir programın içerisindeki farklı öğeler için değil aynı zamanda bir çok programın bir araya gelmesi ile oluşturulan bir yazılım için de geçerlidir.

Yukarıdaki **saatUygulaması** adlı sınıf örneğine dönecek olursak, tanımlamış olan her nesnenin yapısında sınıfın üyesi değişkenlerin bir kopyası bulunmaktadır. Bu kopyaların barındırdıkları değerler farklı olabilir. Nesne yönelimli programlama terminolojisinde bu durum her bir nesne için **örnek değişkenler – instance variables** olarak tanımlanır. Bundan dolayı da her bir örnek değişkenin kendisine özel olan bir değer barındırması da olağandır.

### 9.1.9. Erişim (Accessor) ve Değişim (Mutator) Fonksiyonları

Yukarıda tanımlanmış olan **saatUygulaması** adlı sınıfın üyelerine bir defa daha bakalım. Bu sınıfa üye olan fonksiyonlar arasında yer alan **zamanAyarla** fonksiyonu, kullanıcı tarafından verilen zaman bilgisini uygulamanın zaman bilgisi olarak değiştirmekte idi. Nesne yönelimi programlam paradigmasi ile söyleyecek olursak, üye fonksiyon sınıfına üye olan değişkenlere yeni değerleri atamaktadır. Aynı şekilde, **dakikaDegistir** ve **saniyeDegistir** fonksiyonları da üye değişkenler üzerinde işlem yapmaktadır. Diğer fonksiyonlar olan **zamanOku** ve **zamanYaz** ve **zamanKontrol** fonksiyonları ise sadece sınıfın üyesi olan değişkenlerine erişmektedir. Erişilen sınıfı üyesi değişkenler üzerinde herhangi bir işlem yapmamaktadırlar. Bu durumda sınıfın üyesi olan fonksiyonları değişkenler üzerinde işlem yapma durumlarına göre gruplayabiliriz. Birinci gruptaki üye fonksiyonlar sınıfın üyesi olan değişkenler üzerinde işlem yapabılırken, ikinci gruptaki üye fonksiyonlar ise değişkenlere erişmeye ama değişkeni okumak dışında herhangi bir işlem yapmamaktadırlar.

Bu graplama tür sınıflar için geçerlidir. Bir sınıfın üyesi olan ama sınıfın üyesi olan değişkenler üzerinde herhangi bir değişiklik yapmayan ama sadece değişkenin değerini okuyan fonksiyonlara **erişim fonksiyonları** adı verilir. Bunu tersi olarak da sınıfın üyesi olan değişkenler üzerinde her türlü bir değişiklik yapan fonksiyonlara **değiştirici fonksiyonlar** adı verilir.

Bir erişim fonksiyonu sadece sınıfın üyesi olan değişkenlere erişim sağlamakta ama bu sınıf üyesi olan değişkenler üzerinde herhangi bir işlem yapmamakta olduğu için, programcılar bir önlem olarak **const** özel sözcüğünü fonksiyonların tanımlanması sırasında kullanırlar. Böylece bir üye fonksiyon söz konusu üye değişkenlere eriştiğinde sadece değişkenin değerini okuyabilir ama değişkenin değerini değiştirecek bir eylem gerçekleştiremez.

Bir sınıfın yapısında tanımlanmış olan ve sonunda **const** özel sözcüğü bulunan tüm üye fonksiyonlara **sabit fonksiyon – constant function** adı verilir. Yukarıdaki örnekte yer verilen üye fonksiyonlar olan **zamanOku**, **zamanYaz** ve **zamanKontrol** fonksiyonları bu nedenle **saatUygulaması** sınıfının sabit fonksiyonlarıdır. Bir sınıfın üyesi olan sabit fonksiyonlar sınıfın üyesi olan değişkenlerin değerleri sadece okuyabilir ve üzerinde başka bir işlem yapamazlar, dolayısı ile de bu fonksiyonlar erişim fonksiyonlarıdır. Burada bir noktaya vurgu yapmakta yarar bulunmaktadır: Bir sınıfın üyesi olan bir sabit fonksiyon sadece ve sadece kendisi gibi sınıfın üyesi olan diğer sabit fonksiyonları çağrılabılır. Bu nedenle bir bir fonksiyonu sabit fonksiyon olarak tanımlanacak ise diğer fonksiyonlar ile olan ilişkisine öncelikle dikkat edilmesi zorunludur.

Aşağıdaki örnek programda **saatUygulaması** adlı sınıf kullanılmıştır. Programda sınıf, sınıfın üyesi olan nesneler ve sınıfın üyesi olan fonksiyonlar tanımlanmış ve ayrıca programın gereksinim duyduğu ana fonksiyon da yazılarak program tamamlanmıştır.

```
/*
 * zamanUygulamasi.cxx
 *
 *
 * Copyright 2023 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
 * ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
```

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
VERİ  
\* VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.  
\*  
\*/

```
#include <iostream>
#include <locale>

using namespace std;

class saatUygulaması
{
public:
    void zamanAyarla(int, int, int);
    void zamanOku(int&, int&, int&) const;
    void zamanYaz() const;
    void saniyeDegistir();
    void dakikaDegistir();
    void saatDegistir();
    bool zamanKontrol(const saatUygulaması&) const;

private:
    int sa;
    int dk;
    int sn;
};
```

```
int main()
{
    saatUygulamasi uygulamaSaati;
    saatUygulamasi kullaniciSaati;

    int saat;
    int dakika;
    int saniye;

    uygulamaSaati.zamanAyarla(5,4,30);
    cout << "uygulamaSaati: ";
    uygulamaSaati.zamanYaz();
    cout << endl;

    cout << "kullaniciSaati: ";
    kullaniciSaati.zamanYaz();
    cout << endl;

    kullaniciSaati.zamanAyarla(5,45,16);
    cout << "kullaniciSaati düzenlendikten sonra: ";
    kullaniciSaati.zamanYaz();
    cout << endl;

    if(uygulamaSaati.zamanKontrol(kullaniciSaati))
        cout << "Her iki zaman bilgisi aynıdır." << endl;

    else
        cout << "Her iki zaman bilgisi farklıdır." << endl;

    cout << "Zaman bilgisini saat, dakika ve saniye olarak aralarında boşluk bırakarak giriniz:";
}
```

```
cin >> saat >> dakika >> saniye;
cout << endl;

uygulamaSaati.zamanAyarla(saat, dakika, saniye);
cout << "uygulamaSaati için yeni zaman bilgisi: ";
uygulamaSaati.zamanYaz();
cout << endl;

uygulamaSaati.saniyeDegistir();

cout << "uygulamaSaati için Saniye bir arttırlıdı: ";
uygulamaSaati.zamanYaz(),
cout << endl;

uygulamaSaati.zamanOku(saat, dakika, saniye);

cout << "Saat: " << saat << " Dakika: " << dakika << " Saniye:" << saniye << endl;

return 0;

}

void saatUygulamasi::zamanAyarla(int saat, int dakika, int saniye)
{
    if (0 <= saat && saat < 24)
        sa = saat;
    else
        sa = 0;

    if (0<= dakika && dakika < 60)
        dk = dakika;
```

```
else
    dk = 0;

if (0 <= saniye && saniye < 60)
    sn = saniye;
else
    sn = 0;
}

void saatUygulamasi::zamanOku(int& saat, int& dakika, int& saniye) const
{
    saat = sa;
    dakika = dk;
    saniye = sn;
}

void saatUygulamasi::zamanYaz() const
{
    if (sa < 10)
        cout << "0";

    cout << sa << ":";

    if (dk < 10)
        cout << "0";

    cout << dk << ":";

    if (sn < 10)
        cout << "0";
```

```
cout << sn;
}

void saatUygulamasi::saatDegistir()
{
    sa++;

    if (sa > 23)
        sa = 0;
}

void saatUygulamasi::dakikaDegistir()
{
    dk++;

    if (dk > 59)
    {
        dk = 0;
        saatDegistir();
    }
}

void saatUygulamasi::saniyeDegistir()
{
    sn++;

    if (sn > 59)
    {
        sn = 0;
        dakikaDegistir();
    }
}
```

```
}
```

```
bool saatUygulamasi::zamanKontrol(const saatUygulamasi& digerZaman) const
{
    return (sa == digerZaman.sa && dk == digerZaman.dk && sn == digerZaman.sn);
}
```

Yukarıda gösterilen kaynak kod derlenip çalıştırıldıkten sonra, aşağıdakine benzer bir çıktı elde edilecektir.

```
[goksin@tardis ~/projeler/C++_Notlar/2022-2023/Bahar/09]$ ./zamanuygulamasi
uygulamaSaati: 05:04:30
kullaniciSaati: 545974232:08:00
kullaniciSaati düzelendlendikten sonra: 05:45:16
Her iki zaman bilgisi farklıdır.
Zaman bilgisini saat, dakika ve saniye olarak aralarında boşluk bırakarak
giriniz: 8 34 12

uygulamaSaati için yeni zaman bilgisi: 08:34:12
uygulamaSaati için Saniye bir arttırıldı: 08:34:13
Saat: 8 Dakika: 34 Saniye:13
[goksin@tardis ~/projeler/C++_Notlar/2022-2023/Bahar/09]$
```

Program içerisinde ilgili nesnelerin değerlerinin nasıl düzenlendiğinin daha kolay görülebilmesi için nesne isimleri aynen kullanılmıştır. `kullaniciSaati` nesnesi ilk olarak çağrıldığında kendisi için ayrılmış olan bellek alanını okumaktadır. Dolayısı ile farklı bilgisayarlarda aynı kod derlenip çalıştırıldığında farklı sonuçlar döndürecektir. Burada bir hata bulunmamaktadır.

### 9.1.10.Bir Sınıfta Özel ve Genel Üyelerin Sıralanması

C++ Programlama Dili yapısında genel, özel ve korumalı sınıf üyeleri için herhangi bir hiyerarşî sunmamaktadır. Dolayısı ile programcı söz konusu sınıf üyelerinin istediği sırada yazabilmesi olanaklıdır. Tek unutulmaması ve özellikle dikkat edilmesi gereken nokta bir üye tanımı yapıldığında aksi belirtilmekçe ön tanımlı olarak özel/private olarak tanımlanmakta olduğudur. Bu nedenle bir nesne yönelimli programmanın doğası gereği sınıf üyelerinin arasındaki ilişkilerin tanımlanması bir zorunluluktur. Bu nedenle programın kaynak kodu yazılarken söz konusu üyelerin durumlarının; özel, genel ve korumalı olmak üzere tanımlanması gereklidir. Ayrıca yazılan kodun okunabilir ve anlaşılabilir olması için bu tanımlamalar önemlidir. Aşağıda yukarıda yer verilen programda gösterilen genel ve özel sınıf üyesi tanımlamalarının farklı şekilde tanımlama örnekleri görülmektedir. Her üç örnekte geçerlidir.

Sınıfa ait genel üyelerin ve ardından özel üyelerin tanımlanması:

```
class saatUygulamasi
{
public:
```

```
void zamanAyarla(int, int, int);
void zamanOku(int&, int&, int&) const;
void zamanYaz() const;
void saniyeDegistir();
void dakikaDegistir();
void saatDegistir();
bool zamanKontrol(const saatUygulamasi&) const;

private:
    int sa;
    int dk;
    int sn;
};
```

Sınıfa ait özel üyelerin ve ardından genel üyelerin tanımlanması:

```
class saatUygulamasi
{
    private:
        int sa;
        int dk;
        int sn;

    public:
        void zamanAyarla(int, int, int);
        void zamanOku(int&, int&, int&) const;
        void zamanYaz() const;
        void saniyeDegistir();
        void dakikaDegistir();
        void saatDegistir();
        bool zamanKontrol(const saatUygulamasi&) const;
};
```

Sınıfa ait özel üyelerin özel olarak tanımlanmadan belirtilmesi ve ardından genel üyelerin tanımlanması:

```
class saatUygulaması
{
    int sa;
    int dk;
    int sn;

    public:
        void zamanAyarla(int, int, int);
        void zamanOku(int&, int&, int&) const;
        void zamanYaz() const;
        void saniyeDegistir();
        void dakikaDegistir();
        void saatDegistir();
        bool zamanKontrol(const saatUygulaması&) const;
};
```

Genel olarak pratik bir kural olarak üyelerin özel, genel ve korumalı olmasının tanımlanması zorunludur. Ayrıca sırası ile genel, özel ve korumalı olarak sınıf üyelerinin tanımlanması yazılı olmayan bir kuraldır.

### 9.1.11. Yapıçı Fonksiyonlar

Yukarıdaki örnek programda **kullaniciZamani** nesnesi program içerisinde çağrıldığında döndürdüğü değerler herhangi bir anlam ifade etmiyordu. Bunu nedeni söz konusu nesnenin tanımlanması sırasında herhangi bir ön tanımlı değer atanmamış olması idi. C++ Programlama Dili’nde standarda göre bir nesne tanımlandığında etkinleştirilmez. Bunun nedeni sınıfın özel olarak tanımlanmış olan nesneleri sınıf dışından erişilemez. Yukarıdaki program ise bunlar sınıfın özel olarak tanımlanmış olan değişkenleri idi. Dolayısı ile de programcı bu nesneler etkinleştirilmedi ise, program çalıştırıldığında söz konusu nesneler için ayrılmış bellek alanında bulunan veri döndürülür. Bir çok durumda bu veri önceki bir programdan geriye kalmış olan her türlü veri ve ikili kod olabilir.

Bu durumun neden olabileceği hataların önüne geçilmesi için sınıfın üyelerinin değerlerinin tanımlanarak etkinleştirilmesi gereklidir. Bunun için de **yapıcı fonksiyonlar – constructor functions** kullanılır. Eğer bir yapıçı fonksiyon içerisinde herhangi bir parametre tanımlanmış ise buna **ön tanımlı yapıçı fonksiyon – default constructor** adı verilir.

Yapıcı fonksiyonlarının özellikleri şunlardır:

- Yapıçı fonksiyonun adı ile sınıfın adı aynıdır.
- Yapıçı fonksiyonun herhangi bir tipi söz konusu değildir. Yani ne değer döndüren ne de değer döndürmeyen bir fonksiyondur.
- Bir sınıfın birden çok yapıçı fonksiyonu olabilir. Ancak bu yapıçı fonksiyonların tümün ismi sınıfın ismi ile aynıdır.
- Bir sınıfın birden çok sayıda yapıçı fonksiyonu bulunuyor ise, her bir yapıçı fonksiyonun formal parametreleri farklı olmalıdır. Bu ya formal parametre sayısının farklı olması veya formal parametrelerin veri tipinin farklı tanımlanması ile sağlanır. Bir fonksiyon aşırı yüklenirse, yapıçı fonksiyonun adı aşırı yüklenmiş olur.
- Yapıçı fonksiyonlar, bir nesne yaratıldığında otomatik olarak işler ve nesnenin kapsamındaki tüm üyeleri etkinleştirilir. Yapıçı fonksiyonu tanımlayan bir veri tipi olmadığından çağrılmaması söz konusu değildir.
- Bir sınıfın nesnesi tanımlandığında, hangi değerlerin bu nesneye atanacağı, nesneye aktarılacak değerlerin özelliklerine bağlı olarak değişir.

Yukarıda ye verdigimi programdaki sınıfa iki adet yapıçı fonksiyon ekleyelim.

```
class saatUygulamasi
{
public:
    void zamanAyarla(int, int, int);
    void zamanOku(int&, int&, int&) const;
    void zamanYaz() const;
    void saniyeDegistir();
    void dakikaDegistir();
    void saatDegistir();
    bool zamanKontrol(const saatUygulamasi&) const;
    // Parametreleri olan yapıçı fonksiyon
    saatUygulamasi(int, int, int);
    // Parametreleri olmayan ön tanımlı yapıçı fonksiyon
    saatUygulamasi();

private:
    int sa;
    int dk;
```

```
    int sn;  
};
```

Burada tanımlanmış olan yapıçı fonksiyonların bu şekilde tanımlanmış olması ile programcının işi bitmiş olmamaktadır. Yapıçı fonksiyonların da diğer fonksiyonlar gibi tanımlandıktan sonra yazılması gereklidir.

```
saatUygulamasi::saatUygulamasi(int saat, int dakika, int saniye)
```

```
{  
    if (0 <= saat && saat < 24)  
        sa = saat;
```

```
    else  
        sa = 0;
```

```
    if (0<= dakika && dakika < 60)  
        dk = dakika;  
    else  
        dk = 0;
```

```
    if (0 <= saniye && saniye < 60)  
        sn = saniye;  
    else  
        sn = 0;
```

```
}
```

```
saatUygulamasi::saatUygulamasi()  
{  
    sa=0;  
    dk = 0;  
    sn = 0;  
}
```

Yukarıda yer verilen yapıçı fonksiyonlarının incelenmesinden de görüleceği üzere, ilk yapıçı fonksiyonda tanımlanmış olan parametrelerin değerleri aynen kullanılarak atama yapılmaktadır. Ön tanımlı yapıçı fonksiyon ise basitçe her üç özel değişkenin değerini sıfır olarak atamaktadır. Öte

yandan yapıcı fonksiyonun bir diğer fonksiyonu çağrıp bunun aracılığı ile üye değişkenlerin değerini belirleyebilmekte olanaklıdır. Aşağıdaki örnekte, **zamanAyarla** fonksiyonu çağrılarak yapıcı fonksiyon görevini yerine getirmektedir.

```
saatUygulamasi::saatUygulamasi(int saat, int dakika, int saniye)
{
    zamanAyarla(saat, dakika, saniye);
}
```

### 9.1.12. Yapıcı Fonksiyonların Çağrılması

Bir sınıfta yapıcı fonksiyon tanımlandığında her tanımlanmış olan nesne, tanımlanmış olan ön tanımlı yapıcı fonksiyon tarafından değer atanarak etkinleştirilir. Eğer sınıfın yapısında birden çok sayıda yapıcı fonksiyon tanımlanmış ise hangisinin çağrıacağı önem kazanmaktadır. SOnraki bölüm bu konuyu açıklamaktadır.

### 9.1.13. Ön Tanımlı Yapıcı Fonksiyonların Çağrılması

Bir sınıfta ön tanımlı yapıcı fonksiyonun bulunduğu var sayarak programım içerisinde bu fonksiyonun çağrıldığını düşünelim. Bu işlem aşağıdaki gibi yapılacaktır.

```
sınıfAdı sınıfınNesnesininAdı;
```

Yukarıda yazdığını programı kullanarak bunu yapalım.

```
saatUygulaması kullaniciSati;
```

Bu bildirim **saatUygulaması** adlı sınıfın üyesi olan **kullaniciSati** adlı nesnenin önem tanımlı değerlerini atamaktadır. Burada yapıcı fonksiyonun herhangi bir formal parametresi olmadığı dikkate alınacak olursa, söz konusu olan nesnenin üyesi olan değişkenlerin değerinin tümünün sıfır olacağı görülür. Burada dikkat edilmesi gereken eğer tüm ön tanımlı değerlerin değeri sıfır olacak ise yapıcı fonksiyonun tanımlanırken sonuna “()” eklenmesine gerek olmadığıdır. Eğer “()” kullanılacak olursa bazı derleyiciler bu bildirimin bulunduğu satır için hata mesajı döndürecektil.

### 9.1.14. Yapıcı Fonksiyonların Parametre İle Çağrılması

Bir sınıfta yapıcı fonksiyonların parametreler ile birlikte tanımlandığını var sayarak programım içerisinde yapıcı fonksiyonun çağrıldığını düşünelim. Bu işlem aşağıdaki gibi yapılacaktır.

```
sınıfAdı sınıfınNesnesininAdı(parametre0, parametre1, parametre2, ... );
```

Burada belirtilen parametreler, değişken veya ifade olabilir. Bu nedenle bir yapıcı fonksiyon parametreleri ile birlikte çağrılacak ise şu noktalara dikkat edilmelidir.

- Parametrelerin sayısı ve veri tipi tanımlamaları, fonksiyonun formal parametreleri ile aynı olmalıdır.
- Eğer formal parametreler ile fonksiyonun parametreleri eşleşmiyor ise, bu durumda C++ Programlama tip dönüşümleri yaparak eşleşen parametreleri bulmaya çalışacaktır. Örneğin

bir tam sayı parametre kayar noktalı sayı parametresine dönüştürülebilir. Bu tür hatalarda derleyici tarafından hata döndürülür.

Daha önce açıklandığı üzere aşağıda gösterilen bildirimde **uygulamaSaati** nesnesine sırası ile tam sayı veri tipinde olan üç adet değer aktarılmıştır. Bu parametreler ile formal parametreler eşletiğinden herhangi bir hata olmadan işlem gerçekleşecektir.

saatUygulaması uygulamaSaati (1, 2, 3);

Aşağıda yer verilen örnek kodda, **sepet** adlı sınıf tanımlanmıştır. Sınıfın biri genel diğeri de özel olmak üzere iki ayrı grupta tanımlanmış olan değişkenleri ve nesneleri bulunmaktadır.

```
class sepet
{
public:
    // Yapıçı fonksiyonlar aşağıda tanımlanmıştır.
    urunler();
    urunler(string);
    urunler(string, int, double);
    urunler(string, int, double, int);
    // Sınıfın diğer fonksiyonları aşağıda olacaktır.

private:
    string urunAdi;
    int urunKayitNo;
    double urunFiyat;
    int urunAdet;
};
```

Sınıfın yapısında dört adet yapıçı fonksiyon ve dört adet değişken bulunmaktadır. Yapıçı fonksiyonlar genel bölümünde tanımlanmıştır. Burada yapıçı fonksiyonlarının tanımları aşağıda gösterildiği gibidir.

```
sepet ::urunler()      // ön tanımlı yapıçı fonksiyon
{
    urunAdi = "";
    urunKayitNo = 0;
    urunFiyat = 0.0;
    urunAdet = 0;
```

```
};

sepet ::urunler(string n)
{
    urunAdi = "n";
    urunKayitNo = 0;
    urunFiyat = 0.0;
    urunAdet = 0;

};

sepet ::urunler(string, int adet, double fiyat)
{
    urunAdi = "n";
    urunKayitNo = adet;
    urunFiyat = fiyat;
    urunAdet = 0;

};

sepet ::urunler(string, int adet, double fiyat, int sayi)
{
    urunAdi = "n";
    urunKayitNo = adet;
    urunFiyat = fiyat;
    urunAdet = sayi;

};
```

Bu tanımlamanın ardından, program içerisinde aşağıdaki tanımlamaların yapılmış olduğunu var sayalım.

```
sepet urun1;
```

```

sepet urun2 ("Tava");
sepet urun3 ("Mum", 10, 8.50);
sepet urun4 ("Tuz", 304, 5.50, 2000);

```

Bu tanımlamalara göre ilk olarak urun1 için ön tanımlı yapıçı fonksiyon çağrılmak ve işlem yapacaktır. Çünkü bu bildirimde herhangi bir parametre tanımlanmamış veya kullanılmamıştır. İkinci tanımlamada tek bir parametre kullanılmıştır. Bu parametre, karakter katarı olarak verildiği için ikinci parametreye atanmıştır. Üçüncü bildirimde üç adet parametre kullanıldığı için ilgili değişkenlerin değerleri atanmıştır. Son bildirimde tüm parametreler tanımlandığı için karşılık gelen tüm değişkenlere değer ataması yapılmıştır.

<b>urun1</b>	<b>urun Adı</b>	
	<b>urunKayitNo</b>	<b>0</b>
	<b>urunFiyat</b>	<b>0.0</b>
	<b>urunAdet</b>	<b>0</b>

<b>urun2</b>	<b>urun Adı</b>	<b>Tava</b>
	<b>urunKayitNo</b>	<b>0</b>
	<b>urunFiyat</b>	<b>0.0</b>
	<b>urunAdet</b>	<b>0</b>

<b>urun3</b>	<b>urun Adı</b>	<b>Mum</b>
	<b>urunKayitNo</b>	<b>10</b>
	<b>urunFiyat</b>	<b>8.50</b>
	<b>urunAdet</b>	

<b>urun4</b>	<b>urun Adı</b>	<b>Tuz</b>
	<b>urunKayitNo</b>	<b>304</b>
	<b>urunFiyat</b>	<b>5.50</b>
	<b>urunAdet</b>	<b>2000</b>

### 9.1.15.Yapıcı Fonksiyonlar ve Ön Tanımlı Parametreler

Yapıcı fonksiyonların da diğer fonksiyonlarda olduğu gibi ön tanımlı parametreleri olabilir. Bir fonksiyonun parametreleri için ön tanımlı değerler nasıl tanımlanıyor ise, aynı şekilde bir yapıçı fonksiyonun parametreleri için ön tanımlı değerler aynı şekilde tanımlanabilir. Önceki örnekte kullandığımız ön tanımlı yapıçı fonksiyon ile parametrelere sahip olan yapıçı fonksiyon aşağıdaki gibi yazılabilir.

```

class saatUygulaması
{
    public:
        void zamanAyarla(int, int, int);
        void zamanOku(int&, int&, int&) const;
        void zamanYaz() const;
        void saniyeDegistir();
        void dakikaDegistir();
        void saatDegistir();
        bool zamanKontrol(const saatUygulaması&) const;

        // Parametreler için ön tanımlı değerleri olan yapıçı fonksiyon
        saatUygulaması(int = 0, int = 0, int = 0);

```

```
private:  
    int sa;  
    int dk;  
    int sn;  
};
```

Yukarıdaki kod bloku içerisinde, yapıcı fonksiyon bildiriminde, fonksiyonun üç adet parametresi tanımlanmıştır. Bu parametreler tam sayı veri tipindedir. Parametrelerde ön anımlı değer olarak sıfır atanmıştır. Eğer yukarıda yazdığımız programda ön tanımlı değerler yerine nesnelere başka değerler atayacak olursak, ön tanımlı değerler yerine tanımlanan değerler atanacaktır.

Burada verilen örneklerden yola çıkarak bir yapıcı fonksiyonun herhangi bir parametre tanımlanmadan veya tüm değişkenlere ait ön tanımlı değerlerin parametre olarak tanımlandığında her iki fonksiyonunda **ön tanımlı yapıcı fonksiyon – default constructor** olarak tanımlanabileceği görülmektedir.

### 9.1.16.Yapıcı Fonksiyonlar ve Sınıflar İçin Uyarılar

Sınıfların, nesnelerin ve diğer üyelerin tanımlanması durumunda eğer bir yapıcı fonksiyon tanımlanmamış ise C++ programlama Dili standartı gereğince bir ön tanımlı olarak yapıcı fonksiyon otomatik olarak oluşturulur.. Ancak bu fonksiyonun değişkenlere değer atayarak etkinleştirmesi söz konusu değildir. Burada dikkate edilmesi gereken eğer programcı bir ön tanımlı yapıcı fonksiyon tanımlarsa ise, bu durumda otomatik olarak ön tanımlı yapıcı fonksiyon tanımlanmayacaktır. Programının tanımladığı ön tanımlı yapıcı fonksiyonun parametreleri de tanımlanmak durumundadır. Bu parametrelerin de alacağı değerler ayrıca tanımlanmak durumundadır. Aksi durumda nesneler ve değişkenler tanımlanmış ancak etkinleştirilmemiş olacaktır.

Aşağıdaki örnek kod blokunda bu durum gösterilmiştir.

```
class uydurukSinif  
{  
public:  
    void yazdir() const;  
  
    // Yapıcı fonksiyon  
    uydurukSinif ( int usX, int usY );  
  
private:  
    int x  
    int y;  
};
```

Kaynak kodun incelenmesinden de görüleceği gibi **uydurukSinif** adlı sınıfın yapısında bir tane yapıcı fonksiyon bulunmaktadır. Bu fonksiyonun iki adet parametresi bulunmaktadır. Ancak parametrelerin herhangi bir ön tanımlı değeri bulunmamaktadır. Önceki bölümlerde gördüğümüz gibi bu programın derlenip çalıştırılması durumunda sınıfın yapısında tanımlanan nesneler oluşturulacak ancak etkinleştirilmeyecektir. Bunun nedeni yapıcı fonksiyonun programcı tarafından

tanımlanmış ancak herhangi bir ön tanımlı değerinin ise olmamasıdır. Eğer aşağıdaki bildirim program içerisinde kullanılarak olursa nesne etkinleştirilmiş olacaktır.

```
uydurukSinif uydurukNesne(0, 2); //Geçerli ve doğrudur.
```

Burada dikkat edilmesi gereken nokta ise, uydurukSinif adlı sınıfın ön tanımlı yapıcı fonksiyonun herhangi bir ön tanımlı değeri olmadığıdır. Bu durumda aşağıdaki bildirim hatalı ve geçersizdir.

```
uydurukSinif uydurukNesne; Gecersiz ve hatalıdır.
```

Bu tür hataların önüne geçilebilmesi için bir sınıfın tanımlanması durumunda iki tane yapıcı fonksiyon tanımlaması uygun olacaktır. Birinci ön tanımlı yapıcı fonksiyonun herhangi parametre olmaksızın “()” ile sonlandırılarak tanımlanırken, programının yazacağı ikinci ön tanımlı yapıcı fonksiyon parametreleri ve bu parametrelerin ön tanımlı değerleri ile tanımlanması gereklidir.

### 9.1.17. Sınıf Üyelerin Etkinleştirilmesi ve Ön Tanımlı Yapıcı Fonksiyonlar

C++ Programlam Dili'nde standart gereği, bir sınıfın üyeleri tanımlandığı zaman etkinleştirilebilir. Önceki bölümde yazdığımız programa donecek olursak, aşağıda görüldüğü gibi aynı programı yazabiliriz.

```
class saatUygulaması
{
public:
    void zamanAyarla(int, int, int);
    void zamanOku(int&, int&, int&) const;
    void zamanYaz() const;
    void saniyeDegistir();
    void dakikaDegistir();
    void saatDegistir();
    bool zamanKontrol(const saatUygulaması&) const;

    // Ön tanımlı yağıçı fonksiyon
    saatUygulaması();
    // Parametreler için ön tanımlı değerleri olan yapıcı fonksiyon
    saatUygulaması(int, int, int);

private:
    int sa = 0;
    int dk = 0;
    int sn = 0;
};
```

Bu şekilde yazılan bir programda sınıfın üyesi olan ve değişkenlerin ön tanımlı değerleri belirtilmiş ve değişkenler sınıfın tanımlanması sırasında etkinleştirilmiştir. Bu şekilde yapılan etkinleştirmeye **verilerin sınıfı içi etkinleştirilmesi – in-class initialization of data** adı verilir. Buna

bir sınıfın üyesi olan bir nesne sınıf içerisinde etkinleştirildiğinde tanımlı olan ön tanımlı değerler ile etkinleştirilir.

Buna göre aşağıdaki bildirimini inceleyelim.

saatUygulaması uygulamaSaati;

Bu bildirim işlendiğinde ilk olarak ilgili değişkenler için bellekte yer ayrılacaktır. (a) Ardında bu değişkenlerin değerleri atanır. (b)

uygulamaSaati	sa	
	dk	
	sn	

(a)

uygulamaSaati	sa	0
	dk	0
	sn	0

(b)

Eğer ön tanımlı yapıcı fonksiyonlarda etkinleştirmek için kullanılacak olan değerler bulunuysa bu durumda, yukarıda olduğu gibi nesnelere atanacak olan ön tanımlı değerleri yerine fonksiyonun belirttiği değerler atanır. Aşağıdaki bildirim işlenmesi ile belirtilen durum gerçekleşektir.

saatUygulaması uygulamaSaati( 2, 2, 2 );

Nesnenin durumu aşağıdaki gibi olacaktır:

uygulamaSaati	sa	
	dk	
	sn	

(a)

uygulamaSaati	sa	2
	dk	2
	sn	2

(b)

Yukarıda görülen uygulama sonucunda ön tanımlı yapıcı fonksiyonlara gereksinim olmadığı düşünülebilir. Ancak bu tanımlanın yetersiz kalacağı durum, ön tanımlı değerlere sahip olan nesnelerin tanımlandığı durumlar olacaktır. Eğer aynı sınıfın üyesi olan yeni bir nesnenin tanımlanması gerekiyor ise ve bu nesnenin ön tanımlı değerleri yukarıda gösterildiği gibi aynı sınıfın tanımlı değişkenlerin değerini alacaktır. Bu işlemler için sınıfın ön tanımlı yapıcı fonksiyonunun sınıfın tanımı ile birlikte kullanılması yeterlidir. Aşağıda bu tanımlama şekli görülmektedir.

saatUygulaması uygulamaSaati = saatUygulaması (1, 2, 3);

Bu tanımlama biçimi ön tanımlı yapıcı fonksiyonun tüm sınıfı için tanımlanmış halidir. Sonunda yer alan “{}” ise on tanımlı yapıcı fonksiyonun gövdesini belirtir.

### 9.1.18.Sınıf Üyesi Olarak Diziler ve Yapıcı Fonksiyonlar

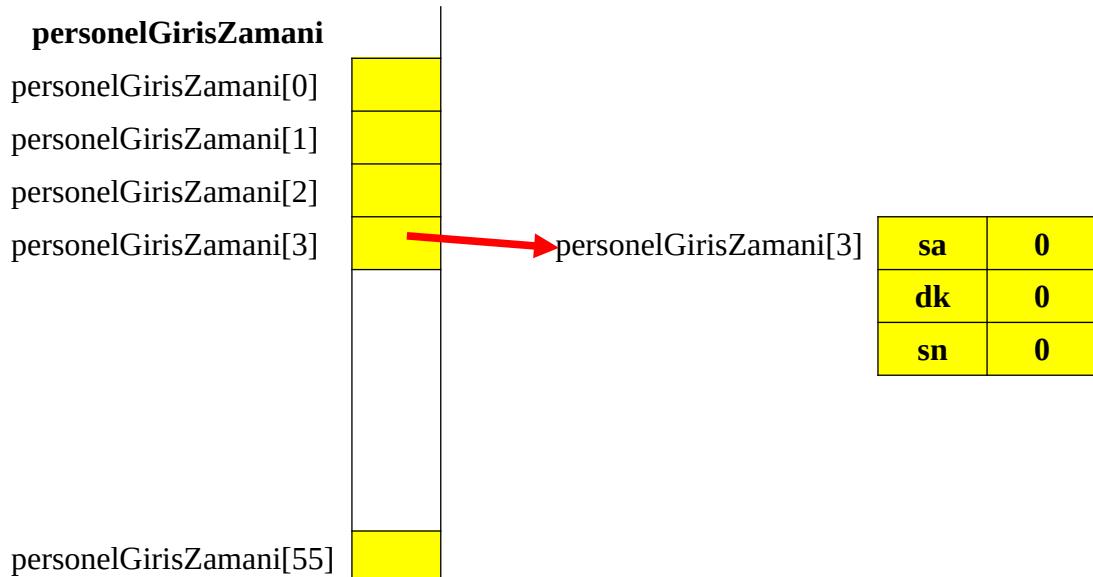
Bir sınıfın yapıcı fonksiyonlara sahip olması gereklidir. Bu da bir sınıfın yapısında tanımlanmış olan nesneleri için de yapıcı fonksiyonların tanımlanması gerektiği anlamına gelir. Ön tanımlı yapıcı fonksiyonun bütün nesnelerin ön tanımlı değerlerini tanımlamak için kullanılır.

Örneğin bir sınıfın yapısında bulunun örnek 200 adet nesnenin her birisi için ön tanımlı yapıcı fonksiyonun tanımlanması pratik olarak olanaksız olmamakla birlikte uygulanabilirlik açısından zor olacağı kolaylıkla anlaşılabilir.

Önceki bölümlerde yazdığımız sınıf uygulaması programını kullanabileceğimiz bir örnek üzerinde duralım. Bir iş yerinde çalışan personelin iş yerine giriş ve çıkış zamanlarını kayıt etmek için kullanılmış olsun. Bu iş yerinde çalışan personel sayısının 56 kişi olduğunu kabul edelim. Bu durumda bütün personelin giriş ve çıkış zamanlarını kayıt altına almak için iki adet dizi tanımlayarak kullanabiliriz. Personelin geliş zamanı bilgisini tutan dizi **personelGirisZamani** ve personelin ayrılma zaman bilgisini tutan dizi **personelCikisZamani** olsun. Bu durumda her dizin elamanı teknik olarak **saatUygulaması** sınıfının bir nesnesi olacaktır.

saatUygulaması uygulamaSaati = saatUygulaması (1, 2, 3);

Bu bildirim ile 56 adet nesne tanımlanacaktır. Her bir nesnenin tanımlandığı andaki durumu da aşağıdaki şekilde gösterilmiştir.



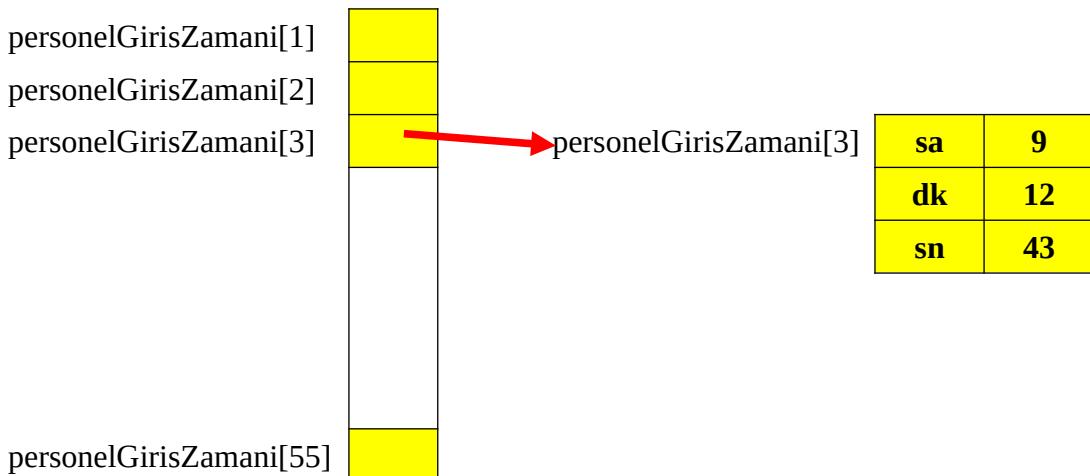
Bu bildirimin işlenmesinin ardından **saatUygulaması** sınıfını kullanarak bu nesneler üzerinde işlem yapabiliriz. Örneğin **personelGirisZamani[3]** nesnesinin değerinin program tarafından okunan zaman bilgisi ile düzenlendiğini var sayalım. Bu işlemin aşağıdaki bildirimdeki gibi olduğunu var sayalım.

saatUygulaması uygulamaSaati = saatUygulaması (1, 2, 3);

Bu bildirim ile söz konusu olan dizi elmanın değerleri yukarıdaki değerleri olacaktır. Bu işlemin tüm personel için yapılabilmesi için bir döngü kurgulanabilir. Döngü yapısında **zamanUygulaması** sınıfının nesnelerini kullanacaktır. Aşağıdaki kod bloku bu döngüyü göstermektedir.

```
for (int i = 0; i < 55, i++)
{
    cout << "Personel " << (j+1) << giriş zamanı: " // alt satırda devam ediyor
    << personelGirisZamani[j].zamanYaz(); // alt satırda devam ediyor
    cout << endl;
}
```





Yukarıdaki döngü personelin giriş zamanını önceki örneklerde görüldüğü gibi saat:dakika:saniye biçiminde yazacaktır. Benzer şekilde personelin iş yerinden ayrıldığı zamanı kayıt etmek için de aynı döngü bu kez **personelCikisZamani** dizisi ile kullanılabilir.

Yukarıdaki örneklerden de anlaşılacağı üzere, diziler aynı zamanda bir sınıfın yapısında bulunan nesneleri yönetmek için de kullanılabilir.

Bölümün başlangıcında bir sınıfın yapısında tanımlana bir dizinin elemanlarının da sınıfın nesneleri olarak kullanılabilir. Ancak bir sınıfın yapısında nesneler tanımlandığı anda bu sınıf için ön tanımlı yapıçı fonksiyonun tanımlanması gereklidir. Bir sınıfta yapıçı fonksiyon tanımlandığında farklı nesneler için de yapıçı fonksiyonların tanımlanması gereklidir. Eğer sınıfın yapısındaki nesneler farklı özelliklere sahip ise bu durumda her bir tekil nesne için yapıçı fonksiyonların tanımlanması gerekecektir. Eğer çok sayıda elemanı olan diziler kullanılıyor ise bu dizinin her bir tekil elemanı için yapıçı fonksiyonun tanımlanması pratik olmaktan uzaktır.

Aşağıdaki örnek bildirimde, iki adet nesne için tanımlanmış olan bir dizi ile söz konusu nesnelerin ön tanımlı değerleri belirtilmiştir.

```
saatUygulaması saatler [2] = {saatUygulaması(8,0,0), saatUygulaması(9,0,0)};
```

Bu bildirim ile iki adet anonim nesne yaratılmıştır. İlk olarak ilk üye değişkenin değeri (8, 0, 0) olarak tanımlanmıştır. Ardından da aynı şekilde ikinci üye değişkenin değerleri de atanmıştır.

Aşağıdaki bildirimde daha önce yer verdiğimiz bir bildirim yer almaktadır. Burada bir nesne yaratılmış ve ön tanımlı değerleri atanmıştır.

```
saatUygulaması uygulamaSaati (1, 2, 3);
```

Bu bildirim teknik olarak aşağıdaki bildirim ile aynıdır.

```
saatUygulaması uygulamaSaati = saatUygulaması (1, 2, 3);
```

Yukarda belirtilen birinci ile ikinci bildirimin karşılaşılması durumunda görüleceği üzere ikinci bildirim önce bir nesnenin yaratılmasını ve ardından bu nesneye ön tanımlı değerlerin atanmasını gerektirmemektedir. Bu nedenle ilk bildirim diğerine göre daha etkin ve verimlidir.

Buradaki açıklamaların birbiri ile çelişen veya bir diğerine göre daha doğru olan bir yönü bulunmamaktadır. Burada özellikle vurgulanmak istenen programlama konusunda gözden kaçan kaça ve

programcının kendisinden kaynaklı hataların neden olabileceği her türlü sorunun önüne geçilmesi amacı ile pratik bir kural olarak bir sınıfın yapısında tanımlanan nesneler için bir yapıcı fonksiyonun tanımlanmasının zorunlu olduğunu söylüyor. Diğer bir deyişle, bir sınıfın yapısında bir yapıcı fonksiyon bulunuyorsa, mutlaka ön tanımlı yapıcısı fonksiyonun bulunması zorunludur.

### 9.1.19. Yıkıcı Fonksiyonlar

Bir sınıf için yapıcısı fonksiyonlarının olması gibi yıkıcı fonksiyonlar da söz konusudur. Yıkıcı fonksiyon, bir sınıfın işlevi sona erdiğinde ortadan kaldırılması için kullanılır. Yukarıdaki örnekte tanımlanan sınıf için yazılacak olan yıkıcı fonksiyon aşağıdaki gibi yazılır. Basitçe söyleyecek olursak sınıfın adının önüne yazılan “~” işaretini ile yıkıcı fonksiyon yazılır.

~saatUygulaması;

Yıkıcı fonksiyonun, adından da anlaşılacağı üzere değer döndürmez veya değer döndürmeyen bir fonksiyon da değildir. Herhangi bir ön tanımı değeri söz konusu değildir. Yıkıcı fonksiyonlar ilerleyen bölümlerde tekrar ele alınacaktır.

## 9.2. Veri Soyutlama, Soyut Veri Tipi ve Sınıflar

Günlük yaşamımızda, otomobili kullanmak için ehliyet sahibi olmak yeterlidir. Otomobili kullanabilmek için otomobilin nasıl çalıştığını tüm ayrıntıların bilinesini gerektirmez. Şöför için sadece direksiyonu, vitesi ve pedalları nasıl kullanması gerektiğini, trafik kurallarını, trafik işaretlerini ve de trafik işaretçilerinin hareketlerinin anlamını bilmesi yeterlidir. Otomobilin motorunun tüm bileşenlerinin nasıl bir araya geldiğini, nasıl bir etkileşim ile çalıştığını, vites kutusunun iç yapısını ve işleyişini, hatta direksiyonun otomobilin yönünü nasıl değiştirdiğini bilmesine gerek yoktur. Sadece yukarıda belirtilen bileşenleri kavramış ve doğru şekilde kullanıyor olması yeterlidir. Buna soyutlama adı verilir. Soyutlamalar ile günlük yaşamımızda kullanmakta olduğumu bir çok araç ve gereçin iç yapısını ve bileşenleri arasındaki etkileşimi bilmek zorunda kalmadan kullanabilmekteyiz.

Soyutlama aynı zamanda veri içinde kullanılabilir. Önceki bölümde saatUygulaması adlı bir sınıf tanımlanmıştır. Bu sınıfın da üç adet değişken üyesi bulunmaktadır. Bu üyeler üzerinde şu işlemler yapılabiliyor:

- Zamanın bilgisinin düzenlenmesi
- Zaman bilgisinin yazdırılması
- Saniye değerinin değiştirilmesi
- Dakika değerinin değiştirilmesi
- Saat değerinin değiştirilmesi
- İki farklı kaynağı ait zaman bilgilerinin karşılaştırılması

Bu işlevlerin nasıl uygulanacağı ise sonraki bölümlerde ele alınmıştır.

Veri soyutlama, verinin uygulama biçiminden üzerinde yapılacak olan işlemlerin ayrılması ile gerçekleşir. saatUygulaması sınıfının tanımlaması ile bu sınıf yapısında gerçekleştirilecek olan

işlemler, verinin **mantıksal özelliklerine** karşılık gelmektedir. Sınıfın üyeleri aracılığı ile veri üzerinde gerçekleştirilecek olan işlemler ile bu işlemlerin algoritmaları ve verinin bilgisayarda nasıl barındırıldığı ise **uygulamaya** karşılık gelmektedir.

Bir veri üzerinde yapılacak olan işlemleri tanımlayan mantıksal özellikler ile bu özellikler üzerinde gerçekleştirilecek olan işlemlerin uygulamasının birbirinden ayrılması **soyut veri tipi – abstract data type** olarak tanımlanır.

Bir soyut veri tipinin, daha önceden gördüğümüz tam sayı verit tipinde olduğu gibi, üç adet özelliği söz konusudur. Bunlar soyut veri tipinin adı yani **veri tipi adı – type name**, soyut veri tipine ait olan **değerlerin kümesi – domain**, bu değerler üzerinde gerçekleştirilecek olan **işlemlerdir – operations**. Tam sayı veri tipinin alabileceği değerleri -2147483648 ile 2147483647 arasındadır ve bu veri üzerinde gerçekleştirilebilecek olan işlemler, %, \*, /, + ve – olarak tanımlıdır. Buradan hareketler saatUygulaması adlı soyut veri tipinin aşağıdaki gibi tanımlanabilmesi olanaklıdır:

### **Veri Tipi Adı**

saatUygulaması

### **Değerler Kümesi**

Her sınıf üyesi zaman bilgisini saat, dakika ve saniye olarak barındırır.

### **İşlemler**

Zaman bilgisini düzenler

Zaman bilgisini okur

Zaman bilgisini döndürür

Saniye değerini değiştirir

Dakika değerini değiştirir

Saat değerini değiştirir

İki arkı zaman bilgisini karşılaştırır

Benzer olarak yaygın bir şekilde kullanılan liste veri yapısı da aşağıdaki gibi tanımlanabilir.

### **Veri Tipi Adı**

Liste

### **Değerler Kümesi**

Sonlu sayıda elemanı olan bir dizi içerisinde tanımlanabilen tüm değerler

### **İşlemler**

Listenin boş olup olmadığını kontrol edilmesi

Listenin dolu olup olmadığını kontrol edilmesi

Listede verilen bir elemanın aranması

- Listeden bir elemanın silinmesi
- Listeye bir elemanın eklenmesi
- Listeye belirtilen bir konma verilen bir elemanın yerleştirilmesi
- Liste elemanlarının sıralanması
- Listenin silinmesi
- Listenin yazdırılması

Buradan da görüleceği gibi veriyi oluşturan değerler ile bu değerler üzerinde yapılabilecek olan işlemlerin birbirinden ayrılması olanaklıdır. Burada şu soru akla gelebilir: "Bir soyut veri tipi bir programda nasıl uygulanabilir?" Bu sorunun yanıtı aslında sorunun içerisinde gizlidir. Soyut veri tipi veri ile bu veri üzerinde gerçekleştirilebilen işlemlerin ayrılması üzerine dayalıdır. O halde veri üzerinde gerçekleştirilebilecek olan işlemler tanımlandığında algoritmalar yapılarak program yazılabilir ve soyut veri tipi de uygulanmış olur.

Önceki bölümde sınıfın yapısında nesneleri ve fonksiyonları tanımlamıştık. Sınıfın tanımında sadece fonksiyonların bildirimi yer alırken, fonksiyonlar sınıfın dışında tanımlanmıştı. Böylece aslında sınıfların soyut veri tiplerinin uygulanması için son derece elverişli olduğu gözlenmektedir.

Liste veri tipi bir çok kaynakta açık olarak tanımlanmış olsa da burada burada kendi veri tipimizi yukarıda tanımlanan özellikleri barındıracak şekilde aşağıdaki sınıf tanımlamasında olduğu gibi tanımlanabilir.

```
class listeOrnek
{
public:
    bool bosListeKontrol() const;
    bool doluListeKontrol() const;
    int listedeAra(int aranan) const;
    void listeyeEkle(int elemanEkle);
    void listedenCikar(int elemanSil);
    void listeSil ();
    void listeYaz() const;
    listeOrnek() // Yapıçı fonksiyon
private:
    int liste[100];
    int buyukluk;
};
```

Aşağıda yukarıda yapılan tanımlamanın UML diagramı olarak ifadesi görülmektedir.

listeOrnek
-liste: [100]
-buyukluk: int
+bosListeKontrol() const: bool
+doluListeKontrol() const: bool
+listedeAra(int aranan) const: int
+listeyeEkle(int elemanEkle): void
+listedenCikar(int elemanSil): void
+listeSil (): void
+listeYaz() const: void
+listeOrnek()

### 9.2.1.Yapılar ve Sınıflar

Bir önceki bölümde yapıları – struct incelemiştir. Yapı, teknik olarak birden çok sayıda ve farklı bileşenden oluşmakta idi. Bu tanım, temel olarak bir yapının sadece üye bileşenlerden oluşmakta olduğunu belirtir. Ancak C++ Programlama Dili’nde yapılar, sınıflara ile benzerlikler taşımaktadır. Sınıfta olduğu gibi bir yapının üyeleri arasında fonksiyonlar yer alabilir. Bu fonksiyonlar arasında yapıçı ve yıkıcı fonksiyonlarda bulunabilir. Bir sınıf ile bir yapı arasındaki temel fark, yapının üyelerinin “genel” özelliğe sahip olması iken, sınıfın üyeleri hem “genel” hem de “özel” olabilmektedir. Benzer olarak bir yapının üyelerini “özel” olarak tanımlamakta olanaklıdır.

C Programlama Dili’nde bulunan yapılar ile C++ programlam Dili’nde bulunan yapılar arasında temel olarak bir fark bulunmamaktadır. Çünkü C++ programlam Dili, C Programlam Dili esas alınarak üzerinde geliştirilmiştir. Dolayısı ile de bir C Programlam Dili’nde yapılmış olan bir yapı, bir C++ Programlam Dili ile yazılmış olan bir programda rahatlıkla kullanılabilir. Ancak C++ Programlam Dili’nde yapılar, C Programlam Dili’nde olduğundan farklı olarak yapısında üye fonksiyonlar, yapıçı ve yıkıcı fonksiyonları da barındıracak şekilde tasarlanmıştır. Bu nedenle de C++ Programlam Dili özelinde bakıldığından yapıların, tamamen soyut veri tiplerini barındıracak şekilde tasarlandığı görülmektedir. Bu da gelecekteki geliştirme sürecinde yapıların C Programlama Dili’nde olduğundan daha farklı bir şekilde evrilerek gelişebilecegi anlamına gelmektedir.

Pratikte C++ Programlam Dili’nde yapılar ile sınıflar teknik olarak çok yakın özelliklere sahip olmalarına rağmen bir çok programcı, yazdığı programlarda yapı kullanımını C Programlam Dili’ndeki şekli ile kullanmaktadır. Bu nedenle de üye fonksiyonlarının yapılarda kullanımına nadiren rastlanmaktadır. Bu nedenle bir sınıfın tüm üyeleri genel olarak tanımlanmış ve sınıfın yapısında herhangi bir üye fonksiyon bulunmuyor ise, bu durumda yapı – struct kullanılarak üyelerin gruplanması genel geçer bir uygulamaya dönüşmektedir. Bu ders notlarında da aynı pratiğe sadık kalınmıştır.

## 9.3.Başlık Dosyaları

Önceki bölümlerde yazdığımız saatUygulaması sınıfını tanımlamıştık. Ardından bu sınıfı aynı adlı bir program içerisinde kullandık. Ancak sınıfın üyesi olan fonksiyonlar ayrıca sınıfın

dışında ana fonksiyonun yapısı içerisinde yazılarak program tamamlanmıştır. Böylece sınıf Uygulaması adlı sınıfın tanımlanması ile uygulanması programın içerisinde yer almış oldu.

Bu yaklaşım ilk bakışta herhangi bir sakınca ortaya çıkarmamış gibi görülebilir. Sınıfın tanımı ile uygulanmasının program içerisinde yer verilmesi özellikle de birde çok sayıda programcının ortak çalıştığı yazılım projelerinde pratikte sorunlara neden olabilir. Tanımlama ile uygulamayı aynı program veya yazılımın yapısında birlikte uygulamak programcıların gerekli gördüğü her durumda tanımlamaları ve uygulamayı değiştirebilecekleri anlamına gelecektir. Bu değişiklikler programın veya yazılımın geliştirme sürecinde sorunlara neden olabileceği kolaylıkla görülebilir. Bu değişikliklerin neden olduğu sorunların önüne geçilebilmesi geliştirme sürecinin temelden değiştirilmesi gerekecektir.

Benzer bir durum donanım üreticileri ile işletim sistemi geliştiricileri açısından da önemlidir. Donanım sürücülerinin yazılması ile işletim sisteminin aygit yönetimi bileşenlerinin yazılması sırasında geliştiricilerin ortak tasarım özelliklerini üzerinde uzlaşması ve gerekli arayüzleri olması gereği gibi tasarlayıp uygulaması zorunludur. Tekil ve bir grup geliştiricinin bu uzlaşılmış tasarım ve uygulama ölçütlerinden farkla şekilde yazılım geliştirmesi uyumsuzluk sorunlarına neden olur.

Geliştirme süreçlerinde bu tür olumsuzlukların ortadan kaldırılması, tekil programcıların aynı problemleri kendi bakış açılarından ve kendi gereksinimlerine göre tekrardan çözmeye çalışması da bir ok uyumsuz veya uyumlu olduğu halde kaynak gereksinimi ve zaman gereksini açısından farklılık gösterecek olan bir çok programın yazılmasına neden olacaktır. Bu sorunların ortadan kaldırılması için tasarıma özel olan ve üzerinde uzlaşmış olan tüm özelliklerin programın yapısından ayrı olarak **başlık dosyası/arayüz dosyası – header file/interface file**, .h olarak uygulanması tercih edilir. Böylece tasarımım temel taşı sayılabilen işlevler ve özellikler hatasız ve doğru şekilde uygulanmış olur.

Uygulama dosyası bir nesne üzerinde işlem yapacak olan fonksiyonların tanımlandığı dosyadır. Dosyanın içerisinde tanımlamlar, ifadeler, bildirimler ve ön işlemci komutları yer alır. C++ Programlam Dili’nde tek bir ana fonksiyon olabileceği için uygulama dosyasında ana fonksiyon bulunmaz. Bu nedenle uygulama dosyasından yararlanarak bir nesne dosyası yani çalıştırılabilir dosya üretilemez. Ayrıca uygulama dosyasının uzantısı “.h” olmak durumundadır. Bu dosyasının adının belirlenmesi sırasında söz konusu fonksiyonları gruplayarak yapısında barındıracağı için bu fonksiyonların kümesini tanımlayan ve işlevini açıklayan bir başlık seçilmelidir. Bu dosyasının yazılan program için kullanılabilmesi için aşağıda gösterildiği gibi programa önişlemci komutları ile eklenebilir.

```
# include "saatuygulamasi.h"
```

Burada dikkatinizi çeken ön işlemci komutu ile arayüz/başlık dosyaları söz konusu olduğunda kullanılmakta olmasıdır. Çünkü bu standart bir ön işlemci komutu olmayıp programcı tarafından hazırlanmış olan bir ön işlemci komutudur. Bu nedenle de “” arasında yapılması gereklidir. Ayrıca bu dosyalarında programım kaynak kodunun bulunduğu dizinde yer olması bir diğer zorunluluktur.

Arayüz/başlık dosyasında fonksiyonların tanımlamaları yer alacaktır. Bu tanımlamlar artık sadece ön işlemci komutu aracılığı ile bağlayıcıya iletilecek ve programın nesne kodu yani çalıştırılabilir koduna bağlanacaktır. Ancak burada dikkat edilmesi gereken ise programın kaynak kodundan

çıkarılan ve arayüz dosyasına yazılan fonksiyonlar asıl program ile olan ilişkisinin soyutlanmış olduğudur. Bu durumda arayüz dosyasında yer verilen fonksiyon tanımlarının belgelendirilmesi zorunludur. Bunun bir yolu, arayüz dosyasında yer alan fonksiyonların kapsamlı açıklamalarının yorum satırı olarak dosyada yer verilmesidir. Aşağıdaki örnekte söz konusu arayüz dosyasına yer alan prototip fonksiyon satırlarında ilgili fonksiyonun açıklamaları, varsa **İşlem öncesi koşullar** ile **İşlem sonrası koşullar – preconditions ve postconditions** – tanımlanmıştır.

**İşlem öncesi koşullar**, adında anlaşılacağı üzere bir işlemin gerçekleştirilemesinden önce kontrol edilen ve geçerli olmaları gereken koşullardır.

**İşlem sonrası koşullar**, adında anlaşılacağı üzere bir işlemin gerçekleştirilemesinden sonra kontrol edilen ve geçerli olmaları gereken koşullardır.

Aşağıdaki örnek programda bir dairenin alanını ve çevresini hesaplayan bir program nesne yönelimli programlama metodolojisi ile yazılmıştır.

**Örnek 1:** Bir dairenin yarı çapı verildiğine göre bu veriyi kullanarak dairenin çevresini ve alanını hesaplayan programı yazınız.

**Cözüm:** İlk olarak daire.h adlı başlık dosyası yazılacaktır. Bu dosyada daire sınıfının genel ve özel değişkenleri, daire sınıfının üyesi olan nesneler ile daire sınıfının üyesi olan fonksiyonlar yazılmaktadır.

```
/*
 * daire.h
 *
 *
 * Copyright 2023 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtım; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
```

```

*
* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
VERİ
* VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
* SORUMLU DEĞİLLERDİR.

*/
*/
```

class daireUygulaması

```

{
```

    public:

```

        void yariCapAta(double r);
        /*
            * Yarı çapın değerini atayan fonksiyon
            * İşlem sonrası koşul:
            * Okunan yarıçap değeri pozitif ve sıfırdan büyük ise
            * değer kullanılır, aksi halde yarıçap değeri olarak sıfır atanır.
            */
        double yariCapOku();
        /*
            * Yarı çapın değerini okuyan fonksiyon
```

```

* İşlem sonrası koşul:
* Okunan yarıçap değeri döndürülür
*/
double alan();
/*
* Dairenin alanını hesaplayan fonksiyon
* İşlem sonrası koşul:
* Dairenin alanı hesaplanır ve döndürülür.
*/
double çevre();
/*
* dairenin çevresini hesaplayan fonksiyon
* İşlem sonrası koşul:
* Dairenin çevresi hesaplanır ve döndürülür.
*/
daireUygulaması(double r = 0);
/*
* Ön tanımlı yapıcı fonksiyon
* Verilen parametreye göre dairenin yarı çapı için
* ön tanımlı değer sıfır olarak seçilmiştir.
* işlem sonrası koşul:
* Dairenin yarı çapı r parametresi ile tanımlanan değerdir.
*/
private:
    double yaricap;
};

// Üye fonksiyonlarının tanımlamaları
void daireUygulaması::yariCapAta(double r)
{
    if (r >= 0)
        yaricap = r;
}

```

```

else
    yaricap = 0;
}

double daireUygulamasi::yariCapOku()
{
    return yaricap;
}

double daireUygulamasi::alan()
{
    return 3.1416 * yaricap * yaricap;
}

double daireUygulamasi::cevre()
{
    return 3.1416 * 2 * yaricap;
}

// Ön tanımlı yapıcı fonksiyon
daireUygulamasi::daireUygulamasi(double r)
{
    yariCapAta(r);
}

```

İkinci olarak da ana fonksiyonun bulunduğu program yazılacaktır.

```

/*
 * daire.cxx
 *
 *
 * Copyright 2023 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:

```

\*

\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM, VERİ

\* VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
```

```
#include <locale>
```

```
#include <iomanip>
```

```
#include "daire.h"
```

```
using namespace std;
```

```
int main()
```

{

```
daireUygulamasi daire1(8);  
daireUygulamasi daire2;  
double yaricap;  
cout << fixed << showpoint << setprecision(2);  
// Ön tanımlı değerler ile yapılan hesaplamalar  
cout << "Birinci daire " << endl;  
cout << "Yarıçap: " << daire1.yariCapOku() << endl;  
cout << "Alan: " << daire1.alan() << endl;  
cout << "Çevresi: " << daire1.cevre() << endl;  
cout << "İkinci daire " << endl;  
cout << "Yarıçap: " << daire2.yariCapOku() << endl;  
cout << "Alan: " << daire2.alan() << endl;  
cout << "Çevresi: " << daire2.cevre() << endl;  
// Kullanıcı tarafından girilen değerler ile yapılan hesaplamalar.  
cout << "Dairenin yarı çapını giriniz: ";  
cin >> yaricap,  
cout << endl;  
daire2.yariCapAta(yaricap);  
cout << "Tanımlanan yarı çap için hesaplanan değerler." << endl;  
cout << "Yarıçap: " << daire2.yariCapOku() << endl;  
cout << "Alan: " << daire2.alan() << endl;  
cout << "Çevresi: " << daire2.cevre() << endl;  
return 0;
```

}

Programın derlenip çalıştırılması için öncelikle nesne dosyasının oluşturulması gereklidir .Bu nedenle geany ile program yazılıp derleniyor ise, bu durumda doğrudan işa etmek yerine önce “**Nesneyi Derle**” komutu ile nesne doyası oluşturulmalı ve ardından “**İnsa Et**” komutu çalıştırılmalıdır.

“**Nesneyi Derle**” komutunun sonucunda “**daire.o**” dosyası elde edilir. Bu dosya tek başına çalıştırılabilir bir dosya değildir .Oluşturulan nesne dosyasının bağlayıcı ile işlemenin geçirilerek çalıştırılabilir dosya elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar/2022-2023/Bahar/09/daire]$ ls -al
total 80
drwxr-xr-x  2 goksin  goksin      512 24 May 22:51 .
drwxr-xr-x  4 goksin  goksin      512 24 May 22:01 ..
-rwxr-xr-x  1 goksin  goksin  31152 24 May 22:51 daire
-rw-r--r--  1 goksin  goksin    2781 24 May 22:49 daire.cxx
-rw-r--r--  1 goksin  goksin    3216 24 May 22:36 daire.h
-rw-r--r--  1 goksin  goksin  32152 24 May 22:49 daire.o
[goksin@tardis ~/projeler/C++_Notlar/2022-2023/Bahar/09/daire]$ ./daire
Birinci daire
Yarıçap: 8.00
Alan: 201.06
Çevresi: 50.27
İkinci daire
Yarıçap: 0.00
Alan: 0.00
Çevresi: 0.00
Dairenin yarı çapını giriniz: 4.2
```

Tanımlanan yarı çap için hesaplanan değerler.

Yarıçap: 4.20  
 Alan: 55.42  
 Çevresi: 26.39

```
[goksin@tardis ~/projeler/C++_Notlar/2022-2023/Bahar/09/daire]$
```

**Örnek:** Önceki bölümlerde yer verilen ve iki zarın atılmasını temsil eden programın nensen yoneliimi programlama metodolojisine göre yazılmış olan hali aşağıda verilmiştir.

```
/*
 * zar.h
 *
 *
 *
 * Copyright 2023 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 *
 * Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
```

```

/*
* * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA
* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,
VERİ

* VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
* SORUMLU DEĞİLLERDİR.

*/
class zar
{
    public:
        void zarAt();
        /*
         * Zar atılması işlemini temsil eden fonksiyon.
         * Rastgele sayı üreticini kullanır.
         * Üretilen sayılar 1 ile 6 kapalı aralığındadır.
         * Üretilen değer sayı adlı değişkende barındırılır.
         */
        int zarOku() const;
        /*
         * Rastgele sayı üreticisinin ürettiği isayı okuyan fonksiyon.
        */
}

```

```

        * Okunacak olan değer sayı adlı değişkendeki değerdir.

    */

    zar();

    /*
     * Ön tanımlı yapıcı fonksiyon
     * Zar atıldığında ön tanımlı değeri bir yapar.
    */

    private:

        double sayı;

};

// Üye fonksiyonların tanımlamaları

void zar::zarAt()
{
    sayı = rand() % 6 + 1;
}

int zar::zarOku() const
{
    return sayı;
}

// Ön tanımlı yapıcı fonksiyon

zar::zar()
{
    sayı = 1;
    srand(time(0));
}

```

İkinci olarak da ana fonksiyonun bulunduğu program yazılacaktır.

```

/*
 * zar.cxx
 *
 *
 * Copyright 2023 Goksin Akdeniz <goksin.akdeniz@gmail.com>

```

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM, VERİ

\* VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

#include <iostream>

```

#include <locale>
#include "zar.h"
using namespace std;
int main()
{
    zar zar1;
    zar zar2;
    cout << "Birinci zar: " << zar1.zarOku() << endl;
    cout << "İkinci zar: " << zar2.zarOku() << endl;
    zar1.zarAt();
    cout << "Birinci zar atıldıktan sonra okunan sayı: " << zar1.zarOku() << endl;
    zar2.zarAt();
    cout << "İkinci zar atıldıktan sonra okunan sayı: " << zar2.zarOku() << endl;
    cout << "Okunan değerlerin toplamı = " << zar1.zarOku() + zar2.zarOku() << endl;
    zar1.zarAt();
    zar2.zarAt();
    cout << "Zarlar tekrar atıldıktan sonra elde edilen toplam = " << zar1.zarOku() +
zar2.zarOku() << endl;
    return 0;
}

```

**Örnek:** Aşağıda verilen başlık dosyası ile ana fonksiyonun yer aldığı programda klavyeden girilen ad ve soyad verisi okunup daha sonra yazdırılmaktadır. Söz konusu başlık dosyası ile ana program için esas alınacak olan uml diagramı aşağıda verilmiştir.

```

/*
 * adsoyad.h
 *
 *
 * Copyright 2023 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

```

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtım; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM, VERİ

\* VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <string>
```

```
using namespace std;
```

```
class adSoyad
```

```
{
```

```

public:
    void yaz() const;
    /*
     * Okunan ad ve soyad verisini sırası ile yazan fonksiyondur.
     * Yazdırma sırası 'ad soyad' şeklindedir.
     */
    void adSoyadAta(string isim, string soyisim);
    /*
     * Okunan ad ve soyad verisini sırası parametrelere
     * geçiren fonksiyondur.
     * İşlem sonrası koşullar:
     * kullaniciAdi = ad; kullaniciSoyadi = soyad;
     */
    string adOku() const;
    /*
     * Okunan adı döndüren fonksiyondur.
     * İşlem sonrası koşullar:
     * ad değeri döndürülür.
     */
    string soyadOku() const;
    /*
     * Okunan soyadı döndüren fonksiyondur.
     * İşlem sonrası koşullar:
     * soyad değeri döndürülür.
     */

    adSoyad(string isim = "", string soyisim = "");
    /*
     * Ön tanımlı yapıçı fonksiyon
     * Ad ve soyad için ön tanımlı değerler boştur.
     */

```

```

*/
private:
    string ad;
    string soyad;
};

// Üye fonksiyonların tanımlamaları

void adSoyad::yaz() const
{
    cout << ad << " " << soyad << endl;
}

void adSoyad::adSoyadAta(string isim, string soyisim)
{
    ad = isim;
    soyad = soyisim;
}

string adSoyad::adOku() const
{
    return ad;
}

string adSoyad::soyadOku() const
{
    return soyad;
}

// Ön tanımlı yapıçı fonksiyon

adSoyad::adSoyad(string isim, string soyisim)
{
    ad = isim;
    soyad = soyisim;}
```

UML diagramı aşağıdaki gibidir:

adSoyad
-ad: string
-soyad :string
+yaz(): void
+adSoyadAta(string, string): void
+adOku() const: string
+soyadOku() const: string
+adSoyad(string = "", string = "")

İkinci olarak da ana fonksiyonun bulunduğu program yazılacaktır.

```
/*
 * adsoyad.hxx
 *
 *
 * Copyright 2023 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 * *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtıımı; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
```

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA  
\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP  
\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM,  
VERİ  
\* VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE  
\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ  
\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI  
\* SORUMLU DEĞİLLERDİR.  
\*  
\*/

```
#include <iostream>
#include <locale>
#include <string>
#include "adsoyad.h"
using namespace std;
int main()
{
    adSoyad kullanici;
    string isim;
    string soyisim;
    cout << "Adınız: ";
    cin >> isim;
    kullanici.adOku();
    cout << "Soyadınız: ";
    cin >> soyisim;
    kullanici.soyadOku();
    kullanici.adSoyadAta(isim, soyisim);
    cout << "Verdiğiniz bilgilere göre adınız ve soyadınız: ";
    kullanici.yaz();
    return 0;
```

}

## 9.4.Sınıfın Statik Olarak Tanımlanan Üyeleri

Önceki bölümlerde değişkenlerin statik ve otomatik olarak gruplandığını görmüştür. Statik değişkenler, programın akışı içerisinde değerleri değişmeden korunurken, otomatik olarak tanımlanan değişkenler ise program akışı içerisinde değişime açık olan değişkenlerdir. Aynı şekilde bir fonksiyonun yapısında statik olarak tanımlanan bir değişken de, fonksiyonun içerisinde değerini korur, değişkenin değeri fonksiyonun işlenmesi boyunca korunur. Bu durum sınıflar içinde geçerlidir. Bir sınıfın yapısında bulunan bir üye değişkenin statik olarak tanımlanması durumunda şu özelliklere dikkat edilmesi gereklidir:

1. Bir sınıfın üyesi olan bir fonksiyon statik olarak tanımlanabilir. Bunun için sınıfın tanımlaması sırasında “**static**” özel sözcüğün fonksiyonun önünde yazılması gereklidir.
2. Bir sınıfın üyesi olan bir değişken statik olarak tanımlanabilir. Bunun için sınıfın tanımlaması sırasında “**static**” özel sözcüğün değişkenin başınd\_comment\_ yazılması gereklidir.
3. Bir sınıfın üyesi genel olarak tanımlanmış ancak statik olma özelliğine de sahip olması isteniyor ise, bu durumda tanımlanırken “**public static**” olarak tanımlanması gereklidir.

Aşağıda verilen başlık dosyasında statik değişkenler ile statik fonksiyonlar tanılanmıştır

```
class goster
{
public:
    static int sayac;
    //public static variable

    void yaz() const;
    //x, y, ve sayac değerlerini yazdırır.

    void ataX(int a);
    //x değerini atar

    //İşlem sonrası durum:
    // x = a;

    static void arttirY();
    //statik fonksiyon

    //y değerini bir arttır.

    //İşlem sonrası durum:
    // y = y + 1

    goster(int a = 0);
```

```

// Ön tanımlı yapıcı fonksiyon
//İşlem sonrası durum; x = a;
//Eğer a için herhangi bir dğer tanımlanmadı ise x = 0;

private:
    int x;
    static int y;
};

int goster::sayac = 0;
int goster::y = 0;
void goster::yaz() const
{
    cout << "x = " << x << ", y =" << y << ", sayac = " << sayac << endl;
}

void goster::ataX(int a)
{
    x = a;
}

void goster::arttirY()
{
    y++;
}

goster::goster(int a)
{
    x = a;
}

```

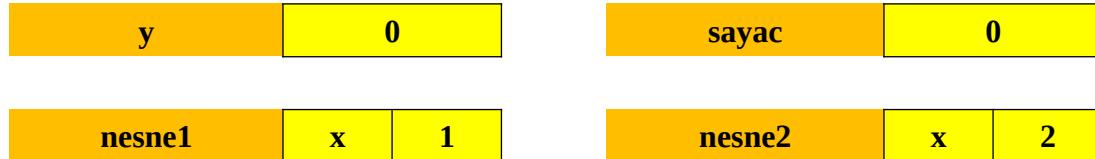
Bir sınıfın yapısında statik ve otomatik üye değişkenlerin tanımlandığı durumda, eğer sınıfın üyesi olan bir nesne tanımlanacak olursa, sadece statik olarak tanımlanmamış değişkenler nesnenin üyesi olabilir. C++ Programlama Dili, her bir statik değişken için tekil bir bellek adresi ayırrır. Tüm nesneler de statik değişkeni barındıran bu tekil bellek adreslerini referans alırlar. Aslında bu sınıfın üyesi olan herhangi bir nesne tanımlanmamış olsa bile, bu statik değişkenler için bellek üzerinde alan ayrırlır. Bu **“public static”** oalrak tanımlanan değişkenler sınıfın dışından, önceki bölümlerde açıklandığı gibi erişilebilir.

Burada hem statik hem de statik olamayan değişkenler için bellek alanının nasıl ayrıldığı üzerinde duralım. Yukarda tanımlanan sınıfın programda kullanılması durumunda, **sayac** ve **y** isimli değişkenler statik olarak tanımlanmış olacaktır. Eğer bu sınıfın üyesi olan iki nesne tanımladığımızı var sayalım.

```
goster nesne1(1);
```

```
goster nesne2(2);
```

Bu bildirimin işlenmesi ile iki adet nesne yaratılmış olacaktır. Bu durumda bellek üzerindeki durum aşağıdaki şekildeki gibi olacaktır.

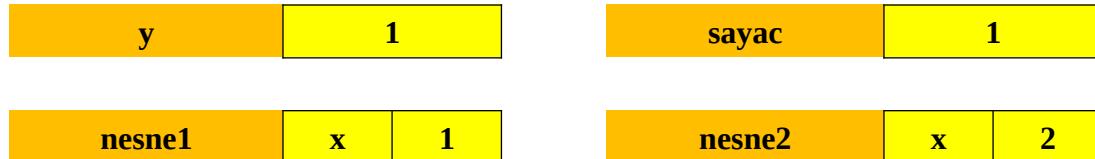


Ardından program içerisinde aşağıda verilen bildirimleri işlendiğini var sayalım.

```
goster::arttirY();;
```

```
goster:: sayac++;
```

Bu durumda bellek üzerindeki değişkenlerin durumu aşağıdaki gibi olacaktır.



Eğer yazdırma fonksiyonunu çağıracak olursak;

```
nesne1:: yaz();
```

Cıktının aşağıdaki benzer şekilde olması gereklidir.

```
x = 1, y = 1, sayac = 1
```

Eğer ikinci nesne için de aynı işlemleri yapacak olursak,

```
nesne1:: yaz();
```

```
x = 2, y = 1, sayac = 1
```

Fonksiyon her bir tekil nesneye ait bellek adreslerinde barındırlan değerleri yazdırırken, aynı zamanda sınıfın statik üyelerinin de değerlerini döndürmektedir. Eğer aşağıdaki bildirim program işlenecek olursa, statik değişkenlerin değeri aşağıdaki şekilde görüldüğü gibi olacaktır.

```
goster:: sayac++;
```



nesne1	x	1
--------	---	---

nesne2	x	2
--------	---	---

Eğer sınıfın yaz fonksiyonu her iki nesne için çağrılsrsa,

```
nesne1.yaz();
```

```
nesne2.yaz;
```

Statik değişkenler ile nesnelerin üyesi olan değişkenlerin durumu döndürülmüş olacaktır.

```
x = 1, y = 1, sayac = 2
```

```
x = 2, y = 1, sayac = 2
```

Yukarıda verilen örnek kodların düzenlenerek program olarak aşağıda verilmiştir. Başlık dosyasının adı **goster.h** olarak seçilmiştir.

```
/*
 * goster.h
 *
 *
 * Copyright 2023 Goksin Akdeniz <goksin.akdeniz@gmail.com>
 *
 * Her hakkı saklıdır.
 *
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği
 * takdirde serbesttir:
 *
 * Kaynak kodun yeniden dağıtımı; yukarıdaki telif hakkı uyarısını,
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.
 *
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.
 *
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA
```

- \* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT
- \* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA
- \* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ
- \* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ
- \* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
- \* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP
- \* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM, VERİ
- \* VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
- \* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
- \* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
- \* SORUMLU DEĞİLLERDİR.
- \*
- \*/

```

using namespace std;

class goster
{
public:
    static int sayac;
    /*
     * genel statik değişken
     */
    void yaz() const;
    /*
     * x, y, ve sayac değerlerini yazdırır.
     */
    void ataX(int a);
    /*
     * x değişkenini atayan fonksiyon.
     * İşlem sonrası durum: x = a;
     */
}

```

```

static void arttirY();
/*
 * Statik fonksiyon.
 * y değerini bir arttırır.
 * İşlem sonrası durum: y = y + 1
 */
goster(int a = 0);
/*
 * Ön tanımlı yapıçı fonksiyon
 * İşlem sonrası durum: x = a;
 * Eğer a için herhangi bir değer tanımlanmamış ise x = 0;
 */
private:
    int x;
    static int y;
};

int goster::sayac = 0;
int goster::y = 0;
void goster::yaz() const
{
    cout << "x = " << x << ", y =" << y << ", sayac = " << sayac << endl;
}
void goster::ataX(int a)
{
    x = a;
}
void goster::arttirY()
{
    y++;
}
goster::goster(int a)

```

```
{  
    x = a;  
}
```

Yukarıda verilen **goster.h** başlık dosyasını kullanan asıl program aynı isim seçilerek **goster.hxx** olarak verilmiştir.

```
/*  
 * goster.hxx  
 *  
 *  
 * Copyright 2023 Goksin Akdeniz <goksin.akdeniz@gmail.com>  
 * *  
 * Her hakkı saklıdır.  
 * Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak  
 * ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği  
 * takdirde serbesttir:  
 *  
 * Kaynak kodun yeniden dağıtıımı; yukarıdaki telif hakkı uyarısını,  
 * şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.  
 *  
 * İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,  
 * şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen  
 * belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.  
 *  
 * İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN  
 * "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİLİK VE ÖZEL BİR AMACA  
 * UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA  
 * KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT  
 * REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA  
 * BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ  
 * SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ  
 * YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA
```

- \* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAI VEYA BİR SEBEP
- \* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM, VERİ
- \* VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE
- \* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ
- \* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI
- \* SORUMLU DEĞİLLERDİR.

\*

\*/

```
#include <iostream>
#include <locale>
#include "goster.h"
using namespace std;
int main()
{
    goster nesne1(1);
    goster nesne2(5);
    goster::arttirY();
    goster::sayac++;
    nesne1.yaz();
    nesne2.yaz();
    cout << "y değişkeni nesne1 ile arttıiyor." << endl;
    nesne1.arttirY();
    nesne1.ataX(4);
    nesne1.yaz();
    nesne2.yaz();
    cout << "y değişkeni nesne2 ile arttıiyor." << endl;
    nesne2.arttirY();
    nesne2.ataX(3);
    nesne1.yaz();
    nesne2.yaz();
```

```
    return 0;  
}
```

Program önceki örneklerde belirtildiği gibi derlenerek çalıştırılabilir. Çalıştırıldığında aşağıdaki çıktı elde edilir.

```
[goksin@tardis ~/projeler/C++_Notlar/2022-2023/Bahar/09/goster]$ /goster  
x = 1, y =1, sayac = 1  
x = 5, y =1, sayac = 1  
y değişkeni nesne1 ile arttılıyor.  
x = 4, y =2, sayac = 1  
x = 5, y =2, sayac = 1  
y değişkeni nesne2 ile arttılıyor.  
x = 4, y =3, sayac = 1  
x = 3, y =3, sayac = 1  
[goksin@tardis ~/projeler/C++_Notlar/2022-2023/Bahar/09/goster]$
```

Yukarıda verilen programın işleyişi çıktıda görülebilir. Kavramların daha iyi anlaşılabilmesi için burada üzerinde durulacaktır. Programda statik üyeleri olan y ve sayac değişkenlerinin değerleri sıfıra eşitlenmektedir. Ardından goster nesne1 ifadesi ile nesne1, goster adlı sınıfın nesnesi olmaktadır. Ardından da sınıfın üyesi olan x değişkenin değeri 3 olarak değiştirilmektedir. Ardından gelen satırda yine goster nesne2 ifadesi ile göster adlı sınıfın bir nesnesi olmaktadır. Bu işlemin ardından da sınıfın üyesi olan x değişkenin değeri 5 olarak değişimektedir.

Sonraki işlem admımda goster sınıfı ile arttirY() fonksiyonu birlikte kullanılarak y değişkenin değeri arttırılmaktadır. Burada sayac adlı değişken üyesi olduğu sınıfın genel değişkeni ve statik olarak tanımlandığını anımsatmakta yarar vardır. Dolayısı ile de goster adlı sınıfı doğrudan nesnenin değerini 1 artırabilmektedir. Bu işlemlerin ardından nesne1.yaz() ve nesne1.yaz() ile söz konusu işlemlerin sonuçları yazdırılmaktadır. Burada dikkat edilmesi gereken her iki nesne için y değişkenin değerinin aynı olduğunu.

Bu yazdırma işleminin ardından komut satırına yazdırılan mesajda sıradaki işlem kullanıcıya bildirilmektedir. Söz konusu olan y değişkeninin değeri nesne1 tarafından arttırılmaktadır. Böylece y değişkenin değeri 2 olmaktadır. Bir sonraki ifade nesne1 aracılığı ile x değişkenin değerini 8 olarak değiştirmektedir. Bu işlemin ardından yapılan değişiklikler fonksiyonlar kullanılarak komut satırına yazdırılmaktadır. Burada dikkat edilmesi gereken y ve sayac değişkenlerinin değerlerinin her iki nesne için aynı olduğunu.

Program son bölümünde gerçekleştirilen işlemler önceki bölümler gerçekleştirilen ile benzerlikler göstermektedir. Değişkenler olan y ve sayac değerleri her iki nesne için aynı iken göster sınıfının statik bir üyesi olmayan x değişkenin değeri 8 iken 23 olarak değiştirilmiştir.

Bir sınıf yapısında statik olarak tanımlanmış ve statik olarak tanımlanmamış değişkenlere barındırılabilir. Bu sınıf kullanılarak nesneler yaratıldığında her nesne sınıfın üyesi olan ama statik olmayan değişkenlerin bir kopyasını barındırırken, sınıfın üyesi olan ve statik olarak tanımlanmış değişkenleri ise paylaşarak kullanacaklardır. Her nesnenin yapısında barındırılan değişkenler nesne ile işlem yaptığı sürece varlıklarını koruyacak ve nesne kaldırıldığında onlarda kaldırılacaktır. Bu tür değişkenlere **örnek değişkenler – instance variables** veya **statik olmayan değişkenler – non-**

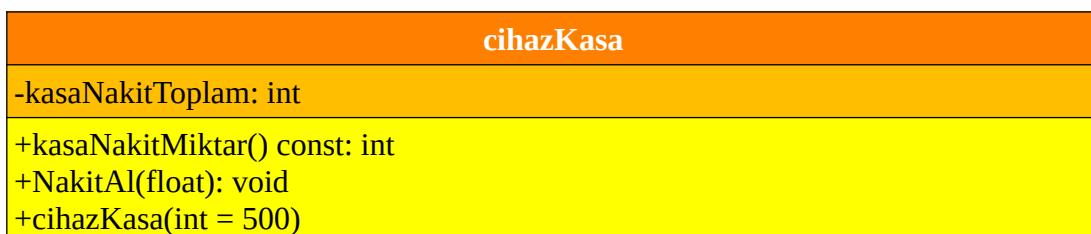
**static variables** adı verilir. Ancak statik olarak tanımlanan sınıf üyeleri sınıf bellekte olduğu sürece varlıklarını sürdürerektir.

**Örnek:** Bir şirket bir sıcak içecek otomatı üretip satmayı planlamaktadır. İçecek otomatının ürünleri şekersiz çay, şekerli çay, sade kahve, sütlü şekersiz kahve, sütsüz şekerli kahve ve sütlü şekerli kahve olarak verilmiştir. Otomat üzerinde çalışacak olan yazılım aşağıdaki işlevleri yerine getirmesi istenmektedir.<sup>12</sup>

1. Müşterinin otomattan satın alabileceği ürünlerin listesi sunulmalıdır.
2. Müşteri ürünler arasından seçim yapmalı ve seçimini girmelidir.
3. Müşterinin seçtiği ürünü ücreti bildirilmelidir.
4. Müşteri ücretini nakit ödeyecektir.
5. Ödeme sonrası müşterinin seçtiği ürün ve varsa para üstü verilecektir.

**Çözüm: Programın Çözümlenmesi ve Algoritma Tasarımı:** Yukarıdaki gereklilikler incelendiğinde programın girdisi olarak ürün seçimi ve ürünün ücreti; çıktı olarak ise müşterinin satın aldığı içecek karşımıza çıkmaktadır. Ayrıca otomatın iki bileşeni olduğu görülmektedir. Bunlar, ürünlerin barındırıldığı servis aygıtı ile ücretin toplandığı kasa kısımdır.

Programın yazılabilmesi için öncelikle bu iki bileşenin özellikleri üzerinde durulmalıdır. İlk olarak kasanın işleyişi ele alınacaktır. Kasa, sistemin düzenli olarak gerçekleştirilen kontrolleri sırasında toplanan ücret alınmakta olsa da, kasada her zaman para üstü verebilmek için bir miktar nakit bulundurulmaktadır. Dolayısı ile kasanın nakit miktarı asla sıfır olmamaktadır. Ayrıca kasada toplanan ücretin miktarı istenilen bir anda döndürülmelidir. Buna göre kasa için bir sınıf yazılabılır. Aşağıda verilen uml diagramında **cihazKasa** adlı sınıf gösterilmiştir.



Aşağıda cihazKasa adlı sınıfın kaynak kodu görülmektedir.

```
class cihazKasa
{
    public:
        float kasaNakitMiktar() const;
        /*

```

<sup>12</sup> C++ Programlama dili kullanılarak Arduino sistemler programlanabilir. Bu otomata benzer bir araç tasarlanıp programlanabilir ve çalıştırılabilir. Aynı şekilde günümüzde kullanılan endüstriyel kontrol ve otomasyon sistemlerinin bazlarında C++ programlama dili kullanılarak otomasyon yazılımları geliştirilebilmektedir.

```

* Kasada o anda bulunan nakit miktarını döndürür.

* işlem sonrası koşullar:

* kasaNakitToplam değerini döndürür.

*/



void nakitKabul (float alinanNakit)

/*
 * Müşteri tarafından ödenen ücreti kasadaki nakit ile toplayarak kasadaki
 * para miktarını güncelleyen fonksiyon
 * İşlem sonrası koşullar:
 * kasaNakitToplam = kasaNakitToplam + alinanNakit;
*/



cihazKasa (float kasaDevir = 500)

/*
 * Yapıçı fonksiyondur. Değişkeni etkinleştirimekte ve ön tanımlı değeri atamaktadır.
 * İşlem sonrası koşullar:
 * kasaNakitToplam = kasaDevir
*/



private:

    float kasaNakitToplam;      // Kasada bulunan nakit miktarıdır.

};


```

Yukarıda sınıfın genel ve özel üyeleri belirtilmiştir. Bu üyeler ile işlem yapacak olan fonksiyonlar belirtilmiş olsa da, fonksiyon tanımlanması gereklidir. Aşağıda söz konusu olan prototip fonksiyonlarının tanımlamaları yapılmıştır.

İlk tanımlanacak olan fonksiyon **kasaNakitMiktar()** adlı fonksiyondur. Kasada herhangi bir andaki para miktarını döndürmektedir. Bunu yaparken sınıfın özel değişkenine erişerek değerini okumaktadır.

```

float cihazKasa::kasaNakitMiktar() const
{
    return kasaNakitToplam;
}

```

Diğer fonksiyon olan **nakitKabul()** adlı fonksiyondur. Müşterinin ödediği ücret miktarı ile kasada bulunan para miktarını toplar ve kasadaki para miktarını günceller. Bu fonksiyon kasaNa-

kitToplam adlı değişken ile alınanNakit adlı değişkeni toplayıp sonucu kasaNakitToplam adlı değişkene atamaktadır.

```
void cihazKasa::nakitKabul( float alinanPara)
{
    kasaNakitToplam = kasaNakitToplam + alinanNakit;
}
```

Sınıfın tanımlamasında bir yapıçı fonksiyon olan **cihazKasa()** tanımlanarak ön tanımlı değer ataması yapılmaktadır. Söz konusu yapısı fonksiyon kasaNakitToplam adlı değişkenin değerini 500 olarak tanımlamaktadır. Burada dikkat edilmesi gereken sınıfın tanımlanmasında yapıçı fonksiyonun ön tanımlı değeri belirtildiği için tekrar yazılmasına gerek yoktur. Aşağıda verilen yapıçı fonksiyonun incelenmesinden görüleceği üzere fonksiyon kasaDevir adlı değişkenin değerinin geçerliliğini kontrol etmektedir. Eğer geçersiz bir değer bulunuyor ise, bu durumda ön tanımlı değer atanmaktadır.

```
cihazKasa::cihazKasa( float kasaDevir)
{
    if (kasaDevir => 0)
        kasaNakitToplam = kasaDevir;
    else
        kasaNakitToplam = 500;
}
```

Söz konusu sınıfın tanımlanması ve sınıfın üye fonksiyonlarının da yazılmasının ardından ürünü servis eden aygıta ilişkin sınıfın tanımlanması ve yazılması aşaması gelmektedir. Servis aygıtı seçilen ürün stokta bulunuyorsa ürünü servis etmektedir. Bu sınıfın yazılabilmesi için ürün miktarının ve ücretlerinin bilinmesi gereklidir. Çünkü aygit ürünü servis ettiğinde ürünün stok miktarı azalacak, kasadaki para miktarı artmış olacaktır. Bu işlemlerde doğrudan bu sınıfın yaptığı işlemler olarak tanımlanabilir. Aşağıda verilen uml diagramında **servisİslemler** adlı sınıf gösterilmiştir.

servisİslemler
-urunSayisi: int
-ucret: float
+urunSayisiOku() const: int
+ucretOku const: float
+satisciIslemi(): void
+servisIslemler(int = 100, float = 1)

Aşağıda **servisİslemler** adlı sınıfın kaynak kodu görülmektedir.

```
class servisIslemeler
{
    public:
        int urunSayisiOku() const;
        /*
         * Otomatta o an da stokta bulunan ürünün sayısını döndürür.
         * işlem sonrası koşullar:
         * ürünün sayısı döndürülür.
        */
        float ucretOku () const;
        /*
         * Müşteriye seçtiği ürünün ücretini döndüren fonksiyondur.
         * İşlem sonrası koşullar:
         * ürünün fiyatı döndürülür;
        */
        void satisIslemi();
        /*
         * Satışı yapılan ürünün sayısını bir azaltır.
         * İşlem sonrası koşullar:
         * urunSayisi--;
        */
        servisIslemeler (int urunSayisiAta = 100, float ucretAta = 1);
        /*
         * Yapıçı fonksiyondur. Değişkenleri etkinleştirmekte ve ön tanımlı değerleri
         * atamaktadır.
         * İşlem sonrası koşullar:
         * urunSayisi = urunSayisiAta;
         * ucret = ucretAta;
        */
    private:
        int urunSayisi; // Otomatta sunulan ürün adedidir.
```

```
    float ucret;           // satılan ürünün ücretidir.  
};
```

Otomat ile satışa sunulan ürün sayısı altı adettir. Bu nedenle programda altı adet nesne tanımlanması gerekecektir. Örneğin yukarıda belirtilen ürünler dışında elma meyve suyu olarak satılacak olursa nesne tanımının isim, miktar ve ücretin tanımlanması gereklidir. Buna göre nesen aşağıdaki gibi tanımlanabilir.

```
servisIslemler elmaSuyu (50, 10)
```

elma	urunSayisi	50
	ucret	10

Bu aşamadan sonra **servisIslemler** adlı sınıfın üyesi olan fonksiyonların yazılması gerekecektir. İlk olarak **urunSayisiOku()** adlı fonksiyon yazılacaktır.

Bu fonksiyon belirli bir ürünün stok miktarını okumakta ve döndürmektedir. Özel olarak tanımlanmış olan **urunSayisi** adlı değişkende veriler barındırılmaktadır. Fonksiyon bu değişkenin barındırdığı değeri okuyup sonucu döndürmektedir.

```
float servisIslemler::urunSayisiOku() const  
{  
    return urunSayisi;  
}
```

Sonraki fonksiyon **ucretOku()** fonksiyonudur. Ürünlerin ücretleri ücret adlı özel değişkende barındırılmaktadır. Fonksiyon değişkenin barındırdığı değeri okuyarak sonucu döndürecektr.

```
int servisIslemler::ucretOku() const  
{  
    return ucret;  
}
```

Bir ürün satışı gerçekleştiğinde aygıtın depolama alanında bulunan ürün sayısı bir azalır. Bundan dolayı **satisIslemi()** fonksiyonu satılan ürünün stok miktarını bir adet azaltacaktır. Bu fonksiyon bu işlemi **satisIslemi** sınıfının özel üyesi olan **urunSayisi** adlı değişkeni azaltarak gerçekleştirilmektedir.

```
void servisIslemler::satisIslemi()  
{  
    urunSayisi--  
}
```

Son olarak sınıfın yapısı fonksiyonu tanımlanacaktır. Bir önceki yapıçı fonksiyonda olduğu gibi, öncelikle geçerli bir değer atanıp atanmadığı kontrol edilecek ve gerekiyorsa ön tanımlı değerler değişkenlere atanacak ve etkinleştirme işlemi sona erecektir.

```
satisIslemi::satisIslemi( int urunSayisiAta, float ucretAta )  
{  
    if (urunSayisiAta => 0)  
        urunSayisi = urunSayisiAta;  
    else  
        urunSayisiAta = 100;  
  
    if (ucretAta => 0 )  
        ucret = ucretAta;  
    else  
        ucret = 1;  
}
```

Sınıflar ve bu sınıfların üyesi olan fonksiyonlar yazıldıktan sonra asıl programın yazılması gereklidir. Burada programın gerçekleştireceği işlemler tanımlanacak ve varsa gerekli olan diğer fonksiyonlar da yazılmamacaktır.

Problemin sunulmasında programın yapacağı işlemler tanımlanmıştır. Bu işlemleri ilgili sınıflara paylaştırıldıkten sonra geriye kalan işlevler ana fonksiyonun yapısında uygulanabilir. Buna göre programın ana fonksiyonu şu işlevleri yerine getirmektedir:

1. Otomat tarafından sunulan ürünleri listelenmelidir.
2. Ürün seçiminin nasıl yapılacağı açıklanmalıdır.
3. Programın sonlandırılması işlemi gerçekleştirilmelidir.

Burada verilen işlevler genel olarak programın asıl fonksiyonunun gerçekleştireceği işlemlerdir. Müşterinin yapacağı satın alma işlemi dikkate alındığında, müşterinin tek bir ürün almayaçağı da düşünülecek olursa, müşterinin her işlemi nasıl yaptığıni anımsamasını ummak yerine programın – sonlanma işlemi hariç olmak üzere – işlem basamaklarını göstermesi gerecektir. Müşteri ürün seçimini yaptıktan sonra ürünün ücretini döndürmesi ve müşterinin parasını ödememesini istemelidir. Müşterinin ödediği para yeterli ise ürün müşteriye sunulmalıdır.

Bu açıklamaları kısaca şu şekilde yeniden yazabiliriz:

1. Müşteriye ürün listesini göster.
2. Müşterinin seçimini oku.
3. Seçim geçerli ise ve ürününden bulunuyor ise ürün satışını yap.

Burada açıklanan işlemleri üç ayrı fonksiyon ile yapabiliriz. İlk fonksiyon ürünü listesini sunan **urunListele()** fonksiyonu, satış işlemini gerçekleştirmek için kullanılacak olan **urunSat()** fonksiyonu ve son olarak da **main()** fonksiyonu. İlk olarak **urunListele()** fonksiyonunu yazacağız.

```
void urunListele()
{
    cout << "***** HOSGELDINIZ *****" << endl;
    cout << "Sectiginiz ürünün yanında bulunan sayisi giriniz." << endl;
    cout << "Sekersiz çay (1.00 TL) için 1 giriniz." << endl;
    cout << "Şekerli çay (1.25 TL) için 2 giriniz." << endl;
    cout << "Sade kahve (1.50 TL) için 3 giriniz." << endl;
    cout << "Sade sütlü kahve (1.75 TL) için 4 giriniz." << endl;
    cout << "Şekerli sade kahve (1.75 TL) için 5 giriniz." << endl;
    cout << "Şekerli sütlü sade kahve (2.00 TL) için 6 giriniz." << endl;
    cout << "*****" << endl;
}
```

Sıradaki fonksiyon **urunSat()** fonksiyonudur. Bu fonksiyon müşteri seçimi yaptıktan sonra servis bölümünde seçilen ürünün bulunup bulunmadığı kontrol edilecektir. Eğer seçilen ürün bulunmuyor ise, müşteriye seçtiği ürünün sunulmadığını belirten bir mesaj döndürecektr. Aksi durumda müşterinden ürünün parasını ödemesi istenecektir.

Müşteri seçtiği ürünün ücretinin tamamını ödememiş ise eksik olan miktarın ödenmesi için bildirimde bulunulacaktır. Eğer müşteri eksik olan kısmını ödemesi için uyarılacaktır. Eğer ödemeyi tamamlamaz ise, bu durumda ödediği para iade edilecektir. Müşteri ücreti tam olarak ödemmiş ise ürün kendisine teslim edilecek, ürün sayısını bir azaltılacak ve kasadaki para miktarı güncellenecektir.

Buradaki açıklamalardan anlaşılacağı üzere, **urunSat()** fonksiyonu **satisIslemi** ve **cihaz-Kasa** sınıflarına erişebilmesi gereklidir. Bu nedenle bu fonksiyonun iki adet parametresi olacaktır. Bu iki parametre ilişkili oldukları sınıfların üyelerine erişmek için kullanılacağından dolayı referans parametresi olarak tanımlanacaktır.

Buna göre söz konusu fonksiyonun algoritması aşağıdaki gibi olacaktır.

- 1 Eğer ürün bulunuyor ise;
  - 1.1 Ürünün ücretini göster ve parasını ödemesini belirt.
  - 1.2 Ödenen para miktarını oku.
  - 1.3 Eğer müşterinin ödediği miktar ücretten az ise;
    - 1.3.1 Müşterinin eksik olan miktarı ödemesini belirt.
    - 1.3.2 Müşteri tarafından ödenen paranın miktarını hesapla

1.3.3      Eğer ödenen para miktarı ücrete eşit ise;

    1.3.3.1     Kasadaki para miktarını güncelle.

    1.3.3.2     Ürünün stok adedini bir azalt.

    1.3.3.3     Mesaj döndür.

1.4      Eğer müşterinin ödediği miktar ücrete eşit değilse, parasını geri ver.

Buna göre urunSat() fonksiyonu aşağıdaki gibi olacaktır.

```
void urunSat(satisIslemi& urun, cihazKasa& odeme)
{
    float miktar           // girilen para miktarı için ilk değişken
    float miktar2          // girilen para miktarı için ikinci değişken
    if (urun.urunSayisiniOku() > 0)      // seçilen ürünün stokta olup olmadığı kontrolü
    {
        cout << "Lütfen " << urun.ucretOku() << "lira ödeyiniz." << endl;
        cin >> miktar;
        if (miktar < urun.ucretOku())
        {
            cout << "Lütfen " << urun.ucretOku() - miktar << " daha ödeyiniz."
<<endl;
        }
        cin >> miktar2;
        miktar = miktar + miktar2;
    }
    if (miktar >= urun.ucretOku())
    {
        odeme.nakitKabul(miktar);
        urun.satisIslemi();
        cout << "Ürününüüz aşağıdan alınız. Bizi tercih ettiğiniz için teşekkürler." << endl;
    }
    else
        cout << "Yapılan ödeme eksiktir. Lütfen ödedığınız ücreti para iade bölümünden alınız." << endl;
}
```

```

    }

else

    cout << "Seçtiğiniz ürün elimizde bulunmamaktadır." << endl;

}

```

Yukarıda diğer fonksiyonlar tanımlandıktan sonra **ana fonksiyonun – main** – tanımlanması aşamasına geçilir. Bu fonksiyonun algoritması aşağıdaki gibidir.

- 1 **cihazKasa** sınıfına erişmek için kullanılacak olan bir nesne yarat.
- 2 Altı adet **satisIslemi** sınıfına ait nesne yarat ve bunları etkinleştir.
- 3 Varsa gereken diğer değişkenleri tanımla.
- 4 **urunListele()** fonksiyonunu çalıştır.
- 5 Müşterinin seçimini oku.
- 6 Çıkış için 0 basılmadığı sürece;
  - 6.1 Ürün satışı için **urunSat()** fonksiyonunu çağır.
  - 6.2 Müşterinin seçim yapabilmesi için **urunListele()** fonksiyonunu çağır.
  - 6.3 Müşterinin seçimini oku.

```

int main()

{
    cihazKasa sayac;

    servisIslemler sekersizCay (100, 1);

    servisIslemler sekerliCay (100, 1.25);

    servisIslemler sadeKahve (100, 1.50);

    servisIslemler sekerliKahveCay (100, 1.75);

    servisIslemler sutluKahve (100, 2);

    servisIslemler sutluSekerliKahve (100, 2.25);

    int secim      // müşterinin seçimini okumak için değişken

    urunListele();

    cin >> secim;

    while (secim !=0)

    {
        switch (secim)

        {

```

```

case 1:
    urunSat(sekersizCay, sayac);
    break;

case 2:
    urunSat(sekerliCay, sayac);
    break;

case 3:
    urunSat(sadeKahve, sayac);
    break;

case 4:
    urunSat(sekerliKahve, sayac);
    break;

case 5:
    urunSat(sutluKahve, sayac);
    break;

case 6:
    urunSat(sutluSekerliKahve, sayac);
    break;

default:
    cout << "Geçersiz seçim yaptınız." << endl;
}

urunListele();
cin >> secim;
}

return 0;
}

```

Yukarıda verilen program örneğinde sınıflar **otomat.h** dosyasında uygulanmıştır. Aşağıda verilmiştir.

```

/*
 * otomat.h
 *

```

\*

\* Copyright 2024 Goksin Akdeniz <goksin.akdeniz@gmail.com>

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM, VERİ

\* VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

```

*/
class cihazKasa
{
public:
    float kasaNakitMiktar() const;
    /*
     * Kasada o anda bulunan nakit miktarını döndürür.
     * işlem sonrası koşullar:
     * kasaNakitToplam değerini döndürür.
    */
    void nakitKabul (float alinanNakit)
    /*
     * Müşteri tarafından ödenen ücreti kasadaki nakit ile toplayarak kasadaki
     * para miktarını güncelleyen fonksiyon
     * İşlem sonrası koşullar:
     * kasaNakitToplam = kasaNakitToplam + alinanNakit;
    */
    cihazKasa (float kasaDevir = 500)
    /*
     * Yapıçı fonksiyondur. Değişkeni etkinleştirilmekte ve ön tanımlı değeri atamaktadır.
     * İşlem sonrası koşullar:
     * kasaNakitToplam = kasaDevir
    */
private:
    float kasaNakitToplam;      // Kasada bulunan nakit miktarıdır.
};

```

Yukarıda verilen program örneğinde fonksiyonlar **otomat.cxx** dosyasında uygulanmıştır. Aşağıda verilmiştir.

```

/*
 * otomat.cxx
 *

```

\*

\* Copyright 2024 Goksin Akdeniz <goksin.akdeniz@gmail.com>

\* \*

\* Her hakkı saklıdır.

\* Değişiklik yapılarak veya yapılmaksızın, yeniden dağıtmak veya kaynak

\* ve ikili biçimlerde kullanmak, aşağıdaki koşullar yerine getirildiği

\* takdirde serbesttir:

\*

\* Kaynak kodun yeniden dağıtımını; yukarıdaki telif hakkı uyarısını,

\* şartlar listesini ve aşağıdaki ret yazısını muhafaza etmelidir.

\*

\* İkili biçimde yeniden dağıtımında; yukarıdaki telif hakkı uyarısı,

\* şartlar listesi ve aşağıdaki ret yazısı dağıtımla birlikte gelen

\* belgelendirmede ve/veya diğer materyallerde tekrarlanmalıdır.

\*

\* İŞBU YAZILIM TELİF HAKKI SAHİPLERİ VE KATKIDA BULUNANLAR TARAFINDAN

\* "OLDUĞU GİBİ" SAĞLANMIŞTIR VE TİCARETE ELVERİŞLİİK VE ÖZEL BİR AMACA

\* UYGUNLUK İÇİN KAPALI BİR TAAHHÜT DE DAHİL OLMAK ÜZERE VE BUNUNLA

\* KISITLI OLMAKSIZIN HER TÜRLÜ AÇIK YA DA KAPALI TAAHHÜT

\* REDDEDİLMİŞTİR. HİÇBİR KOŞULDA TELİF HAKKI SAHİPLERİ VE KATKIDA

\* BULUNANLAR; HER NE SEBEPLE OLUŞMUŞSA VE SÖZLEŞMEDE, KUSURSUZ

\* SORUMLULUKTA, VEYA TAZMİNAT YÜKÜMLÜLÜĞÜNDE OLMAK ÜZERE HANGİ

\* YÜKÜMLÜLÜK KURAMINDA YER ALIRSA ALSIN, İŞBU YAZILIMIN KULLANIMIYLA

\* ORTAYA ÇIKAN DOĞRUDAN, DOLAYLI, TESADÜFİ, ÖZEL, CEZAİ VEYA BİR SEBEP

\* SONUCUNDA OLUŞAN (YEDEK PARÇA VEYA HİZMETLERİN TEMİNİ; KULLANIM, VERİ

\* VEYA RANDIMAN KAYBI; YA DA İŞ KESİNTİSİ DE DAHİL OLMAK ÜZERE VE

\* BUNUNLA KISITLI KALMAKSIZIN) HERHANGİ BİR ZARAR İÇİN, BU GİBİ

\* HASARLARIN OLASILIĞI HAKKINDA UYARILMIŞ OLSALAR BİLE, SİZE KARŞI

\* SORUMLU DEĞİLLERDİR.

\*

```

*/
#include <iostream>
#include <locale>
#include "otomat.h"
using namespace std;

float cihazKasa::kasaNakitMiktar() const
{
    return kasaNakitToplam;
}

void cihazKasa::nakitKabul( float alinanPara)
{
    kasaNakitToplam = kasaNakitToplam + alinanNakit;
}

cihazKasa::cihazKasa( float kasaDevir)
{
    if (kasaDevir => 0)
        kasaNakitToplam = kasaDevir;
    else
        kasaNakitToplam = 500;
}

float servisIslemler::urunSayisiOku() const
{
    return urunSayisi;
}

int servisIslemler::ucretOku() const
{
    return ucret;
}

void servisIslemler::satisIslemi()
{
    urunSayisi--
}

```

```

}

satisIslemi::satisIslemi( int urunSayisiAta, float ucretAta )
{
    if (urunSayisiAta => 0)
        urunSayisi = urunSayisiAta;
    else
        urunSayisiAta = 100;

    if (ucretAta => 0 )
        ucret = ucretAta;
    else
        ucret = 1;
}

void urunListele()
{
    cout << " ***** HOŞGELDİNİZ ***** " << endl;
    cout << "Seçtiğiniz ürünün yanında bulunan sayısını giriniz." << endl;
    cout << "Şekersiz çay (1.00 TL) için 1 giriniz." << endl;
    cout << "Şekerli çay (1.25 TL) için 2 giriniz." << endl;
    cout << "Sade kahve (1.50 TL) için 3 giriniz." << endl;
    cout << "Sade sütlü kahve (1.75 TL) için 4 giriniz." << endl;
    cout << "Şekerli sade kahve (1.75 TL) için 5 giriniz." << endl;
    cout << "Şekerli sütlü sade kahve (2.00 TL) için 6 giriniz." << endl;
    cout << " ***** " << endl;
}

void urunSat(satisIslemi& urun, cihazKasa& odeme)
{
    float miktar          // girilen para miktarı için ilk değişken
    float miktar2         // girilen para miktarı için ikinci değişken
    if (urun.urunSayisiniOku() > 0)      // seçilen ürünün stokta olup olmadığı kontrolü
    {
        cout << "Lütfen " << urun.ucretOku() << "lira ödeyiniz." << endl;
    }
}

```

```

cin >> miktar;

if (miktar < urun.ucretOku())
{
    cout << "Lütfen " << urun.ucretOku() - miktar << " daha ödeyiniz." << endl;
    cin >> miktar2;
    miktar = miktar + miktar2;
}

if (miktar >= urun.ucretOku())
{
    odeme.nakitKabul(miktar);
    urun.satisIslemi();
    cout << "Ürününüzü aşağıdan alınız. Bizi tercih ettiğiniz için teşekkürler." <<
endl;
}

else
{
    cout << "Yapılan ödeme eksiktir. Lütfen ödediğiniz ücreti para iade bölümünden alınız." << endl;
}

else
{
    cout << "Seçtiğiniz ürün elimizde bulunmamaktadır." << endl;
}

int main()
{
    setlocale (LC_ALL, "tr_TR-UTF-8");
    cihazKasa sayac;
    servisIslemler sekersizCay (100, 1);
    servisIslemler sekerliCay (100, 1.25);
    servisIslemler sadeKahve (100, 1.50);
    servisIslemler sekerliKahveCay (100, 1.75);
    servisIslemler sutluKahve (100, 2);
    servisIslemler sutluSekerliKahve (100, 2.25);
}

```

```
int secim      // müşterinin seçimini okumak için değişken
urunListele();
cin >> secim;
while (secim !=0)
{
    switch (secim)
    {
        case 1:
            urunSat(sekersizCay, sayac);
            break;
        case 2:
            urunSat(sekerliCay, sayac);
            break;
        case 3:
            urunSat(sadeKahve, sayac);
            break;
        case 4:
            urunSat(sekerliKahve, sayac);
            break;
        case 5:
            urunSat(sutluKahve, sayac);
            break;
        case 6:
            urunSat(sutluSekerliKahve, sayac);
            break;
        default:
            cout << "Geçersiz seçim yaptınız." << endl;
    }
    urunListele();
    cin >> secim;
}
```

```
    retrun 0;  
}
```

## 10.Kalıtım ve Birleşim

Önceki bölümde soyut veri tipi kavramı tanımlanmış ve C++ programlama Dili içerisinde soyut veri tipinin nasıl uygulanacağı ele alınmıştı. Sınıf ile veri ve bu veri üzerinde gerçekleştirilen tüm işlemler tek bir yapı içerisinde toplanmaktadır. Bu yapı nesne olarak tanımlanmaktadır. Nesne tasarıımı sayesinde bağımsız bir yapıya dönüştürmektedir. İşlemler doğrudan veriye erişebilir ve üzerinde çalışabilirken nesnenin yapısı bu işlemlerden etkilenmemektedir.

Sınıf soyut veri tipinin uygulanmasına olanak verirken aynı zamanda ek özellikler de sunmaktadır. Örneğin tanımlanmış olan bir sınıf varsa, bu sınıfın yararlanarak, var olan sınıfı temel alan yeni sınıflar türetilabilir. Böylelikle bir yazılmış olan bir kaynak kod tekrar yeniden yazılmaya gerek kalmadan tekrar kullanılabilir. C++ Programlama Dili’nde iki veya daha çok sayıdaki sınıf ile bir çok şekilde başka sınıflar arasında ilişki kurulabilir. Bunlar arasında **kalıtım (inheritance)** ve **bileşim (composition/aggregation)** en yaygın kullanılanlardır. Adlarından anlaşılacağı üzere kalıtım kullanılması durumunda temel alınan sınıfa ait değişkenler, fonksiyonlar yeni türetilen sınıf içerisinde de bulunmaktadır. Kalıtım bazı kaynaklarda temel alınan sınıftaki nesnelerin yeni sınıfta da kullanılmakta olduğuna işaret etmem amacıyla miras olarak tanımlanır. Bileşim için ise esas alınan sınıfa ait nesneler yeni sınıfta da bulunmaktadır. Ancak yeni sınıf bu nesneler ile sınırlı olmayıp başka nesnelere de sahip olabilir.

### 10.1.Kalıtım

Önceki bölümde yer verdığımız örneklerden birisinde personelin iş yerine giriş ve çıkış zamanını kayıt eden bir uygulamadan söz etmiştik. Burada iş yerinde çalışan personelin tam zamanlı ve yarı zamanlı çalışan olarak iki gruba ayırmayı düşündürmek istedik. Bu durumda `yariZamanliPersonel` adlı bir sınıf tanımlayabiliriz. Bu sınıfın üyeleri de adından anlaşılacağı üzere yarı zamanlı çalışan personelin adı ve soyadı, saatlik brüt ücreti ile aylık çalışma süresi olsun. Önceki bölümde buna benzer olarak bir personel sınıfı tanımlamıştık. Bu sınıfı kullanarak personelin adını ve soyadını değişken olarak tanımlamıştık. Yarı zamanlı çalışan personel için yazılmış olan program içerisinde yeni bir sınıf tanımlamak yerine var olan `personelBilgileri` sınıfını kullanacağımız. Bunun içinde sınıfa yeni üyeleri (veri ve/veya fonksiyonlar) ekleyeceğiz.

Burada sınıfa eklenecek yeni üyeleri için sınıfı yeniden tasarlamak ve yazmak yerine `yariZamanliPersonel` sınıfını yaratıp sadece gerekli olan üyeleri tanımlayacağımız. Örneğin `personelBilgisi` sınıfı personeli ait olan ad ve soyadı barındıran üyelerde sahip olduğu için bu üyeleri yeni yaratılan sınıfta yer almamaktadır. Bu üyeleri doğrudan `personelBilgilei` sınıfından kalıtım ile alınamaktadır. Bunu ilerideki örnekte göreceğiz.

Önceki bölümde `zamanUygulaması` adını verdığımız sınıfı incelemiştir ve bir program içerisinde sistem saatinden bağımsız olarak bir uygulamada saat, dakika ve saniye değerlerini tanımlamıştık. Örnekte saat, dakika ve saniye olarak üç üye tanımlanmıştır. Bazı uygulamalarda bu değerlerin yanı sıra zaman dilimi bilgisinin de saklanması gerekebilir. Eğer zaman dilimi bilgisinin de barındırılması gerekiyor ise, `zamanUygulaması` sınıfının kapsamının genişletilmesi gerekecektir. Yeni bir değişkenin eklenmesi durumunda `zamanUygulaması` sınıfından türetilen ve zaman dilimi bilgisini de kapsayan genişletilmiş bir zaman uygulamasını belirtmek amacı ile

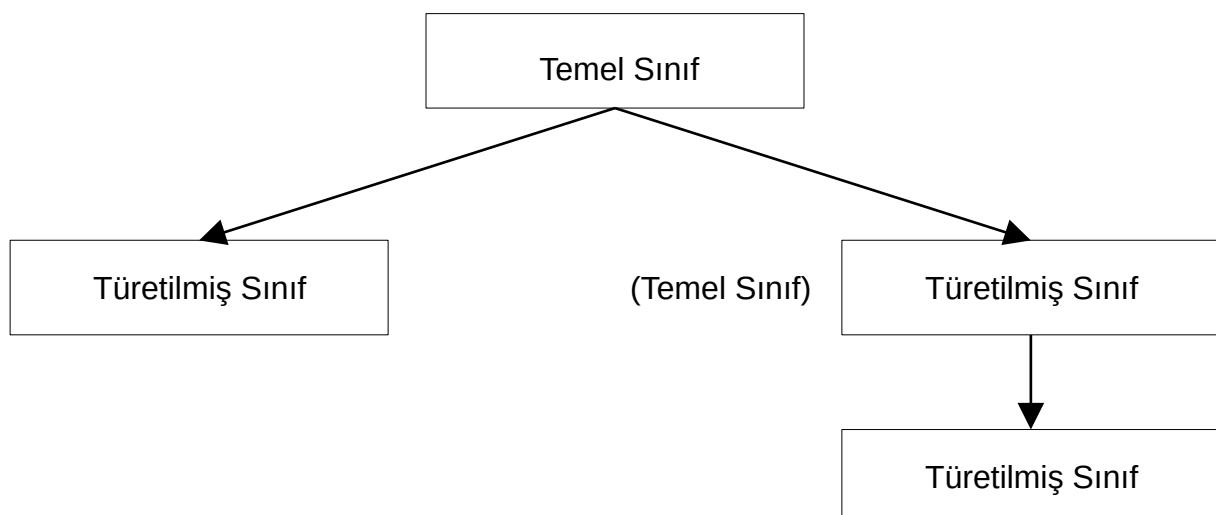
`zdZamanUygulaması` adlı sınıfı türetebiliriz. Bu işlem için `zamanDilimi` adlı bir üye değişken ile gerekli olan diğer fonksiyonlar tanımlanabilir. C++ Programlama Dili’nde bunu gerçekleştirmemizi sağlayan mekanizmaya kalıtım adı verilir. Kalıtım mekanizmasının ana fikri, türetilen sınıfın üyelerinin de temel alınan sınıf üyeleri ile aynı özelliklere sahip olduğunu.

Eğer `personelBilgisi` sınıfından `yariZamanliPersonel` sınıfı türetilmiş ise şunu söylemek olanaklıdır:

“Tam zamanlı olarak çalışanlar personeldir. Yarı zamanlı olarak çalışanlar da personeldir.”

Kalıtım ile var olan sınıflardan yeni sınıflar türetebiliriz. Bu durumda elde edilen yeni sınıflara **türetilmiş sınıflar (derived classes)** adı verilir. Yeni sınıfları oluşturmak için yararlandığımız var ola sınıflar ise **temel sınıf (base class)** olarak isimlendirilir. Böylece her seferinde temelden başlayarak yeni bir sınıf tasarlamak yerine kalıtmadan yararlanarak yazılımın karmaşıklığını azaltmış oluruz.

Her türetilmiş sınıf aynı zamanda bir diğer sınıfın türetilmesi için temel sınıf da olabilir. Bu durumda kalıtım **tekil kalıtım (single inheritance)** veya **çoklu kalıtım (multiple inheritance)** söz konusudur. Tekil kalıtımından söz edildiğinde temel sınıfın tek bir yeni sınıfı türetilmiş olurken, çoklu kalıtımında ise temel sınıfın türetilen yeni sınıf temel alınarak yeni bir sınıf türetilmiş olmaktadır. Bu bölümde tekil kalıtım üzerinde durulacaktır.



**Şekil 11.1:** Kalıtım hiyerarşisi

Kalıtım hiyerarşik olarak bir ağacın gövdesi ve dalları gibi bir yapı olarak gösterilebilir. Ağacın gövdesi temel sınıf olarak tanımlanırken her türetilen sınıf yeni bir dal olarak temsil edilebilir. Eğer türetilmiş bir sınıf esas alınarak yeni bir sınıf daha türetilerek olursa esas alındığı türetilmiş sınıf ise temel sınıf olarak tanımlanır. Şekil 11.1’de bu durum gösterilmiştir.

Türetilmiş bir sınıfın tanımlanması aşağıda gösterildiği gibi yapılır.

```
class sınıfAdı: üyelikDurumuTanımlaması temelSınıfınAdı
{
    sınıfınÜyeleri
```

}

Yukarıdaki tanımlanadan görüleceği üzere üyelikDurumuTanımlaması için **genel (public)**, **özel (private)** ve **korumalı (protected)** olmak üzere üç durum söz konusudur. Eğer üyelikDurumuTanımlaması alanında herhangi bir tanımlama yapılmaz ise **özel (private)** olduğu kabul edilir. **Korumalı (protected)** olması durumunu sonra ele alacağız.

**Örnek:** geometrikSekil adını verdigimiz bir sınıfı tanımladığımız var sayalım. Bu sınıfın yararlanarak genel kalıtım ile daire adlı sınıfı tanımlayalım.

```
class daire: public geometrikSekil
{
    ...
};
```

Eğer bu sınıfı aşağıdaki tanımlanmış olsa idi, özel kalıtım ile tanımlanmış olacaktır.

```
class daire: private geometrikSekil
{
    ...
};
```

Bu şekilde geometrikSekil sınıfının genel üyeleri daire sınıfının özel üyeleri olarak tanımlanmış olacaktır. Eğer yukarıdaki tanımlamada üyelik durumunu belirtmek için “private” veya “public” tanımlaması yapılmamış olsa idi, türetilen sınıfın üyelerinin tamamı var sayılan olarak özel olarak tanımlanmış olacaktır. Yukarıdaki tanımlama ile aşağıdaki ile aynıdır.

```
class daire: geometrikSekil
{
    ...
};
```

---

Türetilmiş sınıflar tasarlarken ve yazılırken aşağıdaki noktalara dikkat etmek gerekmektedir.

1. Eğer temel sınıfın üyeleri özel ise, bu üyeleri daima özel olma durumlarını korurlar. Dolayısı ile de türetilmiş sınıfın üyeleri bu özel üyelerle erişemezler. Aslında temel sınıfın üyeleri aynı zamanda türetilmiş sınıfın üyeleri olmakla birlikte türetilmiş sınıfın üyeleri temel sınıfın özel üyelerine doğrudan erişemezler.
2. Temel sınıfın genel üyeleri türetilmiş sınıfın özel veya genel üyeleri olarak miras alınır. Buna göre türetilmiş sınıfın miras aldığı temel sınıfın genel üyeleri artık türetilmiş sınıf için genel veya özel üye olacaktır.
3. Türetilmiş sınıf yapısında ilave veri ve/veya fonksiyonlar barındırabilir.

4. Türetilmiş sınıf, temel sınıfın genel üyesi olarak tanımlanmış olan fonksiyonlarını yeniden tanımlayabilir. Bu türetilmiş sınıfın, temel sınıfın üyesi olan aynı adlı bir fonksiyona sahip olabileceği anlamına gelmektedir. Bu durumda her iki sınıfta aynı ada, aynı parametrelere sahip iki fonksiyon bulunacaktır. Ancak kural olarak bu fonksiyonların kaynak kodu aynı olmayacağıdır. Bu kural sadece yeniden tanımlanan nesneler için geçerli olup temel sınıftaki nesneleri etkilememektedir.
5. Temel sınıfın üyeleri olan değişkenler aynı zamanda türetilmiş sınıfın da üyesi olan değişkenlerdir. Benzer olarak temel sınıfın üyesi olan fonksiyonlar aynı zamanda türetilmiş sınıfında üyesi olan fonksiyonlardır. Burada dikkate edilmesi gerek nokta eğer türetilmiş sınıfta aynı isim ve parametreler sahip olan bir fonksiyon tekrardan tanımlanmamış olmalıdır. Burada eğer temel sınıfın üyesine türetilmiş sınıftan erişilmek istenirse ilk kural daima göz önünde tutulmalıdır.

---

İzleyen iki bölüm kalıtım ile ilgili iiki önemli noktayı vurgulamaktadır. İlk nokta yukarıda sözü edilen temel sınıfın üyesi olan bir fonksiyonun türetilmiş sınıf içerisinde tekrardan fonksiyon olarak tanımlanmasıdır. Bu konuya incelerken ayrıca temel sınıfın özel üye olarak tanımlanmış değişkenlerine türetilmiş sınıftan nasıl erilebileceğini de inceleyeceğiz. Ele alacağımız ikinci nokta da türetilmiş sınıfın yapısında bulunan yapıçı fonksiyonlarının temel sınıfın özel tanımlı üyelerine erişemeyecek olmasıdır. Bu durumda temel sınıftan kalıtım ile edinilen özel üyelerin türetilmiş sınıfın yapıçı fonksiyonu çalıştırıldığında etkinleştirilmesi gerekecektir.

### 10.1.1.Temel Sınıfın Üyesi Olan Fonksiyonların Yeniden Tanımlanması

Bir temel sınıftan türetilmiş olan bir türetilmiş sınıf bulduğunu varsayılmı. Her iki sınıf yapısında üyesi olan değişkenleri barındırıyor olsun. Bu durumda türetilmiş olan sınıfın kendi üyesi olan değişkenleri yanında temel sınıftan kalıtım ile aldığı üye değişkenler de olacaktır. Burada türetilen sınıfın üyesi olan ve sınıfın üyesi olan değişkenleri yazdırın bir üye fonksiyon de bulunuyor olsun. Bu fonksiyona istenilen isim verilebileceği gibi temel sınıfta da aynı isme sahip olan bir fonksiyon ile aynı isim de verilebilir. Bu şekilde temel sınıfın fonksiyonunun tekrardan tanımlamasına **yeniden tanımlama (redefining/overriding)** adı verilir.

Bir temel sınıfın genel olarak tanımlanmış üye fonksiyonun türetilmiş sınıf tarafından yeniden tanımlanabilmesi için fonksiyonların isimlerinin, parametrelerinin ve veri tiplerinin aynı olması gereklidir. Bir diğer deyişle temel sınıf ile türetilmiş sınıfın üyesi olan fonksiyon her iki sınıfta aynı isim ve veri tipindeki parametreler ile tanımlanmış ise bu durumda **yeniden tanımlama (redefining)** durumu söz konusudur. Ancak temel sınıfın üyesi olan bir fonksiyonun adı ile türetilmiş bir sınıfın üyesi olan fonksiyonun adı aynı olurken parametreleri farklı ise bu durumda **aşırı yükleme (overloading)** söz konusudur.

Aşağıdaki örnekte fonksiyonların yeniden tanımlaması gösterilmiştir.

```
class dortgenSekil
{
public:
```

```
void boyutlariAta(double h, double w);
```

```
// dörtgenin en ve boy değerlerini atayan fonksiyondur.
```

```
// işlem sonrasında en = w; boy = w;
```

```
double okuBoy() const;
```

```
// dörtgenin boyunu döndüren fonksiyon
```

```
// işlem sonrasında boy değeri döndürülür
```

```
double okuEn() const;
```

```
// dörtgenin enini döndüren fonksiyon
```

```
// işlem sonrasında en değeri döndürülür
```

```
double alan() const;
```

```
// dörtgenin alanı döndürülür
```

```
// işlem sonrasında dörtgenin alanı hesaplanmış olur.
```

```
double çevre() const;
```

```
// dörtgenin çevresini hesaplayan fonksiyon
```

```
// dörtgenin çevresinin değeri döndürülür
```

```
void yaz() const;
```

```
// dörtgenin eni ve boyunu yazdırın fonksiyon
```

```
dortgenSekil();
```

```
// varsayılan yapıcı fonksiyon
```

```
// işlem sonrasında boy = 0; en = 0;
```

```
dortgenSekil(double l, double w);
```

```
// parametreleri ile birlikte yapıcı fonksiyon tanımlaması
```

```
// işlem sonrasında boy = l; en = w;
```

```

private:
    double boy;
    double en;
};

```

Aşağıda yukarıda tanımlanan sınıfı ait UML diyagramı görülmektedir.

dortgenSekil
<ul style="list-style-type: none"> <li>- en: double</li> <li>- boy: double</li> </ul>
<ul style="list-style-type: none"> <li>+ boyutlariAta(double, double): void</li> <li>+ okuBoy() const: double</li> <li>+ okuEn() const: double</li> <li>+ alan() const: double</li> <li>+ çevre() const: double</li> <li>+ yaz() const: void</li> <li>+ dortgenSekil()</li> <li>+ dortgenSekil(double, double)</li> </ul>

Yukarıdaki UML diyagramından görüldüğü üzere dortgenSekil adlı sınıfın on adet üyesi bulunmaktadır.

Yukarıda tanımlanan sınıfın üyesi olan fonksiyonların da aşağıdaki tanımlanmış olduğunu varsayıyalım.

```

void dortgenSekil::boyutlariAta(double l, double w)
{
    if (l >= 0)
        boy = l;
    else
        boy = 0;
    if (w >= 0)
        en = w;
    else
        en = 0
}
double dortgenSekil::okuBoy() const
{

```

```

    return boy;
}

double dortgenSekil::okuEn()const
{
    return en;
}

double dortgenSekil::alan() const
{
    return en * boy;
};

double dortgenSekil::cevre() const
{
    return 2 * (en + boy);
}

void dortgenSekil::yaz() const
{
    cout << "Boyu = " << boy;
    cout << "Eni = " << en;
}

dortgenSekil::dortgenSekil(double l, double w)
{
    boyutlariAta(l, w);
}

dortgenSekil::dortgenSekil()
{
    boy = 0;
    en = 0;
}

```

Buradan hareketle bir kutuSekli adında bir sınıfı dorgenSekil sınıfından türetelim.

```

class kutuSekli: public rectangleType
{

```

```
public:
```

```
    void boyutlariAta(double l, double w, double h);  
    // kutunun en, boy ve yüksekliği atayan fonksiyon  
    // işlem sonrasında boy = l; en = w; yükseklik = h;
```

```
    double okuYuksekli() const;  
    // kutunun yükseliğini okuyan fonksiyon  
    // işlem sonrasında kutunun yükseliğini döndürür
```

```
    double alan() const;  
    // kutunun yüzey alanını döndüren fonksiyon  
    // işlem sonrasında kutunun yüzey alanı hesaplanır ve döndürülür.
```

```
    double hacim() const;  
    // kutunun hacmini hesaplayan fonksiyon  
    // işlem sonrasında kutunun hacmi döndürülür
```

```
    void yaz() const;  
    // Boy, en yükseklik yazdırın fonksiyon
```

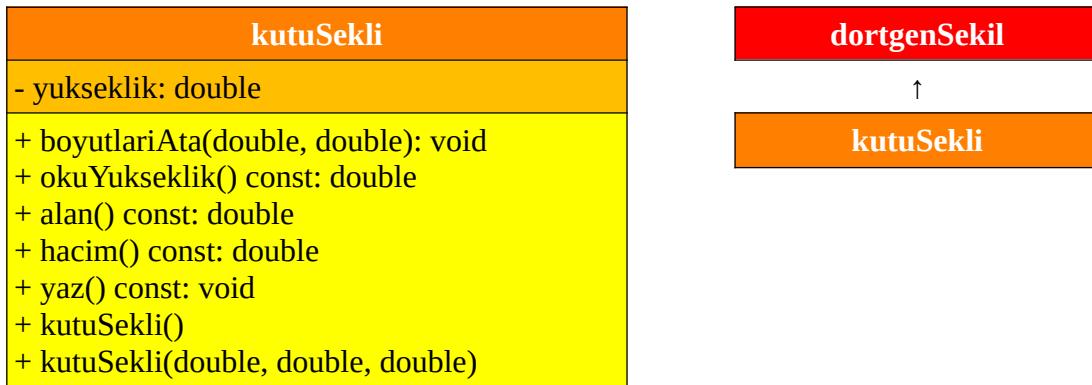
```
    kutuSekli();  
    // varsayılan yazıcı fonksiyon  
    // işlem sonrasında boy = 0; en = 0; yükseklik = 0;
```

```
    kutuSekli(double l, double w, double h);  
    // parametreleri ile birlikte yapıcı fonksiyon tanımlaması  
    // işlem sonrasında boy = l; en = w; yükseklik = h;
```

```
private:
```

```
    double yükseklik;
```

Aşağıda kutuSekli adlı sınıfın UML diyagramı görülmektedir. UML diyagramında kalıtım hiyerarşisi gösterilmektedir.



Yapılan kutuSekli sınıfı tanımlamasından görüleceği üzere bu sınıf dortgenSekil adlı sınıfın genel olarak kalıtım ile türetilmiştir. Bundan dolayı dortgenSekil sınıfının tüm genel üyeleri aynı zamanda kutuSekli adlı sınıfın genel üyeleri durumundadır. Ayrıca kutuSekli sınıfı yaz ve alan fonksiyonlarını yeniden tanımlamaktadır.

Genel olarak bir türetilmiş sınıfın üyesi olan bir fonksiyonun yeniden yazılması söz konusu ise temel sınıfındaki üye fonksiyona erişmek için şunlar yapılmalıdır:

- Eğer türetilen sınıfın temel sınıfındaki fonksiyon yeniden tanımlıyor ise genel üye olarak tanımlı olan fonksiyona erişmek için kapsam çözümleme operatörü kullanılır. Öncelikle temel sınıfın adı ve ardından kapsam çözümleme operatörü ve fonksiyonun adı ile parametreleri yazılır. Örneğin dortgenSekil sınıfındaki alan fonksiyonuna erişmek için “dortgenSekil::alan()” yazılmalıdır.
- Eğer türetilen sınıf temel sınıfın genel üyesi olan fonksiyonu yeniden yazmıyor ise, doğrudan temel sınıfın üyesi olan fonksiyonun adı ve uygun parametreler kullanılarak çağrılır. Burada aslında fonksiyonun yeniden yazılması değil aşırı yüklenmesi durumu söz konusudur. Kullanılan derleyicinin önceliklerine göre burada dikkat edilemesi gerek nokta şudur: Eğer bir türetilmiş sınıf, temel sınıfın üyesi olan bir fonksiyonu aşırı yükleme ile çağrıyor ise temel sınıfın adı ile birlikte kapsam çözümleme operatörü ile fonksiyonun adı yazılarak çağrılmalıdır. Örnek olarak kutuSekli sınıfında tanımlanan boyutlariAta fonksiyonunu incelenebilir.

Sıradaki işlem olarak kutuSekli adlı sınıfın yaz fonksiyonu yazalım. Söz konusu kutuSekli adlı sınıfın üyesi olan üç değişken bulunmaktadır; en, boy ve yükseklik. Türetilmiş sınıfın yaz fonksiyonu yazılırken şu noktalara dikkat edilmesi gereklidir.

- En ve boy, dortgenSekil adlı sınıfın özel üyeleridir. Bu nedenle türetilmiş sınıf olan kutuSekli adlı sınıf içerisindeki doğrudan erişilemezler.
- Temel sınıf olan dortgenSekil için en ve boy özel olarak tanımlı olduğu için doğrudan bu sınıfın ait olan genel üye fonksiyonları ile erişilebilirler. Bu nedenle de türetilmiş sınıfın üyesi olan yaz adlı fonksiyon yazılırken önce dortgenSekil adlı sınıfın yaz adlı fonksiyonu

çağrularak en ve boy yazdırılacak ardından da kutuSekli adlı sınıfın üyesi olan yükseklik yazdırılacaktır.

Türetilmiş sınıfın üyesi olan yaz fonksiyonu yazılırken temel sınıfın yaz fonksiyonu yukarıda tanımlandığı gibi yazılacaktır. Öncelikle temel sınıfın yaz fonksiyonu çağrılabilecek ve bu fonksiyonun işlenmesinin ardından türetilmiş fonksiyonun üyesi olan değişkenlere ait veri yazılacaktır.

```
void kutuSekli::yaz() const
{
    dortgenSekil::yaz();           \\ temel sınıfın yaz fonksiyonu
    cout << "Yükseklik = " << yukseklik;
}
```

Bundan sonra da kutuSekli sınıfının geri kalan üye fonksiyonlarını yazalım. Öncelikle boyutlar, Ata fonksiyonu ile başlayalım.

```
void kutuSekli::boyutlariAta(double l, double w, double h)
{
    dortgenSekli::boyutlariAta(l, w);
    if (h >= 0)
        yukseklik = h;
    else
        yukseklik = 0;
}
```

Yukarıda dikkat edilmesi gereken nokta boyutlariAta fonksiyonu yazılırken, temel sınıfın üyesi olan fonksiyonun çağrılmamasından önce türetilmiş olan sınıfın fonksiyonu kapsam çözümleme operatörü ile yazılmıştır. Burada bir fonksiyonun yeniden yazılması söz konusu olmamakla birlikte kutuSekli sınıfı aynı fonksiyonu aşırı yükleme ile çağırılmakta olduğuna dikkat edilmelidir.

Türetilmiş sınıfın diğer fonksiyonu olan okuYukseklik fonksiyonu aşağıdaki gibidir.

```
double kutuSekli::okuYukseklik() const
{
    return yukseklik;
}
```

Seklin yü<ey alanı kutuSekli sınıfının üyesi olan alan fonksiyonu ile hesaplanmaktadır. Bunun içinde yükseklik yanında en ve boy verisine gereksinim vardır. Bunlar ise dortgenSekil adlı sınıfın genel üyeleridir. Temel sınıfın okuBoy ve okuEn fonksiyonlarına erişilmesi için özel bir tanımlama yapılmasına gerek yoktur.

```
double kutuSekli::alan() const
{
    return 2 * (okuBoy() * okuEn() + okuBoy() * yukseklik + okuEn() * yukseklik);
}
```

Sıradaki fonksiyon ise kutunun hacmini hesaplayan fonksiyondur. Kutunun taban alanı en ile boyunun çarpımından bulunduktan sonra yükseklik ile çarpılarak sonuç elde edilecektir. Şimdi fonksiyonu yazalım. Bunun için temel sınıfın üyesi olan alan fonksiyonunu kullanacağız. Burada türetilmiş sınıf temel sınıfta tanımlı olan fonksiyonu yeniden yazmakta olduğu için temel sınıfın adı ve kapsam çözümleme operatörü ile birlikte fonksiyonun adı birlikte yazılacaktır. Ardın da yükseklik ile çarpılacaktır.

```
double kutuSekli::hacim() const
{
    return dortgenSekli::alan() * yukseklik;
}
```

Sonraki bölümde bir türetilmiş sınıfın yazıcı fonksiyonu yazılırken temel sınıfın yazıcı fonksiyonuna nasıl erişilebileceğini inceleyeceğiz.

### 10.1.2.Türetilmiş ve Temel Sınıfların Yapıçı Fonksiyonları

## EK A: ASCII Tablosu

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	\$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

## Kaynakça

G. Booch, Object-Oriented Analysis and Design, 2nd ed., Addison-Wesley, Reading, MA, 1995.

E. Horowitz, S. Sahni, and S. Rajasekaran, Computer Algorithms C11, Computer Science Press, 1997.

N.M. Josuttis, The C11 Standard Library: A Tutorial and Reference, Addison-Wesley, Reading, MA, 1999.

D.E. Knuth, The Art of Computer Programming, Volume 1: Fundamental Algorithms, 3rd ed., Addison-Wesley, Reading, MA, 1997.

D.E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd ed., Addison-Wesley, Reading, MA, 1998.

D.E. Knuth, The Art of Computer Programming, Volume 3: Searching and Sorting, 2nd ed., Addison-Wesley, Reading, MA, 1998.

B. Stroustrup, The Design and Evolution of C11, Addison-Wesley, Reading, MA, 1994.

Dr. Yalçın Özkan, Nesneye Yönelik Programlama, Alfa Yayıncılık, İstanbul, Türkiye, 2009,

Steve Oualline, Pratik C++ Programlama, Pusula Yayıncılık, İstanbul, Türkiye, 2004

Fahrettin Erdinç, Mühendislik Öğrencileri İçin Temel Kılavuz C++/C, Abaküs Yayınları, 2017

Bülent Çobanoğlu, C/C++ ve JAVA Dilleriyle Algoritma ve Programlama, Abaküs Yayınları, 2018

Harvey M. Deitel, Paul J. Deitel, C ve C++, Sistem Yayıncılık, 2005